REASONING ABOUT EFFECTS OF CONCURRENT ACTIONS

CHITTA BARAL AND MICHAEL GELFOND

▷ Gelfond and Lifschitz introduce a declarative language $\mathcal{A}$ for describing effects of actions and describe translations of theories in this language into extended logic programs. In this paper we extend the language $\mathcal{A}$ and its translation to allow reasoning about the effects of concurrent actions. The logic programming formalization of situation calculus with concurrent actions presented in the paper is of independent interest and may serve as a test bed for the investigation of various transformations and logic programming inference mechanisms. ◁

1. INTRODUCTION

Gelfond and Lifschitz [GL92] introduce a declarative language $\mathcal{A}$ for describing effects of actions and define the semantics of this language based on the notion of a finite automaton. The simplicity of the language and its semantics facilitates the description of the ontology of actions and contributes to establishing correctness (and sometimes completeness) of various logical formalizations of their effects. In particular, a theory of action stated in the language of extended logic programs(ELP's) [GL90] was described in [GL92] as a translation from a subset of $\mathcal{A}$ and proven to be sound w.r.t. the automata based semantics. Kartha [Kar93] recently proved soundness and completeness of this semantics with respect to formalizations proposed earlier by Pednault [Ped89], Reiter [Rei91] and Baker [Bak91].

*Address correspondence to*Department of Computer Science, The University of Texas at El Paso. El Paso, Texas 79968 USA.
E-mail: {chitta,mgelfond}@cs.utep.edu
Web: http://cs.utep.edu/csdept/krgroup.html

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/93/$3.50

Although the language $\mathcal{A}$ is adequate for formalizing several interesting domains, its expressive power is rather limited. In particular, every action is assumed to be executable in any situation and only one action can be performed at a time. In this paper we expand the syntax and semantics of $\mathcal{A}$ to remove these limitations and to allow for a representation of concurrent actions. (For some other recent extensions of $\mathcal{A}$ see [KL94, BG94b, HT93].) Our treatment of concurrency in this paper is along the lines suggested in [GLR91]. As in [GL92], we translate theories in the resulting language $\mathcal{A}_C$ into logic programs and prove correctness of these translations. The translations can be viewed as a logic programming counterpart of situation calculus [MH69] and are interesting in their own right.

The paper is organized as follows: In section 2 we define the syntax and semantics of the language $\mathcal{A}_C$. Section 3 describes the translations of theories from $\mathcal{A}_C$ into logic programs while section 4 illustrates the translations by the way of examples. In section 5 we discuss where and how our paper fits into the state of current research in "reasoning about actions". Proofs of theorems are given in the Appendix.

2. A LANGUAGE $\mathcal{A}_C$

2.1. *Syntax*

First we will recall the syntax of language $\mathcal{A}$ from [GL92].

2.1.1. Syntax of $\mathcal{A}$ The alphabet of $\mathcal{A}$ consists of two disjoint nonempty sets Σ_1 and Σ_2 of symbols, called fluent names and action names. A *fluent literal* is a fluent name possibly preceded by $\neg$.
A *v-proposition* is an expression of the form

$$f \ \textbf{after} \ a_1, \ldots, a_m \tag{1}$$

where f is a fluent literal, and $a_1, \ldots, a_m$ $(m \geq 0)$ are action names. If $m = 0$, (1) is written as

initially f.

An *e-proposition* is an expression of the form

$$a \ \textbf{causes} \ f \ \textbf{if} \ p_1, \ldots, p_n \tag{2}$$

where a is an action name, and each of $f, p_1, \ldots, p_n$ $(n \geq 0)$ is a fluent literal. The literals $p_1, \ldots, p_n$ are called *preconditions* of (2). If $n = 0$, we write this proposition as

a **causes** f.

A *domain description* in $\mathcal{A}$ is a set of v- and e-propositions.

2.1.2. Syntax of $\mathcal{A}_c$ The syntax of $\mathcal{A}_C$ **differs from the syntax of $\mathcal{A}$ only in the definition of action names**. By an *action name* of $\mathcal{A}_C$ we mean a non-empty finite set $\{a_1, \ldots, a_n\}$ of elements of Σ_2. Intuitively, an action name $\{a_i\}$ denotes a *unit action* while an action name $a = \{a_1, \ldots, a_n\}$ where $n > 1$ denotes a *compound action* – a set of unit actions which are performed concurrently and which start and stop cotemporaneously. For simplicity we will often identify a unit action name $\{a_i\}$ with a_i. To illustrate the notion of a domain description in $\mathcal{A}_C$ let us consider the following examples from [GLR91]:

Example 2.1. Mary is lifting a bowl of soup from the kitchen table, while John is opening the door to the dining room.

To represent this story in $\mathcal{A}_C$ let us consider an alphabet consisting of fluent names *lifted* and *opened* and two unit actions *lift* and *open*. The initial situation is described by v–propositions:

initially $\neg lifted$ **initially** $\neg opened$

The effects of the actions can be described by the axioms:

$\{lift\}$ **causes** *lifted* $\{open\}$ **causes** *opened*

The resulting domain description will be denoted by D_1. Intuitively, the effects of the two actions of D_1 are completely independent and so both *lifted* and *opened* should hold after the execution of the compound action $\{lift, open\}$. In a sense, this compound action *inherits* its effect from its subactions. □

The next example describes actions whose effects are mutually dependent.

Example 2.2. Whenever Mary tries to lift the bowl with one hand, she spills the soup. When she uses both hands, she does not spill the soup. We know that the soup is not spilled initially.

This time let us consider an alphabet consisting of a fluent name *spilled* and two unit actions *lift_l* and *lift_r*. The initial situation may be described by a proposition:

initially $\neg$ *spilled*

and the effects of actions are represented by propositions:

$\{lift\_l\}$ **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_2. The last effect law explicitly cancels inheritance of *spilled* by the compound action $\{lift\_l,\ lift\_r\}$. □

Domain descriptions in $\mathcal{A}_C$ are used together with the following informal assumptions:

(a) Changes in the values of fluents can only be caused by execution of actions whose names belong to the alphabet of the language of D.

(b) Effects of an action are either directly specified by the e-propositions in the domain description D or inherited from its sub-actions.

Development of the precise semantics of domain descriptions of $\mathcal{A}_C$ which incorporates these informal assumptions is the subject of the next section.

2.2. Semantics

To describe the semantics of $\mathcal{A}_C$, we will define "models" of a domain description, and when a v-proposition is "true" in a model. If a v-proposition P is true in all models of a domain description D, we say that D *entails* P.

As defined in [GL92], a *state* is a set of fluent names; given a fluent name f and a state σ, we say that f *holds* in σ if $f \in \sigma$; $\neg f$ *holds* in σ if $f \notin \sigma$.

A *transition function* is a mapping Φ of a subset of the set of pairs (a, σ), where a is an action name and σ is a state, into the set of states.[1] As in [GL92], a *structure* is a pair (σ_0, Φ), where σ_0 is a state (called the *initial state* of the structure), and Φ is a transition function. We say that a sequence of action names $a_1, \ldots, a_m$ is *executable* in a structure $M = (\sigma_0, \Phi)$ if for every $1 \leq k \leq m$

$$\Phi(a_k, \Phi(a_{k-1}, \ldots, \Phi(a_1, \sigma_0) \ldots))$$

is defined. The resulting state will be denoted by $M^{(a_1, \ldots, a_m)}$.

We say that a v-proposition (1) is *true* (*false*) in a structure M if

1. $a_1, \ldots, a_m$ is executable in M,

2. f holds (does not hold) in $M^{(a_1, \ldots, a_m)}$.

In particular, the proposition " **initially** f" is true in M iff f holds in the initial state of M.

We say that execution of an action a in a state σ *immediately causes* a fluent literal f if

there is an e–proposition "a **causes** f **if** $p_1, \ldots, p_n$" from the domain D such that for every i, $1 \leq i \leq n$, p_i holds in σ.

We say that execution of an action a in a state σ *causes* a fluent literal f if

1. a *immediately causes* f, or

2. a *inherits* the effect f from its subsets in σ, i.e., there is a $b \subset a$, such that execution of b in σ *immediately causes* f and there is no c such that $b \subset c \subseteq a$ and execution of c in σ immediately causes $\neg f$.

Let a be an action and σ be a state and consider:

$E^+(a, \sigma) = \{f : f$ is a fluent name and execution of a in σ causes $f\}$,

$E^-(a, \sigma) = \{f : f$ is a fluent name and execution of a in σ causes $\neg f$ $\}$.

[1]Recall that in the definition of a transition function in the semantics of $\mathcal{A}$, Φ must be defined on the set of all such pairs.

A structure (σ_0, Φ) will be called a *model* of a domain description D if the following conditions are satisfied:

1. Every v-proposition from D is true in (σ_0, Φ);
2. For every action $a = \{a_1, \ldots, a_n\}$ and every state σ

   (i) if $E^+(a,\sigma) \cap E^-(a,\sigma) = \emptyset$ then $\Phi(a,\sigma)$ is defined and

$$\Phi(a,\sigma) = \sigma \cup E^+(a,\sigma) \setminus E^-(a,\sigma).$$

   (ii) otherwise $\Phi(a,\sigma)$ is undefined.

Observation 2.1. There can be at most one transition function Φ satisfying conditions (i)–(ii). Consequently, different models of the same domain description can differ only by their initial states. □

Example 2.3. Consider the domain description D_1 from Example 1, the initial state $\sigma_0 = \emptyset$ and the transition function Φ defined as follows:

$\Phi(open, \sigma) = \sigma \cup \{opened\}$

$\Phi(lift, \sigma) = \sigma \cup \{lifted\}$

$\Phi(\{open, lift\}, \sigma) = \sigma \cup \{opened, lifted\}$

It is easy to see that the structure (σ_0, Φ) is the only model of the domain description D_1 and therefore D_1 entails v–propositions *opened* **after** $\{open, lift\}$ and *lifted* **after** $\{open, lift\}$. □

Example 2.4. Consider a domain description D_3 containing three unit actions *paint*, *close* and *open*, and two fluents, *opened* and *painted*. The effects of these actions are defined by the following e–propositions:

close **causes** $\neg$*opened*

open **causes** *opened*

paint **causes** *painted*.

Let a transition function Φ be defined as follows:

$\Phi(\emptyset, \sigma) = \sigma$

$\Phi(paint, \sigma) = \sigma \cup \{painted\}$

$\Phi(close, \sigma) = \sigma \setminus \{opened\}$

$\Phi(open, \sigma) = \sigma \cup \{opened\}$

$\Phi(\{paint, close\}, \sigma) = \sigma \cup \{painted\} \setminus \{opened\}$

$\Phi(\{paint, open\}, \sigma) = \sigma \cup \{painted\} \cup \{opened\}$

Notice, that for a pair (A, σ) where σ is an arbitrary state and $\{open, close\} \subseteq A$, Φ is undefined.

It is easy to see that any structure (σ, Φ) where $\sigma \subset \{opened, painted\}$ is a model of D_3 and that D_3 has no other models. □

For a domain description D and its model $M = (\sigma_0, \Phi)$, if $\Phi(a, \sigma)$ is undefined for some action a and state σ, we can consider it to mean that a is not possible in a situation whose state is σ. We can use this to specify conditions when an action is possible and when it is not. The following example makes it clear.

Example 2.5. Let us consider a variant of Example 2, where Mary is unable to lift a heavy box with one hand, while she can lift it using both hands.

Let us consider the alphabet consisting of special fluent names t and another fluent $heavy$ and two unit actions $lift\_l$ and $lift\_r$.

To say that it is impossible to lift a heavy box with only the left hand we have the following e-propositions in our domain description:

$\{lift\_l\}$ **causes** $\neg t$ **if** $heavy$

$\{lift\_l\}$ **causes** t **if** $heavy$

Similarly, to say that it is impossible to lift a heavy box with the right hand we have the following e-propositions in our domain description.

$\{lift\_r\}$ **causes** $\neg t$ **if** $heavy$

$\{lift\_r\}$ **causes** t **if** $heavy$

In general we have the assumption that if an action is not possible then unless otherwise specified, a bigger action containing that action is also not possible. This is automatically captured by the fact that compound actions inherit effects from their subactions.

But to say that it is possible to lift a heavy box using both hands we need to have the following in our domain description:

$\{lift\_l,\ lift\_r\}$ **causes** t

It is easy to see that if $heavy \in \sigma$, then $\Phi(lift\_l, \sigma)$ and $\Phi(lift\_r, \sigma)$ are undefined while $\Phi(\{lift\_l, lift\_r\}, \sigma)$ is defined. □

A domain description is *consistent* if it has a model, and *complete* if it has exactly one model. For instance, domain descriptions D_1 and D_3 from Examples 1 and 4 are consistent, D_1 is complete, and a domain description containing the v–propositions **initially** f and **initially** $\neg f$ is inconsistent.

It is interesting to compare our new semantics with that defined in [GL92]. The comparison of course is only possible for the domain descriptions not containing names for compound actions. But, as demonstrated by the following example, even in this case the new semantics is somewhat more powerful than the old one.

Example 2.6. Consider a domain description D_4 containing an action name a, a fluent name f and two e–propositions

a **causes** f $\qquad$ a **causes** $\neg f$

According to the semantics from [GL92] D is inconsistent while it is easy to check that $M = (\emptyset, \Phi)$ where $\Phi(A, \sigma)$ is undefined for all actions A, is a model of D. $\Box$

The following proposition shows that for descriptions consistent in the sense of [GL92] both semantics coincide. Models of D in the sense of [GL92] will be called *s–models.*

Proposition 2.1. Let D be a domain description not containing compound actions and assume that D has an s–model. Let $M = (\sigma, \Phi)$ and $M^* = (\sigma, \Phi^*)$ where Φ^* is Φ restricted to unit actions.

Then
(i) M is a model of D iff M^* is an s–model of D and

(ii) for every s-model N of D there is a model M of D such that $N = M^*$. $\Box$

PROOF. Directly follows from the definitions of models and s-models. $\Box$

3. FROM $\mathcal{A}_C$ TO LOGIC PROGRAMS

3.1. 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 the negation as failure operator *not* [Cla78] of standard 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 [BG94a, Gel94].

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

$$l_1 \ or \ \dots \ or \ l_k \leftarrow l_{k+1}, \dots l_m, not \ \ l_{m+1}, \dots, not \ \ l_n \tag{1}$$

where $k \geq 0$, and each l_i is a literal, i.e., an atom possibly preceded by $\neg$, and *not* is the negation as failure operator. 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 (1) can be empty. Unless otherwise stated, we assume that rules with variables are used as shorthand for the set consisting of all their ground instantiations. Intuitively the rule can be read as: if $l_{k+1}, \dots, l_m$ are believed and it is not true that $l_{m+1}, \dots, l_n$ are believed then at least one of $\{l_1, \dots, l_k\}$ is believed. For a rule r of the form (1) the sets $\{l_1, \dots, l_k\}$, $\{l_{k+1}, \dots, l_m\}$ and $\{l_{m+1}, \dots, l_n\}$ 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 $Lit(\Pi)$. 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 literals $l, \bar{l}$ belongs to Y. The literals l and $\bar{l}$ are said to be complementary to each other and are referred to as complementary literals.

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

Definition 3.1. [GL91] Let Π be a disjunctive logic program. For any set S of ground 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. □

Definition 3.2. An *answer set* of a disjunctive logic program Π not containing *not* is a minimal (in a sense of set-theoretic inclusion) subset S of Lit such that

(i) for any rule l_1 *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 □

A program[2] 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 ground literal l is *entailed* by a DLP Π (denoted by $\Pi \models l$) 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 DLP's:

Proposition 3.1. [BG94a] For any answer set S of a disjunctive logic program Π:

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

(b) If S is a consistent answer set of Π and $l_i \in S$ for some $1 \leq i \leq k$ then there exists a ground instance of a rule of the type (1) from Π such that
$\{l_{k+1} \ldots l_m\} \subseteq S$, and
$\{l_{m+1} \ldots l_n\} \cap S = \emptyset$, and
$\{l_1 \ldots l_k\} \cap S = \{l_i\}$. □

We now review the definitions of "splitting" and "signing" which we use to analyze properties of the programs obtained by translating a domain description.

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

Definition 3.3. [Tur94] Let Π be a DLP such that no rule in it has an empty head. Let S be a set of literals in the language of Π such that no literals in $head(\Pi)$ appears complemented in $head(\Pi)$. Let $\overline{S}$ denote $Lit \setminus S$. S is said to be a *signing* for Π if each rule $r \in \Pi$ satisfies the following two conditions:

(i) $head(r) \cup pos(r) \subset S$ and $neg(r) \subset \overline{S}$, or
$head(r) \cup pos(r) \subset \overline{S}$ and $neg(r) \subset S$

(ii) If $head(r) \subset S$, then $head(r)$ is a singleton.

If a program has a signing, we say that it is signed. □

Definition 3.4. [Tur94] Let Π be a program. If S is a signing for Π, then

$h_S(\Pi) = \{r \in \Pi \ : \ head(r) \subset S\}$,

$h_{\overline{S}}(\Pi) = \{r \in \Pi \ : \ head(r) \subset \overline{S}\}$. □

Proposition 3.2. **Based on the restricted monotonicity theorem in [Tur94]** Let Π_1 and Π_2 be programs in the same language, both with signing S. If $h_{\overline{S}}(\Pi_1) \subseteq h_{\overline{S}}(\Pi_2)$ and $h_S(\Pi_2) \subseteq h_S(\Pi_1)$, then

if $\Pi_1 \models l$ and $l \in \overline{S}$ then $\Pi_2 \models l$. □

Definition 3.5. (Splitting set) [LT]
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 Π. 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 3.6. (Partial evaluation) [LT]
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 3.7. (Solution) [LT]
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.

□

Lemma 3.1. (Splitting Lemma) [LT]
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. □

3.2. The translation π and its properties

In this section we describe the translation π from domain descriptions to DLP's and discuss its properties.

The translated program πD of a domain description D, uses variables of three sorts: *situation* variables $S, S', \ldots$, *fluent* variables $F, F', \ldots$, and *action* variables $A, A', \ldots$[3]. We also need a sort for *fluent literals* whose terms are of the form f or $\overline{f}$ where f is a term of the type fluent and $\overline{f}$ denotes the fluent term g if $f = \neg g$ and denotes $\neg f$ otherwise. The language includes the situation constant s_0, and the fluent names and action names of D, that become object constants of the corresponding sorts. There are also some predicate and function symbols; their sorts will be clear from their use in the rules below. Of special importance is a function symbol $\{\}$ which will be used to form terms of the action type and a function *result* used to form the terms of the type situation.

The translated programs πD consists of the following translations of the individual propositions from D along with inertia axioms, inheritance axioms and axioms on the full awareness about the initial situation.[4]

1. **Inertia Axioms**:

$$\left.\begin{array}{l} (1a)\ holds(F, result(A,S)) \leftarrow holds(F,S), not\ \ may\_imm\_cause(A, \overline{F}, S), \\ \hfill atomic(A), not\ \ undefined(A,S) \\ (1b)\ \neg holds(F, result(A,S)) \leftarrow \neg holds(F,S), not\ \ may\_imm\_cause(A, F, S), \\ \hfill atomic(A), not\ \ undefined(A,S) \end{array}\right\} (1a){-}(1b)$$

These rules are motivated by the "common-sense law of inertia," [MH69] according to which truth values of fluents are normally not changed by actions. The rules (1a)-(1b) allow us to apply the law of inertia in reasoning "from the past to the future": The first–when a fluent is known to be true in the past, and the second–when it is known to be false. The auxiliary predicate *may_imm_cause* is essentially an "abnormality predicate" [McC86]. The axioms differ from those suggested in

[3]Using a sorted language implies that all atoms in the program are formed in accordance with the syntax of sorted predicate logic. Moreover, when we speak of an *instance* of a rule, we assume that the terms substituted for variables are of the appropriate sorts.

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

[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.

2. **Translating v-propositions**:

A v-proposition "f **after** $a_1, \ldots, a_m$" is translated as

$$(2) \qquad \leftarrow not \;\; h(f, [a_1, \ldots, a_m])$$

where $[a_1, \ldots, a_m]$ stands for the ground term

$result(a_m, result(a_{m-1}, \ldots, result(a_1, s_0) \ldots)),$

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

3. **Translating e–propositions**:

The translation of an e–proposition "a **causes** f if $p_1, \ldots, p_n$" consists of

$$\left.\begin{array}{l} (3a)\ may\_imm\_cause(a, f, S) \leftarrow not\ \ \overline{h(p_1, S)}, \ldots, not\ \ \overline{h(p_n, S)} \\ (3b)\ cause(a, f, S) \leftarrow h(p_1, S), \ldots, h(p_n, S) \end{array}\right\} (3a) - (3b)$$

where, $\overline{holds(p_i, s)}$ is a literal complementary to $holds(p_i, s)$.

Intuitively, $may\_imm\_cause(a, f, s)$ means that if action a is executed in situation s then f may become true as a direct effect of the action a. It is used in disabling the inertia rules (1) in the cases when f can be affected by a. It is also used in defining $undefined$ in rule (3d).

Intuitively, $cause(a, f, s)$ means that if action a is executed in situation s then f will be true as a direct effect of the action a. It is used in the rule (3c) below.

The *Effect Axiom* (3c) and axioms (3d) and (3e) are defined as follows:

$$\left.\begin{array}{l} (3c)\ h(F, result(A, S)) \leftarrow cause(A, F, S), not\ \ undefined(A, S) \\ (3d)\ undefined(A, S) \leftarrow may\_imm\_cause(A, F, S), may\_imm\_cause(A, \overline{F}, S) \\ (3e)\ undefined(A, res(B, S)) \leftarrow undefined(B, S) \end{array}\right\} (3c){-}(3e)$$

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

Axiom (3c) 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.

4. **Inheritance axioms**:

$$\left.\begin{array}{l}
(4a)\ holds(F, result(A,S)) \leftarrow\ subset(B,A), holds(F, Result(B,S)),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad not\ \ noninh(F,A,S), not\ \ undefined(A,S)\\
(4b)\ \neg holds(F, result(A,S)) \leftarrow subset(B,A), \neg holds(F, result(B,S)),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad not\ \ noninh(\overline{F},A,S), not\ \ undefined(A,S)\\
(4c)\ cancels(X,Y,F,S) \leftarrow subset(X,Z), subseteq(Z,Y), cause(Z,\overline{F},S)\\
(4d)\ noninh(F,A,S) \leftarrow subseteq(U,A), may_imm_cause(U,\overline{F},S),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad not\ \ cancels(U,A,\overline{F},S)\\
(4e)\ undefined(A,S) \leftarrow noninh(F,A,S), noninh(\overline{F},A,S)
\end{array}\right\}(4a){-}(4e)$$

Intuitively, $cancels(x,y,f,s)$ 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 in (4) 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.

5. **Full awareness about the initial situation rule**

$$(5)\ holds(F,s_0)\ \ or\ \ \neg holds(F,s_0) \leftarrow$$

6. **Defining subseteq and atomic**:

For any two actions a and b we add $subseteq(a,b)$ if $a \subseteq b$, $subset(a,b)$ if $a \subset b$, $\neg subseteq(a,b)$ if $a \not\subseteq b$, $\neg subset(a,b)$ if $a \not\subset b$, $atomic(a)$ if a is a singleton, and $\neg atomic(a)$ otherwise. Of course this would lead to a very large program.

To make the program more practical we can represent a set of atomic actions as a list (other representations are also possible). In that case *subset* and *atomic* can be defined using the following rules:

$$\left.\begin{array}{l}
\neg subseteq(A,B) \leftarrow member(X,A), not\ \ \ member(X,B)\\
subseteq(A,B) \leftarrow not\ \ \neg subseteq(A,B)\\
eq(X,X)\\
\neg atomic(X) \leftarrow member(Y,X), member(Z,X), not\ \ \ eq(Y,Z)\\
atomic(X) \leftarrow not\ \neg atomic(X)\\
subset(X,Y) \leftarrow subseteq(X,Y), not\ \ subseteq(Y,X)
\end{array}\right\}$$

For a domain description D, the program πD consists of the rules (1a), (1b), (2), (3a) - (3e), (4a) - (4e), (5) and (6).

Observation 3.1. Since in πD we assume full awareness about the initial state (5), *may_imm_cause* and *causes* are equivalent and one can be replaced by the other in the bodies of rules in πD without affecting its answer sets. □

Observation 3.2. The program πD can be split (see Definition 5) using the splitting set consisting of $Lit(\{atomic, subset, subseteq, eq\})$. It is easy to see that the bottom part consists of the rules (6) 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 Theorem 3.1 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. □

Theorem 3.1. **Soundness and completeness of π**
Let D be an arbitrary domain description and $P = f$ **after** $a_1, \ldots, a_n$ be any v-proposition in the language of D such that $a_1, \ldots, a_n$ is executable in any model of D. Then $\pi D \models h(f, [a_1, \ldots, a_n])$ iff $D \models P$.[5] □

The result shows that general-purpose nonmonotonic system of disjunctive logic programs has sufficient power for modeling reasoning about domain descriptions of $\mathcal{A}_C$. Unfortunately, at the moment there is no well-understood query answering mechanism that can answer queries for arbitrary disjunctive databases. For some results in this direction see for instance [Wat94, BEP94].

In the next subsection we consider two other translations of a domain description into subclasses of logic programs for which inference mechanisms are much better understood. These translations are however weaker than π and therefore are incomplete in general.

3.3. The translation π' and its properties

Consider the program $\pi' D$ which differs from πD by having (2')

$$(2') \qquad h(f, [a_1, \ldots, a_m]) \leftarrow$$

instead of (2). Rules of the form 2 are referred to as *constraints*.

Observation 3.3. Since in $\pi' D$ we assume full awareness about the initial state (5) we can replace *may_imm_cause* by *cause* in the body of the rules (1a), (1b), (3d) and (4d) and replace *cause* by *may_imm_cause* in the rule (4c) without affecting its answer sets. □

Perhaps surprisingly, this translation is weaker than π.

Example 3.1. Consider the following domain description D:

initially $\neg f$
a **causes** f **if** p
b **causes** $\neg f$ **if** p
$\{a, b\}$ **causes** f **if** $\neg p$
f **after** $\{a, b\}$

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

[5] Notice that the first occurrence of $\models$ denotes entailment w.r.t. logic programs while the second denotes entailment w.r.t. domain descriptions in $\mathcal{A}_c$.

$\sigma_0 = \{\}$ and $\sigma'_0 = \{p\}$. Let Φ be the transition function in the models of D. It is easy to see that $\Phi(\{a,b\},\emptyset) = \{f\}$, while $\Phi(\{a,b\},\{p\})$ is undefined. It is easy to show that $M' = (\sigma'_0, \Phi)$ is not a model of D and hence, $M = (\sigma_0, \Phi)$ is the only model (by Observation 2.1) of D. Hence, $D \models$ **initially** $\neg p$.

Now let us consider the programs πD and $\pi' D$.

It is easy to show that $\pi' D$ has two answer sets A and A', where
$\{\neg holds(p, s_0), holds(f, [\{a,b\}])\} \subset A$ and
$\{holds(p, s_0), holds(f, [\{a,b\}])\} \subset A'$.

Consider πD. It is easy to show that it has an answer set that contains $\{\neg holds(p, s_0), holds(f, [\{a,b\}])\}$, but it does not have any answer set that contains $holds(p, s_0)$. This is because in the second case by having $holds(p, s_0)$, the answer set is forced to have $noninh(f, \{a,b\}, s_0)$ and $noninh(\neg f, \{a,b\}, s_0)$. Thus there is no way to have $holds(f, [\{a,b\}])$ in that answer set. But then it violates the constraint obtained in step (2) and hence it will not be an answer set.

Hence, in this example the unique answer set of πD correspond to the unique model of D, while $\pi' D$ has an extra answer set. As a result, $\pi D \models \neg holds(p, s_0)$ while $\pi' D \not\models \neg holds(p, s_0)$. □

Although as shown in the above example π' may not lead to a complete translation, we show below that it is *sound*.

Theorem 3.2. **Soundness of** π'
Let D be an arbitrary domain description and $P = f$ **after** $a_1, \ldots, a_n$ be any v-proposition in the language of D such that $a_1, \ldots, a_n$ is executable in any model of D. If $\pi' D \models \pi' P$ then $D \models P$. □

Now we describe a large class of domain descriptions for which translations π and π' are equivalent. We need the following

Definition 3.8. We say that an action a is associated with a domain description D if a is a subset (not necessarily proper) of some action which occurs in some proposition of D. A domain description D will be called *well-founded* (w.r.t. actions) if for any model $M = (\sigma_0, \Phi)$ of D, any action a, associated with D, and any state σ of M, $\Phi(a, \sigma)$ is defined. □

Notice that the domain description D from Example 1 is not well-founded since the action $\{a, b\}$ is associated with D but $\Phi(\{a,b\},\{p\})$, where Φ is the transition function from its model, is undefined.

Theorem 3.3. For any consistent well-founded domain description D the programs πD and $\pi' D$ have the same answer sets. □

Corollary 3.1. For any consistent well-founded domain description D the program $\pi' D$ is complete w.r.t. D, i.e., for any v-proposition p of the form f **after** $a_1, \ldots, a_n$ in the language of D if $a_1, \ldots, a_n$ is executable in D then $\pi' D \models \pi' p$ iff $D \models p$.
□

Computationally, π' is slightly easier to deal with than π. This is due to the fact that the program $\pi' D$ does not contain rules with empty heads which eliminates one computational difficulty.

Answers to queries to the program $\pi' D$ can be computed using the program $ELMO$ developed by Watson [Wat94]. $ELMO$ is built on top of the SLG program by Chen and Warren [WC93] which answers queries for "classical" logic programs without $\neg$ and *or* under the stable model semantics [GL88]. $ELMO$ works correctly for a large class of disjunctive programs with no empty heads. It is easy to show that for any D, $\pi' D$ belongs to this class. As was shown by Inoue [Ino91] (see also recent paper [LT94]) these programs are equivalent to so called abductive logic programs which opens the possibility of answering queries in $\pi' D$ based on abductive reasoning.

There are two substantial sources of inefficiency in $ELMO$. The first one is the presence of disjunctions. The second is the fact that in general, even without disjunction, the query answering methods for programs with negation as failure under stable model semantics are not efficient. This observation leads to the next translation π_f which gives us programs with properties that make them efficiently computable, but much less complete.

3.4. The translation π_f and its properties

The program $\pi_f D$ is obtained from $\pi' D$ by removing all the disjunctions from $\pi' D$. The program loses awareness of some of the fluents of D and concentrates instead only on those whose truth values in the initial situation are known. If the values of many fluents in the initial situation are unknown, this, of course, substantially weakens the program. But at the same time it makes it much more efficient. This is due to the fact that $\pi_f D$ has neither of the computational difficulties present in $\pi' D$. Disjunctions are obviously eliminated, but moreover, the resulting program is easily reducible to a so called *acyclic* program [6] for which the standard computational mechanism of "classical" logic programming, called $SLDNF$ resolution is sound and complete [AB91]. Also, most of the proposed semantics for negation as failure coincide for acyclic programs, and hence no special difficulties related to stable model semantics are present. The resulting program was run on a simple extension of Prolog which allows for treatment of $\neg$. It was also used for constructing and proving the correctness of simple planners capable of producing plans with concurrent actions.

In Observation 3.1 and 3.3 we observed that in πD and $\pi' D$ we can replace may_imm_cause by $cause$ in the body of the rules (1a), (1b), (3d) and (4d) without affecting its answer sets. Such is not the case for $\pi_f D$.

The following example explains the need of having $causes$ instead of may_imm_cause in the body of the rule (4c) in the program $\pi_f D$.

Example 3.2. Consider the following domain description D:

[6] It was originally called "ω locally hierarchical programs" by Cavedon [Cav89] and renamed "acyclic programs" by Apt and Bezem in [AB90] where the authors prove important properties of such programs.

initially $\neg f$
$\{a, c\}$ **causes** f
$\{a, b, c\}$ **causes** $\neg f$ **if** p

Consider $\pi_f D$. Suppose we have *may_imm_cause* in the body of the rule defining *cancels*. Then we would conclude $\neg holds(f, [c])$ using inertia (w.r.t c). Then using inheritance we would conclude that $\neg holds(f, [\{a, b, c\}])$. Since it is possible that p is false in the initial state our conclusion of $\neg holds(f, [\{a, b, c\}])$ will be un-intuitive. This is avoided by having *causes* instead of *may_imm_cause*, in the body of the rule defining *cancels*. □

Theorem 3.4. **Soundness of π_f w.r.t π'**
Let D be an arbitrary domain description and $P = f$ **after** $a_1, \ldots, a_n$ be any v-proposition in the language of D such that $a_1, \ldots, a_n$ is executable in any model of D. $\pi' D \models \pi' P$ if $\pi_f D \models \pi_f P$. □

Proof. (i) It is easy to see that the set $S = Lit(\{may\_imm\_cause, undefined, noninh\})$ is a signing of the programs $\pi' D$ and $\pi_f D$.
(ii) Hence using the restricted monotonicity theorem (Proposition 3.2) we have that for literals with the predicates *holds* and $\neg holds$, the program $\pi_f D$ is sound with respect to $\pi' D$, and hence w.r.t. D. □

To describe the class of domain descriptions for which π_f is complete we introduce a notion of strongly complete domain description.

Definition 3.9. A domain description D is called *strongly complete* if for any fluent in the language of D, **initially** f or **initially** $\neg f$ belongs to D. □

Theorem 3.5. For any consistent and strongly complete domain description D, the programs $\pi_f D$ and $\pi' D$ have the same answer sets. □

Proof. Directly using Lemma 5.1 in the appendix. □

Corollary 3.2. For any consistent, strongly complete and well-founded domain description D, π_f is complete w.r.t. D. □

In the following example we show that π_f is weaker than π'.

Example 3.3. Consider the following domain description D:

initially $\neg f$
a **causes** f **if** p
b **causes** f **if** $\neg p$

It is easy to see that $\pi' D$ will have two answer sets, one containing $\{holds(p, s_0), holds(f, [a]), holds(f, [\{a, b\}])\}$ and another containing $\{\neg holds(p, s_0), holds(f, [b]), holds(f, [\{a, b\}])\}$. Hence, $\pi' D \models holds(f, [\{a, b\}])$. It is easy to see that $\pi_f D \not\models holds(f, [\{a, b\}])$. □

3.5. Relating π, π' and π_f to earlier translations of $\mathcal{A}$

Since $\mathcal{A}$ was introduced in [GL92] several translations of it to logic programming based formalisms have been suggested in the literature. In particular, Gelfond and Lifschitz [GL92] give a sound (but not complete) translation of $\mathcal{A}$ to extended logic programs; Turner [Tur94] gives a sound and complete translation to disjunctive logic programs; Denecker and De Schreye [DDS93] give a sound and complete translation to abductive logic programs (see also Dung [Dun93]); and Thielscher and Holldobler [HT93] give a sound and complete translation to equational logic programs.

Proposition 2.1, together with Theorems 3.3 from the previous section allow us to rigorously compare our new formalizations with the ones mentioned above. Obviously for consistent domain descriptions of $\mathcal{A}$ the corresponding domain description in $\mathcal{A}_c$ has an s-model and by Proposition 2.1 it is well-founded and therefore both, π and π' are conservative extensions of the programs from [Tur94], [DDS93], [HT93], [Dun93]. Since these formalizations use different languages with different semantics, establishing this fact directly, without the use of $\mathcal{A}$ and $\mathcal{A}_C$, will be probably more difficult.

Before $\mathcal{A}$ was introduced there were several works (which includes [AB90, EK89, Eva89]) that "reasoned about actions" using logic programs. In [AB90], Apt and Bezem consider the case when there is complete information about the initial situation. If we consider only such domain descriptions then our π_f is an extension of Apt and Bezem's formalization. This can be easily shown using Theorem 3.5. and Proposition 2.1.

4. SOME DOMAIN DESCRIPTIONS AND THEIR TRANSLATIONS

Example 4.1. Independent Actions

Consider the domain description D_1 from Example 1.

$$\left.\begin{array}{l}\textbf{initially}\ \neg\, lifted\\ \textbf{initially}\ \neg\, opened\\ \{lift\}\ \textbf{causes}\ lifted\\ \{open\}\ \textbf{causes}\ opened\end{array}\right\}\ D_1$$

In Example 3 we have shown that the domain description D_1 entails the v-propositions "lifted **after** {lift, open}" and "opened **after** {lift, open}". Let us demonstrate that the translation of these propositions is entailed by $\pi_f D_1$. For that purpose let us notice that the following rules belong to $\pi_f D_1$.

$$\left.\begin{array}{l}
X1\ \neg holds(lifted, s_0) \leftarrow \\
X2\ \neg holds(opened, s_0) \leftarrow \\
X3\ holds(lifted, [\{lift\}]) \leftarrow \\
X4\ holds(opened, [\{open\}]) \leftarrow \\
X5\ may_imm_cause(\{open\}, opened, S) \leftarrow \\
X6\ may_imm_cause(\{lift\}, lifted, S) \leftarrow \\
X7\ noninh(\neg opened, X, S) \leftarrow subseteq(\{open\}, X), \\
\qquad\qquad\qquad\qquad\qquad\quad may_imm_cause(\{open\}, opened, S), \\
\qquad\qquad\qquad\qquad\qquad\quad not\ \ cancels(\{open\}, X, opened, S) \\
X8\ noninh(\neg lifted, X, S) \leftarrow subseteq(\{lift\}, X), \\
\qquad\qquad\qquad\qquad\qquad\quad may_imm_cause(\{lift\}, lifted, S), \\
\qquad\qquad\qquad\qquad\qquad\quad not\ \ cancels(\{lift\}, X, lifted, S)
\end{array}\right\} \subset \pi_f D_1$$

The rules X1 and X2 are obtained in Step (2'). The rules X3 and X4 are obtained by partial evaluation of rules obtained in Step (3c). The rules X5 and X6 are obtained in Step (3a). The rules X7 and X8 are obtained by considering the rules obtained in Step (4d) and selecting those instances whose body contains a *may_imm_cause* atom that unifies with either X5 or X6.

Now, notice that D_1 is consistent (see Example 3) and therefore, by the Soundness Theorem, $\pi_f D_1$ has a consistent answer set. Let A be an arbitrary answer set of $\pi_f D_1$. According to the inheritance axiom 4.a and Proposition 3.1, to show that $holds(lifted, [\{lift, open\}]) \in A$ it suffices to show that

(a) $holds(lifted, [lift]) \in A$ while

(b) $noninh(lifted, \{lift, open\}, s_0) \notin A$.

(a) follows immediately from X3. Now (b) is the immediate consequence of Proposition 3.1, and the fact that $noninh(lifted, \{lift, open\}, s_0)$ does not occur in the head of any ground instance of a rule from $\pi_f D_1$ whose body has a *may_imm_cause* atom that unifies with either X5 or X6. Similar argument can be used to show that $\pi_f D_1$ entails $holds(opened, [\{lift, open\}])$. □

In the following example we show how our formalism handles the case when the effect of a compound action cancels the effect of the atomic actions.

Example 4.2. Dependent Actions: Cancelation
Consider a slight modification of the domain description D_2 of Example 2.

$$\left.\begin{array}{l}
\textbf{initially}\ has_water \\
\textbf{initially}\ \neg\ spilled \\
\{lift_l\}\ \textbf{causes}\ spilled\ \textbf{if}\ has_water \\
\{lift_r\}\ \textbf{causes}\ spilled\ \textbf{if}\ has_water \\
\{lift_l,\ lift_r\}\ \textbf{causes}\ \neg\ spilled
\end{array}\right\} D_2$$

It is easy to see that D_2 entails

(a) $\neg\ spilled$ **after** $\{lift\_r, lift\_l\}$.

$\pi_f D_2$ entails the translation of (a) since it contains the rule

$holds(\neg\ spilled, [\{lift\_r, lift\_l\}]) \leftarrow$

obtained from e–proposition

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

from D_2.

To see why the Inheritance axiom does not cause inconsistency by inheriting that $holds(spilled, [\{lift\_l, lift\_r\}])$ is true, it suffices to notice that

$noninh(spilled, \{lift\_l, lift\_r\}, s_0)$ is entailed by πD_2.

The effects of compound actions are cancelled in essentially the same way.

Consider a domain description $D_{2.1}$ obtained from D_2 by adding an e–proposition

$\{flip, lift\_r, lift\_l\}$ **causes** $spilled$ **if** has_water.

$\pi_f D_{2.1}$ will contain the rules:

$$
\begin{array}{l}
holds(spilled, [\{flip, lift\_r, lift\_l\}]) \leftarrow cause(\{flip, lift\_r, lift\_l\}, spilled, s_0) \\
cause(\{flip, lift\_r, lift\_l\}, spilled, S) \leftarrow holds(has\_water, S) \\
noninh(\neg spilled, X, S) \leftarrow subseteq(\{flip, lift\_r, lift\_l\}, X), \\
\qquad\qquad may\_imm\_cause(\{flip, lift\_r, lift\_l\}, spilled, S), \\
\qquad\qquad not\ \ cancels(\{flip, lift\_r, lift\_l\}, X, spilled, S)
\end{array}
$$

These rule will make the program entail

$noninh(\neg spilled, \{flip, lift\_r, lift\_l\}, S)$

which blocks the inheritance axioms (4.b) and hence $\{flip, lift\_r, lift\_l\}$ does not inherit "$\neg spilled$" from $\{lift\_r, lift\_l\}$. □

In the next example we show how to represent a compound action whose sub-actions have conflicting effects.

Example 4.3. Conflicting Sub-actions
Consider the domain description D_3 of Example 4.

$$
\left.
\begin{array}{l}
close\ \textbf{causes}\ \neg\ opened \\
open\ \textbf{causes}\ opened \\
paint\ \textbf{causes}\ painted
\end{array}
\right\}\ D_3
$$

Since for any state σ the transition function Φ of D_3 is undefined on $(\{open, close\}, \sigma)$ the effect of performing "$close$" and "$open$" concurrently is unknown. Accordingly, no information about the situation $[\{open, close\}]$ is entailed by $\pi_f D_3$. To show that this is indeed the case let us notice that the following rules belong to $\pi_f D_3$:

$$\left.\begin{array}{l}
Z1\ \neg holds(opened, [\{close\}]) \leftarrow \\
Z2\ holds(opened, [\{open\}]) \leftarrow \\
Z3\ noninh(\neg opened, X, S) \leftarrow subseteq(\{open\}, X), \\
\qquad\qquad\qquad\qquad\qquad may_imm_cause(\{open\}, opened, S), \\
\qquad\qquad\qquad\qquad\qquad not\ \ cancels(\{open\}, X, opened, S) \\
Z4\ noninh(opened, X, S) \leftarrow subset(\{close\}, X), \\
\qquad\qquad\qquad\qquad\qquad may_imm_cause(\{close\}, \neg opened, S), \\
\qquad\qquad\qquad\qquad\qquad not\ \ cancels(\{close\}, X, \neg opened, S)
\end{array}\right\} \subset \pi_f D_3$$

By instantiating Z4 with $X = \{open, close\}$ we obtain the clause:

$noninh(opened, \{open, close\}, S)$

Similarly, from Z3 we obtain the clause

$noninh(\neg opened, \{open, close\}, S)$

Hence, neither $holds(opened, [\{open, close\}])$ nor $\neg holds(opened, [\{open, close\}])$ can be derived from $\pi_f D_3$ using (4b). From Proposition 3.1 and consistency of $\pi_f D_3$ we can conclude that it is unknown if "opened" holds or does not hold in $[\{open, close\}]$.

Notice also that our program neither entails $holds(painted, [\{open, close, paint\}])$ (a translation of the v–proposition $painted$ **after** $\{open, close, paint\}$) nor entails $\neg holds(painted, [\{open, close, paint\}])$. Notice that $\{open, close, paint\}$ is not executable in models of D. This explains the executability condition in the Soundness Theorem. □

5. CONCLUSION

This paper merges ideas from several areas of research and depends substantially on many results in all of these areas. This makes a complete description of related work a difficult if not an impossible task. We would not even attempt to accomplish this task here. Instead, we mention only the work which had direct and immediate influence on us. We hope it will help the reader to better understand the proper place this paper occupies in a large puzzle we, together with many other people, are trying to solve.

Introduction of the language $\mathcal{A}_C$ and its semantics is part of an attempt to develop a framework for systematic development of theories of actions and their effects. There is a large body of work on formalization of reasoning about actions which differ substantially in ontology of actions, in logics (monotonic and nonmonotonic) used for the formalizations, and in the degree of precision and generality attempted by the authors. A typical paper on the subject described a modification of the old (or, possibly an entirely new) approach and illustrates its utility by representing several "canonical" examples, such as the blocks world or the "Yale Shooting" story and its enhancements. Competing approaches were evaluated and compared mostly by their ability to represent knowledge from these examples and by the elegance of the corresponding representations. This methodology proved to be very fruitful. It sharpened our understanding of common-sense reasoning about actions and led to substantial advances in theories of nonmonotonic logics and logic programming. At the same time it left researchers with a great diversity of methods of representation without providing them with a framework for outlining ranges of applicability of these methods and their advantages and disadvantages. The diversity reflects the difficulty and richness of the problem and is probably unavoidable, but we hope that the development of such a unifying framework will help to structure the multitude of different formalisms and to make this diversity manageable. This of course could only happen if such a framework is based on clear mathematical grounds and helps to facilitate mathematical analysis and comparison of different methods of representation.

There are several recent publications, including [Rei91, Rei92, San94, GL92] which attempt the development of such a framework. The attempts differ in scope and methods but share a large number of basic assumptions.

Our work is based on the approach originated in [GL92]. That paper introduces a high-level action description language $\mathcal{A}$, gives its semantics based on the notion of an automata, and outlines the approach of using entailment relations in action description languages for proving properties of various formalisms for modeling reasoning about actions. We hope that by now the connections between [GL92] and the present paper are obvious.

The Toronto group[7] and some others [Elk92] base their unifying framework on the language of situation calculus [McC59] and its extensions (see for instance [GLR91], [PR93] and [Elk92]) as the means of expressing knowledge about actions and their effects, and classical logic as the means of formalization of reasoning.

In further work [Rei93, LLL, Pin94] the simple ontology of situation calculus is enriched by introducing complex actions, actual time line, non-deterministic actions, etc. Effects of actions are described by a collection of axioms written in the syntax of first-order logic. The inertial property of the corresponding dynamical system is described by the so called successor state axioms which determine necessary and sufficient conditions for a property to hold after the execution of an action. This approach provides a solution to the Frame Problem which, though limited in scope, is elegant and applicable to a large class of dynamical systems.

Sandewall [San92, San94] concentrates on the development of a model of reality as a dynamical system viewed as a game between an ego and the world. The rules of the game are determined by the Ego-World Semantics. The semantics is used to describe a taxonomy of dynamical systems which is applied to the analysis of several nonmonotonic (mainly preferred models based) formalisms for representing actions. Unfortunately, the detailed description of the approach (soon to appear in the book) is not yet available to the authors which precludes serious comments on its strengths and weaknesses.[8]

[7] The group at the University of Toronto consists of Reiter, Pinto, Lin, Levesque, Lesperance, Marcu and Scherl and has many recent works on this topic [PR93, Pin94, LR94, LR95, LLL, SL93].

[8] It seems however that the Sandewall's approach shares many underlying assumptions with that from [GL92] and Reiter's [Rei91]. In particular, the dynamical systems under considerations are inertial. On the other hand, the basic ontology of [San92] seems to be built on the notion of

All three approaches enjoyed considerable attention in the last few years. Initial frameworks were extended to include richer ontologies and several logical formalizations were evaluated and compared w.r.t. them. The relation between the frameworks is not yet well understood (see however Kartha [Kar93] and Theischler [Thi94]). The Toronto group's approach and the approach based on $\mathcal{A}$ share the same view of a dynamic world based on the situation calculus model. The difference is in the type of language used to describe actions and in the type of logic associated with this language. The former uses general purpose classical logic while the latter prefers a special purpose, high level language with nonmonotonic, specialized semantics. The main challenge of the Toronto group's approach then is to find proper collections of axioms describing the corresponding ontology, while for us the problem is rather to describe a class of models defining the proper entailment relation. We pay for the simplicity of the language by the necessity to develop this new semantics. We believe that the two approaches shall not be viewed as rivals but rather as mutually complimentary ways to study the same entailment relations. Future research will show if this is a right assessment.

The second theme of the paper is centered around the question of applicability of logic programming languages to representing knowledge about actions. The early attempts on such representation include Eshghi and Kowalski [EK89], Evans [Eva89], Apt and Bezem [AB90], among others. These formalizations used the language of general logic programs and therefore assumed the closed world assumption about all predicates. Formalization in the language of extended logic programs which has no such limitation was suggested in [GL92] and also in [Mil95]. The formalization in [GL92] extends the previous suggestion by Apt and Bezem [AB90] and is shown to be sound w.r.t. semantics of $\mathcal{A}$. This work was generalized to achieve complete formalization of domain descriptions in $\mathcal{A}$ in the languages of extended logic programs with disjunctions [Tur94], abductive [DDS93, Dun93] and equational logic programming [HT93] respectively. In [BG93] we describe a first attempt at generalizing these programs to the dynamic systems with compound actions. In this paper we extend our approach in [BG93]. We hope our work illustrates how the use of action description languages and their semantics facilitates rigorous reasoning about various formalizations, allows to gradually strengthen them, and to investigate the relations between different approaches.

The last theme, related to the treatment of concurrency in the language of situation calculus follows the lines suggested in [GLR91]. A paper addressing the possibility of expressing the results of concurrent actions in situation calculus is [LS92]. The important difference is in the choice of the formalisms – the nonmonotonic approach of [LS92] uses two different non-monotonic logics – circumscription and default logic. In contrast our approaches use single formalisms of domain descriptions or that of logic programs. There are some other differences: for instance, in Example 3 expanded by a v–proposition " **initially** $open$", the formalism of Lin and Shoham uses inertia to entail $holds(open, [\{open, close\}])$ while we believe that "unknown" (produced by our systems) is the more intuitive answer.

Nevertheless, to capture Lin and Shoham's intuition we need to redefine the definition of models by requiring that

actual time line and hence does not allow hypothetical reasoning easily expressible in two other frameworks.

$\Phi(\sigma, [a_1, \ldots, a_n]) = \sigma$ when $E^+ \cap E^- \neq \emptyset$.

In the translation we need to add the following rules.
$holds(F, res(A, S)) \leftarrow holds(F, S), noninh(A, F, S), undefined(A, S)$
$\neg holds(F, res(A, S)) \leftarrow \neg holds(F, S), noninh(A, F, S), undefined(A, S)$

Another approach would be to randomly choose from the intersection of E^+ and E^-, and have the translation such that each answer set corresponds to a particular choice.

The nice feature of Lin and Shoham's formalization is the so called epistemological completeness of their system [LS91] for deterministic actions. Intuitively, a theory of a (deterministic) action is epistemologically complete if, given a complete description of the initial situation, the theory enables us to predict a complete description of the resulting situation when the action is performed. Since some of our actions are not executable we can not expect to have precisely this property but with respect to the naturally modified notion our formalisms are epistemologically complete. In other words, it can be easily shown that given a complete description of the initial situation, our theory enables us to predict a complete description of the resulting situation when a sequence of executable actions is performed.

Another recent paper addressing the possibility of expressing the results of concurrent actions in situation calculus is [ALP94]. The approach in [ALP94] allows concurrent executions of different instances of the same action and treats 'concurrency' from the program execution point of view. It formalizes the notion that if f is true after executing action a followed by action b and also true after executing action b followed by action a then f can be assume to be true after concurrently executing a and b. It seems that such a formalization will not be able to express the effect of executing $lift\_l$ and $lift\_r$ concurrently as described in Example 2. But more study is necessary to further compare the approach in [ALP94] with our approach.

There are of course many open questions left in all of the three directions of research outlined above. In the nearest future we plan to concentrate on two of them: direct extension of the previous work by developing extensions of $\mathcal{A}_C$ to allow representation of knowledge about dynamical systems with richer ontologies and building provably correct logic programs describing the corresponding entailment relations; discovering methods to use these declarative programs to automatize various forms of reasoning utilizing this knowledge.

Acknowledgment

We would like to acknowledge the grants NSF-IRI-92-11-662, NSF-IRI 95-01-577 and NSF-IRI 91-03-112. We also thank V. Lifschitz, G. Kartha, B. Fronhofer, and the anonymous referees for their valuable comments.

REFERENCES

[ALP94.] J. Alferes, R. Li and L. Pereira. Concurrent actions and changes in the situation calculus. In H. Geffner, editors, *Proc. of IBERAMIA 94*, pages 93-104, 1994.

[AB90.] K. Apt and M. Bezem. Acyclic programs. In D. Warren and Peter Szeredi, editors, *Logic Programming: Proc. of the Seventh Int'l Conf.*, pages 617–633, 1990.

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

[Bak91.] A. Baker. Nonmonotonic reasoning in the framework of situation calculus. *Artificial Intelligence*, 49:5–23, 1991.

[BEP94.] R. Ben-Eliyahu and L. Palopoli. Reasoning with minimal models: Efficient algorithms and applications. In *Proceedings of KR 94*, pages 39–50, 1994.

[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.

[BG94a.] C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19,20:73–148, 1994.

[BG94b.] C. Baral and M. Gelfond. Representing actions: Laws, observations and hypothesis. *Journal of Logic Programming*, This volume.

[Cav89.] L. Cavedon. Continuity, consistency, and completeness properties for logic programs. In Giorgio Levi and Maurizio Martelli, editors, *Logic Programming: Proc. of the Sixth Int'l Conf.*, pages 571–584, 1989.

[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 *Proceedings of International Logic Programming Symposium 93, Vancouver*, pages 147–164, 1993.

[Dix91.] J. Dix. Classifying semantics of logic programs. In *Proceedings of International Workshop in Logic Programming and Non-monotonic 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 International Conference on Logic Programming*, pages 222–238, 1993.

[EK89.] K. Eshghi and R. Kowalski. Abduction compared with negation as failure. In Giorgio Levi and Maurizio Martelli, editors, *Logic Programming: Proc. of the Sixth Int'l Conf.*, pages 234–255, 1989.

[Elk92.] C. Elkan. Reasoning about action in first-order logic maintenance systems. In *Proceedings of the Ninth Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, 1992.

[Eva89.] C. Evans. Negation-as-failure as an approach to the Hanks and McDermott problem. In *Proc. of the Second Int'l Symp. on Artificial Intelligence*, 1989.

[Gel94.] M. Gelfond. Logic programming and reasoning with incomplete information. *Annals of Mathematics and Artificial Intelligence*, 12, pages 19-46, 1994.

[GL88.] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, pages 1070–1080, 1988.

[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.

[GLR91.] M. Gelfond, V. Lifschitz, and A. Rabinov. What are the limitations of the situation calculus? In R. Boyer, editor, *Automated Reasoning: Essays in*

Honor of Woody Bledsoe. Kluwer Academic, Dordrecht, 1991.

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

[Ino91.] K. Inoue. Extended logic programs with default assumptions. In *Proc. of International Conference on Logic Programming*, 1991.

[Kar93.] G. Kartha. Soundness and completeness theorems for three formalizations of action. In *International Joint Conference in Artificial Intelligence 93*, pages 724–729, 1993.

[KL94.] G. Kartha and V. Lifschitz. Actions with indirect effects (preliminary report). In *Proceedings of KR*, pages 341–350, 1994.

[LLL.] Y. Lesperance, H. Levesque, and F. Lin. A formalization of ability and knowing how that avoids the frame problem. Manuscript.

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

[LR95.] F. Lin and R. Reiter. How to progress a database II: The STRIPS connection. In *Proceedings of IJCAI*, 2001-2007, 1995.

[LR94.] F. Lin and R. Reiter. How to progress a database (and Why) I: Logical foundations. In *Proc. of KR94*, pages 425–436, 1994.

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

[LS92.] F. Lin and Y. Shoham. Concurrent actions in the situation calculus. In *Proc. of AAAI 92*, pages 590–595, 1992.

[LT.] V. Lifschitz and H. Turner. Splitting a logic program. In *Proc. of ICLP 94*, pages 23-38, 1994.

[LT94.] V. Lifschitz and H. Turner. From disjunctive programs to abduction. Manuscript.

[McC59.] J. McCarthy. Programs with common sense. In *Proc. of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, London, 1959. Her Majesty's Stationery Office.

[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.

[Mil95.] R. Miller. Situation Calculus specifications for event calculus logic programs. In W. Marek, A. Nerode and M. Truszynski, editors, *Proceedings of the Third International Conference on Logic Programming and Non-monotonic Reasoning*, pages 217-230, 1995.

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

[Ped89.] E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In R. Brachman, H. Levesque, and R. Reiter, editors, *Proc. of the First Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 324–332, 1989.

[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.

[Prz88.] T. Przymusinski. Perfect model semantics. In R. Kowalski and K. Bowen, editors, *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, pages

1081–1096, 1988.

[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.

[Rei92.] 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.

[Rei93.] R. Reiter. Proving properties of states in the situation calculus. *Artificial Intelligence*, 64:337-351, 1993.

[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), pages 581-616, 1994.

[SL93.] R. Scherl and H. Levesque. The frame problem and knowledge producing actions. In *AAAI 93*, pages 689–695, 1993.

[Thi94.] M. Thielscher. An analysis of systematic approaches to reasoning about actions and change. Manuscript, 1994.

[Tur94.] H. Turner. Signed logic programs. In *Proc. of ISLP*, 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.

APPENDIX: PROOFS

In this we prove the theorems stated in the earlier sections. First, let us notice that observation 3.2 holds for $\pi' D$ and $\pi_f D$. Hence, in this section we only consider the partially evaluated πD, $\pi' D$ and $\pi_f D$ w.r.t. $Lit(\{atomic, subset, subseteq, eq\})$.

Lemma 5.1. 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. □

Proof. Follows immediately from the definition of answer sets. □

Proposition 5.4. 1. Let D be a domain description. The logic programs $(\pi' D)^+$, and $(\pi_f D)^+$ [9] are locally stratified [Prz88].

[9] For an extended logic program Π the normal logic program Π^+ denotes the program obtained by replacing each occurrence 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^+.

2. If D is a domain description without any v-propositions then the program πD is locally stratified. □

PROOF. (sketch)
For a sequence of actions $\alpha = a_1, \ldots, a_m$ its level is defined as the pair (m, k) where k is $|a_m|$, the cardinality of a_m. We say $(m, k) < (m', k')$ iff $m < m'$ or $m = m'$ 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) For atoms $holds(f, [\alpha])$ and $holds(f, [\alpha'])$ where α and α' are sequence of actions, if $level(\alpha') < level(\alpha)$ then $strata(holds(f, [\alpha'])) < strata(holds(f, [\alpha]))$.

(ii) $strata(holds^+(f, [\alpha])) = strata(causes(a, f, [\alpha])) = strata(cancels(X, Y, f, [\alpha])) <$ $strata(may\_imm\_cause(a, f, [\alpha])) = strata(noninherit(f, X, [\alpha]))$ $= strata(undefined(a, [\alpha])) < strata(holds^+(f, [\alpha|b]))$ □

Lemma 5.2. 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
$[\pi D]_A = \pi D \cup \{holds(f, s_0) \ : \ holds(f, s_0) \in A\} \cup \{\neg holds(f, s_0) \ : \ \neg holds(f, s_0) \in A\}$. □

PROOF. Let $P = \{p \ : p$ is a v-proposition in $D\}$, and let $D' = D \setminus P$.
$\Longrightarrow$
Let A be a consistent answer set of πD. It is easy to see that A is a consistent answer set of $(\pi D')^A$ and A satisfies the constraints πP. 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 $([\pi D']_A)$. The program $([\pi D']_A)^+$ is locally stratified and by Lemma 5.1 we can eliminate the disjunctions in it without affecting the answers sets and therefore it has a unique answer set which is also the unique answer set of $([\pi D]_A)^+$. Hence, A is the 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 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. □

Lemma 5.3. Let D be a domain description. $M = (\sigma_0, \Phi)$ is a model of D iff M is the model of $D_M = D \cup \{$ **initially** $f \ : \ f \in \sigma_0\} \cup \{$ **initially** $\neg f \ : \ f \notin \sigma_0\}$. □

PROOF. $\Longrightarrow$
Let $M = (\sigma_0, \Phi)$ be a model of D. It is easy to see that M is a model of D_M. We

now need to show that it is the only model of D_M. Suppose $M' \neq M$ is also a model of D. It is clear that M' and M only differ in the initial state. But then M' can not satisfy the v-propositions about the initial state in D_M. Hence, M is the only model of D_M.

$\Longleftarrow$
Obvious. $\square$

We now introduce some notations for the forthcoming proofs.

Notation

1. We denote the situation $[a_1, \ldots, a_n]$ by s_n and the state $M^{(a_1,\ldots,a_n)}$ by σ_n.
2. 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.
3. 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 $H(p, s)$.
4. We denote the set of fluent atoms, $\{f :\ a$ immediately causes f in $\sigma\}$ by $direct^+(a, \sigma)$.
5. We denote the set of fluent atoms $\{f :\ a$ immediately causes $\overline{f}$ in $\sigma\}$ by $direct^-(a, \sigma)$.
6. We denote the set of fluent atoms, $\{f :\ a$ inherits the effect f from its subsets in $\sigma\}$ by $inherited^+(a, \sigma)$.
7. We denote the set of fluent atoms, $\{f :\ a$ inherits the effect $\overline{f}$ from its subsets in $\sigma\}$ by $inherited^-(a, \sigma)$.

Note that, $E^+(a, \sigma) = direct^+(a, \sigma) \cup inherited^+(a, \sigma)$ and
$E^-(a, \sigma) = direct^-(a, \sigma) \cup inherited^-(a, \sigma)$.

Lemma 5.4. **Models of D vs Answer sets of πD**
Let $M = (\sigma_0, \Phi)$ be a model of D and A be a consistent answer set of πD such that $\sigma_0 = \{f\ :\ holds(f, s_0) \in A\}$.

If $a_1, \ldots, a_n$ is executable in M then

(a) $f \in direct^+(a_n, \sigma_{n-1})$ iff $may\_imm\_cause(a_n, f, s_{n-1}) \in A$.
(b) $f \in direct^-(a_n, \sigma_{n-1})$ iff $may\_imm\_cause(a_n, \overline{f}, s_{n-1}) \in A$.
(c) $f \in inherited^+(a_n, \sigma_{n-1})$ iff $noninh(\overline{f}, a_n, s_{n-1}) \in A$.
(d) $f \in inherited^-(a_n, \sigma_{n-1})$ iff $noninh(f, a_n, s_{n-1}) \in A$.
(e) $undefined(a_n, s_{n-1}) \notin A$.
(f) $\underline{f \in \sigma_n \Leftrightarrow holds(f, s_n) \in A.}$
(g) $\underline{f \notin \sigma_n \Leftrightarrow \neg holds(f, s_n) \in A.}$

2. If $a_1, \ldots, a_n$ is not executable in M then
$holds(f, s_n) \notin A$ and $\neg holds(f, s_n) \notin A$ and $undefined(a_n, s_{n-1}) \in A$.

$\square$

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

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

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

1. If $a_1, \ldots, a_k$ is executable in M then
   (a) $f \in direct^+(a_k, \sigma_{k-1})$ iff $may\_imm\_cause(a_k, f, s_{k-1}) \in A$.
   (b) $f \in direct^-(a_k, \sigma_{k-1})$ iff $may\_imm\_cause(a_k, \overline{f}, s_{k-1}) \in A$.
   (c) $f \in inherited^+(a_k, \sigma_{k-1})$ iff $noninh(\overline{f}, a_k, s_{k-1}) \in A$.
   (d) $f \in inherited^-(a_k, \sigma_{k-1})$ iff $noninh(f, a_k, s_{k-1}) \in A$.
   (e) $undefined(a_k, s_{k-1}) \notin A$
   (f) $f \in \sigma_k \Leftrightarrow holds(f, s_k) \in A$.
   (g) $f \notin \sigma_k \Leftrightarrow \neg holds(f, s_k) \in A$.
2. If $a_1, \ldots, a_k$ is not executable in M then For any fluent atom f, $holds(f, s_k) \notin A$ and $\neg holds(f, s_k) \notin A$ and $undefined(a_k, s_{k-1}) \in A$.

Induction: level = (n,m)

1. Let $a_1, \ldots, a_n$ be executable in M. Then,
   (a) $f \in direct^+(a_n, \sigma_{n-1})$
   iff
   there is an e-proposition (a_n **causes** f **if** p) in D such that p holds in σ_{n-1} (by definition)
   iff
   $H(p, s_{n-1}) \subset A$ (by IH)
   iff
   $may\_imm\_cause(a_n, f, s_{n-1}) \in A$ (By consistency of A, rule (3a) and Proposition 3.1).
   (b) similar to 1 (a).
   (c) $f \in inherited^+(a_n, \sigma_{n-1})$
   iff
   there exists an e-proposition b **causes** f **if** q in D such that q holds in s_{n-1} and $b \subset a_n$, such that for all c, $b \subset c \subseteq a_n$, if there is an e-proposition c **causes** $\overline{f}$ **if** r in D then r does not hold in s_{n-1} (by definition)
   iff
   $f \in direct^+(b, \sigma_{n-1})$ and $r \notin \sigma_{n-1}$ iff
   $may\_imm\_cause(b, f, s_{n-1}) \in A$ and $H(r, s_{n-1}) \not\subset A$ (by IH)
   iff
   $may\_imm\_cause(b, f, s_{n-1}) \in A$ and $cancels(b, a_n, f, s_{n-1}) \notin A$ (Using consistency of A and applying Proposition 3.1 to rules (4c) and (3b).)
   iff
   $noninh(\overline{f}, a_n, s_{n-1}) \in A$. (By applying Proposition 3.1 to rule (4d))
   (d) similar to 1 (c).
   (e) $a_1, \ldots, a_n$ is executable in M $\Rightarrow$
   $E^+(a_n, \sigma_{n-1}) \cap E^-(a_n, \sigma_{n-1}) = \emptyset$ $\Rightarrow$
   $direct^+(a_n, \sigma_{n-1}) \cap direct^-(a_n, \sigma_{n-1}) = \emptyset$ and
   $inherited^+(a_n, \sigma_{n-1}) \cap inherited^-(a_n, \sigma_{n-1}) = \emptyset$ $\Rightarrow$
   There does not exist f such that both $may\_imm\_cause(a_n, f, s_{n-1})$ and $may\_imm\_cause(a_n, \overline{f}, s_{n-1})$ are in A, and there does not exist f such that both $noninh(f, a_n, s_{n-1})$ and $noninh(\overline{f}, a_n, s_{n-1})$ are in A (Using 1(a) to 1(d) of Lemma 5.4.) $\Rightarrow$

$undefined(a_n, s_{n-1}) \notin A$ (Using Proposition 3.1 on rules (3d) and (4e) of the program.)

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

i. a_n is atomic and $f \in \sigma_{n-1} \setminus E^-(a_n, \sigma_{n-1})$

ii. a_n is not atomic and $f \in \sigma_{n-1} \setminus E^+(a_n, \sigma_{n-1}) \setminus E^-(a_n, \sigma_{n-1})$

iii. $f \in direct^+(a_n, \sigma_{n-1})$

iv. a_n is not atomic and $f \in inherited^+(a_n, \sigma_{n-1}) \setminus E^-(a_n, \sigma_{n-1})$

i. a_n is atomic and $f \in \sigma_{n-1} \setminus E^-(a_n, \sigma_{n-1}) \quad \Rightarrow$
$holds(f, s_{n-1}) \in A$ (by IH) and $may\_imm\_cause(a_n, \overline{f}, s_{n-1}) \notin A$ (by 1(b)) and $undefined(a_n, s_{n-1}) \notin A$ (by 1(e)) $\quad \Rightarrow$
$holds(f, s_n) \in A$ (by using Proposition 3.1 on rule (1a). Note that we can use rule (1a) only because a_n is atomic.).

ii. a_n is not atomic and $f \in \sigma_{n-1} \setminus E^+(a_n, \sigma_{n-1}) \setminus E^-(a_n, \sigma_{n-1})$
$\Rightarrow$
From $f \notin E^+(a_n, \sigma_{n-1})$ and $f \notin E^-(a_n, \sigma_{n-1})$ 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 s_{n-1}; It is easy to see that b is executable in σ_{n-1} and $f \in \Phi(b, \sigma_{n-1})$. By IH this implies that $holds(f, res(b, s_{n-1})) \in A$. From $f \notin inherited^-(a_n, \sigma_{n-1})$ using 1(d) we can conclude that $noninh(f, a_n, s_{n-1}) \notin A$. By 1(e) we have $undefined(a_n, s_{n-1}) \notin A$. Hence using Proposition 3.1 on the rule (4a) we can conclude that $holds(f, s_n) \in A$.

iii. $f \in direct^+(a_n, \sigma_{n-1}) \quad \Rightarrow$
$may\_imm\_cause(a_n, f, s_{n-1}) \in A$ (from 1(a)) $\quad \Rightarrow$
$cause(a_n, f, s_{n-1}) \in A$ (By IH and Proposition 3.1 applied to rules (3a) and (3b)) $\quad \Rightarrow$
$holds(f, s_n) \in A$ (Using 1(e) and applying Proposition 3.1 to rule (3c)).

iv. a_n is not atomic and $f \in inherited^+(a_n, \sigma_{n-1}) \setminus E^-(a_n, \sigma_{n-1})$
$\Rightarrow$
From $f \in inherited^+(a_n, \sigma_{n-1})$ 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 and $f \in direct^+(b, \sigma_{n-1})$; from which we can conclude by (iii) that $holds(f, res(b, s_{n-1})) \in A$. From $f \notin E^-(a_n, \sigma_{n-1})$ we can conclude (by 1(d)) that $noninh(f, a_n, s_{n-1}) \notin A$. Hence, using Proposition 3.1, 1(e) and rule (4a) we can conclude that $holds(f, s_n) \in A$.

$\Longleftarrow$ Let $holds(f, s_n) \in A \quad \Rightarrow$
By Proposition 3.1 at-least one of the following three cases must be true. (Note that $undefined(a_n, s_{n-1}) \notin A$ in all the three cases)

i. $holds(f, s_{n-1}) \in A$, a_n is atomic and $may\_imm\_cause(a_n, \overline{f}, s_{n-1}) \notin A$. (Using rule (1a))

ii. There exists an e-proposition (a_n **causes** f **if** p), such that $H(p, s_{n-1}) \subset A$. (Using rule (3c)).

iii. There exists an action $b \subset a_n$ such that $holds(f, res(b, s_{n-1})) \in A$ and $noninh(f, a_n, s_{n-1}) \notin A$. (using rule (4a)).

i. By IH, $holds(f, s_{n-1}) \in A$ implies $f \in \sigma_{n-1}$.
From $may\_imm\_cause(A_n, \overline{f}, s_{n-1}) \notin A$ using 1(b) we can conclude that $f \notin direct^-(a_n, s_{n-1})$. Since a_n is atomic, $inherited^-(a_n, s_{n-1}) = \emptyset$. Hence, $f \notin E^-(a_n, s_{n-1})$. Hence, $f \in \sigma_n$.

ii. There exists an e-proposition (a_n **causes** f **if** p), such that $H(p, s_{n-1}) \subset A$. By IH we have p holds in σ_{n-1}. Hence, $f \in E^+(a_n, s_{n-1})$. Since, $a_1, \ldots, a_n$ is executable we have that $f \notin E^-(a_n, s_{n-1})$. Hence, $f \in \sigma_n$.

iii. There exists an action $b \subset a_n$ such that $holds(f, res(b, s_{n-1})) \in A$ and $noninh(f, a_n, s_{n-1}) \notin A$.
We will first show that $f \notin direct^-(a_n, \sigma_{n-1})$. Suppose it is not the case. Then using the arguments similar to (iii) of ($\Longrightarrow$) we will have $\neg holds(f, s_n) \in A$. This makes A inconsistent and we have a contradiction.
From, $noninh(f, a_n, s_{n-1}) \notin A$ we conclude (using 1(d))
$f \notin inherited^-(a_n, \sigma_{n-1})$. Hence, $f \notin E^-(a_n, \sigma_{n-1})$.
From $holds(f, res(b, s_{n-1})) \in A$ using IH we can conclude that $f \in \sigma_{n-1}$ or $f \in E^+(b, \sigma_{n-1})$. If $f \in \sigma_{n-1}$, then since we showed $f \notin E^-(a_n, \sigma_{n-1})$, we have $f \in \sigma_n$.
Now suppose $f \in E^+(b, \sigma_{n-1}) \setminus \sigma_{n-1}$. This means there exists an action $b_1 \subset a_n$ such that $f \in direct^+(b_1, \sigma_{n-1})$. Let b_2 be the maximal subaction of a_n such that $f \in direct^+(b_2, \sigma_{n-1})$; i.e., there does not exist an action c, $b_2 \subset c \subseteq a_n$ such that $f \in direct^+(c, \sigma_{n-1})$.
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 σ_{n-1}.

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

Therefore, by definition $f \in E^+(a_n, \sigma_{n-1})$ and hence $f \in \sigma_n$.

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

2. $a_1, \ldots a_n$ is not executable $\Rightarrow$
There are two cases:

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

(case 1) $E^+(a_n, \sigma_{n-1}) \cap E^-(a_n, \sigma_{n-1}) \neq \emptyset$ $\Rightarrow$
(case 1.1) $direct^+(a_n, \sigma_{n-1}) \cap direct^-(a_n, \sigma_{n-1}) \neq \emptyset$ or
(case 1.2) $inherited^+(a_n, \sigma_{n-1}) \cap inherited^-(a_n, \sigma_{n-1}) \neq \emptyset$
(The other two cases are not possible.)

(case 1.1) $direct^+(a_n, \sigma_{n-1}) \cap direct^-(a_n, \sigma_{n-1}) \neq \emptyset$ $\Rightarrow$
Similar to the case 1 (a) we can show that for some f,

$may\_imm\_cause(a_n, f, s_{n-1}) \in A$ and $may\_imm\_cause(a_n, \overline{f}, s_{n-1}) \in A \Rightarrow$ $undefined(a_n, s_n) \in A$ (by rule (3d).)

(case 1.2) $inherited^+(a_n, \sigma_{n-1}) \cap inherited^-(a_n, \sigma_{n-1}) \neq \emptyset \quad \Rightarrow$
There exists $a, a' \subset a_n$ such that $f \in direct^+(a, \sigma_{n-1})$, $f \in direct^-(a', \sigma_{n-1})$, and there does not exist b and b' where $a \subset b \subseteq a_n$, $a' \subset b' \subseteq a_n$, $f \in direct^-(b, \sigma_{n-1})$ and $f \in direct^+(b', \sigma_{n-1})$.
Similar to the case 1 (a) we can show that $may\_imm\_cause(a, f, s_{n-1}) \in A$, $may\_imm\_cause(a', \overline{f}, s_{n-1}) \in A$, $cancels(a, a_n, f, s_{n-1}) \notin A$ and $cancels(a', a_n, f, s_{n-1}) \notin A$. By using (4d) and Proposition 3.1 we have $noninh(f, a_n, s_{n-1}) \in A$ and $noninh(\overline{f}, a_n, s_{n-1}) \in A \Rightarrow$ $undefined(a_n, s_n) \in A$ (by Proposition 3.1 and rules (4e).).

(case 2) Using IH and rule (3e) we get $undefined(a_n, s_n) \in A$.

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

This ends the proof of this lemma. □

Lemma 5.5. For every model $M = (\sigma_0, \Phi)$ of a consistent domain description D, there exists a consistent answer set A of πD such that $\sigma_0 = \{f \ : \ holds(f, s_0) \in A\}$. □

PROOF. Consider $D' = D \setminus$ the set of v-propositions in D and $[\pi' D']_{\sigma_0} = \pi' D' \cup \{holds(f, s_0) \ : \ f \in A\}\{\neg holds(f, s_0) \ : \ f \in A\}$ It is easy to see that M is a model of D'. Since, $([\pi' D']_{\sigma_0})^+$ is locally stratified and contains complete information about the initial state by Lemma 5.1 the disjunctions (about the initial state) in it can be ignored without affecting its answer sets, and hence $([\pi D']_{\sigma_0})^+$ will have a unique answer set, say A^+. We will first show that A^+ is coherent.[10] Incoherency is possible if we derive both $holds(F, S)$ and $holds'(F, S)$ from $([\pi D']_{\sigma_0})^+$. $holds(F, S)$ can be derived using (1a), (3c) and (4a) and $holds'(F, S)$ can be derived using (1b), (3c) and (4b). Using induction on the level of S, it can be easily shown that
(i) the bodies of (1a) and (1b) can not be true at the same time.
(ii) the bodies of (1a) and (3c) can not be true at the same time, because if the body of (3c) is true then the atom with may_imm_cause will become true making the body of (1a) false.
(iii) the bodies of (1a) and (4b) can not be true at the same time, because (1a) is applicable only for atomic actions, while (4b) needs the action to be non-atomic.
(iv) the bodies of (3c) and (1b) can not be true at the same time. (similar reason as in (ii))
(v) We can not have both $holds(f, s)$ and $holds(f, s)$ derived using (3c) because in that case $undefined$ will block the derivation of both.
(vi) the bodies of (3c) and (4b) can not be true at the same time, because if the body of (3c) is true then the atom with $noninh$ will become true making the body of (4b) false.
(vii) the bodies of (4a) and (1b) can not be true at the same time. (similar reason

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

as in (iii))
(viii) the bodies of (4a) and (3c) can not be true at the same time, (similar reason as in (vi))
(ix) the bodies of (4a) and (4b) can not be true at the same time, because *noninh* will block the derivation of at-least one of them.
Hence, A^+ is coherent and hence [GL91] A is consistent and is the unique answer set of $[\pi D']_{\sigma_0}$.

From Lemma 5.2 it is easy to see that A is a consistent answer set of $\pi D'$. Using, Lemma 5.4 w.r.t. D' we have that A and M agree on all v-propositions in D. Since, M is a model of D, M satisfies all v-propositions in D and therefore A must satisfy the constraints in πD. Hence, A is an answer set of πD. □

Lemma 5.6. Let D be a consistent domain description. For every consistent answer set A of πD there exists a model $M = (\sigma_0, \Phi)$ of D such that $\sigma_0 = \{f : holds(f, s_0) \in A\}$. □

PROOF. Let $D' = D \setminus$ the set of v-propositions in D. Let $[D']_A = D' \cup \{$ **initially** $f : holds(f, s_0) \in A\} \cup \{$ **initially** $\neg f : holds(f, s_0) \notin A\}$. It is easy to see that $[D']_A$ is consistent and has a unique model with the initial state σ_0. Let $M = (\sigma_0, \Phi)$ be the model of $[D']_A$. From Lemma 5.3 it is easy to see that M is a model of D'. Since A is a consistent answer set of πD, it is easy to see that A is a consistent answer set of $\pi D'$. Using, Lemma 5.4 w.r.t. D' we have that A and M agree on all v-propositions. Since, A is an answer set of πD, A satisfies the translations of all v-propositions of D in πD and therefore M must satisfy all v-propositions of D. Hence, M is a model of D. □

Theorem 5.1. **Soundness and Completeness of π.**
Let D be an arbitrary domain description and $P = f$ **after** $a_1, \ldots, a_n$ be any v-proposition in the language of D such that $a_1, \ldots, a_n$ is executable in any model of D. Then $\pi D \models h(f, [a_1, \ldots, a_n])$ iff $D \models P$. □

PROOF. Directly from Lemma 5.4, Lemma 5.5 and Lemma 5.6.

Now we will prove properties of the translation π'.

Lemma 5.7. Let D be a consistent domain description. Let $M = (\sigma_0, \Phi)$ be a model of D. Then there exist a consistent answer set A of $\pi' D$ such that:

If $a_1, \ldots, a_n$ is executable in M then

$$f \in \Phi(a_n, [a_1, \ldots, a_{n-1}]) \Rightarrow holds(f, [a_1, \ldots a_n]) \in A$$
$$f \notin \Phi(a_n, [a_1, \ldots, a_{n-1}]) \Rightarrow \neg holds(f, [a_1, \ldots a_n]) \in A$$

otherwise, $holds(f, [a_1, \ldots a_n]) \notin A$ and $\neg holds(f, [a_1, \ldots a_n]) \notin A$ □

PROOF. From Lemma 5.5 and the observation that answer sets of πD are answer sets of $\pi' D$. □

Theorem 5.2. **Soundness of π'**
Let D be an arbitrary domain description and $P = f$ **after** $a_1, \ldots, a_n$ be any v-proposition in the language of D such that $a_1, \ldots, a_n$ is executable in any model of D. If $\pi' D \models \pi' P$ then $D \models P$. □

PROOF. Directly from Lemma 5.7.

The following lemmas are needed to prove a completeness result of $\pi' D$ w.r.t. D.

Lemma 5.8. Let Π be a consistent extended logic program, such that Π^+ is locally stratified and for all atoms p, both p and p' belong to the same strata. Then if $\Pi \models l$ then $\Pi \cup \bar{l}$ is inconsistent.

Lemma 5.9. For every model $M = (\sigma_0, \Phi)$ of a consistent domain description D, there exists a consistent answer set A of $\pi' D$ such that $\sigma_0 = \{f \; : \; holds(f, s_0) \in A\}$. □

PROOF. Similar to Lemma 5.5.

Lemma 5.10. Let D be an well-founded domain description and let $S = \{holds(f, [a_1, \ldots, a_n]) \; : \; a_1, \ldots a_n$ are associated with $D\,\} \cup \{\neg holds(f, [a_1, \ldots, a_n]) \; : \; a_1, \ldots a_n$ are associated with $D\,\}$. Let $D' = D \backslash$ the v-propositions in D.

(1) All answer sets of $\pi' D'$ are complete w.r.t. S. (note $\pi D'$ is the same as $\pi' D'$)

(2)Let A be a set of literals such that for any fluent f either $holds(f, s_0) \in A$ or $\neg holds(f, s_0) \in A$. A is the consistent answer set of $[\pi' D']_A$ and for all v-propositions p in D, $\pi' p \in A$ iff
A is the consistent answer set of $[\pi' D]_A$

PROOF. (1) From well-foundedness of D and Lemma 5.4.

(2) Local stratification guarantees that both $[\pi' D']_A$ and $[\pi' D]_A$ have unique answer sets.
$\Rightarrow$ Obvious
$\Leftarrow$ Because of the uniqueness of answer sets of $[\pi' D']_A$ and $[\pi' D]_A$ it is sufficient if we show that the answer set of $[\pi' D']_A$ is the answer set of $[\pi' D]_A$. Let B be the answer set of $[\pi' D']_A$. Suppose B does not satisfy $l = \pi' p$, for some v-proposition $p \in D$. Since by part (1) B is complete w.r.t. S, we have $[\pi' D']_A \models \bar{l}$. Using Lemma 5.8 we have $[\pi' D]_A$ to be inconsistent. But by Lemma 5.9 we have $[\pi' D]_A$ to be consistent. Hence, we have a contradiction and our assumption about B not satisfying $l = \pi' p$, for some v-proposition $p \in D$ is wrong. Hence, for all v-proposition $p \in D$, $\pi' p \in B$. Hence, B is the answer set of $[\pi' D]_A$. □

Theorem 5.3. For any consistent well-founded domain description D the programs πD and $\pi' D$ have the same answer sets. □

PROOF. A is a consistent answer set of $\pi' D$ iff
A is the consistent answer set of $[\pi' D]_A$ (By reasoning similar to Lemma 5.2) iff
A is the consistent answer set of $[\pi' D']_A$ and for all v-propositions p in D, $\pi' p \in A$ (From part (2) of Lemma 5.10) iff
A is the consistent answer set of $[\pi D']_A$ and for all v-propositions p in D, $\pi' p \in A$ (Since $\pi D'$ and $\pi' D'$ are the same) iff
A is the consistent answer set of $[\pi D]_A$ (Since A satisfies all the constraints) iff
A is a consistent answer set of πD. (By Lemma 5.2) □