Actions with Indirect Effects
(Preliminary Report)

G. Neelakantan Kartha
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

Vladimir Lifschitz
Department of Computer Sciences
and Department of Philosophy
University of Texas at Austin
Austin, Texas 78712-1188

Abstract

We define and study a high-level language for describing actions that extends the language $\mathcal{A}$ introduced by Gelfond and Lifschitz. The new language, $\mathcal{AR}_0$, allows us to describe actions with indirect effects (ramifications) and simple forms of nondeterminism. A translation from $\mathcal{AR}_0$ into a formalism based on circumscription is proved to be sound and complete.

1 Introduction

Describing properties of actions and their effects on the state of the world has long been considered one of the central problems in the theory of knowledge representation. The approaches proposed in the literature differ by the temporal ontologies they use (linear or branching time, time points or intervals, situations, events or histories), by the logic used (classical logic, its nonmonotonic extensions, logic programming), and by other details of the formalization (which objects are reified, which circumscription policy is used, etc.).

In this area of research, it turned out to be difficult to discuss the possibilities and limitations of the available methods in a precise and general way. The tradition is to explain every new approach with reference to a few standard examples, such as the blocks world or the "Yale Shooting" story and its enhancements. Competing approaches are evaluated and compared mostly in terms of their ability to handle these examples. Such analysis does not say much about the range of applicability of each method.

Several recent publications, including [Pednault, 1989], [Lifschitz, 1991], [Lin and Shoham, 1991], [Reiter, 1991] and [Sandewall, 1992], attempt to overcome this problem and to discuss representing action in a methodical and theoretically sound way. Gelfond and Lifschitz [1993] address this issue by introducing a simple "high-level" language, called $\mathcal{A}$, designed specifically for describing the effects of actions. Here, for instance, is the "Yale Shooting" domain encoded in $\mathcal{A}$:

$$\begin{array}{l} \textbf{initially } \neg \mathit{Loaded}, \\ \textbf{initially } \mathit{Alive}, \\ \mathit{Load} \textbf{ causes } \mathit{Loaded}, \\ \mathit{Shoot} \textbf{ causes } \neg \mathit{Alive} \textbf{ if } \mathit{Loaded}, \\ \mathit{Shoot} \textbf{ causes } \neg \mathit{Loaded}. \end{array} \tag{1}$$

The first two lines are "value propositions"—they provide information about the values of fluents, such as *Loaded* or *Alive*, at a specific point in time (in this case, in the initial situation). The other three are "effect propositions"—they describe the effects of actions, such as *Load* or *Shoot*. The semantics of $\mathcal{A}$ allows us to determine which value propositions are "entailed" by this domain description. One such proposition, for instance, is

$$\neg \mathit{Alive} \textbf{ after } \mathit{Load}; \mathit{Wait}; \mathit{Shoot}.$$

The available methods for describing actions in logic can be characterized as translations from high-level action languages into monotonic or nonmonotonic logic-based formalisms. The claim that such a method is adequate turns then into a mathematically verifiable property of the corresponding translation. Competing methods can be compared by comparing the high-level languages to which they are applicable.

In [Gelfond and Lifschitz, 1993], for instance, a new method for representing actions in logic programming is specified as a translation from a subset of $\mathcal{A}$ into a logic programming language, and the translation is proved to be sound relative to the semantics of $\mathcal{A}$. In [Kartha, 1993], the methods for formalizing action in classical logic proposed in [Pednault, 1989] and [Reiter, 1991], as well as the use of circumscription in [Baker, 1991], are described as translations from $\mathcal{A}$, and the soundness and completeness theorems for these translations are stated. In the same spirit, $\mathcal{A}$ is translated into abductive logic programming in [Dung, 1993] and [Denecker and De Schreye, 1993], and into equational logic programming in [Thielscher, 1994].

If a theorem prover, or a query evaluation procedure, is available for the target language of such a translation, then it becomes possible to use the translation for the automation of reasoning about action. For example, the logic programming interpreter XOLDT [Chen and Warren, 1992] is put to such use in [Lifschitz *et al.*, 1993], the abductive procedure SLDNFA in [Denecker and De Schreye, 1993], and the theorem prover of [Boyer and Moore, 1988] in [Subramanian, 1993].

In [Thielscher, 1993], the language $\mathcal{A}$ is related to the ideas of [Sandewall, 1992].

In spite of its extremely simple syntax, $\mathcal{A}$ provides a framework for discussing some interesting cases of reasoning about action, including temporal projection with incomplete information about the initial situation, reasoning from the future to the past ("explanation"), and some forms of hypothetical reasoning ("if a different sequence of actions were performed..."). In many ways, however, the expressive possibilities of this language are limited. This has led to the development of a few enhancements of $\mathcal{A}$. Dung [1993] proposed a "relational" version of $\mathcal{A}$, in which fluents and actions may have arguments. In [Denecker and De Schreye, 1993] and [Thielscher, 1994], dialects of $\mathcal{A}$ are outlined in which one can describe actions with nondeterministic effects. Baral and Gelfond [1993] extend $\mathcal{A}$ by an operator for the concurrent execution of a set of actions. The dialect of $\mathcal{A}$ defined in [Lifschitz, 1993b] provides symbols for time intervals.

This paper addresses yet another limitation of $\mathcal{A}$—its inability to represent *domain constraints*. In $\mathcal{A}$, any combination of fluent values represents a possible state of the system. The extension of $\mathcal{A}$ presented below is, in this respect, different. We will be able, for instance, to extend domain description (1) by the constraint

$$\textbf{always } \mathit{Walking} \supset \mathit{Alive}, \tag{2}$$

where *Walking* is a new fluent name. This constraint tells us that an assignment of truth values to fluents represents a state only if it makes the conditional $\mathit{Walking} \supset \mathit{Alive}$ true.

The domain description consisting of propositions (1) and proposition (2) entails, for instance, the value proposition

$$\neg \mathit{Walking} \textbf{ after } \mathit{Load}; \mathit{Wait}; \mathit{Shoot}.$$

Note that the fluent *Walking* does not occur in the effect propositions. Thus any change in the value of *Walking* is an *indirect effect*, or *ramification*, of the actions that have been performed. The possibility of ramifications is the main feature of the dialect of $\mathcal{A}$ introduced here. We call this language $\mathcal{AR}_0$; $\mathcal{A}$ stands for "actions," $\mathcal{R}$ for "ramifications," and the subscript 0 is used to distinguish this language from the more expressive language $\mathcal{AR}$ that will be defined in the journal version of the paper.

Besides the ability to describe indirect effects, the new language has other expressive possibilities that $\mathcal{A}$ lacks. It allows us to specify not only "fluent preconditions"—the conditions that need to be satisfied in order for an action to affect a specific fluent—but also "action preconditions," which, if violated, make the execution of an action impossible.[1] In $\mathcal{AR}_0$, we can specify a "coordinate frame" in the space of situations, as suggested in [Lifschitz, 1990]; the commonsense law of inertia applies only to the fluents that belong to the frame. Finally, the effects of the actions described in $\mathcal{AR}_0$ can be nondeterministic.

The syntax of the new action language is described in Section 2. Examples illustrating the use of the language are provided in Section 3, and its semantics is defined in Section 4. In Section 5, we show how two well-known properties of first-order logic—the replacement theorem and the theorem on the conservativeness of definitional extensions—can be extended to $\mathcal{AR}_0$. In Section 6, the language $\mathcal{A}$ from [Gelfond and Lifschitz, 1993] is embedded into $\mathcal{AR}_0$.

In Section 7, we use the ideas of [Baker, 1991] to define a translation from $\mathcal{AR}_0$ into the formalism of *nested abnormality theories* [Lifschitz, 1995]. This formalism is based on circumscription, and permits nested applications of the circumscription operator. In this framework, Baker's approach to the frame problem can be presented in a particularly simple form. At the end of the section, a soundness and completeness theorem is stated. It shows that, in the new version, Baker's method is applicable to some examples of nondeterministic actions.

2 Syntax of $\mathcal{AR}_0$

2.1 Formulae and Propositions

To be precise, $\mathcal{AR}_0$, like $\mathcal{A}$, is not a single language, but rather a family of languages. A particular language in this group is characterized by

- a nonempty set of symbols, that are called *fluent names*, or *fluents*,
- a subset of fluent names, that is called the *frame*,
- a nonempty set of symbols, that are called *action names*, or *actions*.

A *formula* is a propositional combination of fluents.

There are four types of *propositions* in $\mathcal{AR}_0$—value propositions, effect propositions, release propositions, and constraints.

A *value proposition* is an expression of the form

$$C \textbf{ after } \overline{A}, \tag{3}$$

where C is a formula, and $\overline{A}$ is a string of actions. Informally, (3) asserts that C holds after the sequence

[1] This terminology belongs to Reiter [1991].

of actions $\overline{A}$ is performed in the initial situation. For instance,

$$\textit{Heads} \textbf{ after } \textit{Toss}$$

is a value proposition, where *Heads* is a fluent and *Toss* is an action.

An *effect proposition* is an expression of the form

$$A \textbf{ causes } C \textbf{ if } P, \tag{4}$$

where A is an action, and C and P are formulae. Intuitively, (4) asserts that A, if executed in a situation in which the precondition P is true, makes C true. For instance,

$$\textit{GetOnBoard} \textbf{ causes } \textit{OnBus} \textbf{ if } \textit{HasTicket}$$

is an effect proposition, where *GetOnBoard* is an action, and *OnBus* and *HasTicket* are fluents.

A *release proposition* is an expression of the form

$$A \textbf{ releases } F \textbf{ if } P, \tag{5}$$

where A is an action, F a fluent which belongs to the frame, and P a formula. Intuitively, (5) says that F is exempt from obeying the commonsense law of inertia when the action A is executed in a situation when the precondition P is true. This is useful for expressing the effects of nondeterministic actions. For instance, the fact that the action *Toss* nondeterministically changes the frame fluent *Heads* is expressed by the release proposition

$$\textit{Toss} \textbf{ releases } \textit{Heads} \textbf{ if } \textit{True}.$$

(*True* stands for some standard tautology, and *False* will denote the negation of *True*.)

Finally, a *constraint* is a proposition of the form

$$\textbf{always } C, \tag{6}$$

where C is a formula. Intuitively, (6) asserts that C holds in all possible situations. We will sometimes identify a constraint (6) with the formula C.

A *domain description*, or *domain*, is a set of propositions.

2.2 Notational Conventions

In formulae, we will omit some parentheses, as customary in classical logic. If $\overline{A}$ in a value proposition (3) is empty, we will write this proposition as

$$\textbf{initially } C.$$

Otherwise, the members of $\overline{A}$ will be separated by semicolons. An effect proposition (4) will be written as

$$A \textbf{ causes } C$$

if P is *True*, and as

$$\textbf{impossible } A \textbf{ if } P$$

if C is *False*. A release proposition (5) will be written as

$$A \textbf{ releases } F$$

if P is *True*.

3 Examples

In this section we show how some systems involving action and change can be described in $\mathcal{AR}_0$. Our comments on the "models" of these domain descriptions will turn into precise statements after the semantics of $\mathcal{AR}_0$ is described in the next section.

The reader familiar with the language $\mathcal{A}$ from [Gelfond and Lifschitz, 1993] will notice that any domain description in $\mathcal{A}$ can be viewed as a domain description in $\mathcal{AR}_0$ in which all fluents belong to the frame; we only need to agree to identify an effect proposition

$$A \textbf{ causes } F \textbf{ if } P_1, \dots, P_n$$

in the sense of $\mathcal{A}$ with

$$A \textbf{ causes } F \textbf{ if } P_1 \wedge \ldots \wedge P_n.$$

In Section 6, this will be discussed in more detail.

The following examples illustrate some of the expressive possibilities of $\mathcal{AR}_0$ that $\mathcal{A}$ lacks.

Example 1: Action Preconditions. The door in a hotel room can be opened by inserting the entry card. In the initial situation, the door is closed. We use two propositional fluents *HasCard* and *DoorOpen*, both belonging to the frame. There is one action *InsertCard*. The propositions are

$$\begin{array}{l} \textbf{initially } \neg \textit{DoorOpen}, \\ \textit{InsertCard} \textbf{ causes } \textit{DoorOpen}, \\ \textbf{impossible } \textit{InsertCard} \textbf{ if } \neg \textit{HasCard}. \end{array}$$

This domain has two models; in one model, *HasCard* is false in the initial situation, and in the other model it is initially true. In the first model, the action of inserting the card cannot be executed.

Example 2: Nonframe Fluents. There are two switches and a light; the light is on only when the switches are both on or both off [Lifschitz, 1990]. This system can be described using three propositional fluents *Switch1, Switch2* and *Light*; only the first two belong to the frame. There are two action names, *Toggle1* and *Toggle2*. The propositions are:

$$\begin{array}{l} \textbf{always } \textit{Light} \equiv (\textit{Switch1} \equiv \textit{Switch2}), \\ \textit{Toggle1} \textbf{ causes } \neg \textit{Switch1} \textbf{ if } \textit{Switch1}, \\ \textit{Toggle1} \textbf{ causes } \textit{Switch1} \textbf{ if } \neg \textit{Switch1}, \\ \textit{Toggle2} \textbf{ causes } \neg \textit{Switch2} \textbf{ if } \textit{Switch2}, \\ \textit{Toggle2} \textbf{ causes } \textit{Switch2} \textbf{ if } \neg \textit{Switch2}. \end{array}$$

This domain has 4 models that differ by the initial states of the switches.

It is essential in this example that the fluent *Light* is not included in the frame. Otherwise, the effects of the actions would be indeterminate. For instance, the effect of *Toggle1* would be to change either the values of *Switch1* and *Light* or, surprisingly, the values of *Switch1* and *Switch2*.

Example 3: Ramifications. As discussed in the introduction, this can be illustrated by the domain description consisting of propositions (1) and proposition (2). All fluents are included in the frame. There are two models that differ by the initial value of *Walking*.

Example 4: Nondeterminism. The "Russian Turkey Shoot" [Sandewall, 1992] is the enhancement of the original shooting story in which there is an additional action of spinning the gun's bullet chamber. More precisely, we think of *Spin* as the action of looking to see whether the chamber is empty, inserting a bullet if it is, and then giving it a spin. We extend (1) by the proposition

$$\mathit{Spin}\ \textbf{releases}\ \mathit{Loaded}.$$

This domain has infinitely many models. In fact, the set of its models has the cardinality of the continuum. To select a specific model, we need to specify, for every string of actions that ends with *Spin*, whether or not *Loaded* is true after these actions are executed sequentially from the initial situation.

In each of the following two examples, the set of models has the cardinality of the continuum also.

Example 5: Restricted Nondeterminism. A part of a table is painted white, and a part black. There are two actions: picking up a block and dropping it onto the table; the latter may put the block entirely within the white region, or entirely within the black region, or touching both the white and black regions. This example was suggested to us by Ray Reiter (personal communication, March 11, 1992). It can be described in $\mathcal{AR}_0$ using two frame fluents, *White* and *Black*, which express that the block is, at least partially, on the corresponding part of the table:

$$\begin{array}{l}\mathit{PickUp}\ \textbf{causes}\ \neg \mathit{White} \wedge \neg \mathit{Black},\\ \mathit{Drop}\ \textbf{causes}\ \mathit{White} \vee \mathit{Black},\\ \mathit{Drop}\ \textbf{releases}\ \mathit{White},\\ \mathit{Drop}\ \textbf{releases}\ \mathit{Black}.\end{array}$$

The second proposition shows that the values of fluents after the execution of *Drop*, although not predetermined by their values in the previous situation, are not totally arbitrary either.

Example 6: Implicit Nondeterminism. Each of two light bulbs is controlled by a switch. The action of turning on the light is nondeterministic, because it is not specified which of the switches is used. In the following description, *Switch1* and *Switch2* belong to the frame, and *Light* does not:

$$\begin{array}{l}\textbf{always}\ \mathit{Light} \equiv (\mathit{Switch1} \vee \mathit{Switch2}),\\ \mathit{TurnOn}\ \textbf{causes}\ \mathit{Light},\\ \mathit{TurnOff}\ \textbf{causes}\ \neg \mathit{Light}.\end{array}$$

There are no release propositions here, and the effect propositions do not include disjunctions; the nondeterminism in this domain description is due entirely to the presence of a constraint.

4 Semantics of $\mathcal{AR}_0$

4.1 States and Transition Functions

In this section, we consider truth-valued functions on the set of fluents. Such a function σ can be extended to arbitrary formulae according to the truth tables of propositional logic.

We say that σ is a *state* of a domain D if it maps every constraint in D to T ("true").

For instance, the domain consisting of propositions (1) and (2) has 6 states; they are the truth-valued functions on

$$\{\mathit{Loaded}, \mathit{Alive}, \mathit{Walking}\}$$

that make the formula $\mathit{Walking} \supset \mathit{Alive}$ true.

The semantics of $\mathcal{AR}_0$ shows how the effect propositions and release propositions of D define a nondeterministic transition system with this set of states, whose input symbols are actions. We will describe this transition system by a function *Res* that maps an action and a state to a set of states. The elements of $\mathit{Res}(A, \sigma)$ are, intuitively, the states that differ from σ as dictated by the effect propositions for A, but, at the same time, differ from σ as little as possible. In determining the "difference" between σ and the elements of $\mathit{Res}(A, \sigma)$, we will consider only the frame fluents not released by the release propositions for A.

As a preliminary step, define $\mathit{Res}_0(A, \sigma)$ to be the set of states σ' such that, for each effect proposition A **causes** C **if** P in D, $\sigma'(C) = \mathsf{T}$ whenever $\sigma(P) = \mathsf{T}$. The set $\mathit{Res}(A, \sigma)$ will be defined as the subset of $\mathit{Res}_0(A, \sigma)$ whose elements are "close" to σ.

In order to make this precise, the following notation is needed. The *distance* $\rho(\sigma_1, \sigma_2)$ between states σ_1 and σ_2 is the set of fluents on which the states differ:

$$\rho(\sigma_1, \sigma_2) = \{F \mid \sigma_1(F) \neq \sigma_2(F)\}.$$

Furthermore, for any action A and any state σ, we define $I(A, \sigma)$ to be the set of frame fluents F such that, for every release proposition A **releases** F **if** P in D, $\sigma(P) = \mathsf{F}$. Intuitively, $I(A, \sigma)$ is the set of frame fluents to which inertia should apply when A is executed in state σ. If D includes no release propositions for A, then $I(A, \sigma)$ is the whole frame. For the domain of Example 4, $I(A, \sigma)$ is the frame $\{\mathit{Loaded}, \mathit{Alive}\}$ unless $A = \mathit{Spin}$, in which case $I(A, \sigma) = \{\mathit{Alive}\}$.

Now the transition function *Res* corresponding to D is defined as follows: $\mathit{Res}(A, \sigma)$ is the set of states $\sigma' \in \mathit{Res}_0(A, \sigma)$ for which $\rho(\sigma, \sigma') \cap I(A, \sigma)$ is minimal relative to set inclusion—in other words, for which there is no $\sigma'' \in \mathit{Res}_0(A, \sigma)$ such that $\rho(\sigma, \sigma'') \cap I(A, \sigma)$ is a proper subset of $\rho(\sigma, \sigma') \cap I(A, \sigma)$.

Consider, for instance, the domain of Example 4. We will represent a state σ by the set of fluents F such that $\sigma(F) = \mathsf{T}$. In this notation,

$$\begin{aligned} Res_0(Load, \{Alive\}) &= \{\{Loaded\}, \{Loaded, Alive\}\}, \\ Res(Load, \{Alive\}) &= \{\{Loaded, Alive\}\}; \end{aligned}$$

$Res_0(Spin, \{Alive\})$ is the set of all states, and

$$Res(Spin, \{Alive\}) = \{\{Alive\}, \{Loaded, Alive\}\}.$$

4.2 Models and Entailment

A *structure* is a partial function from strings of actions to states whose domain is nonempty and prefix-closed. If a structure Ψ is defined on a string $\overline{A}$, we say that $\overline{A}$ is *executable* in Ψ. Intuitively, $\Psi(\overline{A})$ represents the state that results from the execution of the members of $\overline{A}$ sequentially from the initial situation.

A value proposition (3) is *true* in a structure Ψ if $\overline{A}$ is executable in Ψ and $\Psi(\overline{A})(C) = \mathsf{T}$.

A structure Ψ is a *model* of a domain D if every value proposition in D is true in Ψ, and, for every string of actions $\overline{A}$ executable in Ψ and every action A,

- if $\overline{A}A$ is executable in Ψ, then
$$\Psi(\overline{A}A) \in Res(A, \Psi(\overline{A})),$$
- otherwise, $Res(A, \Psi(\overline{A})) = \emptyset$.

Consider, for instance, the domain of Example 1. Its states can be represented by subsets of $\{HasCard, DoorOpen\}$. The only action in this domain is $InsertCard$, and its transition function is defined by the equations

$$Res(InsertCard, \sigma) = \begin{cases} \{\sigma \cup \{DoorOpen\}\}, & \\ \qquad \text{if } HasCard \in \sigma, & \\ \emptyset, \qquad \text{otherwise.} & \end{cases}$$

It is easy to see that this domain has two models. In one model, Ψ_1, the only executable string of actions is the empty string ϵ, and $\Psi_1(\epsilon) = \emptyset$. In the other model, Ψ_2, every string of actions is executable, and

$$\Psi_2(\overline{A}) = \begin{cases} \{HasCard\}, & \text{if } \overline{A} = \epsilon, \\ \{HasCard, DoorOpen\}, & \text{otherwise.} \end{cases}$$

We say that a domain description is *consistent* if it has a model. Two domain descriptions are *equivalent* if they have the same models.

A value proposition is *entailed* by a domain description D if it is true in every model of D.

4.3 Restricted Monotonicity

When an effect proposition, a release proposition, or a constraint is added to a domain description, the set of value propositions entailed by the description often changes in a nonmonotonic fashion: Some of the propositions entailed by the original description are not entailed by the extended one. This cannot happen, however, when a value proposition is added. It is clear from the definition of a model given above that adding a value proposition to a domain description can only make the set of its models smaller, and thus make the set of value propositions entailed by it larger. This is what one would intuitively expect, because an additional value proposition in a domain description expresses merely an additional assumption about the "trajectory" under consideration.

The observation made above is a "restricted monotonicity property" in the sense of [Lifschitz, 1993c]. Propositions of all four types play the role of postulates, and value propositions play the role of sentences, assertions and parameters. A similar fact about the language $\mathcal{A}$ is stated in [Lifschitz, 1993c] as Proposition 3.

5 Two Theorems about Domain Descriptions

In this section, we show how two ideas familiar from classical logic—the "replacement property" and the notion of an explicit definition—can be applied to the language $\mathcal{AR}_0$.

5.1 Replacement

Let T be a set of constraints. We say that formulae A and B are *equivalent* with respect to T if the formula $A \equiv B$ is a propositional consequence of T. If, for instance, T includes the constraint from Example 6, then *Light* is equivalent to $Switch1 \vee Switch2$ with respect to T.

Theorem 1. *Let D be a domain description, let T be a subset of the constraints in D, and let A and B be formulae equivalent with respect to T. If a domain description D' is obtained from D by replacing A by B in some or all occurrences in any value propositions, effect propositions, and constraints that do not belong to T, then D and D' are equivalent.*

This theorem shows, for instance, that the domain of Example 6 is equivalent to

$$\begin{array}{l} \textbf{always } Light \equiv (Switch1 \vee Switch2), \\ TurnOn \textbf{ causes } Switch1 \vee Switch2, \\ TurnOff \textbf{ causes } \neg Switch1 \wedge \neg Switch2 \end{array} \qquad (7)$$

and to

$$\begin{array}{l} \textbf{always } Light \equiv (Switch2 \vee Switch1), \\ TurnOn \textbf{ causes } Light, \\ TurnOff \textbf{ causes } \neg Light \end{array}$$

(take $T = \emptyset$).

Note that Theorem 1 does *not* allow us to replace a frame fluent F in a release proposition (5) by an equivalent one. In fact, such a replacement can produce a non-equivalent domain. This is shown by the following example:

$$\begin{array}{l} \textbf{always } F_1 \equiv F_2, \\ A \textbf{ releases } F_1, \\ A \textbf{ releases } F_2. \end{array}$$

Replacing F_1 by F_2 in the second line would change the set of models.

5.2 Explicit Definitions

A constraint in a domain description D is an *explicit definition* of a fluent F if it has the form

$$\textbf{always } F \equiv C, \tag{8}$$

and F

- does not belong to the frame,
- does not occur in C,
- does not occur in any other propositions in D.

For instance, the constraint in (7) is an explicit definition of *Light*.

Theorem 2. *Let D be a domain description that includes an explicit definition of a fluent F, and let D' be the domain description obtained from D by deleting this definition and deleting F from the language. A value proposition that does not contain F is entailed by D if and only if it is entailed by D'.*

Thus in $\mathcal{AR}_0$, as in classical logic, definitional extensions are conservative.

6 Relation to $\mathcal{A}$

We claimed in Section 3 that the language $\mathcal{A}$ from [Gelfond and Lifschitz, 1993] can be viewed as a subset of $\mathcal{AR}_0$. Now this assertion can be given a precise meaning.

In this section, the terms that have different definitions in the languages $\mathcal{A}$ and $\mathcal{AR}_0$, such as "proposition," "structure," "entailed" and "consistent," will be understood in the sense of the language $\mathcal{A}$, unless stated otherwise.

First, we need to review the syntax and semantics of $\mathcal{A}$. We begin with two nonempty sets of symbols, called *fluents* and *actions*. A *fluent expression* is a fluent possibly preceded by $\neg$. A *value proposition* is of the form

$$F \textbf{ after } \overline{A}, \tag{9}$$

where F is a fluent expression, and $\overline{A}$ is a string of actions. An *effect proposition* is of the form

$$A \textbf{ causes } F \textbf{ if } P_1, \ldots, P_n, \tag{10}$$

where A is an action, and each of $F, P_1, \ldots, P_n$ $(n \geq 0)$ is a fluent expression. This proposition is said to *describe the effect of A on F*, and the fluent expressions $P_1, \ldots, P_n$ are said to be its *preconditions*. A *proposition* is a value proposition or an effect proposition. A *domain description* is a set of propositions.

The semantics of $\mathcal{A}$ can be described as follows. A *state* is a truth-vlaued function on the set of fluents. Given a fluent F and a state σ, we say that F *holds* in σ if $\sigma(F) = \mathsf{T}$; $\neg F$ *holds* in σ if $\sigma(F) = \mathsf{F}$. A *transition function* is a function from pairs (A, σ), where A is an action and σ is a state, to states. A *structure* is a pair (σ_0, Φ), where σ_0 is a state (the *initial state* of the structure), and Φ is a transition function.

For any structure M and any string of actions $A_1 \ldots A_m$, by $M^{A_1 \ldots A_m}$ we denote the state

$$\Phi(A_m, \Phi(A_{m-1}, \ldots, \Phi(A_1, \sigma_0) \ldots)),$$

where Φ and σ_0 are the transition function and the initial state of M. We say that a value proposition (9) is *true* in M if F holds in the state $M^{\overline{A}}$, and that it is *false* otherwise.

A structure (σ_0, Φ) is a *model* of a domain description D if every value proposition in D is true in (σ_0, Φ), and, for every action A, every fluent F, and every state σ, the following two conditions are satisfied:

(i) if D includes an effect proposition describing the effect of A on F $(\neg F)$ whose preconditions hold in σ, then F $(\neg F)$ holds in $\Phi(A, \sigma)$;

(ii) otherwise, F holds in $\Phi(A, \sigma)$ if and only if F holds in σ.

As in the case of $\mathcal{AR}_0$, we say that D is *consistent* if it has a model; a value proposition is said to be *entailed* by D if it is true in every model of D.

For a consistent domain description in the language $\mathcal{A}$, there is a simple one-one correspondence betweens its models as defined here and its models in the sense of Section 4:

Theorem 3. *Let D be a consistent domain description in the language $\mathcal{A}$. If M is a model of D in the sense of $\mathcal{A}$, then the function*

$$\overline{A} \mapsto M^{\overline{A}} \tag{11}$$

is a model of D in the sense of $\mathcal{AR}_0$. Moreover, every model of D in the sense of $\mathcal{AR}_0$ has form (11) for exactly one model M of D.

Corollary. *Let D be a consistent domain description in the language $\mathcal{A}$. A value proposition is entailed by D in the sense of $\mathcal{A}$ if and only if it is entailed by D in the sense of $\mathcal{AR}_0$.*

Without assuming the consistency of D, the assertions of Theorem 3 and of the corollary would be incorrect. For instance, the domain description

$$\begin{array}{l} A \textbf{ causes } F, \\ A \textbf{ causes } \neg F, \end{array}$$

although inconsistent in $\mathcal{A}$, has models in the sense of $\mathcal{AR}_0$ (with ϵ as the the only executable string of actions).

7 Translating from $\mathcal{AR}_0$ into Circumscription

In this section, we use $\mathcal{AR}_0$ to study the range of applicability of the approach to representing actions by circumscriptive theories developed in [Baker, 1991]. Baker's method is applied in [Lifschitz, 1991] to a class of reasoning problems characterized in terms of their syntactic form, and in [Kartha, 1993] to the problems expressible in $\mathcal{A}$. Here we further extend that work by showing how the method can be applied to $\mathcal{AR}_0$.

We restrict attention to *finite* domain descriptions, that is, to domain descriptions with finitely many fluents, actions and propositions.

7.1 Nested Abnormality Theories

The formalism of nested abnormality theories is introduced in [Lifschitz, 1995]. Its use is demonstrated there by recasting several familiar applications of circumscription in its framework—inheritance hierarchies, the domain closure assumption, and the causal minimization approach to the frame problem.

The difference between this formalism and earlier uses of circumscription for formalizing knowledge can be characterized as follows. A "circumscriptive theory" is usually defined by a list of axioms $A_1, \ldots, A_n$ that may contain the abnormality predicate Ab, and by a list of predicate and/or function constants[2] $C_1, \ldots, C_m$ that are "described" by the axioms and thus are allowed to vary in the process of circumscribing Ab [McCarthy, 1986]. A possible syntax for such theories is

$$C_1, \ldots, C_m : A_1, \ldots, A_n. \qquad (12)$$

The circumscription operator allows us to translate (12) into the language of classical logic by forming the circumscription of the abnormality predicate Ab relative to the conjunction of the axioms $A_1 \wedge \ldots \wedge A_n$ with $C_1, \ldots, C_m$ allowed to vary; we denote this circumscription by

$$\mathrm{CIRC}[A_1 \wedge \ldots \wedge A_n; Ab; C_1, \ldots, C_m]. \qquad (13)$$

(See [Lifschitz, 1993a] for the definition of the circumscription operator.) Unfortunately, this is not general enough for the purpose of representing defaults, and "prioritized circumscription" is proposed in [McCarthy, 1986] as a more general representation tool. In nested abnormality theories, we generalize (12) in a different way: each A_i in (12) is allowed to be a "block" of form (12), so that axioms become "nested."

[2] Object constants are viewed as function constants of arity 0.

Intuitively, each block can be viewed as a group of axioms that describes a certain collection of predicates and functions, and the embedding of blocks reflects the dependence of these descriptions on each other.

It is also convenient to replace the predicate constant Ab in (13) by an existentially quantified predicate variable. The abnormality predicate usually plays an auxiliary role in a formalization; what we are actually interested in are the logical consequences of (13) that do not include Ab. To put it differently, if (13) is denoted by $F(Ab)$, and ab is a predicate variable of the same arity as Ab, then what we are interested in are the consequences of the sentence $\exists ab F(ab)$. The effect of this modification in the context of nested abnormality theories is that the abnormality predicate becomes local to the block in which it is used. The syntax of simple abnormality theories is defined in such a way that Ab may even have different arities in different blocks.

We turn now to the formal treatment of the subject. The definitions below are reproduced from [Lifschitz, 1995].

Consider a language L of classical logic which may have variables of several sorts and higher-order variables. We assume that L does *not* include Ab among its symbols. For any list $\tau_1, ..., \tau_k$ of sorts of variables available in L, by $L_{\tau_1 \ldots \tau_k}$ we denote the language obtained from L by adding Ab as a k-ary predicate constant taking arguments of the sorts $\tau_1, ..., \tau_k$. *Blocks* are defined recursively as follows: For any list of sorts $\tau_1, ..., \tau_k$ and any list of function and/or predicate constants $C_1, \ldots, C_m$ $(m \geq 0)$ of L, if each of $A_1, \ldots, A_n$ $(n \geq 0)$ is a formula of $L_{\tau_1 \ldots \tau_k}$ or a *block*, then

$$\{C_1, \ldots, C_m : A_1, \ldots, A_n\}$$

is a *block*. A *nested abnormality theory* is a set of blocks, called its *axioms*.

The semantics of nested abnormality theories is characterized by a map φ that translates blocks into formulae. It is convenient to make φ defined also on formulae of the languages $L_{\tau_1 \ldots \tau_k}$. If A is such a formula, then φA stands for the universal closure of A. For blocks we define, recursively:

$$\varphi\{C_1, \ldots, C_m : A_1, \ldots, A_n\} = \exists ab F(ab),$$

where

$$F(Ab) = \mathrm{CIRC}[\varphi A_1 \wedge \ldots \wedge \varphi A_n; Ab; C_1, \ldots, C_m].$$

Note that, for any block A, φA is a sentence not containing Ab.

A sentence A of L will be identified with the block $\{: A\}$. It is easy to see that $\varphi\{: A\}$ is equivalent to A.

For any nested abnormality theory T, φT stands for $\{\varphi A \mid A \in T\}$. A *model* of T is a model of φT in the sense of classical logic. A *consequence* of T is a sentence of L that is true in all models of T.

7.2 Language

We will transform a finite domain D in the sense of the language $\mathcal{AR}_0$ into a nested abnormality theory whose language L has variables of three sorts: for fluents, actions and situations. These variables will be denoted respectively by $f, f_1, f_2, \ldots$; $a, a_1, a_2, \ldots$; $s, s_1, s_2, \ldots$. In addition, L has predicate variables $\lambda, \lambda_1, \lambda_2, \ldots$ that range over properties of fluents.

The language has the following object constants:

- the fluents of D,
- the actions of D,
- the situation constants S_0 and $\perp$.

It also includes the binary function constant *Result*, which takes an action and a situation as arguments, and produces a situation.

The constant $\perp$ is needed because we want to talk about actions that may not be executable, such as *InsertCard* in Example 1. Intuitively, $\perp$ represents the value "undefined", so that the assertion that a is not executable in situation s will be represented by

$$Result(a, s) = \perp .$$

Finally, L has two predicate constants:

- the unary predicate *FrameFluent*, whose argument is a fluent,
- the binary predicate *Holds*, whose two arguments are a fluent and a situation.

Note that "formulae" as defined in Section 2.1 are not among the formulae of L. To avoid confusion, we will refer to formulae in the sense of Section 2.1 as "domain formulae." For any domain formula C, by $T(C, s)$ we will denote the formula of L obtained from C as the result of replacing each fluent name F by $Holds(F, s)$. For instance, $T(\neg Alive \wedge Loaded, s)$ stands for

$$\neg Holds(Alive, s) \wedge Holds(Loaded, s).$$

For any string of actions $A_1 \ldots A_m$, by $[A_1 \ldots A_m]$ we will denote the ground term

$$Result(A_m, Result(A_{m-1}, \ldots, Result(A_1, S_0), \ldots)).$$

7.3 Law of Inertia and Existence of Situations

Our translation β from $\mathcal{AR}_0$ into nested abnormality theories uses two defaults that play an important role in applications of default reasoning to the frame problem.

According to the *commonsense law of inertia*, the value of a frame fluent normally remains unchanged after performing an action. Formulas describing how actions change the world represent exceptions to this default. The commonsense law of inertia is expressed by the formula

$$FrameFluent(f) \wedge Result(a, s) \neq \perp \wedge \neg Ab(f, a, s) \supset Holds(f, Result(a, s)) \equiv Holds(f, s).$$

This formula will be denoted by LI.

According to the *existence of situations principle*, for any assignment of values to fluents, there normally exists a situation in which every fluent takes the assigned value. Recognizing the role of this principle was a major contribution of [Baker, 1991]. Constraints represent exceptions to this default. For instance, the constraint

$$Holds(Walking, s) \supset Holds(Alive, s) \qquad (14)$$

shows that there is no situation in which *Walking* is true and *Alive* is false. The existence of situations principle is expressed by the formula

$$\neg Ab(\lambda) \supset \exists s[s \neq \perp \wedge \forall f(Holds(f, s) \equiv \lambda(f))],$$

This formula will be denoted by ES.

As defined below, the nested abnormality theory βD representing a finite domain D consists of several formulas, including LI and ES, arranged into a system of blocks. We should not worry about by the fact that Ab has three arguments in LI and only one argument in ES: these formulas will appear in two different blocks.

7.4 Translating Propositions

We will define how to construct, for each proposition P in D, the corresponding formula βP. These formulas will be included in the translation βD of D.

If P is a value proposition (3), then βP is

$$[\overline{A}] \neq \perp \wedge T(C, [\overline{A}]).$$

The first term expresses that the sequence of actions $\overline{A}$ is executable. For instance, the proposition

Heads **after** *Toss*

is translated as

$$Result(Toss, S_0) \neq \perp \wedge Holds(Heads, Result(Toss, S_0)).$$

If P is an effect proposition (4), then βP is

$$T(P, s) \wedge Result(A, s) \neq \perp \supset T(C, Result(A, s)).$$

For instance, the proposition

Shoot **causes** $\neg$*Alive* **if** *Loaded*

is translated as

$$Holds(Loaded, s) \wedge Result(Shoot, s) \neq \perp \supset \neg Holds(Alive, Result(Shoot, s)).$$

If P is a release proposition (5), then βP is

$$T(P, s) \supset Ab(F, A, s).$$

This formula will accompany the commonsense law of inertia LI, so that the law of inertia will be disabled in application to F when A is executed in the presence of the precondition P. For instance, the proposition

$$\mathit{Spin}\ \mathbf{releases}\ \mathit{Loaded}$$

is translated as

$$Ab(\mathit{Loaded}, \mathit{Spin}, s)$$

(if we disregard the trivial antecedent $T(\mathit{True}, s)$, which equals True).

Finally, if P is a constraint (6), then βP is $T(C, s)$. For instance, (2) is translated as (14).

7.5 Definition of βD

Now we are ready to define the representation βD of a finite domain D as a nested abnormality theory.

By $\mathbf{F}$ we will denote the set of fluents in the language of D, by $\mathbf{Fr}$ the set of frame fluents, by $\mathbf{A}$ the set of actions. Furthermore, let D_v, D_e, D_r and D_c be the parts of D consisting of its value propositions, effect propositions, release propositions and constraints, so that

$$D = D_v \cup D_e \cup D_r \cup D_c.$$

The axioms of βD are as follows.

Group 1. Unique names axioms:

$$F_1 \neq F_2$$

for all pairs of distinct fluents $F_1, F_2 \in \mathbf{F}$,

$$A_1 \neq A_2$$

for all pairs of distinct actions $A_1, A_2 \in \mathbf{A}$, and

$$S_0 \neq \perp .$$

Group 2. Domain closure axioms:

$$\bigvee_{F \in \mathbf{F}} f = F,$$

$$\bigvee_{A \in \mathbf{A}} a = A.$$

Group 3. The universe of situations is assumed to be sufficiently large:

$$\exists s_1, \ldots, s_N \bigwedge_{1 \leq i < j \leq N} s_i \neq s_j,$$

where $N = 2^{|\mathbf{F}|} + 1$.

Group 4. Translations of the value propositions:

$$\beta P \qquad (P \in D_v).$$

Group 5. Characterization of $\mathit{FrameFluent}$:

$$
\begin{array}{l}
\{\mathit{FrameFluent} : \\
\quad \mathit{FrameFluent}(f) \supset Ab(f), \\
\quad \mathit{FrameFluent}(F) \qquad (F \in \mathbf{Fr}) \\
\}.
\end{array}
$$

By enclosing the formulae $\mathit{FrameFluent}(F)$ for all F in $\mathbf{Fr}$ in a block along with

$$\mathit{FrameFluent}(f) \supset Ab(f),$$

we express the "closed world assumption" for $\mathit{FrameFluent}$: a fluent satisfies this predicate *only* if it represented by a constant from $\mathbf{Fr}$.

Group 6. Characterization of Holds and Result:

$$
\begin{array}{l}
\{\mathit{Result} : \\
\quad LI, \\
\quad \beta P \qquad (P \in D_r), \\
\quad \{\mathit{Result} : \\
\quad\quad \mathit{Result}(a, s) = \perp \supset Ab(a, s), \\
\quad\quad \mathit{Result}(a, \perp) = \perp, \\
\quad\quad \beta P \qquad (P \in D_e), \\
\quad\quad \{\mathit{Holds} : \\
\quad\quad\quad ES, \\
\quad\quad\quad \beta P \qquad (P \in D_c) \\
\quad\quad \} \\
\quad \} \\
\}.
\end{array}
$$

The innermost block represents the basic assumption about the universe of situations: for any assignment of truth values to fluents, there exists a situation in which every fluent takes the assigned value, unless such a situation is declared impossible by one of the constraints in D. In the intermediate block, the formula

$$\mathit{Result}(a, s) = \perp \supset Ab(a, s)$$

expresses that, normally, an action is executable. Thus the intermediate block tells us that an action can be executed unless this is prohibited by the propositions in D. The outermost block says that the values of fluents do not change unless this is required by the propositions in D.

The nesting of blocks reflects our intention to decide first what kinds of situations exist, then which actions can be executed, and then what the effects of actions are.

7.6 Soundness and Completeness

Consider a finite domain D. Let Ψ be a model of D, and let M be a model of βD. We say that M is *similar* to Ψ if, for every value proposition P, P is true in Ψ if and only if M satisfies βP.

Theorem 4. *Let D be a finite domain. For any model Ψ of D, there exists a model M of βD similar to Ψ.*

For any model M of βD, there exists a model Ψ of D such that M is similar to Ψ.

The following corollary expresses the soundness and completeness of the translation.

Corollary. *For any finite domain D and any value proposition P, βD entails βP if and only if D entails P.*

8 Conclusion

In addition to the expressive possibilities of $\mathcal{A}$, the language $\mathcal{AR}_0$ allows us to represent actions that may be impossible to execute, fluents that are exempt from inertia, actions with indirect effects and simple nondeterministic actions. It has properties similar to the replacement theorem and to the theorem on the conservativeness of definitional extensions familiar from classical logic. There is a sound and complete translation from $\mathcal{AR}_0$ into the language of nested abnormality theories.

The treatment of value propositions in the semantics of $\mathcal{AR}_0$ is analogous to the "filtering" method advocated by Sandewall [1989], [1992]. Indeed, the value propositions included in D do not affect the construction of the corresponding transition function Res; they are taken into account only when we define which of the "trajectories" described by this transition function are counted as the models of D. Similarly, in the nested abnormality theory βD, the translations of the value propositions form a separate group of axioms that is not placed in the range of any circumscriptions.

The use of nested abnormality theories, instead of conjunctions of circumscriptions as in [Baker, 1991] and [Lifschitz, 1991], leads to a simpler and more natural presentation of Baker's method. Moreover, the use of conjunctions of circumscriptions along the lines of that earlier work does not seem to lead to satisfactory results when actions can be nondeterministic. This will be discussed in detail elsewhere.

Acknowledgements

We are grateful to Michael Gelfond, Enrico Giunchiglia, Fangzhen Lin, Norman McCain and Hudson Turner for comments on drafts of this paper. This work was partially supported by National Science Foundation under grants IRI-9101078 and IRI-9306751.

References

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

[Baral and Gelfond, 1993] Chitta Baral and Michael Gelfond. Representing concurrent actions in extended logic programming. In *Proc. of IJCAI-93*, pages 866–871, 1993.

[Boyer and Moore, 1988] Robert Boyer and J Strother Moore. *A computational logic handbook.* Academic Press, 1988.

[Chen and Warren, 1992] Weidong Chen and David S. Warren. A goal-oriented approach to computing well-founded semantics. In Krzysztof Apt, editor, *Proc. Joint Int'l Conf. and Symp. on Logic Programming*, pages 589–603, 1992.

[Denecker and De Schreye, 1993] Mark Denecker and Danny De Schreye. Representing incomplete knowledge in abductive logic programming. In Dale Miller, editor, *Logic Programming: Proceedings of the 1993 Int'l Symposium*, pages 147–163, 1993.

[Dung, 1993] Phan Minh Dung. Representing actions in logic programming and its applications in database updates. In *Logic Programming: Proceedings of the Tenth Int'l Conf. on Logic Programming*, pages 222–238, 1993.

[Gelfond and Lifschitz, 1993] Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.

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

[Lifschitz *et al.*, 1993] Vladimir Lifschitz, Norman McCain, and Hudson Turner. Automated reasoning about action: A logic programming approach. In Dale Miller, editor, *Proc. of ILPS-93*, page 641, 1993.

[Lifschitz, 1990] Vladimir Lifschitz. Frames in the space of situations. *Artificial Intelligence*, 46:365–376, 1990.

[Lifschitz, 1991] Vladimir Lifschitz. Towards a metatheory of action. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proc. of the Second Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 376–386, 1991.

[Lifschitz, 1993a] Vladimir Lifschitz. Circumscription. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *The Handbook of Logic in AI and Logic Programming*, volume 3, pages 298–352. Oxford University Press, 1993.

[Lifschitz, 1993b] Vladimir Lifschitz. A language for describing actions. In *Working Papers of the Second Symposium on Logical Formalizations of Commonsense Reasoning*, 1993.

[Lifschitz, 1993c] Vladimir Lifschitz. Restricted monotonicity. In *Proc. AAAI-93*, pages 432–437, 1993.

[Lifschitz, 1995] Vladimir Lifschitz. Nested abnormality theories. *Artificial Intelligence*, 74:351–365, 1995.

[Lin and Shoham, 1991] Fangzhen Lin and Yoav Shoham. Provably correct theories of action (preliminary report). In *Proc. AAAI-91*, pages 349–354, 1991.

[McCarthy, 1986] John McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 26(3):89–116, 1986. Reproduced in [McCarthy, 1990].

[McCarthy, 1990] John McCarthy. *Formalizing common sense: papers by John McCarthy.* Ablex, Norwood, NJ, 1990.

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

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

[Sandewall, 1989] Erik Sandewall. Filter preferential entailment for the logic of action in almost continuous worlds. In *Proc. of IJCAI-89*, pages 894–899, 1989.

[Sandewall, 1992] Erik Sandewall. Features and fluents: A systematic approach to the representation of knowledge about dynamical systems. Technical Report LiTH-IDA-R-92-30, Linköping University, 1992.

[Subramanian, 1993] Sakthi Subramanian. *A Mechanized Framework for Specifying Problem Domains and Verifying Plans.* PhD thesis, University of Texas, Austin, Department of Computer Science, 1993.

[Thielscher, 1993] Michael Thielscher. An analysis of systematic approaches to reasoning about actions and change. Manuscript, 1993.

[Thielscher, 1994] Michael Thielscher. Representing actions in equational logic programming. In *Proc. ICLP-94*, 1994. To appear.