In: P. Van Hentenryck, ed., Proc. of the Int. Conf. on Log. Prog., 207–224, Santa Margherita Ligure, Italy, 1994.

Representing Actions in Equational Logic Programming

Michael Thielscher
Intellektik, Informatik, TH Darmstadt
Alexanderstraße 10, D–64283 Darmstadt, Germany
E-mail: mit@intellektik.informatik.th-darmstadt.de

Abstract

A sound and complete approach for encoding the action description language $\mathcal{A}$ developed by M. Gelfond and V. Lifschitz in an equational logic program is given. Our results allow the comparison of the resource-oriented equational logic based approach and various other methods designed for reasoning about actions, most of them based on variants of the situation calculus, which were also related to the action description language recently.

A non-trivial extension of $\mathcal{A}$ is proposed which allows to handle uncertainty in form of non-deterministic action descriptions, i.e. where actions may have alternative randomized effects. It is described how the equational logic programming approach forms a sound and complete encoding of this extended action description language $\mathcal{A}_{ND}$ as well.

1 Introduction

Understanding and modelling the ability of humans to reason about actions, change, and causality is one of the key issues in Artificial Intelligence and Cognitive Science. Since logic appears to play a fundamental rôle for intelligent behavior, many deductive methods for reasoning about change were developed and thoroughly investigated. It became apparent that a straightforward use of classical logic lacks the essential property that facts describing a world state may change in the course of time. To overcome this problem, the truth value of a particular fact has to be associated with a particular state. This solution brings along the famous technical frame problem which captures the difficulty of expressing that the truth values of facts not affected by some action are not changed by the execution of this action.

Many deductive methods for reasoning about change are based on the ideas underlying the situation calculus. These approaches require one or more axioms accounting for the frame problem, e.g. a set of frame axioms as in [26, 11, 20], successor state axioms as in [27], a nonmonotonic law of inertia as in [25] and [10], or persistence assumptions as in [21]. Recently, three resource-oriented deductive methods were proposed which do not require additional frame axioms, namely the linear connection method [3], the use of a certain fragment of linear logic [24], and an approach based on equational logic programs [15]. The three approaches treat logic formulas as resources which can be produced and consumed through the execution

of actions [14]. It has been proved that they are equivalent for conjunctive planning problems, i.e. problems where situations as well as conditions and effects of actions are conjunctions of atomic facts [12, 29].

In spite of the equivalence result for the resource-oriented approaches, the expressiveness of the equational logic approach, say, appears not to be satisfactorily clarified as there is a much deeper rift between this approach and methods based on the situation calculus. Recently, M. Gelfond and V. Lifschitz initiated a comparison of a variety of deductive frameworks for reasoning about change wrt a semantical approach based on the action description language $\mathcal{A}$ [10]. This language overcomes some restrictions of former work in so far as it supports reasoning about the past as well as handling partial information about situations. Furthermore, [10] presents a sound encoding of $\mathcal{A}$ in terms of extended logic programs with two kinds of negation. In [8], a logic program including a single law of inertia is used to encode $\mathcal{A}$. It is shown that, by using a partial completion semantics, the program is sound and complete wrt the semantics of $\mathcal{A}$. Independently, a similar sound and complete abductive logic program is constructed in [7]. [19] showed soundness and completeness as regards $\mathcal{A}$ for, among others, a circumscription based method [2] and R. Reiter's formalism [27].

Recently, the expressiveness of the resource-oriented equational logic approach was substantially enhanced by introducing the concept of specificity which allows to handle several descriptions of one and the same action, depending on the particular situation in which this action is performed [16, 17]. In this paper, we show how this extended approach can be used as a sound and complete method for encoding $\mathcal{A}$. This result bridges the gap between the resource-oriented approaches and all the methods enumerated in the previous paragraph as it rigorously shows the equivalence with regard to the problem class determined by $\mathcal{A}$.

A broad spectrum of characteristic examples for reasoning about change is presented in [28]. Many of them are covered by $\mathcal{A}$. However, some more complicated domains require indeterminism in form of randomized effects of actions. As a second major contribution of this paper, we extend the action description language $\mathcal{A}$ such that this kind of reasoning can be performed. We illustrate how the equational logic approach can be used to encode it as well. The extended action description language $\mathcal{A}_{ND}$ can be considered as a basis for investigating if and how the various methods which were already related to $\mathcal{A}$ can cope with problems of indeterminism and uncertainty.

A recapitulation of the equational logic programming based approach augmented by the notion of specificity is given in the following Section 2. The action description language $\mathcal{A}$ is briefly described in Section 3, and a formalization in terms of equational logic programming is presented in Section 4. The proofs of the various results are omitted due to lack of space. They can be found in a detailed technical report [31]. In Section 5 we extend the action description language to express uncertainty regarding the effects of actions, and in Section 6 we briefly illustrate how this extended

version $\mathcal{A}_{ND}$ can be modelled using the particular equational logic program developed in Section 2. Finally, in Section 7 we shall discuss some merits of our approach and outline conceivable future extensions as regards $\mathcal{A}$.

2 The Equational Logic Programming Approach

Throughout this paper we illustrate the various basic notions and results with the help of the Yale Shooting scenario [13] and some of its variants such as the Stanford Murder Mystery [1] or the Russian Turkey domain [28].

The most significant feature of the equational logic based approach is that a complete situation is represented by a single term using a binary function symbol $\circ$ which connects the various atomic facts, called *resources*, which hold in this situation. For example, the term [1]

$$loaded \circ dead\,. \tag{1}$$

describes a situation where the gun is loaded and the turkey is already shot dead. Intuitively, the order of the various subterms describing a situation should be irrelevant. To this end, the function $\circ$ is required to be associative, i.e. $X \circ (Y \circ Z) = (X \circ Y) \circ Z$, commutative, i.e. $X \circ Y = Y \circ X$, and to have a unit element $\emptyset$ which denotes the situation where nothing is known, i.e. $X \circ \emptyset = X$. These three axioms (AC1) essentially define the data structure multiset, i.e. the so-called *AC1-term* (1) can be adequately interpreted as the multiset $\mathcal{S} = \{\!|loaded, dead|\!\}$. [2] Based on this concept, actions are defined by a multiset of conditions along with a multiset of effects. For instance, the *shoot* action can be specified by the triple

$$\langle\ \{\!|loaded|\!\}\ ,\ shoot\ ,\ \{\!|unloaded|\!\}\ \rangle \tag{2}$$

where $\{\!|loaded|\!\}$ denotes the multiset of conditions, *shoot* is the name of the action, and $\{\!|unloaded|\!\}$ denotes the multiset of effects. Such a description $\alpha = \langle \mathcal{C}, a, \mathcal{E} \rangle$ of an action a is applicable in a situation $\mathcal{S}$ iff $\mathcal{C} \dot{\subseteq} \mathcal{S}$. If an action description is applicable then it is executed by removing the conditions from the actual situation and adding the effects afterwards, i.e. applying the action description α to a situation $\mathcal{S}$ yields the new situation $(\mathcal{S} \dot{-} \mathcal{C}) \dot{\cup} \mathcal{E}$. [3] In our example, (2) is applicable in the situation represented by (1) since $\{\!|loaded|\!\} \dot{\subseteq} \{\!|loaded, dead|\!\}$. The resulting situation is $(\ \{\!|loaded, dead|\!\} \dot{-} \{\!|loaded|\!\}\ ) \dot{\cup} \{\!|unloaded|\!\} = \{\!|unloaded, dead|\!\}$. No additional axioms for solving the technical frame problem are needed since each resource which is not affected by an action — here the resource *dead* — is available after having applied the two multiset operations.

In [16] it is argued that there is often more than simply a single specification such as (2) of one action because the effects of a particular action may vary from situation to situation. For example, (2) is formally applicable also in case of $\{\!|loaded, alive|\!\}$. Its application yields $\{\!|unloaded, alive|\!\}$ which is undesired since shooting with a previously loaded gun should cause *alive* to

be replaced by *dead*. (It possibly could be that turkeys do not fully agree with this view.) The notion of *specificity* accounts for this problem. First, another action description for *shoot* is introduced, viz.

$$\langle\ \{\!|\mathit{loaded}, \mathit{alive}|\!\}\ ,\ \mathit{shoot}\ ,\ \{\!|\mathit{unloaded}, \mathit{dead}|\!\}\ \rangle. \tag{3}$$

Now, (3) is applicable whenever the original action description (2) is applicable but not vice versa. Hence, (3) is said to contain a *more specific* information and, thus, should be preferred whenever it can be used in a particular situation. To this end, we formally define a partial ordering on a set of action descriptions with regard to specificity as follows. An action description $\alpha_1 = \langle \mathcal{C}_1, a_1, \mathcal{E}_1 \rangle$ is said to be *more specific* than an action description $\alpha_2 = \langle \mathcal{C}_2, a_2, \mathcal{E}_2 \rangle$ iff $a_1 = a_2$ and $\mathcal{C}_1 \dot{\supset} \mathcal{C}_2$. Taking into consideration this definition, an action description should be applied only if there is no more specific description which is also applicable. If an action description can be applied to a situation $\mathcal{S}$ on this condition then we say that it is *most specific wrt* $\mathcal{S}$. Note that this requirement does not rule out the existence of more than one most specific action description. In Section 6 we illustrate how multiple most specific descriptions capture the problem of indeterminism.

A second useful application of specificity is to avoid inconsistencies via the so-called *completion* mechanism. For instance, one of the action descriptions of *load* should be

$$\langle\ \{\!|\ |\!\}\ ,\ \mathit{load}\ ,\ \{\!|\mathit{loaded}|\!\}\ \rangle\,. \tag{4}$$

This works fine in situations like $\{\!|\mathit{alive}|\!\}$ or $\{\!|\mathit{dead}|\!\}$, i.e. if nothing is known about the state of the gun. However, the application of (4) to $\{\!|\mathit{unloaded}|\!\}$, say, yields the unintended situation $\{\!|\mathit{unloaded}, \mathit{loaded}|\!\}$. This can be avoided by *completing* (4) to obtain the additional action description $\langle \{\!|\mathit{unloaded}|\!\}, \mathit{load}, \{\!|\mathit{loaded}|\!\} \rangle$ which is more specific than (4). The Yale Shooting scenario is discussed more thoroughly in Section 4.

The reasoning process described so far can be encoded using the equational logic program depicted in Figure 1 (see [16, 17]): The various action descriptions of a domain are encoded using a ternary predicate *action*. The ternary predicate $\mathit{causes}(i, [a_1, \ldots, a_n], g)$ expresses the fact that the application of the sequence $[a_1, \ldots, a_n]$ of actions to the initial situation i yields the goal situation g. If $n \geq 1$ then an action description α of a_1 with conditions c is selected such that c is contained in i — i.e. $\exists V.\, c \circ V =_{\mathrm{AC1}} i$ —, α is most specific wrt i, and the recursive call uses the resulting situation after applying α to i, along with the sequence $p = [a_2, \ldots, a_n]$. Finally, the notion of specificity is encoded using the predicate $\mathit{non\_specific}(a, c, i)$ which expresses the fact that there is a description of action a which is more specific wrt the situation i than the particular description with conditions c. Following the definition of specificity, such a more specific action description must have conditions c' such that c' is applicable in i and the multiset corresponding to c' is a strict superset of the multiset corresponding to c. In terms of AC1-representation this is true iff $\exists V.\ c' \circ V =_{\mathrm{AC1}} i$ and $\exists W \neq_{\mathrm{AC1}} \emptyset.\ c \circ W =_{\mathrm{AC1}} c'$ hold.

$$
\begin{array}{l}
\mathit{action}(c_1, a_1, e_1). \\
\quad\vdots \\
\mathit{action}(c_m, a_m, e_m). \\
\\
\mathit{causes}(I, [\,], I). \\
\mathit{causes}(I, [A|P], G) \;\leftarrow\; \mathit{action}(C, A, E),\;\; C \circ V =_{\mathrm{AC1}} I, \\
\qquad\qquad\qquad\qquad\quad \neg \mathit{non\_specific}(A, C, I),\;\; \mathit{causes}(V \circ E, P, G). \\
\\
\mathit{non\_specific}(A, C, I) \;\leftarrow\; \mathit{action}(C', A, E'),\;\; C' \circ V =_{\mathrm{AC1}} I, \\
\qquad\qquad\qquad\qquad\quad\;\; C \circ W =_{\mathrm{AC1}} C',\;\; W \neq_{\mathrm{AC1}} \emptyset.
\end{array}
$$

Figure 1: The equational logic program P_A used to reason about change, where $A = \{\langle \mathcal{C}_i, a_i, \mathcal{E}_i \rangle \,|\, 1 \leq i \leq m\}$ is a finite set of action descriptions and c_i (resp. e_i) are AC1-terms representing $\mathcal{C}_i$ (resp. $\mathcal{E}_i$). The binary predicate $=_{\mathrm{AC1}}$ denotes equality modulo our equational theory (AC1).

Our equational program includes negative literals. Therefore, the adequate computation mechanism is *SLDENF-resolution*, i.e. SLD-resolution augmented by negation-as-failure along with an extended unification procedure which unifies wrt a concrete equational theory [30, 17, 32]. Moreover, we sometimes employ *constructive negation* [5] to avoid the problem of floundering. Semantics is defined by K. Clark's *completion* [6] along with a so-called *unification complete* theory $\mathrm{AC1}^*$ which allows to derive inequality of two terms whenever they are not unifiable [18, 30, 17]. Let $(P_A^*, \mathrm{AC1}^*)$ denote the completion of our program depicted in Figure 1.

In [17] it is argued that we can restrict ourselves to models of $(P_A^*, \mathrm{AC1}^*)$ where terms which are built up from the AC1-function $\circ$ are interpreted as multisets. Let $\mathcal{I}$ denote such an interpretation. In [17] it is shown that $(P_A^*, \mathrm{AC1}^*)$ models actions, change, and specificity as intended:

Theorem 2.1 *Let A be a finite set of action descriptions and P_A be as in Figure 1. Then, $(P_A^*, \mathrm{AC1}^*) \models \mathit{causes}(i, [a_1, \ldots, a_n], g)$ iff there are multisets $\mathcal{S}_0, \ldots, \mathcal{S}_n$ such that $\mathcal{S}_0 \doteq i^{\mathcal{I}}$, $\mathcal{S}_n \doteq g^{\mathcal{I}}$, and $\mathcal{S}_j \doteq (\mathcal{S}_{j-1} \dot{-} \mathcal{C}_j) \dot{\cup} \mathcal{E}_j$, where $\langle \mathcal{C}_j, a_j, \mathcal{E}_j \rangle \in A$ is most specific wrt $\mathcal{S}_{j-1}$ ($1 \leq j \leq n$).*

3 The Action Description Language $\mathcal{A}$

In this section we give a brief introduction to the ideas underlying the semantical approach described in [10] which is based on an action description language called $\mathcal{A}$. We use the Stanford Murder Mystery domain [1] to illustrate the expressiveness of $\mathcal{A}$ regarding reasoning about the past and handling incomplete information about the initial situation. This example describes the reasoning process which has to be performed to conclude that

the gun must have been loaded if the turkey was alive at the beginning and is observed to be dead after shooting and waiting.

The basic elements of $\mathcal{A}$ are action names, e.g. *load*, *wait*, and *shoot*, and fluent names, e.g. *loaded* and *alive*, along with expressions like

$$\begin{array}{lllll} \mathit{load} & \texttt{causes} & \mathit{loaded} & & \\ \mathit{shoot} & \texttt{causes} & \neg\mathit{alive} & \texttt{if} & \mathit{loaded} \\ \mathit{shoot} & \texttt{causes} & \neg\mathit{loaded} & & \end{array} \tag{5}$$

These expressions describe the effect of a particular action, e.g. *shoot*, on a single fluent, e.g. *alive*, provided a number of conditions, e.g. *loaded*, hold. In general, these so-called *e-propositions* (i.e. *effect* propositions) are of the form a `causes` e `if` $c_1, \ldots, c_n$, where a is an action name, e is a fluent name occurring either affirmatively or negatively, and $c_1, \ldots, c_n$ are affirmative or negated fluent names ($n \geq 0$). The set of e-propositions describing a domain is used to define a *transition function* which maps states into states given a particular action name. A *state* σ is a set of fluent names, and a positive fluent f (resp. a negative fluent $\neg f$) is said to *hold* in σ iff $f \in \sigma$ (resp. $f \notin \sigma$). For instance, $\{\mathit{alive}\}$ describes the state where *alive* and $\neg\mathit{loaded}$ hold, and the transition function Φ determined by (5) is

$$\begin{array}{rcl} \Phi(\mathit{load}, \sigma) & = & \sigma \cup \{\mathit{loaded}\} \\ \Phi(\mathit{shoot}, \sigma) & = & \begin{cases} \sigma - \{\mathit{loaded}, \mathit{alive}\}, & \text{if } \mathit{loaded} \in \sigma \\ \sigma, & \text{otherwise.} \end{cases} \\ \Phi(\mathit{wait}, \sigma) & = & \sigma \end{array} \tag{6}$$

In general, Φ is designed such that (a) if there is an e-proposition describing the effects of a on a positive fluent f (resp. a negative fluent $\neg f$) whose conditions hold in σ then $f \in \Phi(a, \sigma)$ (resp. $f \notin \Phi(a, \sigma)$) and (b) if there is no such e-proposition then $f \in \Phi(a, \sigma)$ iff $f \in \sigma$. It is noteworthy that given a set of e-propositions either the corresponding transition function is definitely determined or there is no transition function satisfying (a) and (b). The reader is invited to verify that the transition function (6) satisfies these conditions as regards the e-propositions (5).

Apart from defining the effects of actions there is a possibility to describe the value of a single fluent in a particular state using so-called *v-propositions* (i.e. *value* propositions). For example,

$$\begin{array}{ll} \texttt{initially} & \mathit{alive} \\ \neg\mathit{alive} & \texttt{after}\ [\mathit{shoot}, \mathit{wait}] \end{array} \tag{7}$$

describe the facts that the turkey is alive in the initial state and is dead after executing the sequence of actions $[\mathit{shoot}, \mathit{wait}]$, respectively, which precisely corresponds to the Stanford Murder Mystery scenario. In general, a v-proposition is of the form f `after` $[a_1, \ldots, a_n]$ where $a_1, \ldots, a_n$ are action names, and if $n = 0$ then the expression `initially` f is used instead.

A set of e-propositions and v-propositions is called a *domain*. A *structure* consists of an *initial* state σ_0 and a transition function Φ. A structure

(σ_0, Φ) is a *model* of a domain $\mathcal{D}$ iff Φ is determined by the e-propositions in $\mathcal{D}$, and Φ together with σ_0 satisfy the v-propositions occurring in $\mathcal{D}$: A v-proposition `initially` f is satisfied in (σ_0, Φ) iff f holds in σ_0, and a v-proposition f `after` $[a_1, \ldots, a_n]$ is satisfied iff f holds in $\Phi([a_1, \ldots, a_n], \sigma_0)$, which abbreviates $\Phi(a_n, \Phi(a_{n-1}, \ldots, \Phi(a_1, \sigma_0)\ldots))$. A domain is called *consistent* if it admits at least one model.

In our example, Φ is defined as in (6) and — due to (7) — (σ_0, Φ) is a model of the Stanford Murder Mystery iff $\mathit{alive} \in \sigma_0$ and $\mathit{alive} \notin \Phi([\mathit{shoot}, \mathit{wait}], \sigma_0)$. It is easy to verify that $\mathit{loaded} \in \sigma_0$ holds in each such model due to (6), i.e. we are allowed to conclude that the gun was necessarily loaded in the initial state. In other words, the v-proposition `initially` *loaded* is said to be *entailed* by the domain description $\mathcal{D}$ given by (5) and (7); this is written $\mathcal{D} \models$ `initially` *loaded*.

4 Encoding $\mathcal{A}$

The first step towards the formalization of a particular domain via the equational logic approach consists in the definition of an underlying set of resources. As the equational logic approach does not support explicit negation, we need two different resources for each fluent name of a domain description in $\mathcal{A}$. Thus, in case of the Yale Shooting scenario we deal with the set $\{\mathit{loaded}, \overline{\mathit{loaded}}, \mathit{alive}, \overline{\mathit{alive}}\}$, where the independent resource $\overline{f}$ should be interpreted as the negation of resource f, i.e. $\overline{\mathit{loaded}}$ and $\overline{\mathit{alive}}$ take the rôle of *unloaded* and *dead*, respectively, used in Section 2. A situation $\mathcal{S}$ built up from these resources is said to *correspond* to a state σ (and vice versa) iff for each fluent name f we find that if $f \in \sigma$ then $f \in \mathcal{S}$ and if $f \notin \sigma$ then $\overline{f} \in \mathcal{S}$. For instance, $\{\!|\mathit{alive}, \overline{\mathit{loaded}}|\!\}$ corresponds to $\{\mathit{alive}\}$.

The second step in formalizing a domain consists in fixing consistency criteria. In view of the fact that states in $\mathcal{A}$ are sets, no resource should occur more than once in a situation. Moreover, no resource together with its negation may occur, and each fluent name occurs either affirmatively or negatively in a situation. [4] This is formalized by the clauses depicted in Figure 2.

The third and final step consists in creating a set of action descriptions. Given a set of action names along with a set of e-propositions the following Algorithm 1 automatically generates such a set. For notational simplicity, it is assumed that the e-propositions are given in terms of our method, i.e. whenever a negated fluent name $\neg f$ occurs then it is replaced by the term $\overline{f}$.

Algorithm 1 For each action name a let

$$\left\{ \begin{array}{lllll} a & \texttt{causes} & e_1 & \texttt{if} & F_1 \\ & & \vdots & & \\ a & \texttt{causes} & e_m & \texttt{if} & F_m \end{array} \right\}$$

be the set of all e-propositions wrt action name a, where each F_i is a sequence of resources $f_{i1}, \ldots, f_{in_i}$ $(1 \leq i \leq m)$. Note that this set may be

$inconsistent(f_i \circ f_i \circ V)$.

$inconsistent(\overline{f_i} \circ \overline{f_i} \circ V)$.

$inconsistent(f_i \circ \overline{f_i} \circ V)$.

$inconsistent(S) \leftarrow \neg holds(f_i, S),\ \neg holds(\overline{f_i}, S)$.

$holds(F, F \circ V)$.

Figure 2: The clauses defining inconsistency, where $F = \{f_i \mid 1 \leq i \leq m\}$ is a finite set of fluent names and the first four clauses are generated separately for each member of F.

empty, e.g. in case of a being *wait*. In what follows, a multiset is called consistent iff it contains no multiple occurrences of an element and no f along with its negation $\overline{f}$.

1. For all (possibly empty) consistent combinations

$$\mathcal{C} \ := \ F_i \cup \ldots \cup F_j \tag{8}$$

where $\{i, \ldots, j\}$ is a subset of $\{1, \ldots, m\}$, let $\mathcal{E}$ contain the effects determined by the conditions $\mathcal{C}$, along with those conditions which are not affected by the action, i.e. [5]

$$\mathcal{E} \ := \ \Big\{\!\Big|\ e_k \mid F_k \subseteq \mathcal{C}\ \Big|\!\Big\} \ \cup \ \Big\{\!\Big|\ f \in \mathcal{C} \mid \overline{f} \notin \{\!|e_k \mid F_k \subseteq \mathcal{C}|\!\}\ \Big|\!\Big\}$$

where $\overline{\overline{f}}$ should be interpreted as f. Then, create the action description $\langle \mathcal{C}, a, \mathcal{E} \rangle$. All action descriptions generated in this step are called *pure*.

For instance, as regards the *shoot* action there are two possible combinations of conditions, viz. $\mathcal{C}_1 = \{\!|\,|\!\}$ and $\mathcal{C}_2 = \{\!|loaded|\!\}$ determining the effects $\mathcal{E}_1 = \{\!|\overline{loaded}|\!\}$ and $\mathcal{E}_2 = \{\!|\overline{alive}, \overline{loaded}|\!\}$, respectively (c.f. (5)).

2. Apply the following completion procedure (c.f. the paragraph below equation (4) in Section 2) to each pure action description $\alpha = \langle \mathcal{C}, a, \mathcal{E} \rangle$ of a: Let $\mathcal{F}$ be the set of "free" effects, i.e. those elements of $\mathcal{E}$ which neither occur affirmatively nor negatively in $\mathcal{C}$, along with their negations, i.e.

$$\mathcal{F} := \{\!|e \mid e \in \mathcal{E} \wedge e \notin \mathcal{C} \wedge \overline{e} \notin \mathcal{C}|\!\} \cup \{\!|\,\overline{e} \mid e \in \mathcal{E} \wedge e \notin \mathcal{C} \wedge \overline{e} \notin \mathcal{C}|\!\}$$

Then, for each consistent non-empty subset $F \subseteq \mathcal{F}$ create the *completion* $\langle \mathcal{C} \cup F, a, \mathcal{E} \rangle$ of $\alpha = \langle \mathcal{C}, a, \mathcal{E} \rangle$ — provided there is no pure action description generated in Step 1 of the same action and with conditions $\tilde{\mathcal{C}}$ such that $\mathcal{C} \cup F \supseteq \tilde{\mathcal{C}} \supset \mathcal{C}$ (this will be clarified below).

For example, for the pure description $\langle \{\!|loaded|\!\}, shoot, \{\!|\overline{alive}, \overline{loaded}|\!\} \rangle$ we have $\mathcal{F} = \{\!|\overline{alive}, alive|\!\}$ which has two consistent non-empty subsets. Hence, we obtain the completions $\langle \{\!|loaded, \overline{alive}|\!\}, shoot, \{\!|\overline{alive}, \overline{loaded}|\!\} \rangle$ and $\langle \{\!|loaded, alive|\!\}, shoot, \{\!|\overline{alive}, \overline{loaded}|\!\} \rangle$, respectively. ∎

Given a domain description $\mathcal{D}$ we refer to the set of action descriptions generated via Algorithm 1 by $A_{\mathcal{D}}$. For instance, applying this algorithm to the Yale Shooting domain which consists of the action names *load*, *shoot*, and *wait* along with the e-propositions (5) yields four pure action descriptions, viz.

$$\begin{array}{ll} \langle \{\!|\,|\!\}, \mathit{wait}, \{\!|\,|\!\}\rangle & \langle \{\!|\,|\!\}, \mathit{shoot}, \{\!|\overline{\mathit{loaded}}|\!\}\rangle \\ \langle \{\!|\,|\!\}, \mathit{load}, \{\!|\mathit{loaded}|\!\}\rangle & \langle \{\!|\mathit{loaded}|\!\}, \mathit{shoot}, \{\!|\overline{\mathit{alive}}, \overline{\mathit{loaded}}|\!\}\rangle \end{array} \tag{9}$$

Step 2 then yields the following completions:

$$\begin{array}{ll} \langle \{\!|\mathit{loaded}|\!\}, \mathit{load}, \{\!|\mathit{loaded}|\!\}\rangle & \langle \{\!|\overline{\mathit{loaded}}|\!\}, \mathit{shoot}, \{\!|\overline{\mathit{loaded}}|\!\}\rangle \\ \langle \{\!|\overline{\mathit{loaded}}|\!\}, \mathit{load}, \{\!|\mathit{loaded}|\!\}\rangle & \langle \{\!|\mathit{loaded}, \overline{\mathit{alive}}|\!\}, \mathit{shoot}, \{\!|\overline{\mathit{alive}}, \overline{\mathit{loaded}}|\!\}\rangle \\ & \langle \{\!|\mathit{loaded}, \mathit{alive}|\!\}, \mathit{shoot}, \{\!|\overline{\mathit{alive}}, \overline{\mathit{loaded}}|\!\}\rangle \end{array} \tag{10}$$

The restriction at the end of Step 2 prevents us from using the pure action description $\langle \{\!|\,|\!\}, \mathit{shoot}, \{\!|\overline{\mathit{loaded}}|\!\}\rangle$ to create $\langle \{\!|\mathit{loaded}|\!\}, \mathit{shoot}, \{\!|\overline{\mathit{loaded}}|\!\}\rangle$ because a pure description of *shoot* with conditions $\tilde{\mathcal{C}} = \{\!|\mathit{loaded}|\!\}$ is already included in (9). This is in fact desired since after shooting we definitely know that the turkey is dead whenever the gun was loaded before.

Some observations concerning the output of Algorithm 1 are necessary to obtain our first main result: The set $A_{\mathcal{D}}$ does not contain two action descriptions for one and the same action with the same multiset of conditions and different effects. The most important consequence of this observation is that given a situation and an action name there is always a single most specific description of this action wrt this particular situation. (Note that there is always at least one applicable action description since $\mathcal{C}$ in (8) can be empty.) Furthermore, the application of this most specific action description to a consistent situation again yields a consistent situation with regard to the consistency criterion defined in Figure 2, provided the original domain $\mathcal{D}$ is not inconsistent itself. Now, the following equivalence result for the transition function of a domain description and our approach can be proved. For a more thorough discussion we must refer the reader to [31].

Proposition 4.1 *Let $\mathcal{D}$ be a consistent domain description with transition function Φ, σ be a state,* a *an action name, and $\mathcal{S}_\sigma$ the situation corresponding to σ. Then, $(\mathcal{S}_\sigma \dot{-} \mathcal{C}) \dot{\cup} \mathcal{E}$ corresponds to $\Phi(\mathrm{a}, \sigma)$, where $\alpha = \langle \mathcal{C}, \mathrm{a}, \mathcal{E}\rangle \in A_{\mathcal{D}}$ is the most specific action description of* a *wrt $\mathcal{S}_\sigma$.*

Beside translating the e-propositions into a set of action descriptions we have to consider the v-propositions which describe a particular scenario. To this end, we use a unary predicate *observations* with the intended meaning that $\mathit{observations}(i)$ is true if i represents a consistent situation such that each v-proposition is satisfied wrt i. Furthermore, the binary predicate $\mathit{satisfiable}(f, [a_1, \ldots, a_n])$ is used to express the fact that the additional v-proposition f `after` $[a_1, \ldots, a_n]$ holds in some model of the domain. The clauses defining these two predicates are depicted in Figure 3.

$$
\begin{array}{lll}
\mathit{observations}(I) & \leftarrow & \neg \mathit{inconsistent}(I), \\
 & & \mathit{causes}(I, [a_{11}, \ldots, a_{1n_1}], f_1 \circ V_1), \\
 & & \vdots \\
 & & \mathit{causes}(I, [a_{m1}, \ldots, a_{mn_m}], f_m \circ V_m).
\end{array}
$$

$$
\begin{array}{lll}
\mathit{satisfiable}(F, P) & \leftarrow & \mathit{observations}(I), \\
 & & \mathit{causes}(I, P, F \circ V).
\end{array}
$$

Figure 3: The given v-propositions $\{f_i \texttt{ after } a_{i1}, \ldots, a_{in_i} \mid 1 \leq i \leq m\}$ are collected in a single clause defining the unary predicate *observations* which is used in the definition of the predicate *satisfiable*.

For example, the two value propositions (7) are encoded via

$$
\begin{array}{lll}
\mathit{observations}(I) & \leftarrow & \neg \mathit{inconsistent}(I), \quad \mathit{causes}(I, [\,], \mathit{alive} \circ V_1), \\
 & & \mathit{causes}(I, [\mathit{shoot}, \mathit{wait}], \overline{\mathit{alive}} \circ V_2).
\end{array}
$$

If $\mathcal{D}$ is a domain description then let $P_{\mathcal{D}}$ consist of the clauses depicted in Figure 1, Figure 2, and Figure 3. Then, $(P_{\mathcal{D}}, \mathrm{AC1})$ can be shown to form a sound and complete encoding of $\mathcal{A}$ wrt the completion semantics:

Theorem 4.2 (Soundness & Completeness) *If $\mathcal{D}$ is a consistent domain description then $\mathcal{D} \models f \texttt{ after } [a_1, \ldots, a_n]$ iff*

$$
(P^*_{\mathcal{D}}, \mathrm{AC1}^*) \models \neg\, \mathit{satisfiable}(\overline{f}, [a_1, \ldots, a_n]) \,. \tag{11}
$$

Proof (sketch): Let Φ be the transition function determined by $\mathcal{D}$. If i is a term representing a situation $\mathcal{I}$ then, according to Theorem 2.1, Proposition 4.1, and Figure 3, $(P^*_{\mathcal{D}}, \mathrm{AC1}^*) \models \mathit{observations}(i)$ if and only if there is a state σ_0 corresponding to $\mathcal{I}$ such that (σ_0, Φ) is a model of $\mathcal{D}$. Hence, (11) is true iff there is no model of $\mathcal{D}$ such that $\overline{f}$ holds in $\Phi([a_1, \ldots, a_n], \sigma_0)$, i.e. iff $\mathcal{D} \models f \texttt{ after } [a_1, \ldots, a_n]$. ∎

For the complete proof the reader is referred to [31].

The part of the consistency criterion in Figure 2 where a situation is required to be complete in the sense that it must contain either the positive or the negative version of each fluent seems not particularly satisfactory. Fortunately, this condition can be omitted in many applications, e.g. in all of the various examples in [10]. However, the requirement is necessary to obtain completeness in examples like

$$
\begin{array}{llllll}
a & \texttt{causes} & f_1 & \texttt{if} & h & \qquad\qquad b \texttt{ causes } g \texttt{ if } f_1 \\
a & \texttt{causes} & f_2 & \texttt{if} & \neg h & \qquad\qquad b \texttt{ causes } g \texttt{ if } f_2
\end{array}
$$

These e-propositions entail $g \texttt{ after } [a, b]$ which cannot be obtained in our approach until each initial situation is forced to either contain h or $\overline{h}$.

It is noteworthy that when using schema (11) to decide the entailment of a v-proposition, this proposition is encoded within the query. Thus, one might object that v-propositions have to be guessed before proving their entailment. However, this is not necessary in general since the various answers to the query $\exists I.\, \mathit{observations}(I)$ provide the set of possible initial situations.

5 $\mathcal{A}$ + Non-Determinism = $\mathcal{A}_{ND}$

A basic assumption underlying $\mathcal{A}$ is that the effects of an action are always completely known and deterministic. Surely, one cannot adhere to this idealistic view of the real world in general since it is impractical, and even impossible due to a theoretical result of physics, to refine descriptions of the world until the effects of an arbitrary action can always be explicitly computed. The ability of humans to handle uncertainty, indeterminism, surprising effects etc. very flexibly contrasts sharply with the necessity of completely determining the effects of actions.

In this section we extend the action description language $\mathcal{A}$ such that effect propositions can express indeterminism. This approach is, of course, again idealistic as it is assumed that all possible alternatives are known and can be enumerated, and that stating priorities in form of, say, probabilistic values for the individual alternatives is not supported. Rather the extended action description language $\mathcal{A}_{ND}$ should be regarded as a first step in the large and open area of uncertainty, indeterminism, and disjunctive planning.

As the running example of this section we use the Russian Turkey scenario as formalized in [28]. The set of actions is augmented by the action *spin* with the intended meaning that spinning causes the gun to become randomly loaded or unloaded regardless of its state before. This action can be formalized in $\mathcal{A}_{ND}$ using the two expressions

$$\begin{array}{lll} \mathit{spin} & \texttt{alternatively causes} & \mathit{loaded} \\ \mathit{spin} & \texttt{alternatively causes} & \neg\mathit{loaded} \end{array} \tag{12}$$

In general, a `alternatively causes` $e_1, \ldots, e_m$ `if` $c_1, \ldots, c_n$ is called an *extended* effect proposition in $\mathcal{A}_{ND}$. The intended meaning is as follows: Let

$$\left\{ \begin{array}{llll} a & \texttt{alternatively causes} & E_1 & \texttt{if} \quad C_1 \\ & \vdots & & \\ a & \texttt{alternatively causes} & E_k & \texttt{if} \quad C_k \end{array} \right\} \tag{13}$$

be the set of all extended e-propositions such that $C_1, \ldots, C_k$ simultaneously hold in a particular state σ. If a is executed in σ then exactly one of the various sets of effects $E_1, \ldots, E_k$ will become true in the resulting state.

Recall that a set of e-propositions in the language $\mathcal{A}$ determines a transition function Φ. Now, the possibility of alternative effects forces a redefinition of the notion of transition. At first glance one might suggest for allowing the existence of several different transition functions, each of them modelling one of the various alternative effects of an action. Consider the e-propositions (5) of Section 3 and (12) for the Russian Turkey scenario. Φ

could be designed as in (6) augmented by either $\Phi(spin, \sigma) = \sigma \cup \{loaded\}$ or $\Phi(spin, \sigma) = \sigma - \{loaded\}$ for each σ separately. However, this idea does not correctly capture the intuition: If Φ is such a transition function in a particular model then the result of spinning the gun, say, will be fixed forever regarding a particular state, e.g. it would be impossible to find a model where `initially` $\neg loaded$, $\neg loaded$ `after` $[spin]$, and $loaded$ `after` $[spin, spin]$ are simultaneously true. This is of course unintended.

For this reason, we propose to drop the idea of Φ being a function and use the notion of Φ as a *relation* between a pair of states and an action name instead, such that two states σ, σ' and action name a are related whenever the application of a to σ might yield σ'. For instance, for the Russian Turkey scenario we obtain the following transition relation Φ:

$$
\begin{array}{lll}
(\sigma, spin, \sigma') \in \Phi & \text{iff} & \sigma' = \sigma \cup \{loaded\} \ \text{ or } \ \sigma' = \sigma - \{loaded\} \\
(\sigma, load, \sigma') \in \Phi & \text{iff} & \sigma' = \sigma \cup \{loaded\} \\
(\sigma, shoot, \sigma') \in \Phi & \text{iff} & \sigma' = \begin{cases} \sigma - \{loaded, alive\}, & \text{if } loaded \in \sigma \\ \sigma, & \text{otherwise.} \end{cases} \\
(\sigma, wait, \sigma') \in \Phi & \text{iff} & \sigma' = \sigma
\end{array}
\tag{14}
$$

In general, Φ is designed such that a triple (σ, a, σ') is element of Φ if and only if the following conditions are satisfied:

(a) If a `causes` f `if` $c_1, \ldots, c_n$ (resp. a `causes` $\neg f$ `if` $c_1, \ldots, c_n$) is an effect proposition such that $c_1, \ldots, c_n$ hold in σ then f (resp. $\neg f$) holds in σ'.

(b) Let (13) be the set of all extended e-propositions such that each element occurring in $C_1, \ldots, C_k$ holds in σ. If $k > 0$ then it is possible to select a $\lambda \in \{1, \ldots, k\}$ such that the members of E_λ hold in σ.

(c) Let f be a fluent name such that neither f nor $\neg f$ is forced to hold in σ' by (a) or (b) then $f \in \sigma'$ iff $f \in \sigma$.

The reader is invited to verify that the transition relation (14) satisfies these conditions wrt the e-propositions (5) and (12).

Having defined the notion of transition, we now concentrate on defining models in $\mathcal{A}_{ND}$. The issue of models is, in general, to provide a view of the real world. Usually, there exist several models which all satisfy some fundamental properties, observations, and maybe subjective impressions, but which differ in unknown or uninteresting things. All these models describe possible worlds although reality is captured by only one of them. In $\mathcal{A}$, where no indeterministic and randomized effects are allowed, the only task left to nature is to design the initial state. Now, however, the rôle of nature is much more appreciable because it has to decide which effects occur whenever alternatives are allowed. To this end, we employ an additional component for each model, namely a function φ which states the behavior of nature in this model in case of alternative effects. For instance, if the initial state is

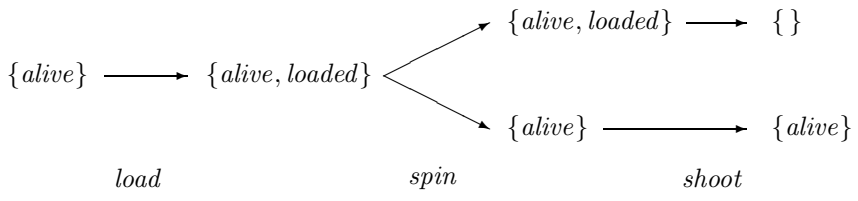

Figure 4: The possible developments in the Russian Turkey scenario.

known to be $\{alive\}$ and we are interested in the consequences of executing the sequence of actions $[load, spin, shoot]$ then the set of models of this domain can be divided into two classes: either the gun remains loaded after spinning, or it becomes unloaded. Hence, if we additionally observe that the turkey is as lively as before after loading, spinning, and shooting then no model of the former class can explain this. Thus it is reasonable to conclude that the gun was necessarily unloaded after $[load, spin]$. This is illustrated in Figure 4.

In the sequel we extend the formal definitions concerning $\mathcal{A}$ to $\mathcal{A}_{ND}$. A *structure* is a triple $(\sigma_0, \Phi, \varphi)$ where σ_0 denotes the initial state as before, Φ is a transition relation, and φ is a mapping from pairs consisting of a finite sequence of actions and a state into the set of states. A structure $(\sigma_0, \Phi, \varphi)$ is a *model* of a domain $\mathcal{D}_{ND}$ iff Φ is determined by the e-propositions in $\mathcal{D}_{ND}$, φ satisfies the two conditions

1. $\varphi([\,], \sigma_0) = \sigma_0$ and
2. $(\varphi([a_1, \ldots, a_{n-1}], \sigma_0), a_n, \varphi([a_1, \ldots, a_n], \sigma_0)) \in \Phi$,

and φ together with σ_0 satisfy the v-propositions of $\mathcal{D}_{ND}$: A v-proposition `initially` f is satisfied in $(\sigma_0, \Phi, \varphi)$ iff f holds in σ_0, and a v-proposition f `after` $[a_1, \ldots, a_n]$ is satisfied iff f holds in $\varphi([a_1, \ldots, a_n], \sigma_0)$. A domain description in $\mathcal{A}_{ND}$ is *consistent* if it admits at least one model $(\sigma_0, \varphi, \Phi)$ such that for any σ and any a there is at least one σ' such that $(\sigma, a, \sigma') \in \Phi$. For instance, a structure $(\sigma_0, \Phi, \varphi)$ is a model of (5) and (12) along with the two v-propositions

$$\begin{array}{l} \texttt{initially}\ \ alive \\ alive\ \ \texttt{after}\ \ [load, spin, shoot] \end{array} \tag{15}$$

iff Φ is as in (14), φ is appropriately defined, and $alive \in \sigma_0$ as well as $alive \in \varphi([load, spin, shoot], \sigma_0)$. Obviously, $\neg loaded \in \varphi([load, spin], \sigma_0)$ holds in each such model, i.e. $\mathcal{D}_{ND} \models \neg loaded$ `after` $[load, spin]$ (see again Figure 4).

At the very end of [7], the authors suggest a similar way of handling indeterminism using propositions of the form a `possibly causes` f `if` $c_1, \ldots, c_n$ but without giving a formal description of transition and entailment. A detailed examination and a comparison to our approach remains to be done.

6 Encoding $\mathcal{A}_{ND}$

In this section, we briefly illustrate how the equational logic program described in Section 2 can just as well be used to encode domain descriptions in the extended language $\mathcal{A}_{ND}$. Recall that the set of action descriptions generated by Algorithm 1 is designed such that for each action and each situation there is exactly one most specific description. Now, the existence of several most specific action descriptions addresses the existence of alternatives. In our example, we generate the pure descriptions

$$\langle\ \{\!|\ |\!\}\ ,\ spin\ ,\ \{\!|loaded|\!\}\ \rangle \quad \text{and} \quad \langle\ \{\!|\ |\!\}\ ,\ spin\ ,\ \{\!|\overline{loaded}|\!\}\ \rangle \tag{16}$$

according to (12). In general, Algorithm 1 can be straightforwardly modified such that each possible alternative for a fixed set of conditions determines a unique pure action description. The completion step simply remains as it stands.

Nonetheless, the kind of reasoning illustrated with the Russian Turkey domain cannot be modelled yet. If the query $\neg satisfiable(loaded, [load, spin])$ is used to prove entailment of $\neg loaded$ `after` $[load, spin]$ in the spirit of Theorem 4.2 then a negative answer is obtained. The reason for this undesired result is that we are free to choose the effect of spinning independently such that both $alive$ `after` $[load, spin, shoot]$ as well as $loaded$ `after` $[load, spin]$ can be satisfied by applying different most specific action description of $spin$.

The crucial point is that we have to take into account the reaction of nature, which is implicitly determined by the observations (15). Our solution to this problem is as follows: Instead of using simply the action name such as $spin$, we add an argument to denote a particular effect of this action such as in $spin(1)$. The additional argument is intended to take the rôle of the index λ used in the definition of transition in $\mathcal{A}_{ND}$ (see the paragraph below equation (14)). All actions are extended in this way.

In our running example, the following list includes all such action descriptions of $spin$, i.e. the two descriptions (16) — augmented by the additional argument — along with their completions:

$$\begin{array}{rr}
\langle\{\!|\ |\!\},\ spin(1)\ ,\{\!|loaded|\!\}\ \rangle & \langle\{\!|\ |\!\},\ spin(2)\ ,\{\!|\overline{loaded}|\!\}\rangle \\
\langle\{\!|loaded|\!\},\ spin(1)\ ,\{\!|loaded|\!\}\rangle & \langle\{\!|loaded|\!\},\ spin(2)\ ,\{\!|\overline{loaded}|\!\}\rangle \\
\langle\{\!|\overline{loaded}|\!\},\ spin(1)\ ,\{\!|loaded|\!\}\rangle & \langle\{\!|\overline{loaded}|\!\},\ spin(2)\ ,\{\!|\overline{loaded}|\!\}\rangle
\end{array} \tag{17}$$

The other action descriptions (9) and (10) are extended in a similar way.

The two predicates $observations$ and $satisfiable$ make use of these additional arguments. In our example (15) they are defined as follows:

$$\begin{array}{rl}
observations(I, X_1, X_2, X_3) \leftarrow & \neg inconsistent(I),\ causes(I, [\,], alive \circ V_1), \\
& causes(I, [load(X_1), spin(X_2), shoot(X_3)], alive \circ V_2).
\end{array}$$

$$\begin{array}{rl}
satisfiable(F, P, X_1, X_2, X_3) \leftarrow & observations(I, X_1, X_2, X_3), \\
& causes(I, P, F \circ V).
\end{array}$$

Now, let $P_{\mathcal{D}_{ND}}$ consist of the clauses depicted in Figure 1 — where the set of action descriptions is given as described above — and Figure 2 along with the clauses above. Then it is easy to prove that

$$\neg\exists X_1, X_2, X_3.\ \mathit{satisfiable}(\mathit{loaded}, [\mathit{load}(X_1), \mathit{spin}(X_2)], X_1, X_2, X_3)$$

is a logical consequence of $(P^*_{\mathcal{D}_{ND}}, \mathrm{AC1}^*)$. A more detailed analysis and a formal proof of a result similar to Theorem 4.2 can be found in [31].

7 Discussion

The soundness and completeness of the equational logic approach with respect to the action description language developed by M. Gelfond and V. Lifschitz constitutes the main result of this paper. This achievement illustrates that our approach is not only suitable for temporal projection but can also be used to reason about former situations, to find explanations for observations, and to deal with incomplete knowledge about situations. In particular the way incompletely specified situations are modelled appears to be very elegant: We merely have to add a variable to an AC1-term describing a situation. Nonetheless, an important aspect of incomplete information has not been discussed yet. Suppose nothing is known about the state of the gun then our program answers `no` when asked whether the v-proposition $\neg\mathit{alive}$ `after` $[\mathit{shoot}]$ is entailed. This appears to be too optimistic but can be easily improved. Recall that our program entails a v-proposition only if the contrary cannot be consistently assumed. Instead of answering `no` if this is not the case the answer could be `maybe` whenever both the v-proposition itself and the contrary are satisfiable wrt the domain description.

Probably the most important application of approaches to reasoning about actions is the field of planning. Our equational logic program can be directly used to search for plans: One simply employs an appropriate instance of the predicate *causes* with the middle argument left uninstantiated, i.e. an answer to the query $\exists P.\, \mathit{causes}(i, P, g)$ provides a plan whose application to i yields g (see [15]). In addition, as our approach can handle partial information about the initial situation, say, it is able to deal with questions such as *what do I need to achieve a certain goal?* Furthermore, values can be assigned to the various available resources so that the system is required to find an initial situation which is as cheap as possible (wrt the cost of the chosen resources). All these features are available in a quite compact logic program where no additional frame axioms increase the search space or make derivations more difficult in general.

The way planning is performed in our approach suggests an extension of $\mathcal{A}$ which provides a new kind of reasoning about the past. Hitherto, $\mathcal{A}$ supports reasoning about facts in former situations. But suppose we observe a lively turkey which suddenly drops dead, then it seems to be reasonable to conclude, from all of our knowledge, that a *shoot* action must have been performed. This can be easily obtained by using the query

$\exists A, I, V.\,(\neg \mathit{inconsistent}\,(I \circ \mathit{alive}) \wedge \mathit{causes}\,(I \circ \mathit{alive}, [A], \overline{\mathit{alive}} \circ V))$ yielding a single solution, namely $\{A \mapsto \mathit{shoot},\ I \mapsto \mathit{loaded},\ V \mapsto \overline{\mathit{loaded}}\}$.

Planning with uncertainty and incomplete information about the effects of actions is somewhat more difficult. The ideas presented in Section 6 cannot be directly adopted because if the additional argument of each action name is left variable then the system is always free to choose the desired effect. This is, of course, too optimistic. Rather, a *cautious agent* should be defined who is content only if solutions to the planning problem in each possible alternative can be found. This may require the creation of different subplans, each of them solving the problem in only some of the several alternatives, which is not feasible in our equational logic program of Section 2. In [4] it is shown how a cautious agent can be modelled in our method. The basic idea is to introduce a second function symbol, again embedded in a particular equational theory, which denotes a *disjunctive* connection of AC1-terms, each of them describing a possible situation. This can be regarded as the first step in view of weakening our restriction that situations are represented by conjunctions of atomic facts.

There are a variety of other aspects concerning upgrades of existing approaches such as concurrent actions, multiple agents, complex and non-inertial actions, probabilistic values for alternative effects etc. We have hopes that the equational logic based approach enables us to carry out experiments with respect to these and other ontological aspects as well.

Acknowledgements

The author would like to thank Wolfgang Bibel, Stefan Brüning, Steffen Hölldobler, Vladimir Lifschitz, and Aaron Rothschild for valuable comments on an earlier version of this paper. The author was partially supported by ESPRIT within basic research action MEDLAR-II under grant no. 6471 and by the German Research Community (DFG) within project KONNEKTIONSBEWEISER under grant no. Bi 228/6-2.

Notes

1. Throughout this paper, we use a PROLOG–like syntax, i.e. constants and predicates are in lower cases whereas variables are denoted by upper case letters. Moreover, free variables are assumed to be universally quantified and, as usual, the term $[h \mid t]$ denotes a list with head h and tail t.

2. Multisets are depicted using the brackets $\{\!|\ |\!\}$, and $\dot\cup$, $\dot-$, $\dot\subseteq$, $\doteq$, etc. denote the multiset extensions of the usual set operations and relations.

3. Thus, planning in this approach is closely related to planning in STRIPS [9, 22] except that multisets are used instead of sets and that planning is performed in a purely deductive context. As argued in [12] multisets are more adequate solutions to the frame problem. We therefore do not require the function symbol $\circ$ to be idempotent. The fundamental difference, however,

is that STRIPS was designed for planning only and cannot be used to perform the kind of general reasoning which is necessary for modelling $\mathcal{A}$, say.

4. The latter is a rather strict assumption which is required by $\mathcal{A}$ and therefore necessary to obtain completeness. We will take up this problem later.

5. Observe that $\mathcal{C}$ as well as $\mathcal{E}$ are multisets although set operations such as $\cup$ and $\subseteq$ are used to define them, which is to ensure that neither $\mathcal{C}$ nor $\mathcal{E}$ contain elements more than once.

References

[1] A. B. Baker. A simple solution to the Yale shooting problem. In *Proceedings of the Int.'l Conf. on Knowledge Representation and Reasoning*, 11–20, 1989.

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

[3] W. Bibel. A Deductive Solution for Plan Generation. *New Generation Computing*, 4:115–132, 1986.

[4] S. Brüning, S. Hölldobler, J. Schneeberger, U. Sigmund, and M. Thielscher. Disjunction in Resource–Oriented Deductive Planning. In D. Miller, ed., *Proc. of the ILPS*, page 670, Vancouver, 1993. MIT Press. (Poster presentation.)

[5] D. Chan. Constructive Negation Based on the Completed Database. *Proc. of the IJCSLP*, 111–125, 1988.

[6] K. L. Clark. Negation as Failure. In H. Gallaire and J. Minker, ed.'s, *Workshop Logic and Data Bases*, 293–322. Plenum Press, 1978.

[7] M. Denecker and D. de Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. In D. Miller, ed., *Proc. of the ILPS*, 147–163, Vancouver, 1993. MIT Press.

[8] P. M. Dung. Representing Actions in Logic Programming and its Applications in Database Updates. In D. S. Warren, ed., *Proc. of the ICLP*, 222–238, Budapest, 1993. MIT Press.

[9] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5(2):189–208, 1971.

[10] M. Gelfond and V. Lifschitz. Representing Action and Change by Logic Programs. *Journal of Logic Programming*, 17:301–321, 1993.

[11] C. Green. Application of theorem proving to problem solving. In *Proc. of the IJCAI*, 219–239, Los Altos, CA, 1969. Morgan Kaufmann Publishers.

[12] G. Große, S. Hölldobler, J. Schneeberger, U. Sigmund, and M. Thielscher. Equational Logic Programming, Actions, and Change. In K. Apt, ed., *Proc. of the IJCSLP*, 177–191, Washington, 1992. MIT Press.

[13] S. Hanks and D. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

[14] S. Hölldobler. On Deductive Planning and the Frame Problem. In A. Voronkov, ed., *Proc. of the Int.'l Conf. on Log. Prog. and Autom. Reasoning (LPAR)*, 13–29. Springer, volume 624 of LNAI, 1992.

[15] S. Hölldobler and J. Schneeberger. A New Deductive Approach to Planning. *New Generation Computing*, 8:225–244, 1990.

[16] S. Hölldobler and M. Thielscher. Actions and Specificity. In D. Miller, ed., *Proc. of the ILPS*, 164–180, Vancouver, 1993. MIT Press.

[17] S. Hölldobler and M. Thielscher. Computing Change and Specificity with Equational Logic Programs. *Annals of Mathematics and Artificial Intelligence*, special issue on Processing of Declarative Knowledge, 1994. (To appear.)

[18] J. Jaffar, J.-L. Lassez, and M. J. Maher. A theory of complete logic programs with equality. *Journal of Logic Programming*, 1(3):211–223, 1984.

[19] G. N. Kartha. Soundness and Completeness Theorems for Three Formalizations of Actions. In R. Bajcsy, ed., *Proc. of the IJCAI*, 724–729, Chambéry, France, 1993. Morgan Kaufmann.

[20] R. Kowalski. *Logic for Problem Solving*, volume 7 of *Artificial Intelligence Series*. Elsevier, 1979.

[21] R. Kowalski and M. Sergot. A logic based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[22] V. Lifschitz. On the Semantics of STRIPS. In M. P. Georgeff and A. L. Lansky, ed.'s, *Proc. of the Workshop on Reasoning about Actions & Plans*. Morgan Kaufmann, 1986.

[23] J. W. Lloyd. *Foundations of Logic Programming*. Series Symbolic Computation. Springer, second, extended edition, 1987.

[24] M. Masseron, C. Tollu, and J. Vauzielles. Generating Plans in Linear Logic. In *Foundations of Software Technology and Theoretical Computer Science*, 63–75. Springer, volume 472 of LNCS, 1990.

[25] J. McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[26] J. McCarthy and P. J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence*, 4:463–502, 1969.

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

[28] E. Sandewall. Features and Fluents. Technical Report LiTH-IDA-R-92-30, University of Linköping, Sweden, 1992.

[29] J. Schneeberger. *Plan Generation by Linear Deduction*. PhD thesis, FG Intellektik, TH Darmstadt, 1992.

[30] J. C. Shepherdson. SLDNF–Resolution with Equality. *Journal of Automated Reasoning*, 8:297–306, 1992.

[31] M. Thielscher. Modelling theories of actions by Equational Logic Programs. Technical Report AIDA–93–18, FG Intellektik, TH Darmstadt, 1993. Available via anonymous ftp from `130.83.26.1` in `/pub/AIDA/Tech-Reports/1993`.

[32] M. Thielscher. SLDENF–Resolution. 1994 (submitted). Available via anonymous ftp from `130.83.26.1` in `/pub/AIDA/Tech-Reports/OTHER`.