In: B. Nebel, L. Dreschler-Fischer, ed.'s, Proc. of the German Annual Conference on AI.
Springer LNAI 861, p. 16–27, Sep. 1994. (Best Paper Award.)

Representing Concurrent Actions and Solving Conflicts*

Sven-Erik Bornscheuer Michael Thielscher

FG Intellektik, FB Informatik, Technische Hochschule Darmstadt
Alexanderstraße 10, D–64283 Darmstadt (Germany)
E-mail: {sven,mit}@intellektik.informatik.th-darmstadt.de

Abstract. As an extension of the well–known *Action Description language* $\mathcal{A}$ introduced by M. Gelfond and V. Lifschitz [7], C. Baral and M. Gelfond recently defined the dialect $\mathcal{A}_C$ which allows the description of concurrent actions [1]. Also, a sound but incomplete encoding of $\mathcal{A}_C$ by means of an extended logic program was presented there. In this paper, we work on interpretations of contradictory inferences from partial action descriptions. Employing an interpretation different from the one implicitly used in $\mathcal{A}_C$, we present a new dialect $\mathcal{A}_C^+$, which allows to infer non-contradictory information from contradictory descriptions and to describe nondeterminism and uncertainty. Furthermore, we give the first sound and complete encoding of $\mathcal{A}_C$, using equational logic programming, and extend it to $\mathcal{A}_C^+$ as well.

1 Introduction

Intelligent beings are able to treat contradictory information more or less appropriately. For instance, imagine yourself asking two passers-by for the shortest way to the train station. The first one answers: "Turn right, and you will get there in five minutes." while the second one answers: "Turn right, and you will get there in ten minutes." Reasoning about these answers you find out that they are contradictory, i.e. the provided information is inconsistent and cannot be true. However, since both passers-by are in agreement with their recommendation to turn right, you would assume this part of the information to be true; you are only left in uncertainty about the time it takes to reach the station.

One should be aware of the difference between uncertain information explicitly stated as such like "you will arrive in five or in ten minutes" and contradictory information like the answers above. Contradictory information cannot be true; so, it has to be interpreted appropriately if you nonetheless want to derive some benefit from it.

When machines are used to reason about some complex information, it makes no sense to assume this information to be consistent in general; we know,i.a., from

* The second author was supported in part by ESPRIT within basic research action MEDLAR-II under grant no. 6471 and by the Deutsche Forschungsgemeinschaft (DFG) within project KONNEKTIONSBEWEISER under grant no. Bi 228/6-1.

Software Engineering that in general formalizations of something non-trivial are incorrect. Therefore, if a machine detects the incorrectness of the information it reasons about, this only gives certainty about circumstances which had to be assumed anyway. But it still has to be decided how this machine has to act in such a situation.

A typical field where the detection of the inconsistence of the used information has to be expected, is the one of reasoning about the execution of concurrent actions in dynamic systems.

Most complex dynamic systems include concurrent actions. Therefore, the ability for describing concurrent actions is of central interest in AI. For instance, to open a door locked by an electric door opener an autonomous robot has to press a button and to push the door concurrently. Hence, only knowing the effects of the seperate execution of each action, will not enable to open the door.

Since it is of course impractical to define the effects of the concurrent execution of each possible tuple of actions explicitly, it is preferable to infer most of these effects from the various descriptions of the involved individual actions. In certain cases, some of these descriptions may propose contradictory effects. The crucial question is again how to interpret such contradictions.

We discuss this question in the first part of this paper, and, for that, use a formalism $\mathcal{A}_C$ introduced by C. Baral and M. Gelfond. $\mathcal{A}_C$ allows the describtion of concurrent actions and was developed in [1] as an extension of the *Action Description Language* $\mathcal{A}$ [7] which was introduced in 1992 and became very popular since. Both $\mathcal{A}$ and $\mathcal{A}_C$ attract by the simple, elegant, and natural way in which the effects of actions are described. The execution of actions adds or removes elements from a particular set of facts representing some situation in the world; all non-affected facts continue to hold in the resulting situation due to the common assumption of persistence. The language $\mathcal{A}_C$ is defined in Section 2.

In Section 3 we examine the possibilities of interpreting contradictory inferences from partial action descriptions. Coming from a different point of view than the one implicitly underlying $\mathcal{A}_C$, we present an extension of $\mathcal{A}_C$ which we call $\mathcal{A}_C^+$. This new dialect allows to infer the non-contradicted information from contradictory descriptions, while such inference is not possible in $\mathcal{A}$ nor in $\mathcal{A}_C$. Moreover, $\mathcal{A}_C^+$ allows the describtion of nondeterministic actions with randomized effects and uncertain knowledge.

In the second part of this paper, we present the first sound and complete translation from $\mathcal{A}_C$ into a logic program with an underlying equational theory and, by modifying this translation in an appropriate way, an encoding of $\mathcal{A}_C^+$ as well. This translation allows to automate reasoning about dynamic systems following the concepts determined by $\mathcal{A}_C^+$ (and $\mathcal{A}_C$, respectively). Moreover, the translation of such high-level languages into different approaches designed for reasoning about dynamic systems, actions, and change, allows to compare the possibilities and limitations of these approaches in a precise and uniform way, which is in favorable contrast to the traditional way of explaining new approaches with reference to a few standard examples, such as some in the blocksworld or the famous "Yale Shooting Scenario" and its enhancements [7, 16, 17].

To this end, $\mathcal{A}$ was translated into several formalisms [7, 5, 14, 4, 19]. In [1], a sound but, unfortunately, incomplete encoding of $\mathcal{A}_C$ using extended logic programs following the lines of [7] was presented. In this paper, we extend the work [19] and show how an approach based on *equational logic programming* (ELP) [10, 9] can be used as a sound and even complete method for encoding $\mathcal{A}_C$ and $\mathcal{A}_C^+$.

ELP is a deductive approach using first-order-logic for describing actions. Similar to $\mathcal{A}$ and $\mathcal{A}_C$, the effects of actions are described by adding or removing resources from a single term representing a complete situation in the world (see also [2]). ELP is introduced in Section 4, and the translations encoding $\mathcal{A}_C$ and $\mathcal{A}_C^+$, respectively, are evolved in Section 5. Finally, our results are summarized in Section 6.

2 $\mathcal{A}_C$

We briefly review the concepts underlying the language $\mathcal{A}_C$ as defined in [1].

A *domain description* D in $\mathcal{A}_C$ consists of two disjoint sets of symbols, namely a set F_D of *fluent names* and a set A_D of *unit action names*, along with a set of *value propositions* (v-propositions) — each denoting the value of a single fluent in a particular state — and a set of *effect propositions* (e-propositions) denoting the effects of *actions*. A (compound) *action* is a non-empty subset of A_D with the intended meaning that all of its elements are executed concurrently. A v-proposition is of the form

$$f \texttt{ after } [a_1, \ldots, a_m] \tag{1}$$

where $a_1, \ldots, a_m$ ($m \geq 0$) are (compound) actions and f is a *fluent literal*, i.e. a fluent name possibly preceded by $\neg$. Such a v-proposition should be interpreted as: f has been observed to hold after having executed the sequence of actions $[a_1, \ldots, a_m]$. In case $m = 0$, (1) is written as `initially` f.

An e-proposition is of the form

$$a \texttt{ causes } f \texttt{ if } c_1, \ldots, c_n \tag{2}$$

where a is an action and f as well as $c_1, \ldots, c_n$ ($n \geq 0$) are fluent literals. (2) should be read as: Executing action a causes f to hold in the resulting state provided the conditions $c_1, \ldots, c_n$ hold in the actual state.

Example. You can open a door by running into it if at the same time you activate the electric door opener; otherwise, you will hurt yourself by doing this. The dog sleeping beside the door will wake up when the door opener is activated. You can close the door by pulling it. To formalize this scenario in $\mathcal{A}_C$, consider the two sets $A_{D_1} = \{activate,\ pull, run\_into\}$ and $F_{D_1} = \{open, sleeps, hurt\}$. The initial situation is partially described by the v-proposition `initially` *sleeps*, and the effects of the actions can be described by the e-propositions

$\{activate\}$	`causes`	$\neg sleeps$
$\{run\_into\}$	`causes`	$hurt$ `if` $\neg open$
$\{pull\}$	`causes`	$\neg open$
$\{activate, run\_into\}$	`causes`	$open$
$\{activate, run\_into\}$	`causes`	$\neg hurt$ `if` $\neg hurt$

Informally, the last e-proposition is needed to restrict the application of the

second one (which we call to *overrule* an e-proposition). Let D_1 denote the domain description given by these propositions.

Given a domain description D, a *state* σ is simply a subset of the set of fluent names F_D. For any $f \in F_D$, if $f \in \sigma$ then f is said to *hold* in σ, otherwise $\neg f$ holds. For instance, *sleeps* and $\neg$*open* hold in $\{sleeps, hurt\}$. A *structure* M is a pair (σ_0, Φ) where σ_0 is a state — called the *initial* state — and Φ is a partially defined mapping — called a *transition function* — from pairs consisting of an action and a state into the set of states. If $\Phi(a, \sigma)$ is defined then its value is interpreted as the result of executing a in σ.

Let $M^{(a_1,\ldots,a_k)}$ be an abbreviation of $\Phi(a_k, \Phi(a_{k-1}, \ldots, \Phi(a_1, \sigma_0)\ldots))$ where $M = (\sigma_0, \Phi)$, then a v-proposition like (1) is *true* in M iff

$$\begin{array}{l} \forall 1 \leq k \leq m\,.\ M^{(a_1 \ldots a_k)} \text{ is defined and} \\ f \text{ holds in } M^{(a_1,\ldots,a_m)}\,. \end{array}$$

The given set of e-propositions determines how a transition function should be designed which is suitable for a domain description. If a is an action, f a fluent literal, and σ a state then we say (executing) a *causes* f *in* σ iff there is an action b such that a causes f *by* b in σ. We say that a causes f by b in σ iff

1. $b \subseteq a$,
2. there is an e-proposition b `causes` f `if` $c_1, \ldots, c_n$ such that each $c_1, \ldots, c_n$ holds in σ.
3. there is no action c such that $b \subset c \subseteq a$ and a causes $\neg f$ by c in σ.

(3)

If 3. does not hold then action b is called to be *overruled* (by action c).

Using the two sets

$$\begin{array}{rcl} B_f(a,\sigma) & := & \{f \in F_D \mid a \text{ causes } f \text{ in } \sigma\} \\ B'_f(a,\sigma) & := & \{f \in F_D \mid a \text{ causes } \neg f \text{ in } \sigma\}, \end{array} \tag{4}$$

a structure $M = (\sigma_0, \Phi)$ is called a *model* of a domain description iff

1. every v-proposition is true in M and
2. for every action a and every state σ, $\Phi(a, \sigma)$ is only defined in case $B_f(a,\sigma) \cap B'_f(a,\sigma) = \{\}$.
   If it is defined then $\Phi(a,\sigma) = \sigma \cup B_f(a,\sigma) \setminus B'_f(a,\sigma)$.

(5)

A domain description admitting at least one model is said to be *consistent*. A v-proposition ν like (1) is *entailed* by a domain description D, written $D \models \nu$, if ν is true in every model of D.

Example (continued). The transition function determined by the e-propositions in our domain description D_1 is defined as follows. Let σ be an arbitrary state then

$$\begin{array}{ll} \Phi(\{\}, \sigma) & = \sigma \\ \Phi(\{run\_into\}, \sigma \cup \{open\}) & = \sigma \cup \{open\} \\ \Phi(\{run\_into\}, \sigma \setminus \{open\}) & = \sigma \setminus \{open\} \cup \{hurt\} \\ \Phi(\{pull\}, \sigma) & = \sigma \setminus \{open\} \\ \Phi(\{activate\}, \sigma) & = \sigma \setminus \{sleeps\} \\ \Phi(\{activate, pull\}, \sigma) & = \sigma \setminus \{sleeps, open\} \\ \Phi(\{run\_into, pull\}, \sigma \cup \{open\}) & = \sigma \setminus \{open\} \end{array}$$

$$\begin{aligned}
&\Phi(\{run_into\,, pull\}, \sigma \setminus \{open\}) = \sigma \setminus \{open\} \cup \{hurt\} \\
&\Phi(\{activate\,, run_into\}, \sigma) \qquad = \sigma \cup \{open\} \\
&\Phi(\{activate\,, run_into\,, pull\}, \sigma) \quad \text{is undefined}
\end{aligned}$$

D_1 has four models, viz.

$$\begin{array}{ll}
(\{sleeps\}, \Phi) & (\{open\,, sleeps\}, \Phi) \\
(\{sleeps\,, hurt\}, \Phi) & (\{open\,, sleeps\,, hurt\}, \Phi)
\end{array} \tag{6}$$

If, for instance, the v-proposition $\neg hurt$ `after` $\{run\_into\}$ is added to D_1 then the only remaining model is $(\{open\,, sleeps\}, \Phi)$ since for all other structures in (6) we find that $hurt \in \Phi(\{run\_into\}, \sigma_0)$. Hence, the v-proposition `initially` *open* , say, is entailed by this extended domain.

3 Interpreting and handling contradictions and a robust language $\mathcal{A}_{\mathrm{C}}^{+}$

In this section, we present different ways of interpreting descriptions of actions which may be executed concurrently. To illustrate our exposition, we use the terms of $\mathcal{A}_C$; nevertheless, the differences we work out classify other languages describing possibly concurrent actions as well.

Suppose a rather complex description of a part of the world has to be constructed. Because of the combinatorial explosion it obviously is impractible to describe the effects of all possible combinations of unit actions. Therefore, the effects of compound actions have to be inferred from the descriptions given seperately for the various involved actions. Combining these action descriptions might yield a contradiction among their effects.[2] In terms of $\mathcal{A}_C$ this means that $B_f \cap B'_f \neq \{\}$ and, hence, the particular compound action is not executable (see (5)). For instance, recall our domain description D_1. The e-propositions describing the effects of the elements of $\{activate\,, pull\,, run\_into\}$ propose both *open* and $\neg open$.

There are several different ways of inferring the effects of a compound action from such contradictory partial descriptions. Therefore, languages describing actions can be classified according to the explicit resp. implicit methods they use to draw these conclusions.

Explicit methods provide further information about the effects of certain compound actions. In terms of $\mathcal{A}_C$, additional e-propositions may

1. add a fluent to B_f or B'_f : obviously, the set $B_f \cap B'_f$ will remain nonempty, i.e., no conflicts will be solved;
2. remove a fluent from B_f or B'_f : this allows to remove predicted conflicts, but not to redefine facts not mentioned by the unit action descriptions (the approach [15] uses this method)
3. add or remove a fluent from B_f or B'_f : this allows to give a complete new definition of B_f and B'_f (used in $\mathcal{A}_C$, $\mathcal{A}_C^+$, and in the State Event Logic [8]).

[2] Of course, this prolem might even occur without concurrency involved, i.e. if several descriptions of the same unit action are used to infer the effects of this single action. If such an inference yields a contradiction, the semantics of $\mathcal{A}$, for instance, define the whole domain description to be inconsistent.

Example (continued). The e-proposition $\{activate, run\_into\}$ `causes` *open* adds the fluent *open* to the set $B_f(\{activate, run\_into\}, \sigma)$ [3] while the e-proposition $\{activate, run\_into\}$ `causes` $\neg hurt$ `if` $\neg hurt$ removes the fluent *hurt* [4] from $B_f(\{activate, run\_into\}, \sigma)$ by overruling the unit action description. Our example can only be modelled by using both addition and cancellation of effects.

Suppose the effects are not defined explicitly for all possible compound actions. In this case, it can happen that certain actions still are proposed to have contradictory effects. This might indicate that these actions are not executable in the world.[5] On the other hand, if such actions are observed then they indicate that the descriptions of their effects are wrong, uncertain or include nondeterministic actions.[6] In this case, depending on the chosen interpretation and the extent of certainty required one has to regard

1. the whole domain description (State Event Logic [8]),
2. the whole situation ($\mathcal{A}_C$ and [15])
3. the effects of the conflicting actions, or
4. the contradictory fluents ($\mathcal{A}_C^+$).

as unreliable.

Example (continued). Recall our example. Of course, it is conceivable that the door opener is activated, the door is pulled, and somebody runs into it at the same moment. The domain description D_1 proposes both *open* and $\neg open$ to be an effect of this compound action. Hence, D_1 is incomplete with respect to the world it describes. In fact, without further information we cannot say wether the door will be closed after executing this action or not.

However, we are sure that the dog will not sleep afterwards since we know that $\{activate\}$ `causes` $\neg sleeps$, and there is no proposition contradicting this.

In our example, by using the semantics of $\mathcal{A}_C$ it cannot be inferred that the dog does not sleep after executing $\{activate, pull, run\_into\}$. As an extreme case, imagine an agent in Saarbrücken executing this action and, concurrently, another agent in Franfurt switching off a light. Again, by $\mathcal{A}_C$ it cannot be inferred that the light is switched off in Frankfurt because the description used proposes contradictory states of a door in Saarbrücken. Nonetheless it seems to be reasonable to draw some conclusions about the resulting state instead of declaring it to be totally undefined, as it is done in $\mathcal{A}_C$.

We therefore weaken the basic assumption which says that $\Phi(a, \sigma)$ is undefined whenever the corresponding sets $B_f(a, \sigma)$ and $B'_f(a, \sigma)$ share one or more elements. To this end, we adopt a concept which has been introduced in [19]

---

[3] Note that the fluent *open* is not mentioned by the unit action descriptions $\{activate\}$ `causes` $\neg sleeps$ and $\{run\_into\}$ `causes` *hurt* `if` $\neg open$, respectively.

[4] postulated by $\{run\_into\}$ `causes` *hurt* `if` $\neg open$

[5] For instance, closing and opening the same door concurrently is not possible; these actions themselves are contradictory with respect to concurrent execution.

[6] In our example, running into the door, activating the door opener, and pulling the door concurrently might be regarded as a nondeterministic action wrt. the truth value of *open* . Also, D_1 could be intended to express uncertain knowledge about the effect of this action.

where $\mathcal{A}$ has been extended by integrating nondeterministic actions. The crucial idea is to drop the notion of a single resulting state determined by an action and a state, and to define a collection of possible resulting states instead. We use a ternary transition *relation* Φ such that an action a and two states σ, σ' are related if the execution of a in σ *possibly* yields σ'. Informally, if no conflicts occur wrt. a and σ then there is only one possible resulting state which should be exactly as in $\mathcal{A}_C$. If, on the other hand, there are conflicts, i.e. if the corresponding set $B_f(a,\sigma) \cap B'_f(a,\sigma)$ is not empty, then each combination of the truth values of the controversial fluent names determines one possible result.

By using this transition relation Φ, it becomes necessary to define which action names occuring in different v-propositions denote one and the same execution of actions (having distinct effects) and which do not. To this end, as in [19] we premise each two sequences of actions $a_1, \ldots, a_k, a_{k+1}, \ldots, a_m$ and $a_1, \ldots, a_k, a_{m+1}, \ldots, a_n$ occuring in the same domain description to refer to one and the same execution of $a_1, \ldots, a_k$ and, consequently, augment each structure by a function φ which maps each sequence $[a_1, \ldots, a_m]$ to a distinct resulting state $M^{(a_1,\ldots,a_m)}$ wrt. Φ and this premise.

The following definition of the dialect $\mathcal{A}_C^+$ makes these ideas manifest.

Definition 1. $\mathcal{A}_C^+$ is defined by the syntax and semantics of $\mathcal{A}_C$, but where
- a structure is a tripel $(\sigma_0, \Phi, \varphi)$
- a structure $(\sigma_0, \Phi, \varphi)$ is a model of a domain description D iff

$\varphi([]) = \sigma_0$,
$(\varphi([a_1, \ldots, a_{m-1}])\ ,\ a_m\ ,\ \varphi([a_1, \ldots, a_m]))\ \in\ \Phi$,
$(\sigma, a, \sigma') \in \Phi$ iff $\sigma' = ((\sigma \cup B_f) \setminus B'_f) \cup B^{\lightning}$ for some $B^{\lightning} \subseteq B_f \cap B'_f$,
and for all $\mathrm{v}-$propositions (1) in D, f holds in $\varphi([a_1, \ldots, a_m])$.

$\mathcal{A}_C^+$ is a proper extension of $\mathcal{A}_C$ in so far as whenever a v-proposition is entailed by a consistent domain description in $\mathcal{A}_C$ then it is also entailed wrt. the semantics of $\mathcal{A}_C^+$.

Example (continued). If our domain description D_1 is augmented by either the v-proposition *open* `after` $\{$*activate*, *pull*, *run_into*$\}$ or the contrary proposition $\neg$*open* `after` $\{$*activate*, *pull*, *run_into*$\}$ then both extended domains have models (with different functions φ) according to the semantics of $\mathcal{A}_C^+$. On the other hand, if D_1 is augmented by *sleeps* `after` $\{$*activate*, *pull*, *run_into*$\}$ then there is no model wrt. $\mathcal{A}_C^+$. Hence, as intended we can conclude that $D_1 \models_{\mathcal{A}_C^+} \neg$*sleeps* `after` $\{$*activate*, *pull*, *run_into*$\}$.

Note that $\mathcal{A}_C^+$ does not distinguish between intentionally expressed nondeterminism of actions and the interpretation of contradictory defined actions as to have uncertain effects. For instance, D_1 could be augmented by the e-propositions *activate* `causes` $\{$*bark*$\}$ and *activate* `causes` $\{\neg$*bark*$\}$ for describing that the dog possibly starts or stops barking when the door opener is activated. In fact, for someone or something reasoning about a domain description it makes no difference, whether the producer of this domain description was conscious of the uncertainty of the described effects of an action or not.

4 The ELP Approach

The equational logic programming approach to reasoning about actions and change [10, 11] is based on using reification to represent a complete situation by a single term $t_1 \circ \cdots \circ t_n$ where $t_1, \ldots, t_n$ are the facts holding in this situation. Since the order in such terms should be irrelevant, the connection function $\circ$ is required to be associative (A) and commutative (C). In addition, it admits a unit element (1), viz. the constant $\emptyset$ denoting the empty situation.

Actions are defined and executed in a STRIPS-like fashion [6] using a ternary predicate[7]

$$action\,(\,V \circ c_1 \circ \cdots \circ c_l\,,\,a_1 \circ \cdots \circ a_m\,,\,V \circ e_1 \circ \cdots \circ e_n\,) \tag{7}$$

meaning that the compound action[8] $a_1 \circ \cdots \circ a_m$ transfers every situation $V \circ c_1 \circ \cdots \circ c_l$ [9] into the situation $V \circ e_1 \circ \cdots \circ e_n$ (in other words, if $a_1 \circ \cdots \circ a_m$ is executed in a situation in which the *conditions* $c_1 \circ \cdots \circ c_l$ hold, it removes these conditions and adds the *effects* $e_1 \circ \cdots \circ e_n$). Thus, all facts which are not amongst the conditions hold after the application of (7) if they did so before. Therefore, no additional axioms for solving the frame problem are needed although dealing with a purely deductive method.

The result of executing a sequence of actions is then defined by the two clauses (12) (see further below) which conform to the semantics of $\mathcal{A}_C$.

Based on similar ideas, a logic program associated with the equational theory (AC1) was presented in [19] which forms a sound and complete encoding of the original Action Description Language $\mathcal{A}$. In the following section we evolve an analogous program encoding of domain descriptions given in $\mathcal{A}_C$ or $\mathcal{A}_C^+$, respectively. Due to the large number of compound actions implicitly described by a domain description in $\mathcal{A}_C$, it seems impractical to describe all of them explicitly in ELP. Therefore, definitions (7) are represented implicitly by clause (8) in the translations defined in the following section.

5 Translating $\mathcal{A}_C$ and $\mathcal{A}_C^+$ into ELP

In $\mathcal{A}_C$, classical negation of fluent symbols determines these fluents to be false. Since in the ELP based method fluents are reified and, hence, cannot be negated in this way we represent negated fluent symbols by defining a new complementary symbol for each fluent name. Let $\overline{F_D}^{\varphi}$ denote a set of symbols such that $F_D \cap \overline{F_D}^{\varphi} = \{\}$ then we define a bijective mapping φ over $F_D \cap \overline{F_D}^{\varphi}$ such that $x \in F_D \Leftrightarrow \varphi(x) \in \overline{F_D}^{\varphi}$ and $\varphi(\varphi(x)) = x$. For instance, if $F_D = \{open, sleeps, hurt\}$ then we might use $\overline{F_D}^{\varphi} = \{closed, awake, safe\}$,

[7] Throughout this paper, we use a PROLOG-like syntax, ie. 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 $[h \,|\, t]$ denotes a list with head h and tail t.

[8] In [12] the concurrent execution of actions is not taken into consideration; this extension is obviously necessary for encoding $\mathcal{A}_C$.

[9] i.e. every situation unifiable with $V \circ c_1 \circ \cdots \circ c_l$ wrt. to the underlying equational theory (AC1)

say, along with $\varphi(open) = closed$, $\varphi(sleeps) = awake$, $\varphi(hurt) = safe$, and vice versa.

Now, we are able to map sequences of fluent literals using a function τ_{φ_D} into a single term based on the (AC1)–function $\circ$:

$$\tau_{\varphi_D}(f_1, \ldots, f_m, \neg f_{m+1}, \ldots, \neg f_n) \;:=\; f_1 \circ \cdots \circ f_m \circ \varphi_D(f_{m+1}) \circ \ldots \circ \varphi_D(f_n)$$

where $f_i \in F_D$.

In $\mathcal{A}_C$, a state in the world is described as a set σ of fluent symbols f_i implying that the fluent literals f_i are true and the fluent literals corresponding to $F_D \setminus \sigma$ are false in this state. Therefore, a state σ is represented by an (AC1)–term corresponding to $\sigma \cup \{\varphi(f) \mid f \notin \sigma\}$:

$$\gamma_{\varphi_D}(\{f_1, \ldots, f_m\}) \;:=\; f_1 \circ \ldots \circ f_m \circ \varphi_D(f_{m+1}) \circ \ldots \circ \varphi_D(f_n),$$

where $\{f_1, \ldots, f_n\} = F_D$.

Finally, we represent compound actions by simply connecting the unit action names using again our (AC1)–function:

$$\mu_{\varphi_D}(\{a_1, \ldots, a_k\}) \;:=\; a_1 \circ \cdots \circ a_k$$

where $\{a_1, \ldots, a_k\} \subseteq A_D$.

Using the definitions above, we are now prepared for translating $\mathcal{A}_C$ domain descriptions into an equational logic program. For each fluent name, we use a unit clause to relate it to its counterpart in the set $\overline{F_D}^{\varphi}$:

$$\varphi_D^{ELP} \;:=\; \{complement(f \circ \varphi_D(f)). \mid f \in F_D\}$$

For each e-proposition we use a unit clause stating its conditions, its action name, and its effect:

$$EPROP_{\varphi_D} \;:=\; \{eprop(\tau_{\varphi_D}(c_1, \ldots, c_n), \mu_{\varphi_D}(a), \tau_{\varphi_D}(f)). \mid a \texttt{ causes } f \texttt{ if } c_1, \ldots, c_n \in D\}$$

To encode the semantics of $\mathcal{A}_C$ we use a ternary predicate $action(i, a, h)$ intending that executing action a in state i yields state h. It is defined as follows.

$$\begin{aligned} action(I, A, H) \leftarrow\; & \neg overruled(H, I, \emptyset, A), \\ & \neg non\_inertial(H, I, A), \\ & \neg inconsistent(H). \end{aligned} \tag{8}$$

where the resulting state h is required

1. not to be *overruled*, i.e. there is no non-overruled e-proposition applicable to i and postulating the complement of a fluent literal in h.
2. not to be *non-inertial*, i.e. there is no resource in h but not in i which is not postulated by an applicable e-proposition, and
3. not to be *inconsistent*, i.e. h contains exactly one element of each pair of complementary resources and, thus, is of the form $\tau_{\varphi_D}(\sigma)$ for some σ.

The predicates *overruled*, *non_inertial*, and *inconsistent* are defined by

$$\begin{aligned} & overruled(F \circ H, C \circ J, A, A \circ B \circ D) \\ & \qquad \leftarrow eprop(C, A \circ B, G), \\ & \qquad\quad F \neq_{AC1} \emptyset, \\ & \qquad\quad B \neq_{AC1} \emptyset, \\ & \qquad\quad complement(F \circ G), \\ & \qquad\quad \neg overruled(G, C \circ J, A \circ B, A \circ B \circ D). \end{aligned} \tag{9}$$

In words, the effect F of an action A is overruled by an *eprop* postulating the effect $G = \varphi_D(F)$ of an action $B \supset A$ if B is not overruled with respect to G. The termination of the decision about $\mathit{overruled}(H, I, B, B \circ A)$ for all H, I, B, A can be shown by induction over the strictly increasing third argument, if a fair selection rule is used.

$$\begin{array}{l} \mathit{non\_inertial}(F \circ G, H \circ J, A) \leftarrow \mathit{complement}(F \circ H)\,, \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \neg \mathit{overruled}(H, H \circ J, \emptyset, A)\,. \end{array} \tag{10}$$

In words, the truth value of a fluent literal F of the initial state $H \circ J$ is changed although no non-overruled e-proposition postulates this change.

$$\begin{array}{ll} \mathit{inconsistent}(G \circ G \circ H) & \leftarrow G \neq_{AC1} \emptyset\,. \\ \mathit{inconsistent}(F \circ G) & \leftarrow \mathit{complement}(F)\,. \\ \mathit{inconsistent}(H) & \leftarrow \mathit{complement}(F \circ G)\,, \\ & \quad F \neq_{AC1} \emptyset\,,\ G \neq_{AC1} \emptyset\,, \\ & \quad \neg \mathit{holds}(F, H)\,,\ \neg \mathit{holds}(G, H)\,. \\ \mathit{holds}(F, H \circ F)\,. & \end{array} \tag{11}$$

In words, a situation term is inconsistent if it contains a resource twice or if it contains a resource along with its counterpart $\varphi(f)$ or if it neither contains f nor $\varphi(f)$ for some $f \in F_D$.

The transition of σ_0 into the state $M^{(a_1,\ldots,a_m)}$ (the result G of executing $[a_1, \ldots, a_m]$ in the initial state I) is modelled by

$$\begin{array}{ll} \mathit{result}(I, [], G) & \leftarrow I =_{AC1} G\,. \\ \mathit{result}(I, [A \mid P], G) & \leftarrow \mathit{action}(I, A, H)\,, \\ & \quad \mathit{result}(H, P, G)\,. \end{array} \tag{12}$$

The termination of the decision about $\mathit{result}(H, P, G)$ for all H, P, G can be shown by induction over the strictly decreasing second argument.

Finally, let n be the number of v-propositions of the form (1) then these are translated into the clause

$$\begin{array}{rl} \mathit{model}(I) \leftarrow & \neg \mathit{inconsistent}(I)\,, \\ & \mathit{result}(I, [\mu_{\varphi_D}(a_{11}), \ldots, \mu_{\varphi_D}(a_{1m_1})], \tau_{\varphi_D}(f_1) \circ G_1)\,, \\ & \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ & \mathit{result}(I, [\mu_{\varphi_D}(a_{n1}), \ldots, \mu_{\varphi_D}(a_{nm_n})], \tau_{\varphi_D}(f_n) \circ G_n)\,. \end{array} \tag{13}$$

with the intended meaning that $\mathit{model}(i)$ is true if i represents a consistent initial state such that all v-propositions are satisfied.

To summarize, a domain description D in $\mathcal{A}_C$ is translated into the set of clauses $P = \varphi_D^{ELP} \cup \mathit{EPROP}_{\varphi_D} \cup \{(8)\text{–}(13)\}$. As we have negative literals in the body of some clauses, the adequate computational mechanism for P is SLDNF–resolution where, due to our equational theory (AC1), standard unification is replaced by a theory unification procedure. Following [18], the semantics of our program is then given by its *completion* (cf. [3]) $(P_D^*, AC1^*)$ where $AC1^*$ denotes a *unification complete* theory wrt. AC1 (see [13] or [18]).

The following theorem forms the basis of our soundness and completeness result regarding the completion of our constructed equational logic program.

Theorem 2. *Let D be a domain description in $\mathcal{A}_C$ determining a transition function Φ; then,there exists a σ_0 such that the structure (σ_0, Φ) is a model of D and some $I = \gamma_{\varphi_D}(\sigma_0)$ iff*

$$(P_D^*, AC1^*) \models model(I)\,.$$

Example(continued) Let φ_{D_1} be defined by

$$\varphi_{D_1}(sleeps) = \overline{sleeps}\,, \quad \varphi_{D_1}(hurt) = \overline{hurt}\,, \quad \varphi_{D_1}(open) = \overline{open}$$

Then $P_{D_1} =$

$complement(sleeps \circ \overline{sleeps})\,.$
$complement(hurt \circ \overline{hurt})\,.$
$complement(open \circ \overline{open})\,.$
$eprop(\emptyset, activate, \overline{sleeps})\,.$
$eprop(\overline{open}, run\_into, hurt)\,.$
$eprop(\emptyset, pull, \overline{open})\,.$
$eprop(\emptyset, activate \circ run\_into, open)\,.$
$eprop(\overline{hurt}, activate \circ run\_into, \overline{hurt})\,.$
$model(I) \leftarrow \neg inconsistent(I)\,,$
$result(I, \emptyset, sleeps \circ G_1)\,.$

$\cup(8) - (12)$

The equational logic program developed above can be easily modified to simulate the semantics defined by $\mathcal{A}_C^+$. We use the very same translation, but the literal $\neg overruled(H, I, \emptyset, A)$ in the body of (8) is replaced by $\neg impossible(H, I, A)$ and the following clause is added.

$$\begin{aligned} impossible(F \circ H, I, A) \leftarrow\ & overruled(F, I, \emptyset, A)\,, \\ & complement(F \circ G)\,, \\ & \neg overruled(G, I, \emptyset, A)\,. \end{aligned}$$

In words, *a causes $\neg f$ in σ* and $\neg$(*a causes f in σ*) where $I = \gamma_{\varphi_D}(\sigma)$
Also, according to Definition 1, the head of (8) is replaced by $action(I, A(H), H)$ and the action names in (9) have to be labeled similarly with variables such that two action names are labeled with the same variable iff they denote one and the same execution of an action (see also [19]).

6 Summary

We investigated possible interpretations of partially contradictory descriptions of the effects of concurrently executed actions. Our analysis lead to a new language $\mathcal{A}_C^+$ describing concurrent actions which extends the work of C. Baral and M. Gelfond: $\mathcal{A}_C^+$ allows to infer all the non-contradicted information from contradictory descriptions, whereas $\mathcal{A}$ and $\mathcal{A}_C$ does not. Moreover, $\mathcal{A}_C^+$ enables one to describe nondeterministic actions and uncertain knowledge.

Furthermore, we presented sound and complete encodings of the languages $\mathcal{A}_C$ and (by a simple modification) $\mathcal{A}_C^+$ by means of equational logic programs.

Acknowledgements. The authors would like to thank Wolfgang Bibel, Gerd Große and Wulf Röhnert for valuable discussions and comments on an earlier version of this paper.

References

1. C. Baral and M. Gelfond. Representing Concurrent Actions in Extended Logic Programming. In R. Bajcsy, ed., *Proc. of IJCAI*, p. 866–871, Chambéry, August 1993. Morgan Kaufmann.
2. W. Bibel. A Deductive Solution for Plan Generation. *New Generation Computing*, 4:115–132, 1986
3. K. L. Clark. Negation as Failure. In H. Gallaire and J. Minker, ed.'s, *Workshop Logic and Data Bases*, p. 293–322. Plenum Press, 1978.
4. M. Denecker and D. de Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. In D. Miller, ed., *Proc. of ILPS*, p. 147–163, Vancouver, October 1993. MIT Press.
5. P. M. Dung. Representing Actions in Logic Programming and its Applications in Database Updates. In D. S. Warren, ed., *Proc. of ICLP*, p. 222–238, Budapest, June 1993. MIT Press.
6. 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.
7. M. Gelfond and V. Lifschitz. Representing Action and Change by Logic Programs. *Journal of Logic Programming*, 17:301–321, 1993.
8. G. Große. Propositional State-Event Logic. To appear JELIA'94, Springer LNAI.
9. G. Große, S. Hölldobler, J. Schneeberger, U. Sigmund, and M. Thielscher. Equational Logic Programming, Actions, and Change. In K. Apt, ed., *Proc. of IJCSLP*, p. 177–191, Washington, 1992. MIT Press.
10. S. Hölldobler and J. Schneeberger. A New Deductive Approach to Planning. *New Generation Computing*, 8:225–244, 1990.
11. S. Hölldobler and M. Thielscher. Actions and Specificity. In D. Miller, ed., *Proc. of ILPS*, p. 164–180, Vancouver, October 1993. MIT Press.
12. 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).
13. 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.
14. G. N. Kartha. Soundness and Completeness Theorems for Three Formalizations of Actions. In R. Bajcsy, ed., *Proc. of IJCAI*, p. 724–729, Chambéry, France, August 1993. Morgan Kaufmann.
15. F. Lin and Y. Shoham. Concurrent Actions in the Situation Calculus. In *Proc. of AAAI*, p. 590–595, South Lake Tahoe, California, 1992.
16. E. Sandewall. Features and Fluents. Technical Report LiTH-IDA-R-92-30, Institutionen för datavetenskap, Technical University Linköping, Schweden, 1992.
17. E. Sandewall. The range of applicability of nonmonotonic logics for the inertia problem. In R. Bajcsy, ed., *Proc. of IJCAI*, p. 738–743, Chambéry, France, August 1993. Morgan Kaufmann.
18. J. C. Shepherdson. SLDNF-Resolution with Equality. *Journal of Automated Reasoning*, 8:297–306, 1992.
19. M. Thielscher. Representing Actions in Equational Logic Programming. In P. Van Hentenryck, ed., *Proc. of ICLP*,p. 207–224, Santa Margherita Ligure, Italy, 1994. MIT Press.