Reasoning about actual and hypothetical occurances of concurrent and non-deterministic actions

Chitta Baral, Michael Gelfond, Richard Watson
Department of Computer Science
University of Texas at El Paso
El Paso, Texas 79968 U.S.A.
{chitta,mgelfond,rwatson}@cs.utep.edu

Abstract

We propose extension $\mathcal{L}_2$ of the action description language $\mathcal{L}_1$ that can express both actual and hypothetical situations, concurrent execution of actions, observations of the truth values of fluents in these situations (as opposed to hypothetical values of fluents expressible in $\mathcal{A}$ and $\mathcal{A}_C$), observations of actual occurrences of (possibly non-deterministic combination of) actions. The corresponding entailment relation formalizes various types of common-sense reasoning about actions and their effects not modeled by the previous approaches. We then present a translation of domain descriptions in $\mathcal{L}_2$ to disjunctive logic programs.

1 Introduction

To perform nontrivial reasoning an intelligent agent situated in a changing domain needs the knowledge of causal laws that describe effects of actions changing the domain, the ability to observe and record occurrences of these actions, and the truth values of fluents[1] at particular moments of time. Discovery of methods of representing this kind of information in a form allowing various types of reasoning about the dynamic world and at the same time tolerant to future updates is one of the central problems of knowledge representation.

Recently, there have been several efforts towards **systematic** development of **provably correct** methods [GL92, San92, LLL$^+$94] to reason about actions. This paper can be viewed as a continuation of the work started in [GL92] where the authors introduced the high-level *action description language* $\mathcal{A}$ capable of expressing causal laws describing effects of actions as well as statements about values of fluents in possible states of the world.

[1] By fluents in this paper we mean propositions whose truth values depend on time.

In the last few years the syntax and semantics of $\mathcal{A}$ were expanded to allow descriptions of the effects of concurrent and non-deterministic actions as well as descriptions of global constraints expressing time independent relations between fluents [BG93, KL94a, BT94, Bar95, GL95, BT94, HT93, MT94]. We also have by now a collection of sound and often complete translations from domain descriptions in these languages into disjunctive, abductive, and equational logic programs [Dun93, DDS93, HT93, Tur94, ALP94]. This work (together with works based on direct axiomatizations of theories of actions in classical logic and its nonmonotonic extensions [PR93, Pin94, Rei91, LS91, MS94, Pro96]) helped increase understanding of the underlying ontological principles of reasoning about actions as well as advantages and limitations of general-purpose non-monotonic formalisms. This work also allowed the establishment of equivalence between some of the previously known theories of actions seemingly based on different intuitions, languages and logics [Kar93] and stimulated work on the theory and implementation of logic programming languages [AB91, Tur93, LT94, LMT93].

In [BGP95] Baral et al. propose a language inspired by $\mathcal{A}$, called $\mathcal{L}_1$, that can express actual situations, observations of the truth values of fluents in these situations (as opposed to hypothetical values of fluents expressible in $\mathcal{A}$), and observations of actual occurrences of actions. We believe that this adaption[2] significantly improves applicability of action description languages to various areas of AI.

The goal of this paper is to extend $\mathcal{L}_1$ even further to allow representation of concurrent and "non-deterministic" actions. Although our formalization of concurrent action in this paper is based on the language $\mathcal{A}_C$ [BG93], the proposed language $\mathcal{L}_2$ is not an extension of $\mathcal{A}_C$. This is because we do not allow hypothetical statements in domain descriptions of $\mathcal{L}_2$, while such statements were allowed in the domain descriptions of $\mathcal{A}_C$. On the other hand $\mathcal{L}_2$ allows actual observations. We describe the semantics of the proposed language $\mathcal{L}_2$ (defined as a natural extension of $\mathcal{L}_1$) and demonstrate, by way of examples, that the corresponding entailment allows one to capture various interesting forms of reasoning about actions. Our proposal differs from several other extensions of $\mathcal{A}$ dealing with concurrent and non-deterministic actions in the following way:

1. To the best of our knowledge this is the first proposal which allows reasoning about actual occurrences of concurrent actions.

2. Representation of non-deterministic actions is different (and we believe more natural in many cases) than those presented in the literature.

We also present a translation of domain descriptions in $\mathcal{L}_2$ into disjunctive logic programs [GL91]. The use of disjunctive logic programs is important because of the availability of interpreters [Wat94, WC93] for a large class of disjunctive programs.

We are currently working on investigating applicability of $\mathcal{L}_2$ to the design of architecture for intelligent agents capable of observing, planning, and acting in a changing environment and on discovering efficient methods of computing the corresponding entailment relation. Another direction of research is aimed at combining $\mathcal{L}_2$ with other extensions and dialects of $\mathcal{A}$.

[2] It should be noted that $\mathcal{L}_1$ is not an extension of $\mathcal{A}$; mainly because it does not allow hypothetical statements in its domain descriptions.

2 Syntax of $\mathcal{L}_2$

We will start with the description of a language $\mathcal{L}_2$ capable of expressing actual observations, compound actions, and hypotheses.

The alphabet of $\mathcal{L}_2$ consists of three disjoint nonempty sets of symbols $\mathcal{F}$, $\mathcal{A}$ and $\mathcal{S}$, called *fluents*, *actions*, and *actual situations*. Elements of $\mathcal{A}$ and $\mathcal{S}$ will be denoted by (possibly indexed) letters a and s respectively. We will also assume that $\mathcal{S}$ contains two special situations s_0 and s_N called *initial* and *current* situations. The 'N' in s_N corresponds to the word 'Now'.

A *fluent literal* is a fluent possibly preceded by $\neg$. Fluent literals will be denoted by (possibly indexed) letters f and p (possibly preceded by $\neg$). $\neg\neg f$ will be equated with f.

By an *action* we mean an arbitrary set of unit actions. A set $\{a_i\}$ where a_i is a unit action will be identified with a_i. Actions of the form $\{a_1, \ldots, a_n\}$ with $n > 1$ are called *compound* and interpreted as sets of actions which are performed concurrently and which start and stop contemporaneously. Unless otherwise specified, by an action we mean a compound action. An expression $[a_1, \ldots, a_n]$ will denote a sequence $a_1, \ldots, a_n$ of actions.

There are three kinds of propositions in $\mathcal{L}_2$; causal laws, facts and hypotheses.

An *effect law* is an expression of the form

$$a \ \textbf{causes} \ f \ \textbf{if} \ p_1, \ldots, p_n \tag{1}$$

where a is an action, and $f, p_1, \ldots, p_n$ $(n \geq 0)$ are fluent literals. $p_1, \ldots, p_n$ are called *preconditions* of (1). We will read this law as "f is guaranteed to be true after the execution of an action a in any state of the world in which $p_1 \ldots p_n$ are true". If $n = 0$, we write the effect law as $a \ \textbf{causes} \ f$ (1a)

An atomic *fluent fact* is an expression of the form

$$f \ \textbf{at} \ s \tag{2}$$

where f is a fluent literal and s is a situation. (Unless otherwise stated by situations we will mean actual situations.) The intuitive reading of (2) is "f is observed to be true in situation s".

An atomic *occurrence fact* is an expression of the form

$$\alpha \ \textbf{occurs\_at} \ s \tag{3}$$

where α is a sequence of actions, and s is a situation. It states that "the sequence α of actions was observed to have occurred in situation s". (We assume that actions in the sequence follow the next action in the sequence immediately).

An atomic *precedence fact* is an expression of the form

$$s_1 \ \textbf{precedes} \ s_2 \tag{4}$$

where s_1 and s_2 are situations. It states that situation s_2 occurred after situation s_1.

Propositions of the type (1) express general knowledge about effects of actions and hence are referred to as *laws*. Propositions (2), (3) and (4) are called *atomic facts* or *observations*. A *fact* is a propositional combination of atomic facts.

A hypothesis is an expression of the form

$$f \ \textbf{after} \ \ [a_1, \dots, a_n] \ \textbf{at} \ \ s \tag{5}$$

where a_i's are actions and s is a situation. The above expression reads as "Assuming that the sequence of actions $[a_1, \dots, a_n]$ occurs starting at the situation s, fluent f would be true in the resulting situation."

A collection of laws and facts is called a *domain description* of $\mathcal{L}_2$. (i.e. Domain descriptions do not include hypotheses, but hypotheses can be inferred from a domain description.) The sets of laws and facts of a domain description, D, will be denoted by D_l and D_f respectively. We will only consider domain descriptions whose propositions do not contain the situation constant s_N.

To see how the domain descriptions of $\mathcal{L}_2$ can be used to represent knowledge about actions let us consider the following example:

Example 2.1 Suppose that we are given a series of observations about $Fred$:

(a) when the water pistol was squirted, $Fred$ was seen to be $alive$ and dry,

(b) at the same time a shot was fired at $Fred$.

Suppose also that it is generally known that

(c) *squirting* makes $Fred$ wet, (d) *shooting* makes $Fred$ $dead$

The above information can be represented by a domain description D_1 consisting of the following propositions:

$$\left.\begin{array}{ll} (p1)\ alive\ \ \textbf{at}\ \ s_0 & (p2)\ dry\ \ \textbf{at}\ \ s_0 \\ (p3)\ squirt\ \ \textbf{occurs\_at}\ \ s_0 & (p4)\ s_0\ \ \textbf{precedes}\ \ s_1 \\ (p5)\ shoot\ \ \textbf{occurs\_at}\ \ s_0 & (p6)\ squirt\ \ \textbf{causes}\ \ \neg dry \\ (p7)\ shoot\ \ \textbf{causes}\ \ \neg alive & \end{array}\right\} D_1$$

To complete the description of D_1 we need to define its language. For simplicity we assume that this language contains only the fluents and actions explicitly mentioned in the propositions of D_1. Unless stated otherwise the same assumption will be made in other examples throughout this paper. □

Domain descriptions in $\mathcal{L}_2$ are used in conjunction with the following informal assumptions which clarify the description's meaning:

(a) Changes in the values of fluents can only be caused by execution of actions.

(b) There are no actions except those from the language of the domain description.

(c) Effects of an action (possibly compound) are specified by the causal laws in the domain description or inherited from its sub-actions.

(d) No actions occur except those needed to explain the facts in the domain description.

(e) Actions may happen concurrently.

These assumptions give a intuitive understanding of domain descriptions of $\mathcal{L}_2$.

Consider for instance domain description D_1 from Example 2.1. It is easy to see that D_1 together with assumption (d) implies that *squirt* and *shoot* are the only action which occur between s_0 and s_1. Using D_1 with the assumptions (a) - (e) we can conclude that at the moment s_1 *Fred* is *wet* and *dead*.

Our goal in this paper is to build a mathematical model which will help us to better understand and eventually mechanize these types of arguments. As the first step, we suggest a semantics of domain descriptions of $\mathcal{L}_2$ which precisely specifies the sets of acceptable conclusions which can be reached from such descriptions together with assumptions (a)-(e).

3 Semantics of $\mathcal{L}_2$

In this section we introduce a semantics of a domain description in $\mathcal{L}_2$. We start with defining causal models of D and proceed by explaining when facts are true in these models.

A *state* is a set of fluent names. A *causal interpretation* is a partial function Ψ from sequences of actions to states such that:

(1) Empty sequence [ ] belongs to the domain of Ψ and (2) Ψ is prefix-closed[3].

$\Psi([\,])$ is called the initial state of Ψ. The partial function Ψ serves as an interpretation of the laws of D. If α belongs to the domain of Ψ we say that α is *possible* in the initial state of Ψ.

Given a fluent f and a state σ, we say that f *holds* in σ (f is *true* in σ) if $f \in \sigma$; $\neg f$ *holds* in σ (f is *false* in σ) if $f \notin \sigma$. The *truth* of a propositional formula, C, with respect to σ is defined as usual.

To better understand the role Ψ plays in interpreting domain descriptions let us first use it to define *models* of descriptions consisting entirely of effect laws. To achieve this goal we will attempt to carefully define effects of actions as determined by such a description, D, and our informal assumptions; (a)-(e).

A fluent f is an **immediate effect** of (executing) a in σ if there is an effect law "a **causes** f **if** $p_1, \ldots, p_n$" in D whose preconditions hold in σ.

For unit actions these are the only effects possible. For compound actions effects may also be determined by the assumption (c) which says that compound actions may inherit effects of their sub-actions. The meaning of this statement however is somewhat ambiguous and needs some refinement. To see the problem consider, for instance, domain description

a **causes** f

[3]By prefix closed we mean that for any sequence of actions α and action a, if $\alpha \circ a$ is in the domain of Ψ then so is α. (Recall that $\circ$ denotes concatenation, and $\alpha \circ a$ means the sequence of actions where a follows α).

$\{a, b\}$ **causes** $\neg f$

and an action $\{a, b, c\}$. Does this action inherit the effect of a, the effect of $\{a, b\}$, both or neither? We follow [BG93] and opt for the the second alternative on the basis of specificity principle which says that "more specific information overrides less specific information". This leads to the following definition:

Definition 3.1 A fluent f is an **inherited effect** of (executing) a in σ if there is $b \subset a$ such that
(a) f is an immediate effect of b in σ.
(b) There is no action c such that $b \subset c \subseteq a$ and $\neg f$ is an immediate effect of c in σ.

□

Definition 3.2 f is an **effect** of (executing) a in σ if either f is an **immediate effect** of (executing) a in σ or f is an **inherited effect** of (executing) a in σ. □

Let, $E_a^+(\sigma) = \{f : f \text{ is an effect of } a \text{ in } \sigma\}$, $\quad E_a^-(\sigma) = \{f : \neg f \text{ is an effect of } a \text{ in } \sigma\}$ and $Res(a, \sigma) = \sigma \cup E_a^+(\sigma) \setminus E_a^-(\sigma)$.

The following definition captures the meaning of effect laws of D.

Definition 3.3 A causal interpretation Ψ **satisfies** the effect laws of D if for any sequence $\alpha \circ a$ from the language of D

$\Psi(\alpha \circ a) = Res(a, \Psi(\alpha))$ if $E_a^+(\Psi(\alpha)) \cap E_a^-(\Psi(\alpha)) = \emptyset$
and undefined otherwise.

We say that Ψ is a **causal model** of D if it satisfies all the effect laws of D □

Example 3.1 Suppose we are given the following general information about effects of actions performed by Mary:
Whenever Mary tries to lift the bowl with one hand, she spills the soup on the table and makes the table wet. When she uses both hands, she does not spill the soup.

To specify these effects by a domain description let us consider an alphabet consisting of a fluent name *spilled* and two unit actions $lift\_l$ and $lift\_r$, and three effect laws:

$\{lift\_l\}$ **causes** *spilled*

$\{lift\_r\}$ **causes** *spilled*

$\{lift\_l,\ lift\_r\}$ **causes** $\neg$ *spilled* **if** $\neg$ *spilled*

The resulting domain description will be denoted by D_1.

It is easy to see that the only causal model of D_1 is defined by a transition function Ψ where

$\Psi(\alpha \circ \{lift\_l\}) = \Psi(\alpha \circ \{lift\_r\}) = \{spilled\}$ if $\Psi(\alpha) = \emptyset$

and for any other action a and sequence α

$\Psi(\alpha \circ a) = \Psi(\alpha)$ □

Example 3.2 Consider a domain description consisting of the following propositions:

a **causes** f
b **causes** $\neg f$

It is easy to see for any α, $\Psi(\alpha \circ \{a, b\})$ is undefined. This is because $E^+_{\{a,b\}}(\Psi(\alpha)) = E^-_{\{a,b\}}(\Psi(\alpha)) = \{f\}$. □

Example 3.3 Consider a domain description consisting of the following propositions:

$\{a, b\}$ **causes** f
$\{b, c\}$ **causes** g

It is easy to see that for any α, $\Psi(\alpha \circ \{a, b, c\}) = \{f, g\}$. This is because $E^+_{\{a,b,c\}}(\Psi(\alpha)) = \{f, g\}$ and $E^-_{\{a,b\}}(\Psi(\alpha)) = \emptyset$. □

3.1 Interpreting observations

Let D be an arbitrary domain description and let a causal interpretation Ψ be a causal model of D. To interpret the observations of D we first need to define the meaning of situation constants $s_0, s_1, s_2, \ldots$ from $\mathcal{S}$. To do that we consider a mapping Σ from $\mathcal{S}$ to sequences of actions from the language of D. This mapping will be called *a situation assignment of* $\mathcal{S}$ if it satisfies the following properties:

1. $\Sigma(s_0) = [\ ]$ 2. for every $s_i \in \mathcal{S}$, $\Sigma(s_i)$ is a prefix of $\Sigma(s_N)$.

Definition 3.4 An *interpretation* M of $\mathcal{L}_2$ is a pair (Ψ, Σ), where Ψ is a causal model of D, Σ is a situation assignment of $\mathcal{S}$ and $\Sigma(s_N)$ belongs to the domain of Ψ. □

$\Sigma(s_N)$ will be called the *actual path* of M and for simplicity will often be denoted by Σ_N.

Now we can define truth of facts of D w.r.t. an interpretation M. Facts which are not true in M will be called *false* in M.

Definition 3.5 For any interpretation $M = (\Psi, \Sigma)$.

(1) (f **at** s) is *true* in M (or satisfied by M) if f is true in $\Psi(\Sigma(s))$.

(2) atomic fact ($[a_1, \ldots, a_n]$ **occurs_at** s) is true in M if there is a sequence $\beta = [b_1, \ldots b_n]$ of actions such that

(a) The sequence $\Sigma(s) \circ \beta$ is a prefix of the actual path of M, and

(b) for any i, $1 \leq i \leq n$, $a_i \subseteq b_i$.

(3) (s_1 **precedes** s_2) is true in M if $\Sigma(s_1)$ is a proper prefix of $\Sigma(s_2)$

(4) Truth of non-atomic facts in M is defined as usual. □

A set of facts is true in interpretation M if all its members are true in M.

To complete the definition of the model we need only to formalize the following underlying assumption of domain descriptions of $\mathcal{L}_2$:

(d) No actions occur except those needed to explain the facts in the domain description.

This is done by imposing a minimality condition on the situation assignments of $\mathcal{S}$ which leads to the following definitions.

Definition 3.6 Let $\alpha = [a_1, \ldots, a_n]$ and $\beta = [b_1, \ldots, b_m]$ be sequences of actions. We say that $\alpha \leq \beta$ if there exists a subsequence[4] $[b_{i_1}, \ldots, b_{i_n}]$ of β such that for every $a_k \in \alpha$, $a_k \subseteq b_{i_k}$. □

Definition 3.7 An interpretation $M = (\Psi, \Sigma)$ will be called a *model* of a domain description D in $\mathcal{L}_2$ if the following conditions are satisfied:

(1) Ψ is a causal model of D,
(2) facts of D are *true* in M, and
(3) there is no other interpretation $N = (\Psi, \Sigma')$ such that N satisfies the previous two conditions and $\Sigma'(s_N) \leq \Sigma(s_N)$. □

The following proposition shows that for a model (Ψ, Σ) of a domain description in $\mathcal{L}_2$, Ψ is completely determined by its initial state.

Proposition 3.1 Let $M_1 = (\Psi_1, \Sigma_1)$ and $M_2 = (\Psi_2, \Sigma_2)$ be two models of a domain description D in language $\mathcal{L}_2$. If $\Psi_1([\,]) = \Psi_2([\,])$ then $\Psi_1 = \Psi_2$. □

Corollary 3.1 Let D be a domain description in language $\mathcal{L}_2$. If for all models of D, $\Psi([\,])$ is uniquely defined then Ψ is also uniquely defined. □

A domain description D is said to be *consistent* if it has a model.

Definition 3.8 A domain description D entails a fact p (written as $D \models p$) iff p is *true* in all models of D. □

Definition 3.9 A domain description D is said to define a unique actual path if for any two situations s_1 and s_2 that are explicitly mentioned in D, $D \models s_1$ **precedes** s_2 or $D \models s_2$ **precedes** s_1. □

Lemma 1 Let D be a domain description such that
(1) D defines a unique actual path, and
(2) the only atomic fluent facts which occur in propositions of D are of the form f **at** s_0.
Then situation assignments of all models of D coincide on s_N and on all the situations explicitly mentioned in D. □

[4] Given a sequence $X = x_1, \ldots, x_m$, another sequence $Z = z_1, \ldots, z_n$ is a subsequence of X if there exists a strictly increasing sequence $i_1, \ldots, i_n$ of indices of X such that for all $j = 1, 2 \ldots, n$, we have $x_{i_j} = z_j$.

The following proposition shows that the empty set of actions is also an action and it behaves like the wait action with no effects on the fluents. We refer to its as the *wait* action.

Example 3.4 Let D be the following domain description
s_0 **precedes** s_1
s_1 **precedes** s_2
a **occurs_at** s_0
b **occurs_at** s_0

Then $D \models \{\}$ **occurs_at** s_1. $\square$

3.2 Examples

In this section we illustrate by way of examples how domain descriptions are used to represent information and how the above notion of entailment captures informal arguments based on the information from these descriptions and the informal assumptions (a) - (e). We start with Example 2.1 from Section 2.

Proposition 3.2 Consider the domain description D_1 from Example 2.1. We have

$D_1 \models ((\neg dry \wedge \neg alive)$ **at** $s_N)$ $\square$

Proof. Consider a causal interpretation, Ψ, and the situation assignment, Σ, defined as follows:

$$\Psi([\,]) = \{alive, dry\}$$

$$\Psi(\alpha \circ squirt) = \begin{cases} \{alive\} & \text{if } alive \in \Psi(\alpha) \\ \emptyset & \text{otherwise} \end{cases}$$

$$\Psi(\alpha \circ shoot) = \begin{cases} \{dry\} & \text{if } dry \in \Psi(\alpha) \\ \emptyset & \text{otherwise} \end{cases}$$

$$\Psi(\alpha \circ \{squirt, shoot\}) = \emptyset$$

$\Sigma(s_0) = [\,]$

$\Sigma(s_N) = \Sigma(s_1) = [\{squirt, shoot\}]$

It is easy to check that the interpretation $M = (\Psi, \Sigma)$ satisfies Definition 3.7 and hence is a model of D_1.
Let us show that it is the only model (i.e. for any model $M_1 = (\Psi_1, \Sigma_1)$ of D_1, $\Psi = \Psi_1$, and $\Sigma = \Sigma_1$). Since by definition any situation assignment maps s_0 into $[\,]$, statements (p1) and (p2) of D_1 will be satisfied only if $\Psi([\,]) = \Psi_1([\,]) = \{alive, dry\}$. Since Ψ and Ψ_1 are

causal models of D_1 and causal models are uniquely determined by their initial values, we have that $\Psi = \Psi_1$.
To establish that $\Sigma = \Sigma_1$ notice that to satisfy conditions (p3) - (p5) the actual path $\Sigma_1(s_N)$ of M_1 must start with a compound action which is a superset of $\{squirt, shoot\}$ followed (not necessarily immediately) by another action (possibly $wait$). But then by definition $\Sigma(s_N) \leq \Sigma_1(s_N)$. M_1 is a model of D_1 and hence Σ_1 satisfies the minimality condition 3 from 3.7. This implies that $\Sigma(s_N) = \Sigma_1(s_N) = [\{squirt, shoot\}]$. Now recall that, by definition of situation assignment, $\Sigma_1(s_1)$ must be a prefix of $[\{squirt, shoot\}]$. Since (p3) and (p5) are true in M_1 we have that $\Sigma(s_1) = \Sigma_1(s_1) = [\{squirt, shoot\}]$.
To complete the proof it suffices to notice that $((\neg dry \wedge \neg alive)$ **at** $s_N)$ is true in M.

Example 3.5 (Reasoning by cases) Let us consider a modification of Example 2.1 where there is a precondition of being loaded for the shoot action to be deadly and where there are two guns at least one of which is initially loaded.

$$\left.\begin{array}{l} (q1)\ alive\ \textbf{at}\ \ s_0 \\ (q2)\ loaded_1\ \textbf{at}\ \ s_0 \vee loaded_2\ \textbf{at}\ \ s_0 \\ (q3)\ \{shoot_1, shoot_2\}\ \ \textbf{occurs\_at}\ \ s_0 \\ (q4)\ shoot_1\ \ \textbf{causes}\ \ \neg alive\ \textbf{if}\ \ loaded_1 \\ (q5)\ shoot_2\ \ \textbf{causes}\ \ \neg alive\ \textbf{if}\ \ loaded_2 \end{array}\right\} D_2$$

$\Box$

Proposition 3.3 $D_2 \models \neg alive$ **at** s_N. $\Box$

Example 3.6 (Explaining observations) Let us now consider a modified version of Example 2.1 where instead of

(b) At the same moment a shot was fired at $Fred$, we have

(b′) In a later moment $Fred$ was observed to be dead

and where we assume that our domain contains unit actions $a_1, \ldots, a_n$ different from $squirt$ and $shoot$.

The resultant story can be represented by a domain description D_3 consisting of the propositions (p1)– (p4) and (p6) - (p7) of D_1 and the following proposition $(p5')$.

$(p5')\ \neg alive$ **at** s_1 $\Box$

Proposition 3.4 $D_3 \models ([squirt, shoot]\ \textbf{occurs\_at}\ \ s_0) \vee ([\{squirt, shoot\}]\ \textbf{occurs\_at}\ \ s_0)$.
i.e. D_3 entails that either $squirt$ occured at s_0 and then $shoot$ occured or $squirt$ and $shoot$ both occured simultaneously at s_0. $\Box$

Consider the following three examples which are modifications of example 3.1. In all three cases we will assume the bowl is initially on the table (i.e. $\neg lifted$) and the soup is not spilled.

Example 3.7 Consider the case when the bowl is lifted by both sides at once. The domain, D_4, can be defined as follows:

$$\left.\begin{array}{l}(r1)\ lift\_l\ \textbf{causes}\ \ spilled\\ (r2)\ lift\_r\ \textbf{causes}\ \ spilled\\ (r3)\ lift\_l\ \textbf{causes}\ \ lifted\\ (r4)\ lift\_r\ \textbf{causes}\ \ lifted\\ (r5)\ \{lift\_l, lift\_r\}\ \textbf{causes}\ \ \neg spilled\ \textbf{if}\ \ \neg spilled\\ (r6)\ \{lift\_l, lift\_r\}\ \textbf{causes}\ \ lifted\\ (r7)\ initially\ \neg spilled\\ (r8)\ initially\ \neg lifted\\ (r9)\ \{lift\_l, lift\_r\}\ \textbf{occurs\_at}\ \ s_0\end{array}\right\} D_4$$

□

Proposition 3.5 $D_4 \models \neg spilled\ \textbf{at}\ \ s_N$ and $D_4 \models lifted\ \textbf{at}\ \ s_N$. □

Example 3.8 Next consider the example where first the left side is lifted and then the right side is lifted. Our new domain, D_5, can be represented by propositions (r1) - (r8) from our previous example and three new propositions, (r10) - (r12).

$(r10)\ s_0\ \textbf{precedes}\ \ s_1$
$(r11)\ lift\_l\ \textbf{occurs\_at}\ \ s_0$
$(r12)\ lift\_r\ \textbf{occurs\_at}\ \ s_1$ □

Proposition 3.6 $D_5 \models spilled\ \textbf{at}\ \ s_N$ and $D_5 \models lifted\ \textbf{at}\ \ s_N$ □

Example 3.9 Let us now consider the case where we aren't told what actions occured, only that in a later moment we observe that the bowl is lifted and the soup is not spilled. Our new domain, D_6, can be represented by propositions (r1) - (r8), (r10), and two new propositions, (r13) and (r14).

$(r13)\ lifted\ \ \textbf{at}\ \ s_1$
$(r14)\ \neg spilled\ \ \textbf{at}\ \ s_1$ □

Proposition 3.7 $D_6 \models \{lift\_l, lift\_r\}\ \textbf{occurs\_at}\ \ s_0$ □

3.3 Hypothetical Reasoning

Even though domain descriptions of $\mathcal{L}_2$ can express types of knowledge and reasoning not easily expressible in other variants of $\mathcal{A}$, they lack the ability of the latter to do hypothetical reasoning. To do hypothetical reasoning we allow *hypothesis* of the form (5) in the query language of $\mathcal{L}_2$. Note that —according to these definitions— the query language of $\mathcal{L}_2$ is strictly stronger than that of $\mathcal{A}$ (and $\mathcal{A}_C$) while, as a means of specifying domain descriptions, neither $\mathcal{L}_2$ nor $\mathcal{A}$ is more powerful.

For convenience, we introduce the following additional notation:
If s in (5) is the current situation (i.e. s_N) then we simply write

$$f \textbf{ after } [a_1, \ldots, a_n] \tag{6}$$

If n in (6) is 0, then we write

$$\textbf{currently } f \tag{7}$$

The notions of model of a domain description of $\mathcal{L}_2$ and of the truth of atomic facts in such models remain unchanged. To define answers to queries from $\mathcal{L}_2$ we need the following:

Definition 3.10 Let D be a domain description in $\mathcal{L}$ and $M = (\Psi, \Sigma)$ be an interpretation of D. We say that a *hypothesis* (5) *is true in an interpretation* M if f is true in $\Psi(\Sigma(s) \circ [a_1, \ldots, a_n])$. □

Definition 3.11 Let D be a domain description and H be a hypothesis in $\mathcal{L}_2$. We say $D \models H$ iff H is true in all models of D. □

Truth of arbitrary queries and sets of queries in $\mathcal{L}$ is defined as usual.

A reasoner with knowledge formulated in a domain description D can use hypothetical queries for various purposes. We believe the most important of them is planning. Suppose that our reasoning agent is given a domain description, D, and a collection of fluent literals, G, viewed as a goal to be achieved by performing actions from D. Hypothetical statements of $\mathcal{L}_2$ and $\mathcal{L}_2$ entailment can be used to define a notion of a plan:

Definition 3.12 Let D be a domain description in $\mathcal{L}_2$ and G be a set of fluent literals. A sequence, α, of actions is a *plan* for achieving a goal G if $D \models f$ **after** α for every fluent literal $f \in G$. □

A planning program should be able to generate sequences of actions viewed as possible plans and test them using the entailment relation of $\mathcal{L}_2$. Consider for instance the following example.

Example 3.10 Consider a variant D_2' of domain description D_2

$$\left.\begin{array}{l} (q1)\ alive \textbf{ at } s_0 \\ (q2)\ loaded_1 \textbf{ at } s_0 \vee loaded_2 \textbf{ at } s_0 \\ (q4)\ shoot_1 \textbf{ causes } \neg alive \textbf{ if } loaded_1 \\ (q5)\ shoot_2 \textbf{ causes } \neg alive \textbf{ if } loaded_2 \end{array}\right\} D_2'$$

and a goal $G = \{\neg alive\}$ a reasoner can come up with a "candidate plan" $[\{shoot_1, shoot_2\}]$ for achieving this goal. The candidate plan will be tested by proving the (hypothetical) statement "$\neg alive$ **after** $[\{shoot_1, shoot_2\}]$", i.e. by checking whether $D_2' \models \neg alive$ **after** $[\{shoot_1, shoot_2\}]$. It is easy to check that the statement is true and therefore $[\{shoot_1, shoot_2\}]$ is indeed a plan for G. Notice that, the resulting domain description (obtained from D_2' by adding $[\{shoot_1, shoot_2\}]$ **occurs_at** s_0) entails G **at** s_N. □

As in the case of $\mathcal{L}_1$, the entailment relation of $\mathcal{L}_2$ also allows modeling of more sophisticated forms of hypothetical reasoning. This aspect is discussed in [BGP95].

Finally, the following definitions and notations can be useful in our further discussions.

Let H_1 and H_2 be two sets of hypotheses. We say that the premise H_1 entails conclusion H_2 in D if H_2 is true in every model of D in which H_1 is true. We will denote this by $H_1 \models_D H_2$.

Proposition 3.8 $\emptyset \models_D H$ iff $D \models H$ $\square$

A set of hypotheses H is is *inconsistent* w.r.t. a domain description D if no model of D satisfies H.

It is important to notice that the entailment relation ($\models_D$) defined by a domain description D is *monotonic* - addition of new hypothesis to the set of hypotheses H_1 can only decrease the set of models of D satisfying it and hence can only increase the set of conclusions. Non-monotonicity occurs only when new information about the real world (i.e. new laws or new facts) are added to a reasoner's knowledge.

Example 3.11 Consider the following domain description, D_7, consisting of (p1) and (p2):

(p1) $shoot$ **causes** $\neg alive$ **if** $loaded$ (p2) $load$ **causes** $loaded$

Suppose that, given the domain description D_7, a reasoner would like to know if Fred would be dead after shooting under the assumption that initially the gun is loaded. Notice that both statements are hypothetical and therefore are naturally represented as follows:

$H_1 = loaded$ **after** $[\,]$ **at** s_0 $H_2 = \neg alive$ **after** $[shoot]$ **at** s_0

The question can be formulated as $H_1 \models_{D_7} H_2$? The answer is obviously *yes*. $\square$

3.4 Relating $\mathcal{L}_2$ and $\mathcal{L}_1$

It was our goal in this paper to extend the language $\mathcal{L}_1$. We now formally state the relation between $\mathcal{L}_1$ and $\mathcal{L}_2$.

Theorem 1 Let D be a domain description in $\mathcal{L}_2$ which does not contain any compound actions. For any query Q in $\mathcal{L}_2$ which does not contain compound actions, $\mathcal{L}_2 \models Q$ iff $\mathcal{L}_1 \models Q$. (i.e., $\mathcal{L}_2$ is a conservative extension of $\mathcal{L}_1$). $\square$

Proof:
Follows trivially from the semantics of $\mathcal{L}_1$ and $\mathcal{L}_2$. $\square$

4 From $\mathcal{L}_2$ to Logic Programs

4.1 Extended Logic Programs and Disjunctive Logic Programs

In this section we review necessary definitions and results from the theory of declarative logic programming. In addition to negation as failure, *not*, [Cla78] of "classical" logic programming languages we consider two other connectives: classical (strong, explicit) negation,

$\neg$, of [GL90] and epistemic disjunction, *or*, of [GL91]. Both connectives are needed to allow representation of various forms of incomplete information. There is no complete agreement on the nature and semantics of these connectives and their interrelation with negation as failure. Several different proposals were discussed in the literature. (see, for instance, Minker et al. [LMR92], Pereira et al. [PCA90], Dix [Dix91], Przymusinski [Prz90], Gelfond and Lifschitz [GL91]). We will follow [GL91]. Applicability of this approach to representation of incomplete information is discussed in [BG94, Gel94].

A disjunctive logic program (DLP) is a collection of rules of the form

$$l_0 \ or \ \ldots \ or \ l_k \leftarrow l_{k+1}, \ldots, l_m, not \ l_{m+1}, \ldots, not \ l_n \tag{8}$$

where each l_i is a literal (i.e. an atom possibly preceded by $\neg$) and *not* is the negation as failure. Expression on the left hand (right hand) side of $\leftarrow$ is called the *head* (the *body*) of the rule. Both the head and the body of (8) can be empty. Intuitively the rule can be read as: if $l_{k+1}, \ldots, l_m$ are believed and it is not true that $l_{m+1}, \ldots, l_n$ are believed then at least one of $\{l_0, \ldots, l_k\}$ is believed. For a rule r of the form (8) the sets $\{l_0, \ldots, l_k\}$, $\{l_{k+1}, \ldots, l_m\}$ and $\{l_{m+1}, \ldots, l_n\}$ are referred to as $head(r)$, $pos(r)$ and $neg(r)$ respectively. $lit(r)$ stands for $head(r) \cup pos(r) \cup neg(r)$. For any DLP Π, $head(\Pi) = \bigcup_{r \in \Pi} head(r)$. For a set if predicates S, $Lit(S)$ denotes the set of literals with predicates from S. For a DLP Π, $Lit(\Pi)$ denotes the set of literals with predicates from the language of Π. When it is clear from the context we write Lit instead of $Lit(\Pi)$. Given a literal, l, of the form a or $\neg a$ (where a is an atom), $\bar{l}$ denotes $\neg a$ if $l = a$ and a if $l = \neg a$. For sets of literals X and Y, we say Y is *complete* in X if for every literal $l \in X$, at least one of the complementary literals $l, \bar{l}$ belongs to Y.

A program determines a collection of *answer sets* – sets of ground literals representing possible beliefs of the program.

Definition 4.1 *[GL91]* Let Π be a disjunctive logic program without variables. For any set S of literals, let Π^S be the logic program obtained from Π by deleting

(i) each rule that has a formula *not* l in its body with $l \in S$, and

(ii) all formulas of the form *not* l in the bodies of the remaining rules. $\square$

Definition 4.2 An *answer set* of a disjunctive logic program Π not containing *not* is a smallest (in a sense of set-theoretic inclusion) subset S of Lit such that

(i) for any rule $l_0 \ or \ \ldots \ or \ l_k \leftarrow l_{k+1}, \ldots, l_m$ from Π, if $l_{k+1}, \ldots, l_m \in S$, then for some i, $0 \leq i \leq k$, $l_i \in S$;

(ii) if S contains a pair of complementary literals, then $S = Lit$.

A set, S, of literals is an answer set of an arbitrary disjunctive logic program, Π, if S is an answer set of Π^S $\square$

A program[5] is consistent if it has an answer set not containing contradictory literals. As was shown in [Gel94] if a program is consistent then all of its answer sets are consistent. A

[5]Henceforth by "program" we mean a disjunctive logic program.

ground literal l is *entailed* by a DLP if it belongs to all of its answer sets. When all the rules in a DLP have $k = 0$ then it is referred to as an extended logic program [GL90, PW89].

In our further discussion we will need the following proposition about DLPs:

Proposition 4.1 *[BG94]* For any answer set S of a disjunctive logic program Π:

(a) For any ground instance of a rule of the type (8) from Π, if
$\{l_{k+1} \dots l_m\} \subseteq S$ and
$\{l_{m+1} \dots l_n\} \cap S = \emptyset$
then there exists an i, $0 \leq i \leq k$ such that $l_i \in S$.

(b) If S is a consistent answer set of Π and $l_i \in S$ for some $0 \leq i \leq k$ then there exists a ground instance of a rule from Π such that
$\{l_{k+1} \dots l_m\} \subseteq S$, and
$\{l_{m+1} \dots l_n\} \cap S = \emptyset$, and
$\{l_0 \dots l_k\} \cap S = \{l_i\}$. □

4.2 Approximating the Entailment Relation of $\mathcal{L}_2$

Our methodology for computing the entailment relation of $\mathcal{L}_2$ is based on translating a domain description D of $\mathcal{L}_2$ into a declarative logic program ΠD and proving soundness (and if possible completeness) of this translation w.r.t. entailment in D.

In this section we limit ourself to domain descriptions in which all facts are atomic, occurance facts do not contain action sequences, and which have an explicit actual path. The following definition clarifies what is meant by an explicit actual path.

Definition 4.3 Let D be a consistent domain description with situation constants $s_0, \dots, s_k$. We will say that D has an *explicit actual path* if for any $0 < i \leq k$ and any compound or unit action a

$(s_{i-1}$ **precedes** $s_i) \in D$

$D \models (a$ **occurs_at** $s_i)$ iff $(a$ **occurs_at** $s_i) \in D$ and $i < k$.

□

It is easy to see that any D satisfying these conditions contains the statements
a_0 **occurs_at** $s_0, \dots, a_{k-1}$ **occurs_at** s_{k-1} and has a unique actual path, i.e. for any model (Ψ, Σ) of D and for every $0 < i \leq k$, $\Sigma(s_i) = [a_0, \dots, a_{i-1}]$ and $\Sigma(s_N) = \Sigma(s_k)$.

Now we will assume that D has explicit actual path and describe a program ΠD approximating the entailment relation of D.

The alphabet of ΠD consists of symbols for actions, fluent literals and situations from the language of D, and predicates *holds, occurs_at, causes, imm_follows, undefined, cancells, noninh, subset, eq, subseteq, member,* and *atomic*. We will use the following typed variables:

S	for situations
A	for actions
F, G	for fluent literals
P	for lists of fluent literals

All the variables can be indexed. The corresponding lower case letters will denote constants of respective types.

Let D be a domain description with the explicit actual path $a_0, \dots a_k$. The logic programming approximation ΠD of entailment of D will consists of the following rules:

1. **Domain Dependent Axioms**

   (a) **(AL) Description of Actual Path**

   $imm\_follows(s_1, s_0) \leftarrow$

   $\vdots$

   $imm\_follows(s_k, s_{k-1}) \leftarrow$

   $occurs\_at(a_0, s_0) \leftarrow$

   $\vdots$

   $occurs\_at(a_{k-1}, s_{k-1}) \leftarrow$

   (b) **(BC) Boundary Conditions** Each atomic fluent fact, f **at** s in D, is translated as a rule of the form $h(f, s_i) \leftarrow$

   where, for any literal g, the atom $h(g, s)$ denotes the literal $holds(g, s)$ if g is a positive literal and the literal $\neg holds(|g|, s)$ if g is a negative literal; and for any fluent f, $|f| = f$ and $|\neg f| = f$.

   (c) **(CL) Causal Laws**

   The translation of an effect law "a **causes** f if $p_1, \dots, p_n$" consists of

   $causes(a, f, S) \leftarrow h(p_1, S), \dots, h(p_n, S)$

   Intuitively, $causes(a, f, s)$ means that if action a is executed in situation s then f will be true as a direct effect of the action a.

2. **Domain Independent Axioms**

   (a) **(EA) Effects of actions**

   The *Effect Axioms* are defined as follows:

   $(i)\ h(F, Result(A, S)) \leftarrow causes(A, F, S), not\ undefined(A, S)$
   $(ii)\ undefined(A, S) \leftarrow causes(A, F, S), causes(A, \overline{F}, S)$
   $(iii)\ undefined(A, res(B, S)) \leftarrow undefined(B, S)$

   The effect axiom allows us to prove that f will hold after a, if the preconditions are satisfied.

The first axiom differs from the one suggested in [GL92] only by allowing terms for compound actions and by using the predicate "undefined". The next axioms are new. They describe how the effects of individual actions are related to the effects of these actions performed concurrently.

(b) **(IA) Inheritance Axioms**

$$(i)\ h(F, result(A, S)) \leftarrow subsetof(B, A), h(F, result(B, S)), not\ noninh(F, A, S), not\ undefined(A, S)$$

$$(ii)\ cancells(X, Y, F, S) \leftarrow subsetof(X, Z), subseteq(Z, Y), causes(Z, \overline{F}, S)$$

$$(iii)\ noninh(F, A, S) \leftarrow subseteq(U, A), causes(U, \overline{F}, S), not\ cancells(U, A, \overline{F}, S)$$

$$(iv)\ undefined(A, S) \leftarrow noninh(F, A, S), noninh(\overline{F}, A, S)$$

Intuitively, $cancells(x, y, f, s)$ means that an action z, $x \subset z \subseteq y$ causes $\overline{f}$ thus cancelling the inheritance of the effect f by the action y from its sub-action x .

Intuitively, $noninh(f, x, s)$ means that the action x does not inherit the effect f from its sub-actions in situation s.

The non-inheritance axioms are essential for the correct treatment of concurrent actions.

According to these axioms the effects of compound actions are normally inherited from the effects of their components.

(c) **(FI) First Inertia Axiom**

$$h(F, result(A, S)) \leftarrow h(F, S), not\ causes(A, \overline{F}, S), atomic(A), not\ undefined(A, S)$$

This rule is motivated by the "common-sense law of inertia," [MH69] according to which fluents normally are not changed by actions. The rule allows us to apply the law of inertia in reasoning "from the past to the future." The auxiliary predicate *causes* is essentially an "abnormality predicate" [McC86]. The axiom differs from those suggested in [GL92] only in the use of predicate "atomic" to restrict the inertia rules to unit actions and the predicate "undefined" to restrict inertia rules to executable actions.

(d) **(SI) Second Inertia Axiom**

$$h(F, S2) \leftarrow imm\_follows(S2, S1), occurs\_at(A1, S1), h(F, result(A1, S1))$$

The second inertia axiom assures that the effects of actions and inertia are passed to named situations.

3. **(FA) Full Awareness of Initial State Axiom**

Turner [Tur94] called it "complete initial situations rule". We prefer to call it "full awareness about the initial situation rule" which means that the program is aware of all fluents of the language in the initial situation.

$holds(F, s_0)$ *or* $\neg holds(F, s_0) \leftarrow$

4. **(LA) Library Axioms**

$\neg subset(A, B) \leftarrow member(X, A), not\ \ member(X, B)$
$subset(A, B) \leftarrow not\ \neg subset(A, B)$
$\neg atomic(X) \leftarrow member(Y, X), member(Z, X), not\ \ eq(Y, Z)$
$atomic(X) \leftarrow not\ \neg atomic(X)$
$eq(X, X)$
$subseteq(X, Y) \leftarrow subset(X, Y)$
$subseteq(X, Y) \leftarrow eq(X, Y)$
$member(X, [X|\_])$
$member(X, [\_|T]) \leftarrow member(X, T)$

4.3 Properties of ΠD

In this section we prove various relationships between D and ΠD. The main results are given as propositions and theorems, and lemmas are used in the proofs of the propositions and theorems.

It should be noted that rules of ΠD are ground instantiations of the schemas describing ΠD in the previous subsection, where substitutions of variables by ground terms are done in accordance with definitions of their types.

We first give an overview of "splitting sets" and show a result that will allow us to ignore the "Library Axioms" in our later proofs.

Definition 4.4 *(Splitting set) [LT94]*
A *splitting set* for a program Π is any set U of literals such that, for every rule $r \in \Pi$, if $head(r) \cap U \neq \emptyset$ then $lit(r) \subset U$. If U is a splitting set for Π, we also say that U splits P. The set of rules $r \in \Pi$ such that $lit(r) \subset U$ is called the *bottom* of Π relative to the splitting set U and denoted by $b_U(\Pi)$. The subprogram $\Pi \setminus b_U(\Pi)$ is called *the top of* Π relative to U. □

Definition 4.5 *(Partial evaluation) [LT94]*
The partial evaluation of a program Π with splitting set U w.r.t. a set of literals X is the program $e_U(\Pi, X)$ defined as follows. For each rule $r \in \Pi$ such that:

$$(pos(r) \cap U) \subset X \quad \wedge \ (neg(r) \cap U) \cap X = \emptyset$$

put in $e_U(\Pi, X)$ all the rules r' that satisfy the following property:

$$head(r') = head(r),\ pos(r') = pos(r) \setminus U,\ neg(r') = neg(r) \setminus U$$

□

Definition 4.6 *(Solution) [LT94]*
Let U be a splitting set for a program Π. A solution to Π w.r.t. U is a pair $\langle X, Y \rangle$ of literals such that:

- X is an answer set for $b_U(\Pi)$;
- Y is an answer set for $e_U(\Pi \setminus b_U(\Pi), X)$;
- $X \cup Y$ is consistent.

$\square$

Lemma 2 *(Splitting Lemma) [LT94]*
Let U be a splitting set for a program Π. A set A of literals is a consistent answer set for Π if and only if $A = X \cup Y$ for some solution $\langle X, Y \rangle$ to Π w.r.t. U. $\square$

Observation 1 The program ΠD can be split (see Definition 4.4) using the splitting set consisting of $Lit(\{atomic, subset, \neg subset, \neg atomic, subseteq, eq, member\})$. It is easy to see that the bottom part consists of the rules (LA) and the top part consists of the rest. It is easy to see that the bottom part has a unique answer set. Hence by virtue of Lemma 2 in the rest of the paper we only consider the top part of the program ΠD partially evaluated w.r.t. the unique answer set of the bottom part. $\square$

Lemma 3 Let Π be an extended logic program, $\mathcal{D}$ be a set of disjunctions, and A be a set of literals. If for every disjunction of the form a_1 *or* $\ldots$ *or* a_n in $\mathcal{D}$ there exists an i such that $a_i \in A$, then $\Pi \cup \mathcal{D} \cup A$ and $\Pi \cup A$ have the same answer sets. $\square$

Proof: Follows immediately from the definition of answer sets. $\square$

We now introduce some notations for the forthcoming proofs.

Notation

1. A set of fluent literals $p = \{p_1, \ldots, p_n\}$ is said to hold in a state σ if every p_i holds in σ. Otherwise, we say that p does not hold in σ.
2. For a set of fluent literals $p = \{p_1, \ldots, p_n\}$ we denote the set $\{h(p_1, s), \ldots, h(p_n, s)\}$ by $H(p, s)$.
3. ${direct_a}^+(\sigma) = \{f : f$ is an immediate effect of a in $\sigma\}$.
4. ${direct_a}^-(\sigma) = \{f : \overline{f}$ is an immediate effect of a in $\sigma\}$.
5. ${inherited_a}^+(\sigma) = \{f : f$ is an inherited effect of a in $\sigma\}$.
6. ${inherited_a}^-(\sigma) = \{f : \overline{f}$ is an inherited effect of a in $\sigma\}$.
7. Given a model, $M = (\Psi, \Sigma)$, an actual situation, s_i, and a sequence of actions, $\alpha_n = [a_1, \ldots, a_n]$, we denote the state $\Psi(\Sigma(s_i))$ by σ_{s_i} and the state $\Psi(\Sigma(s_i) \circ \alpha_n)$ by σ_{s_i,α_n}

8. Given a sequence of actions $\alpha_n = [a_1, \ldots, a_n]$, we denote the situation $result(a_n, result(a_{n-1}, \ldots, result(a_1, s_i) \ldots))$ by $\alpha_n(s_i)$.

Note that, $E_a{}^+(\sigma) = direct_a{}^+(\sigma) \cup inherited_a{}^+(\sigma)$ and $E_a{}^-(\sigma) = direct_a{}^-(\sigma) \cup inherited_a{}^-(\sigma)$.

Proposition 4.2 Let D be a domain description. The logic program $(\Pi D)^+$ [6] is locally stratified. $\square$

Proof(sketch)
For a sequence of actions, $\alpha = a_1, \ldots, a_m$, and an actual situation s_i its level is defined as the pair $(m+i, k)$ where k is $|a_m|$, i.e. the cardinality of a_m. We say $(m+i, k) < (m'+i', k')$ iff $m+i < m'+i'$ or $m+i = m'+i'$ and $k < k'$.

It is easy to see that any assignment of strata to atoms which satisfies the following conditions gives us a local stratification:

(i) Atoms of the form $imm\_follows(s2, s1)$ and $occurs\_at(a, s)$ belong to the lowest strata.

(ii) For atoms $holds(f, \alpha(s_i))$ and $holds(f, \alpha'(s_i'))$ where α and α' are sequence of actions and s_i and s_i' are actual situations, if $level(\alpha', s_i') < level(\alpha, s_i)$ then $strata(holds(f, \alpha'(s_i'))) < strata(holds(f, \alpha(s_i)))$.

(iii) $strata(holds^+(f, \alpha(s_i))) = strata(causes(a, f, \alpha(s_i))) = strata(cancells(X, Y, f, \alpha(s_i))) < strata(noninh(f, X, \alpha(s_i))) = strata(undefined(a, \alpha(s_i))) < strata(holds^+(f, (\alpha \circ b)(s_i)))$
$\square$

Lemma 4 Let D be a domain description and let A be a set of literals in the language of ΠD. A is a consistent answer set of ΠD iff A is the consistent answer set of
$[\Pi D]_A = \Pi D \cup \{holds(f, s_0) \ : \ holds(f, s_0) \in A\} \cup \{\neg holds(f, s_0) \ : \ \neg holds(f, s_0) \in A\}$. $\square$

Proof:
$\Longrightarrow$
Let A be a consistent answer set of ΠD. By definition, A is a consistent answer set of $(\Pi D)^A$. It is easy to see that if a program Π does not contain *not*, then if A is an answer set of Π then A is an answer set of $\Pi \cup B$ for any $B \subseteq A$. Hence, A is a consistent answer set of $([\Pi D]_A)^A$. Therefore, by definition A is a consistent answer set of $([\Pi D]_A)$.

$\Longleftarrow$
Let A be the consistent answer set of $[\Pi D]_A$. It is easy to see that A satisfies $(\Pi D)^A$. Suppose A is not the minimal set that satisfies $(\Pi D)^A$. Let $A' \subset A$ satisfy $(\Pi D)^A$.
(case 1) A and A' differ on facts about the initial state.
Consider the case when $holds(f, s_0) \in A$ and $holds(f, s_0) \notin A'$. (The arguments for the other case is similar.) But then, since either $holds(f, s_0) \in A'$ or $\neg holds(f, s_0) \in A'$ we conclude

[6] For an extended logic program Π the normal logic program Π^+ denotes the program obtained by replacing each occurance of a literals $\neg l$ in Π by l'. See [GL90] for relation between answer sets of an extended logic program Π and the stable models of the corresponding normal logic program Π^+.

that $\neg holds(f, s_0) \in A'$. By consistency of A we have $\neg holds(f, s_0) \notin A$. This contradicts our assumption that $A' \subset A$.
(case 2) A and A' do not differ on facts about the initial state. But then A' will satisfy $([\Pi D]_A)^A$ contradicting our assumption that A is an answer set of $[\Pi D]_A$.
Hence, A is the minimal set that satisfies $(\Pi D)^A$ and therefore is an answer set of ΠD. $\square$

Let D be a domain description with explicit actual path $s_0, \dots, s_k$. Consider ground instantiations of all the rules of ΠD except the Second Inertia Axiom (SI), the Description of Actual Path (AL), and the Full Awareness Axiom (FA). The set of all such instantiations not containing any other situation constants except s_i will be denoted by H_i. It is easy to see that Π_D consists of the union of the sets $H_0, \dots, H_k$ together with ground instantiations of SI, AL, and FA.

Lemma 5 Models of D vs Answer sets of H_i
Let $M = (\Psi, \Sigma)$ be a model of D and for some $0 \leq i \leq k$, let A_i be a consistent answer set of $H_i \cup I$ where $I = \{h(f, s_i) : M \models f \textbf{ at } s_i\}$

1. If $a_1, \dots, a_n$ is executable from s_i in M then
   (a) $f \in {direct_{a_n}}^+(\sigma_{s_i,\alpha_{n-1}})$ iff $causes(a_n, f, \alpha_{n-1}(s_i)) \in A_i$.
   (b) $f \in {direct_{a_n}}^-(\sigma_{s_i,\alpha_{n-1}})$ iff $causes(a_n, \overline{f}, \alpha_{n-1}(s_i)) \in A_i$.
   (c) $f \in {inherited_{a_n}}^+(\sigma_{s_i,\alpha_{n-1}})$ iff $noninh(\overline{f}, a_n, \alpha_{n-1}(s_i)) \in A_i$.
   (d) $f \in {inherited_{a_n}}^-(\sigma_{s_i,\alpha_{n-1}})$ iff $noninh(f, a_n, \alpha_{n-1}(s_i)) \in A_i$.
   (e) $undefined(a_n, \alpha_{n-1}(s_i)) \notin A_i$.
   (f) $f \in \sigma_{s_i,\alpha_n} \Leftrightarrow holds(f, \alpha_n(s_i)) \in A_i$.
   (g) $f \notin \sigma_{s_i,\alpha_n} \Leftrightarrow \neg holds(f, \alpha_n(s_i)) \in A_i$.
2. If $a_1, \dots, a_n$ is not executable from s_i in M then
   $holds(f, \alpha_n(s_i)) \notin A_i$ and $\neg holds(f, \alpha_n(s_i)) \notin A_i$ and $undefined(a_n, \alpha_{n-1}(s_i)) \in A_i$.

$\square$

Proof:
We will prove this theorem using induction on the level of sequence of actions.

Base case: level = (i,0)
Directly from the conditions of the theorem.

Induction Hypothesis: $level < (n + i, m)$
Let us assume that by IH

1. If $a_1, \dots, a_k$ is executable from s_i in M then
   (a) $f \in {direct_{a_k}}^+(\sigma_{s_i,\alpha_{k-1}})$ iff $causes(a_k, f, \alpha_{k-1}(s_i)) \in A_i$.

(b) $f \in direct_{a_k}{}^-(\sigma_{s_i,\alpha_{k-1}})$ iff $causes(a_k, \overline{f}, \alpha_{k-1}(s_i)) \in A_i$.

(c) $f \in inherited_{a_k}{}^+(\sigma_{s_i,\alpha_{k-1}})$ iff $noninh(\overline{f}, a_k, \alpha_{k-1}(s_i)) \in A_i$.

(d) $f \in inherited_{a_k}{}^-(\sigma_{s_i,\alpha_{k-1}})$ iff $noninh(f, a_k, \alpha_{k-1}(s_i)) \in A_i$.

(e) $undefined(a_k, \alpha_{k-1}(s_i)) \notin A_i$

(f) $f \in \sigma_{s_i,\alpha_n} \Leftrightarrow holds(f, \alpha_k(s_i)) \in A_i$.

(g) $f \notin \sigma_{s_i,\alpha_n} \Leftrightarrow \neg holds(f, \alpha_k(s_i)) \in A_i$.

2. If $a_1, \ldots, a_k$ is not executable from s_i in M then for any fluent atom f, $holds(f, \alpha_k(s_i)) \notin A_i$ and $\neg holds(f, \alpha_k(s_i)) \notin A_i$ and $undefined(a_k, \alpha_{k-1}(s_i)) \in A_i$.

Induction: level = (n+i,m)

1. Let $a_1, \ldots, a_n$ be executable from s_i in M. Then,

(a) $f \in direct_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}})$
iff
there is an effect law (a_n **causes** f **if** p) in D such that p holds in $\sigma_{s_i,\alpha_{n-1}}$ (by definition)
iff
$H(p, \alpha_{n-1}(s_i)) \subset A_i$ (by IH)
iff
$causes(a_n, f, \alpha_{n-1}(s_i)) \in A_i$ (By consistency of A_i, rule CL, and Proposition 4.1).

(b) similar to 1 (a).

(c) $f \in inherited_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}})$
iff
$\exists b \subset a_n$ s.t. an effect law b **causes** f **if** $q \in D$, q holds in $\sigma_{s_i,\alpha_{n-1}}$, and there is no $b \subset c \subseteq a_n$ where there is an effect law c **causes** $\overline{f}$ **if** $r \in D$ s.t. r holds in $\sigma_{s_i,\alpha_{n-1}}$ (by definition)
iff
$\exists b \subset a_n$ s.t. $f \in direct_b{}^+(\sigma_{s_i,\alpha_{n-1}})$ and $\neg\exists c : b \subset c \subseteq a_n$ s.t. $f \in direct_c{}^-(\sigma_{s_i,\alpha_{n-1}})$ (by definition)
iff
$\exists b \subset a_n$ s.t. $causes(b, f, \alpha_{n-1}(s_i)) \in A_i$ and $\neg\exists c : b \subset c \subseteq a_n$ s.t. $causes(c, \overline{f}, \alpha_{n-1}(s_i)) \in A_i$ (by IH)
iff
$\exists b \subset a_n$ s.t. $causes(b, f, \alpha_{n-1}(s_i)) \in A_i$ and $cancells(b, a_n, f, \alpha_{n-1}(s_i)) \notin A_i$ (Using consistency of A and applying Proposition 4.1 to rule IA(ii).)
iff
$noninh(\overline{f}, a_n, \alpha_{n-1}(s_i)) \in A_i$. (By applying Proposition 4.1 to rule IA(iii))

(d) similar to 1 (c).

(e) $a_1, \ldots, a_n$ is executable from s_i in M $\Rightarrow$
$E_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}}) \cap E_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}}) = \emptyset$ $\Rightarrow$
$direct_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}}) \cap direct_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}}) = \emptyset$ and
$inherited_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}}) \cap inherited_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}}) = \emptyset$ $\Rightarrow$
There does not exist f such that both $causes(a_n, f, \alpha_{n-1}(s_i))$ and $causes(a_n, \overline{f}, \alpha_{n-1}(s_i))$ are in A_i, and there does not exist f such that both $noninh(f, a_n, \alpha_{n-1}(s_i))$ and $noninh(\overline{f}, a_n, \alpha_{n-1}(s_i))$ are in A_i (Using 1(a) to 1(d) of Lemma 5.) $\Rightarrow$
$undefined(a_n, \alpha_{n-1}(s_i)) \notin A_i$ (Using Proposition 4.1 on rules EA(ii) and IA(iv) of the program.)

(f) $\Longrightarrow$ Let $f \in \sigma_{s_i,\alpha_n}$ $\Rightarrow$
$f \in \sigma_{s_i,\alpha_{n-1}} \cup E_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}}) \setminus E_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$ $\Rightarrow$
At least one of the following cases is true.

i. a_n is atomic and $f \in \sigma_{s_i,\alpha_{n-1}} \setminus E_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$
ii. a_n is not atomic and $f \in \sigma_{s_i,\alpha_{n-1}} \setminus E_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}}) \setminus E_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$
iii. $f \in direct_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}})$
iv. a_n is not atomic and $f \in inherited_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}}) \setminus E_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$

i. a_n is atomic and $f \in \sigma_{s_i,\alpha_{n-1}} \setminus E_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$ $\Rightarrow$
$holds(f, s_{n-1}) \in A_i$ (by IH) and $causes(a_n, \overline{f}, \alpha_{n-1}(s_i)) \notin A_i$ (by 1(b)) and $undefined(a_n, \alpha_{n-1}(s_i)) \notin A_i$ (by 1(e)) $\Rightarrow$
$holds(f, \alpha_n(s_i)) \in A_i$ (by using Proposition 4.1 on rule FI. Note that we can use rule FI only because a_n is atomic.).

ii. a_n is not atomic and $f \in \sigma_{s_i,\alpha_{n-1}} \setminus E_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}}) \setminus E_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$ $\Rightarrow$
From $f \notin E_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}})$ and $f \notin E_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$ we can conclude that there exists an atomic action b in a_n such that there is no effect axioms of the form b **causes** $\overline{f}$ **if** p where p holds in $\sigma_{s_i,\alpha_{n-1}}$; It is easy to see that α_{n-1}, b is executable from s_i in M and $f \in \sigma_{s_i,\alpha_{n-1},b}$. By IH this implies that $holds(f, res(b, \alpha_{n-1}(s_i))) \in A_i$. From $f \notin inherited_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$ using 1(d) we can conclude that
$noninh(f, a_n, \alpha_{n-1}(s_i)) \notin A_i$. By 1(e) we have $undefined(a_n, \alpha_{n-1}(s_i)) \notin A_i$. Hence using Proposition 4.1 on the rule IA(i) we can conclude that $holds(f, \alpha_n(s_i)) \in A_i$.

iii. $f \in direct_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}})$ $\Rightarrow$
$causes(a_n, f, \alpha_{n-1}(s_i)) \in A_i$ (from 1(a)) $\Rightarrow$
$holds(f, \alpha_n(s_i)) \in A_i$ (Using 1(e) and applying Proposition 4.1 to rule EA(i)).

iv. a_n is not atomic and $f \in inherited_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}}) \setminus E_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$ $\Rightarrow$
From $f \in inherited_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}})$ it is easy to show that, there exists an action $b \subset a_n$ such that $a_1, \ldots, a_{n-1}, b$ is executable from s_i in M and $f \in direct_b{}^+(\sigma_{s_i,\alpha_{n-1}})$; from which we can conclude by (iii) that
$holds(f, res(b, \alpha_{n-1}(s_i))) \in A_i$. From $f \notin E_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$ we can conclude (by 1(d)) that $noninh(f, a_n, \alpha_{n-1}(s_i)) \notin A_i$. Hence, using Proposition 4.1, 1(e), and rule IA(i) we can conclude that $holds(f, \alpha_n(s_i)) \in A_i$.

$\Longleftarrow$ Let $holds(f, \alpha_n(s_i)) \in A_i \quad \Rightarrow$

By Proposition 4.1 at-least one of the following three cases must be true. (Note that $undefined(a_n, \alpha_{n-1}(s_i)) \notin A_i$ in all the three cases)

i. $holds(f, \alpha_{n-1}(s_i)) \in A_i$, a_n is atomic and $causes(a_n, \overline{f}, \alpha_{n-1}(s_i)) \notin A_i$. (Using rule FI)

ii. There exists an effect law (a_n **causes** f **if** p), such that $H(p, \alpha_{n-1}(s_i)) \subset A_i$. (Using rule EA(i)).

iii. There exists an action $b \subset a_n$ such that $holds(f, res(b, \alpha_{n-1}(s_i))) \in A_i$ and $noninh(f, a_n, \alpha_{n-1}(s_i)) \notin A_i$. (using rule IA(i)).

i. By IH, $holds(f, \alpha_{n-1}(s_i)) \in A_i$ implies $f \in \sigma_{s_i,\alpha_{n-1}}$. From $causes(A_n, \overline{f}, \alpha_{n-1}(s_i)) \notin A_i$ using 1(b) we can conclude that $f \notin direct_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$. Since a_n is atomic, $inherited_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}}) = \emptyset$. Hence, $f \notin E_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$. Hence, $f \in \sigma_{s_i,\alpha_n}$.

ii. There exists an effect law (a_n **causes** f **if** p), such that $H(p, \alpha_{n-1}(s_i)) \subset A_i$. By IH we have p holds in $\sigma_{s_i,\alpha_{n-1}}$. Hence, $f \in E_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}})$. Since, $a_1, \ldots, a_n$ is executable we have that $f \notin E_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$. Hence, $f \in \sigma_{s_i,\alpha_n}$.

iii. There exists an action $b \subset a_n$ such that $holds(f, res(b, \alpha_{n-1}(s_i)) \in A_i$ and $noninh(f, a_n, \alpha_{n-1}(s_i)) \notin A_i$.

We will first show that $f \notin direct_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$. Suppose it is not the case. Using arguments similar to (iii) of ($\Longrightarrow$) we will have $\neg holds(f, \alpha_n(s_i)) \in A_i$. This makes A_i inconsistent and we have a contradiction.

From $noninh(f, a_n, \alpha_{n-1}(s_i)) \notin A_i$ we conclude (using 1(d)) $f \notin inherited_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$. Hence, $f \notin E_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$.

From $holds(f, res(b, \alpha_{n-1}(s_i))) \in A_i$ using IH we can conclude that $f \in \sigma_{s_i,\alpha_{n-i}}$ or $f \in E_b{}^+(\sigma_{s_i,\alpha_{n-1}})$. If $f \in \sigma_{s_i,\alpha_{n-i}}$, then, since we showed $f \notin E_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$, we have $f \in \sigma_{s_i,\alpha_n}$.

Now suppose $f \in E_b{}^+(\sigma_{s_i,\alpha_{n-1}}) \setminus \sigma_{s_i,\alpha_{n-1}}$. This means there exists an action $b_1 \subset a_n$ such that $f \in direct_{b_1}{}^+(\sigma_{s_i,\alpha_{n-1}})$. Let b_2 be the maximal sub-action of a_n such that $f \in direct_{b_2}{}^+(\sigma_{s_i,\alpha_{n-1}})$; i.e. there does not exist an action c, $b_2 \subset c \subseteq a_n$ such that $f \in direct_c{}^+(\sigma_{s_i,\alpha_{n-1}})$.

We now claim that

(*) for all actions g, $b_2 \subset g \subseteq a_n$, if ($g$ **causes** $\overline{f}$ **if** q) in D then q does not hold in $\sigma_{s_i,\alpha_{n-1}}$.

Suppose our claim is false. Then there exists an action g, $b_2 \subset g \subseteq a_n$, such that $f \in direct_g{}^-(\sigma_{s_i,\alpha_{n-1}})$. From maximality of b_2 we have $f \in inherited_{a_n}{}^-(\sigma_{s_i,\alpha_{n-1}})$. By 1(d) we have $noninh(f, a_n, \alpha_{n-1}(s_i)) \in A_i$. This contradicts our original assumption that $noninh(f, a_n, \alpha_{n-1}(s_i)) \notin A_i$. Hence our claim (*) is true.

Therefore, by definition $f \in E_{a_n}{}^+(\sigma_{s_i,\alpha_{n-1}})$ and hence $f \in \sigma_{s_i,\alpha_n}$.

(g) Similar to the proof of 1(f).

2. $a_1, \ldots a_n$ is not executable from s_i in $M \quad \Rightarrow$

There are two cases:

(case 1) $a_1, \ldots, a_{n-1}$ is executable from s_i in M.
(case 2) $a_1, \ldots, a_{n-1}$ is not executable from s_i in M.

(case 1) ${E_{a_n}}^+(\sigma_{s_i,\alpha_{n-1}}) \cap {E_{a_n}}^-(\sigma_{s_i,\alpha_{n-1}}) \neq \emptyset \quad \Rightarrow$
(case 1.1) ${direct_{a_n}}^+(\sigma_{s_i,\alpha_{n-1}}) \cap {direct_{a_n}}^-(\sigma_{s_i,\alpha_{n-1}}) \neq \emptyset$ or
(case 1.2) ${inherited_{a_n}}^+(\sigma_{s_i,\alpha_{n-1}}) \cap {inherited_{a_n}}^-(\sigma_{s_i,\alpha_{n-1}}) \neq \emptyset$
(The other two cases are not possible.)

(case 1.1) ${direct_{a_n}}^+(\sigma_{s_i,\alpha_{n-1}}) \cap {direct_{a_n}}^-(\sigma_{s_i,\alpha_{n-1}}) \neq \emptyset \quad \Rightarrow$
Similar to the case 1 (a) we can show that for some f, $causes(a_n, f, \alpha_{n-1}(s_i)) \in A_i$ and $causes(a_n, \overline{f}, \alpha_{n-1}(s_i)) \in A_i \Rightarrow$
$undefined(a_n, \alpha_n(s_i)) \in A_i$ (by rule EA(ii).)

(case 1.2) ${inherited_{a_n}}^+(\sigma_{s_i,\alpha_{n-1}}) \cap {inherited_{a_n}}^-(\sigma_{s_i,\alpha_{n-1}}) \neq \emptyset \quad \Rightarrow$
There exists $a, a' \subset a_n$ such that $f \in {direct_a}^+(\sigma_{s_i,\alpha_{n-1}})$, $f \in {direct_{a'}}^-(\sigma_{s_i,\alpha_{n-1}})$, and there does not exist b and b' where $a \subset b \subseteq a_n$, $a' \subset b' \subseteq a_n$, $f \in {direct_b}^-(\sigma_{s_i,\alpha_{n-1}})$ and $f \in {direct_{b'}}^+(\sigma_{s_i,\alpha_{n-1}})$.

Similar to the case 1 (a) we can show that $causes(a, f, \alpha_{n-1}(s_i)) \in A_i$, $causes(a', \overline{f}, \alpha_{n-1}(s_i)) \in A_i$, $cancells(a, a_n, f, \alpha_{n-1}(s_i)) \notin A_i$ and $cancells(a', a_n, f, \alpha_{n-1}(s_i)) \notin A_i$. By using IA(iii) and Proposition 4.1 we have $noninh(f, a_n, \alpha_{n-1}(s_i)) \in A_i$ and $noninh(\overline{f}, a_n, \alpha_{n-1}(s_i)) \in A_i \Rightarrow$
$undefined(a_n, \alpha_n(s_i)) \in A_i$ (by Proposition 4.1 and rules IA(iv).).

(case 2) Using IH and rule EA(iii) we get $undefined(a_n, \alpha_n(s_i)) \in A_i$.

Since for any f, all rules with either $holds(f, \alpha_n(s_i))$ or $\neg holds(f, \alpha_n(s_i))$ in their head, have *not* $undefined(a_n, \alpha_{n-1}(s_i))$ in their body, $holds(f, \alpha_n(s_i)) \notin A_i$ and $\neg holds(f, \alpha_n(s_i)) \notin A_i$.

This ends the proof of this lemma. □

Lemma 6 For every model $M = (\Psi, \Sigma)$ of a consistent domain description D, there exists a consistent answer set A_i of $H_i \cup I$ where $I = \{h(f, s_i) : M \models f \textbf{ at } s_i\}$ s.t.

1. $A_i \models h(f, s_i) \Leftrightarrow M \models f \textbf{ at } s_i$.
2. $A_i \models h(f, \alpha(s_i)) \Leftrightarrow M \models f \textbf{ after } \alpha \textbf{ at } s_i$.

□

Proof:
From Proposition 4.2, it can be seen that $[H_i \cup I]^+$ is locally stratified. Note that since $[H_i \cup I]^+$ also contains complete information about the initial state and has no disjunctions, it will have a unique answer set, say ${A_i}^+$. We will first show that ${A_i}^+$ is coherent.[7] Incoherency

[7] ${A_i}^+$ is coherent iff there does not exist an atom, l, such that both l and l' are in ${A_i}^+$. For an ELP Π, consistency of answer set of Π corresponds to the coherence of answer sets of Π^+.

is possible if we derive both $holds(F, S)$ and $holds'(F, S)$ from $[H_i \cup I]^+$. $holds(F, S)$ and $holds'(F, S)$ can be derived using FI, EA(i), and IA(i). Using induction on the level of S, it can be easily shown that
(i) to derive both $holds(F, S)$ and $holds'(F, S)$ using FI the previous situation would have to be incoherent, violating the inductive hypothesis.
(ii) FI and EA(i) can not lead to incoherency. If the body of EA(i) is true for $holds(F, S)$ ($holds'(F, S)$ respectively) then the atom with $causes$ will block FI from deriving $holds'(F, S)$ ($holds(F, S)$ respectively).
(iii) the bodies of FI and IA(i) can not be true at the same time since FI is applicable only for atomic actions, while IA(i) needs the action to be non-atomic.
(iv) we can not have both $holds(f, s)$ and $holds'(f, s)$ derived using EA(i). In that case $undefined$ will block the derivation of both.
(v) EA(i) and IA(i) can not lead to incoherency. If the body of EA(i) is true for $holds(F, S)$ ($holds'(F, S)$ respectively) then the atom with $noninh$ will become true blocking IA(i) from deriving $holds'(F, S)$ ($holds(F, S)$ respectively).
(vi) we can not have both $holds(F, S)$ and $holds'(F, S)$ derived from IA(i) at the same time, $noninh$ will block the derivation of at-least one of them.
Hence, ${A_i}^+$ is coherent and hence [GL91] A_i is consistent and is the unique answer set of $H_i \cup I$.

Using Lemma 5 w.r.t. D we have that A_i and M agree on all fluent facts and all hypothesis of the form f **after** α **at** s_i in D. □

Lemma 7 Let D be a consistent domain description with explicit actual path $s_0, \ldots, s_k$ and let Π_1 be obtained from ΠD by replacing (SI) by

$h(F, s_i) \leftarrow h(F, result(a_{i-1}, s_{i-1}))$

where $0 < i \leq k$ and (a_{i-1} **occurs_at** s_{i-1}) $\in D$.

Then Π_1 is equivalent to ΠD. □

Lemma 8 Let D be a consistent domain description with explicit actual path $s_0, \ldots, s_k$, let r_i be the rule $h(f, s_i) \leftarrow h(f, result(a_{i-1}, s_{i-1}))$, let $M = (\Psi, \Sigma)$ be a model of D, and let $T_m = (H_0 \cup I_0) \cup (r_1 \cup H_1) \cup \ldots \cup (r_m \cup H_m)$ where $I_0 = \{h(f, s_0) : M \models f$ **at** $s_0\}$. For any $0 \leq m \leq k$, the program T_m has a unique consistent answer set A_m s.t.

1. $A_m \models h(f, s_i) \Leftrightarrow M \models f$ **at** s_i.
2. $A_m \models h(f, \alpha(s_i)) \Leftrightarrow M \models f$ **after** α **at** s_i.

□

Proof. We use induction on m.

Base: $m = 0$. Conclusion of the Lemma follows immediately from Lemma 6.

Inductive step: Obviously, $T_m = T_{m-1} \cup (r_m \cup H_m)$. By inductive hypotheses T_{m-1} has a unique consistent answer set A_{m-1}. Notice that $head(T_{m-1})$ forms a splitting set [LT94] for T_m and hence

(i) A_m is an answer set of T_m iff $A_m = A_{m-1} \cup C$ where C is an answer set of the partial evaluation $e(T_m, A_{m-1})$ of T_m w.r.t. A_{m-1}. It is easy to see that

(ii) $e(T_m, A_{m-1}) = H_m \cup I_m$ where

$I_m = \{h(f, s_m) \ : \ A_{m-1} \models h(f, result(a_{m-1}, s_{m-1}))\}$

By inductive hypotheses, (2) implies that for any fluent literal f s.t. $h(f, s_m) \in I_m$, $M \models f$ **after** a_{m-1} **at** s_{m-1}. Let $\Sigma(s_k)$ be the actual path of D. Then $\Sigma(s_m) = \Sigma(s_{m-1}) \circ a_{m-1}$ and hence $M \models f$ **at** s_m. Therefore, I_m satisfies the conditions of Lemma 6 and hence $H_m \cup I_m$ has a unique consistent answer set, C, which satisfies conditions (i) and (ii). This, together with (1), implies the conclusion of Lemma 3 for T_m. □

Lemma 9 For every model $M = (\Psi, \Sigma)$ of a consistent domain description D with explicit actual path $s_0, \ldots, s_k$ there exists a unique consistent answer set, A, of ΠD s.t.

1. $A \models h(f, s_i) \ \Leftrightarrow \ M \models f$ **at** s_i.
2. $A \models h(f, \alpha(s_i)) \ \Leftrightarrow \ M \models f$ **after** α **at** s_i.

□

Proof: By Lemma 7, A is an answer set of ΠD iff A is an answer set of Π_1. Notice that Π_1 can be split using the splitting set $U = Lit(\{imm\_follows, occurs\_at\})$. It can be seen that the bottom contains only the rules describing the actual path, (AL), and the top contains the rest of Π_1. It can easily be seen that the bottom part has a unique answer set, which we will denote as B. Let us consider $e_U(\Pi_1 \setminus b_u(\Pi_1), B)$. Notice that the top and bottom contain no common literals, hence $e_U(\Pi_i \setminus b_u(\Pi_1), B) = \Pi_1 \setminus (AL)$. We will denote $\Pi_1 \setminus (AL)$ by Π_2. Let $T_k = (H_0 \cup I_0) \cup (r_1 \cup H_1) \cup \ldots \cup (r_k \cup H_k)$ where $I_0 = \{h(f, s_0) : M \models f$ **at** $s_0\}$ and r_i be the rule $h(f, s_i) \leftarrow h(f, result(a_{i-1}, s_{i-1}))$. By Lemma 8, T_k has a unique answer set which we will denote by, C. Notice that by Lemma 4, C is an answer set of Π_2 iff C is an answer set of $[\Pi_2]_C$. $[\Pi_2]_C$ contains complete information about about the initial state, and hence, by Lemma 3, we can remove the disjunctions, (FA), and still have the same answer sets. But it can easily be seen that, $[\Pi_2]_C \setminus (FA) = T_k$. Following backward through our proof, C is therefore the unique answer set of Π_2 w.r.t M. By Lemma 2, since both B and C are unique, $A = B \cup C$ is the unique answer set of Π_1 for M and hence is the unique answer set of ΠD for M.

□

Theorem 2 For any consistent domain description D with explicit actual path, its logic programming approximation ΠD is sound w.r.t. D. □

Proof: Follows directly from Lemma 9. □

The following example shows that, without further restrictions, ΠD is not complete.

Example 4.1 Consider the following domain description D_8:

$$\left.\begin{array}{l} s_0 \textbf{ precedes } s_1 \\ \{a,b\} \textbf{ occurs\_at } s_0 \\ \neg f \textbf{ at } s_0 \\ a \textbf{ causes } f \textbf{ if } p \\ b \textbf{ causes } \neg f \textbf{ if } p \\ \{a,b\} \textbf{ causes } f \textbf{ if } \neg p \\ f \textbf{ at } s_1 \end{array}\right\} D_8$$

There are two candidates for the initial state of a model of D_8:

$\Psi(\Sigma(s_0)) = \{\}$ and $\Psi'(\Sigma(s_0)) = \{p\}$. It is easy to see that $\Psi(\Sigma(s_0) \circ \{a,b\}) = \{f\}$, while $\Psi'(\Sigma(s_0) \circ \{a,b\})$ is undefined. It is easy to show that $M' = (\Psi', \Sigma)$ is not a model of D_8 and hence, $M = (\Psi, \Sigma)$ is the only model of D_8. Hence, $D_8 \models \neg p$ **at** s_0 and $D_8 \models f$ **after** $\{a,b\}$ **at** s_0.

Now let us consider the program ΠD_8.

It is easy to show that ΠD_8 has two answer sets, A and A', where
$\{\neg holds(p, s_0), holds(f, s_1), holds(f, result(\{a,b\}, s_0))\} \subset A$ and
$\{holds(p, s_0), holds(f, s_1), undefined(\{a,b\}, s_0)\} \subset A'$.
Since $holds(p, s_0)$ and $holds(f, result(\{a,b\}, s_0))$ each belong to only one of the answer sets of ΠD_8, $\Pi D_8 \not\models \neg p$ **at** s_0 and $\Pi D_8 \not\models f$ **after** $\{a,b\}$ **at** s_0 and is therefore not complete.
$\square$

Now we will show that by adding a stronger condition on the initial state we can achieve both soundness and completeness.

Definition 4.7 A domain description D is said to be **strongly complete** if for any fluent f in the language either f **at** s_0 or $\neg f$ **at** s_0 is in D. $\square$

Theorem 3 For any strongly complete, consistent domain description D with explicit actual path its logic programming approximation ΠD is sound and complete w.r.t. D. $\square$

Proof: Soundness was proven in Theorem 2.
To prove completeness, notice that it can easily be seen that D has only one model, say M. Since M is unique

1. $D \models f$ **at** $s_i \;\Leftrightarrow\; M \models f$ **at** s_i
2. $D \models f$ **after** α **at** $s_i \;\Leftrightarrow\; M \models f$ **after** α **at** s_i

By Lemma 9, there exist a unique answer set, A, of ΠD s.t.

1. $A \models h(f, s_i) \;\Leftrightarrow\; M \models f$ **at** s_i.
2. $A \models h(f, \alpha(s_i)) \;\Leftrightarrow\; M \models f$ **after** α **at** s_i.

and therefore

1. $A \models h(f, s_i) \;\Leftrightarrow\; D \models f$ **at** s_i.

2. $A \models h(f, \alpha(s_i)) \;\Leftrightarrow\; D \models f$ **after** α **at** s_i.

To complete the proof we need to show that A is the only answer set of ΠD. From Proposition 4.2 ΠD is locally stratified and by Lemma 3 the disjunctions can be removed without changing the answer sets, hence ΠD has a unique answer set, A. □

5 Actions with non-predictable effects

In this section we discuss another extension of our language which will allow representation of actions with non-predictable (non-deterministic) effects. Let us start with the following example from [San94]:

Example 5.1 This is a version of shooting scenario from previous examples. This time the gun is a revolver (to save space we assume that it is a two-shooter, i.e. has two chambers), with a top chamber aligned with the gun's barrel. An action *shoot* is successful (i.e. results in the death of Fred) only if there is a bullet in the top chamber. Action *load* puts the bullet in the top chamber of the cylinder and a new actions, *spin*, rotates the cylinder. As a result of *spin* top chamber may or may not contain a bullet.

Spin is a typical example of an action with non-predictable effects. There are at least two explanations for this phenomena: According to the first one *spin* is a single indivisible action which can cause the chamber containing the bullet to be in any of the possible positions but the actual effect of performing *spin* may be different at different times. This approach seems to be adopted in [KL94a, San94]. Another explanation (see [Pin94]) views *spin* not as a single action, but rather as a collection of actions with precise deterministic effect. Unpredictability of the outcome is caused simply by the lack of knowledge which (unique) action from this collection was actually performed. Our treatment will follow the second view which we find more intuitive in certain cases. □

To represent knowledge about the occurrences and effects of non-deterministic actions we will augment the syntax and semantics of $\mathcal{L}_2$. First we expand the alphabet of $\mathcal{L}_2$ by a new symbol $|$. An expression a of the form $a_1 \mid \ldots \mid a_n$ (where $a_1 \ldots a_n$ are actions) will be called a *non-deterministic action*, with (possible) *instances* $a_1, \ldots, a_n$. By *generalized actions* we mean deterministic and non-deterministic actions. Propositions of the new language $\mathcal{L}_3$ are defined as follows:

1. Occurrence facts and hypothesis of $\mathcal{L}_3$ are expressions of the form:

$[a_1, \ldots, a_n]$ **occurs_at** s

f **after** $[a_1, \ldots, a_n]$ **at** s

where $a_1, \ldots, a_n (n > 0)$ are **generalized** actions, f is a fluent literal, and s is a situation.

2. All other propositions are exactly the same as in $\mathcal{L}_2$.

Notice that non-deterministic actions can not occur in causal laws. *The effect of executing such action in a particular situation is determined by the effects of its (deterministic) instances.* The semantics of domain descriptions of $\mathcal{L}_3$ refines this intuition.

The notion of interpretation of $\mathcal{L}_3$, i.e. the definition of transition function and situation assignment, is the same as in $\mathcal{L}_2$. To define a model we only need to expand the definition of validity to occurrence facts containing non-deterministic action.

We will say that a sequence $[a_1^*, \ldots, a_n^*]$ of deterministic actions is a (possible) instance of a sequence of generalized actions $[a_1, \ldots, a_n]$ if, for any $1 \leq i \leq n$, $a_i^* = a_i$ or a_i^* is an instance of a_i.

An atomic fact, A, of the form $([a_1, \ldots, a_n]\ \textbf{occurs\_at}\ \ s)$ is *true* in an interpretation, M, if there is an instance $[a_1^*, \ldots, a_n^*]$ of $[a_1, \ldots, a_n]$ such that $([a_1^*, \ldots, a_n^*]\ \textbf{occurs\_at}\ \ s)$ is true in M.

To deal with hypothetical reasoning in $\mathcal{L}_3$ we also need to expand the notion of truth for hypothesis containing non-deterministic actions.

We say that A, of the form $(f\ \textbf{after}\ \ [a_1, \ldots, a_n]\ \textbf{at}\ \ s)$, is *true* (*false*) in an interpretation, M, if for any instance $[a_1^*, \ldots, a_n^*]$ of $[a_1, \ldots, a_n]$,
$(f\ \textbf{after}\ \ [a_1^*, \ldots, a_n^*]\ \textbf{at}\ \ s)$ is true (false) in M.

Notice that in this formulation, hypothetical queries can be answered as *yes*, *no*, and *unknown* (the latter corresponding to neither true nor false case.)

To illustrate the definition, let us consider the following representation of the above example:

Example 5.2

$$\left.\begin{array}{l}
Facts: \\
(f1)\ alive\ \textbf{at}\ \ s_0 \\
(f2)\ s_0\ \textbf{precedes}\ \ s_1 \\
(f3)\ load(c_1)\ \textbf{occurs_at}\ \ s_1 \\
(f4)\ spin\ \textbf{occurs_at}\ \ s_2 \\
\\
Laws: \\
(l0)\ load(X)\ \textbf{causes}\ \ bullet_in(X) \\
(l1)\ load(X)\ \textbf{causes}\ \ top(X) \\
(l2)\ shoot\ \textbf{causes}\ \ dead\ \textbf{if}\ \ top(X), bullet_in(X) \\
(l3)\ shoot\ \textbf{causes}\ \ top(c_1)\ \textbf{if}\ \ top(c_2) \\
(l3')\ shoot\ \textbf{causes}\ \ \neg top(c_1)\ \textbf{if}\ \ \neg top(c_2) \\
(l4)\ shoot\ \textbf{causes}\ \ top(c_2)\ \textbf{if}\ \ top(c_1) \\
(l4')\ shoot\ \textbf{causes}\ \ \neg top(c_1)\ \textbf{if}\ \ \neg top(c_2) \\
(l5)\ move_on_top(X)\ \textbf{causes}\ \ top(X)
\end{array}\right\} D_9$$

Here variable X ranges over two chambers c_1 and c_2 and spin stands for a non-deterministic action $move\_on\_top(c_1)|move\_on\_top(c_2)$. □

Proposition 5.1 For the above domain description D_9 we have

1. $D_9 \models (dead$ **after** $[shoot, shoot])$

2. $D_9 \models (bullet\_in(c_1) \wedge top(c_1)$ **at** $s_2)$

3. $D_9 \models (top(c_1) \vee top(c_2)) \wedge bullet\_in(c_1)$ **at** $s_N)$

$\square$

6 Conclusion

In this paper we extended the language $\mathcal{L}_1$ to allow compound actions – actions that consist of simultaneous execution of a set of unit actions. The extended language $\mathcal{L}_2$ allows both hypothetical reasoning about effect of actions (by allowing hypothetical statements in the queries), which is useful in planning, and reasoning about actual occurances of actions, which is useful in formalizing reactive behavior of a robot and in formalizing execution of transactions in databases. Moreover, it can reason with compound actions and elegantly capture the associated frame problem, where we do not want to explicitly specify the effect of all possible compound actions; rather we incorporate inheritance mechanisms such that a compound action normally inherits the effects of its sub-actions.

Some of the recent work that are close to our work in terms of allowing both concurrent actions and actual observations are in [McC, KM96]. But, it seems like that in both these works the only kind of compound actions that are allowed are the ones where the sub-actions do not conflict with each other or with the overall action. But since both the works are ongoing it is not appropriate at this point to give a detailed comparison between them and our proposal.

References

[AB91] K. Apt and M. Bezem. Acyclic programs. *New Generation Computing*, 9(3,4):335–365, 1991.

[ALP94] J Alferes, R. Li, and L Pereira. Concurrent actions and changes in the situation calculus. In H. Geffner, editor, *Proc of IBERAMIA 94*, pages 93–104. McGraw Hill, 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.

[BG94] C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19,20:73–148, 1994.

[BGP95] C. Baral, M. Gelfond, and A. Provetti. Representing Actions : Laws, Observations and Hypothesis. *Journal of Logic Programming (to appear)*, 1996.

[BT94] S. Bornscheuer and M. Thielscher. Representing concurrent actions and solving conflicts. In *Proc. of German Conference on AI*, 1994.

[Cla78] K. Clark. Negation as failure. In Herve Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

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

[Dix91] J. Dix. Classifying semantics of logic programs. In *Proceedings of* International Workshop in logic programming and nonmonotonic reasoning*, Washington D.C.*, pages 166–180, 1991.

[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.

[Gel94] M. Gelfond. Logic programming and reasoning with incomplete information. *Annals of Mathematics and Artificial Intelligence*, 12:19–116, 1994.

[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.

[GL95] E. Giunchiglia and V. Lifschitz. Dependent fluents. In *Proc. of IJCAI 95*, pages 1964–1969, 95.

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

[Kar93] G. Kartha. Soundness and completeness theorems for three formalizations of action. In *IJCAI 93*, pages 724–729, 1993.

[KL94a] G. Kartha and V. Lifschitz. Actions with indirect effects: Preliminary report. In *KR 94*, pages 341–350, 1994.

[KM96] A. Kakas and R. Miller. A simple declarative language for describing narratives with actions. *Journal of Logic Programming (to appear)*, 1996.

[LLL+94] Y. Lesperance, H. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl. A logical approach to high level robot programming – a progress report. In *Working notes of the 1994 AAAI fall symposium on Control of the Physical World by Intelligent Systems (to appear), New Orleans, LA*, November 1994.

[LMR92] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of disjunctive logic programming.* The MIT Press, 1992.

[LMT93] V. Lifschitz, N. McCain, and H. Turner. Automation of reasoning about action: a logic programming approach. In *Posters of the International Symposium on Logic Programming*, 1993.

[LS91] F. Lin and Y. Shoham. Provably correct theories of actions: preliminary report. In *Proc. of AAAI–91*, 1991.

[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.

[McC] J. McCarthy. Overcoming an unexpected obstacle. manuscript, 1992.

[McC86] J. McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 26(3):89–116, 1986.

[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.

[MS94] R. Miller and M. Shanahan. Narratives in the situation calculus. *Journal of Logic and Computation*, 4(5):513–530, october 1994.

[MT94] N. McCain and H. Turner. Language independence and language tolerance in logic programs. In *Proc. of the Eleventh Intl. Conference on Logic Programming*, pages 38–57, 1994.

[PCA90] L. Pereira, L. Caires, and J. Alferes. Classical negation in logic programs. In *7 Simposio Brasiliero de Inteligencia Artificial*, 1990.

[Pin94] J. Pinto. *Temporal Reasoning in the Situation Calculus.* PhD thesis, University of Toronto, Department of Computer Science, February 1994. KRR-TR-94-1.

[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.

[Pro96] A. Provetti. Hypothetical reasoning about actions: from situation calculus to event calculus. *Computational Intelligence*, 12(3), 1996.

[Prz90] T. Przymusinski. Stationary semantics for disjunctive logic programs and deductive databases. In *North American Conference on Logic Programming*, pages 40–62, 1990.

[PW89] D. Pearce and G. Wagner. Reasoning with negative information 1 – strong negation in logic programming. Technical report, Gruppe fur Logic, Wissentheorie and Information, Freie Universitat Berlin, 1989.

[Rei91] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380. Academic Press, 1991.

[San92] E. Sandewall. Features and fluents: A systemetic approach to the representation of knowledge about dynamical systems. Technical report, Institutionen for datavetenskap, Universitetet och Tekniska hogskolan i Linkoping, Sweeden, 1992.

[San94] E. Sandewall. The range of applicability of some non-monotonic logics for strict inertia. *Journal of Logic and Computation*, 4(5):581–616, october 1994.

[Tur93] H. Turner. A monotonicity theorem for extended logic programs. In D. S. Warren, editor, *Proc. of 10th International Conference on Logic Programming*, pages 567–585, 1993.

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

[Wat94] R. Watson. An Inference Engine for Epistemic Specifications, 1994. M.S Thesis, Department of Computer Science, University of Texas at El Paso.

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