*

INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

Explicit and Implicit Indeterminism: Reasoning About Uncertain and Contradictory Specifications of Dynamic Systems

Sven-Erik Bornscheuer and Michael Thielscher

TR-96-009

February 1996

Abstract

A high-level action semantics to specify and reason about dynamic systems is presented which supports both uncertain knowledge (taken as *explicit* indeterminism) and contradictory information (taken as *implicit* indeterminism). We start by developing an action description language for intentionally representing nondeterministic actions in dynamic systems. We then study the different possibilities of interpreting contradictory specifications of concurrent actions. We argue that the most reasonable interpretation which allows for exploiting as much information as possible is to take such conflicts as implicit indeterminism.

As the second major contribution, we present a calculus for our resulting action semantics based on the logic programming paradigm including negation-as-failure and equational theories. Soundness and completeness of this encoding wrt the notion of entailment in our high-level action language is proved by taking the completion semantics for equational logic programs with negation.

* Address of the first author: Wissensverarbeitung, Informatik, TU Dresden, 01062 Dresden (Germany). The second author is currently on leave from FG Intellektik, TH Darmstadt.

1 Introduction

Uncertainty is a general challenge which comes with different faces. If an agent reasons about a given representation of a dynamic system, he might be uncertain about the effects of particular actions; one possible reason for such an uncertainty is the designer of this representation has intentionally specified these actions to have nondeterministic effects. There are good reasons for doing this: the designer might not know the exact causal relationship between the action and the observed effects, or the action might be chaotic, etc. In the very first part of this paper, we develop a high-level representation language and semantics which allows for intentionally specifying nondeterministic actions with randomized effects.

Our language is based on the *Action Description Language* $\mathcal{A}$ [Gelfond and Lifschitz, 1993], which is appealing because of the simple, elegant and natural way in which the effects of actions are described. A formal introduction to this language can be found in the next section, 2, and our extended language dealing with explicit indeterminism, $\mathcal{A}_N$, is then developed in Section 3.

Aside from being faced with explicitly represented indeterminism, an agent might also be uncertain about a given specification when it turns out to be contradictory. Intelligent beings are most often able to evaluate contradictory information to an appropriate extent. 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; the provided information is inconsistent and, hence, 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 sound; you are only left with 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 two answers above. Contradictory information cannot be true, so it has to be interpreted appropriately if nonetheless some benefit shall be derived from it. Surely, any such interpretation may become delicate and, therefore, has to be carefully selected in view of the application at hand. When machines are used to reason about complex domains, it is highly likely that an inconsistency occurs in the corresponding formal specification; e.g., we know from Software Engineering that in general formalizations of non-trivial scenarios are incorrect. Therefore, if a reasoning system detects an inconsistency in the information that has been provided, this only gives certainty about circumstances which had to be assumed anyway. But it still has to be decided how this system has to act in such a situation.

A typical field where contradictory specifications have to be expected is reasoning about the concurrent execution of actions in dynamic systems. Most complex dynamic systems include some kind of concurrency, which is why the ability of describing simultaneous 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, knowing the effects of the separate execution of these actions only is not sufficient to be able to open the door. Since it is of course impractical to define the effects of the concurrent execution of each possible combination of actions explicitly, it is necessary to infer these effects from the various descriptions of the individual actions that are involved. In certain cases, some of these descriptions may however propose contradictory effects. The crucial question then is how to interpret such contradictions.

This question will be discussed in Section 4. To this end, we use a recent extension of the Action Description Language $\mathcal{A}$ which is called $\mathcal{A}_C$ and supports representing and reason-

ing about the concurrent execution of actions [Baral and Gelfond, 1993]. In Subsection 4.1, we discuss different explicit methods which enable the designer of a representation to prevent the aforementioned conflicts by providing more specific information regarding particular concurrently executed actions. We will argue that $\mathcal{A}_C$ uses the most expressive way and, hence, is most suitable as a basis for our further discussions. The language $\mathcal{A}_C$ is recapitulated in Subsection 4.2.

In Subsection 4.3, we then examine the various possibilities to interpret contradictory inferences caused by combining action descriptions. Suggesting a different point of view than the one implicitly underlying $\mathcal{A}_C$, we present a new language called $\mathcal{A}_{NCC}$, which combines our preceding development regarding *explicit* indeterminism, the language $\mathcal{A}_N$, with $\mathcal{A}_C$ and defines a new way of successfully reasoning about inconsistent specifications of concurrently executed actions.[1] The crucial idea is to interpret such contradictions as *implicit* indeterminism. To this end, we consider uncertain the pieces of information which cause the contradiction, while all effects on which the involved action descriptions agree are assumed to occur as specified. Thereby, our language enables us to still infer reasonable information from contradictory descriptions, while such inferences are neither possible in $\mathcal{A}$ nor in $\mathcal{A}_C$.

As the second major contribution of this paper, we then present a sound and complete translation from domains specified in our high-level action language $\mathcal{A}_{NCC}$ into logic programs. Our translation follows an approach originally introduced in [Hölldobler and Schneeberger, 1990], which is based on the reification of entire situation descriptions by formally treating them as terms. In contrast to situation calculus [McCarthy, 1963; McCarthy and Hayes, 1969], where situation terms are abstract objects, the former approach employs situation terms consisting of an explicit collection of those fluents which hold in the situation being represented. Executing actions is then modeled by manipulating such collections of fluents, which is why we call the underlying method *fluent calculus*, $\mathcal{FC}$.[2] An equational logic program suitable for encoding $\mathcal{A}_{NCC}$, consequently named $\mathcal{FC}_{NCC}$, is developed in Subsection 5.1. In Subsection 5.2, we analyze the semantics of this program given by its completion, and in Subsection 5.3 we prove its soundness and completeness wrt the high-level action semantics given by $\mathcal{A}_{NCC}$. Finally, in Subsection 5.4 we discuss an adequate computation mechanism four our program, namely, *SLDENF-resolution* [Shepherdson, 1992; Thielscher, 1996a], which is based on SLD-resolution but with the standard unification procedure replaced by a special equality unification algorithm and negation-as-failure used to treat negative subgoals.

Our translation allows automated reasoning about dynamic systems following the concepts captured by $\mathcal{A}_{NCC}$. 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. As argued in, e.g., [Gelfond and Lifschitz, 1993; Sandewall, 1993; Sandewall, 1994; Thielscher, 1994; Thielscher, 1995b], doing this is in favorable contrast to the traditional way of justifying new approaches with reference to a few standard examples such as the blocksworld or the famous "Yale Shooting Scenario" and its enhancements. To this end, translations of $\mathcal{A}$ and some of its extensions, for instance, into a number of existing action calculi have recently been used for the purpose of comparison and to study their range of applicability (see, e.g., [Gelfond and Lifschitz, 1993; Dung, 1993; Kartha, 1993; Denecker and de Schreye, 1993; Baral and Gelfond, 1993;

[1] $\mathcal{A}_N$ means "action formalism supporting nondeterministic (N) actions based on the Action Description Language ($\mathcal{A}$)," and $\mathcal{A}_{NCC}$ is as $\mathcal{A}_N$ but in addition supports concurrent actions and conflict solving.

[2] We are grateful to Stuart Russell for suggesting this name.

Kartha and Lifschitz, 1994]).

2 Describing Simple Action Scenarios

We briefly review the concepts underlying the Action Description language $\mathcal{A}$ as defined in [Gelfond and Lifschitz, 1993].

Definition 1 A domain description D consists of two disjoint and non-empty sets of symbols F_D and A_D called *fluent names* and *unit actions*, respectively. A *fluent literal* is a fluent name or its negation, the latter of which is denoted by $\overline{f}$.

Furthermore, D consists of a set V_D of *value propositions* (*v-propositions*, for short), which are expressions of the form

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

where $a_1, \ldots, a_m$ ($m \geq 0$) are unit actions and ℓ is a fluent literal.

Finally, D includes a set E_D of *effect propositions* (*e-propositions*, for short), which are expressions of the form

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

where a is a unit action and ℓ as well as $c_1, \ldots, c_n$ ($n \geq 0$) are fluent literals. ■

A v-proposition (1) should be interpreted as: ℓ has been observed to hold after having executed the sequence of unit actions $[a_1, \ldots, a_m]$. In case $m = 0$, (1) is usually written `initially` ℓ. An e-proposition (2) should be read as: Executing unit action a causes ℓ to hold in the resulting state provided the conditions $c_1, \ldots, c_n$ hold in the current state.

Example 1 The *Yale Shooting* domain [Hanks and McDermott, 1987] can be modeled using the fluent names $F_{D_1} = \{loaded, alive\}$ denoting the state of a gun and a turkey, respectively. The effects of the unit actions $A_{D_1} = \{load, wait, shoot\}$ are specified by these three e-propositions:

$$\begin{array}{lll} load & \texttt{causes} & loaded \\ shoot & \texttt{causes} & \overline{loaded} \\ shoot & \texttt{causes} & \overline{alive} \quad \texttt{if} \quad loaded \end{array} \tag{3}$$

In words, loading the gun causes it to be loaded, shooting with the gun causes it to become unloaded and also shoots the turkey provided the gun was loaded. Waiting is assumed to have no effects at all. The following two v-propositions then encode the *Stanford Murder Mystery* [Baker, 1989]:

$$\begin{array}{l} \texttt{initially} \ alive \\ \overline{alive} \ \texttt{after} \ [wait, shoot] \end{array} \tag{4}$$

In words, the turkey was alive at the beginning but not after having executed *wait* followed by *shoot*. ■

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 $\overline{f}$ holds. For instance, *alive* and $\overline{loaded}$ hold in the state $\sigma = \{alive\}$.

The given effect propositions implicitly determine the causal behavior of the dynamic system being modeled:

Definition 2 Let D be a domain description in $\mathcal{A}$. Furthermore, let $a \in A_D$ be a unit action, $\ell \in F_D \cup \{\overline{f} \mid f \in F_D\}$ a fluent literal, and $\sigma \subseteq F_D$ a state. Then we say that a *causes* ℓ *in* σ iff E_D contains an e-proposition a `causes` ℓ `if` $c_1, \ldots, c_n$ such that each of $c_1, \ldots, c_n$ holds in σ. Let

$$\begin{array}{rcl} B_f(a,\sigma) & := & \{f \in F_D \mid a \text{ causes } f \text{ in } \sigma\} \\ \overline{B_f}(a,\sigma) & := & \{f \in F_D \mid a \text{ causes } \overline{f} \text{ in } \sigma\} \end{array} \tag{5}$$

and let Φ be a mapping from pairs consisting of a unit action and a state into the set of states, that is, $\Phi : A \times 2^{F_D} \mapsto 2^{F_D}$. Then Φ is a *transition function* for D iff, for each $a \in A$ and $\sigma \subseteq F_D$,

1. $B_f(a,\sigma) \cap \overline{B_f}(a,\sigma) = \{\}$ and
2. $\Phi(a,\sigma) = (\sigma \setminus \overline{B_f}(a,\sigma)) \cup B_f(a,\sigma)$.

■

In words, $B_f(a,\sigma)$ contains all fluent names that some e-proposition claims to become true when executing a in σ while $\overline{B_f}(a,\sigma)$ contains all fluent names that become false. Apart from considering new truth values for these affected fluent names, the assumption of persistence is applied to all remaining fluents.

Example 1 Given the e-propositions in (3), we have, for instance, $B_f(load, \{alive\}) = \{loaded\}$ and $\overline{B_f}(load, \{alive\}) = \{\}$; hence, $\Phi(load, \{alive\}) = \{alive, loaded\}$. The following definition provides a complete description of the transition function for the Yale Shooting scenario according to Definition 2:

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

■

Based on the concept of transition, the semantics for $\mathcal{A}$ provides a notion of entailment given a domain specification:

Definition 3 Let D be a domain description in $\mathcal{A}$. A *structure* M is a pair (σ_0, Φ) where $\sigma_0 \subseteq F_D$ —called the *initial* state—and $\Phi : A \times 2^{F_D} \mapsto 2^{F_D}$. Let $M^{[a_1,\ldots,a_m]}$ be an abbreviation for $\Phi(a_m, \Phi(a_{m-1}, \ldots, \Phi(a_1, \sigma_0)\ldots))$ then a v-proposition f `after` $[a_1, \ldots, a_m]$ is *true* in structure M iff f holds in the state $M^{[a_1,\ldots,a_m]}$.

A structure $M = (\sigma_0, \Phi)$ is then called a *model* of D iff Φ is a transition function for D and every v-proposition in V_D is true in M. A v-proposition ν is *entailed* by D iff ν is true in every model of D. ■

Example 2 Let Φ be as in (6). Then both structures $M_1 = (\{alive\}, \Phi)$ and $M_2 = (\{alive, loaded\}, \Phi)$ are models of the first v-proposition in (4). Since $M_1^{[wait,shoot]} = \{alive\}$, our second v-proposition in (4) is not true in M_1, whereas $M_2^{[wait,shoot]} = \{\}$ shows that M_2 is (the only) model for our entire example domain. Since $M_2^{[]} = \{alive, loaded\}$, this domain entails, among others, the v-proposition `initially` *loaded*, and so can be taken as a solution to the Stanford Murder Mystery. ■

3 Nondeterministic Actions: The Language $\mathcal{A}_N$

A basic assumption underlying $\mathcal{A}$ is that the effects of an action are always completely known and deterministic. As argued in the introduction, one cannot adhere to this idealistic view of the real world in general since it is impossible 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. This insight has recently led to several proposals for integrating nondeterministic actions into existing frameworks, e.g. [Brüning *et al.*, 1993; Sandewall, 1994; Kartha and Lifschitz, 1994; Kartha, 1994; Baral, 1995; Łukaszewicz and Madalińska-Bugaj, 1995]. In this section we extend the action description language $\mathcal{A}$ so that indeterminism can be explicitly represented; we call the resulting dialect $\mathcal{A}_N$.

To begin with, the possibility to express nondeterministic actions requires an extended notion of effect propositions:

Definition 4 Let F_D be a set of fluents and A_D a set of unit actions. An *effect proposition* is either of the form

$$a \texttt{ causes } e \texttt{ if } c_1, \ldots, c_n$$

(in what follows called *strict* e-proposition) or of the form

$$a \texttt{ alternatively causes } e_1, \ldots, e_m \texttt{ if } c_1, \ldots, c_n$$

(in what follows called *alternative* e-proposition) where $a \in A_D$ and $e, e_1, \ldots, e_m$ as well as $c_1, \ldots, c_n$ are fluent literals ($m, n \geq 0$). ■

Example 2 We marginally extend the *Russian Turkey* scenario as formalized in [Sandewall, 1994] and take this as the running example of this section. To this end, the set of unit actions used in Example 1 is augmented by an action called *spin*. The intended meaning is that spinning causes the gun to become randomly loaded or unloaded regardless of its state before, and if it becomes unloaded then the person operating it becomes nervous.[3] The latter is represented by the additional fluent name *nervous*. The effects of the new unit action can be specified in $\mathcal{A}_N$ using these two alternative e-propositions:

$$\begin{array}{llll} \mathit{spin} & \texttt{alternatively} & \texttt{causes} & \mathit{loaded} \\ \mathit{spin} & \texttt{alternatively} & \texttt{causes} & \overline{\mathit{loaded}}, \mathit{nervous} \end{array} \tag{7}$$

■

The intended meaning of a set of alternative e-proposition is as follows: If a is a unit action and σ a state then let

$$\left\{ \begin{array}{llllll} a & \texttt{alternatively} & \texttt{causes} & E_1 & \texttt{if} & C_1 \\ & & \vdots & & & \\ a & \texttt{alternatively} & \texttt{causes} & E_k & \texttt{if} & C_k \end{array} \right\} \tag{8}$$

be the set of all alternative e-propositions describing a such that $C_1, \ldots, C_k$ simultaneously hold in σ, where each of $E_1, C_1, \ldots, E_k, C_k$ is a finite (possibly empty) sequence of fluent

[3] For sake of simplicity, we assume the gun's cylinder consist of two chambers, exactly one of which contains a bullet. Furthermore, executing the action *load* shall be interpreted as manually selecting the chamber that is loaded.

literals. Now, if a is executed in σ then nondeterministically one $E_i \in \{E_1, \ldots, E_k\}$ becomes true in the resulting state (that is, all fluent literals $e_{i1}, \ldots, e_{im_i} = E_i$ hold in this state). For instance, if *spin* is executed in state $\{alive, loaded\}$ then, following (7), either *loaded* or else both $\overline{loaded}$ and *nervous* will be true afterwards. Hence, the possible resulting states are $\{alive, loaded\}$ and $\{alive, nervous\}$, respectively.

Recall that a set of e-propositions in the language $\mathcal{A}$ determines a unique transition function Φ. Now, however, the possibility of alternative effects forces a redefinition of this notion. At first glance one might suggest allowing the existence of several different transition functions, each of which models one of the various alternative effects of an action. Each particular model (σ_0, Φ) would then have to select among these possibilities. E.g., given the e-propositions (7), Φ could be designed such that either $\Phi(spin, \sigma) = \sigma \cup \{loaded\}$ or $\Phi(spin, \sigma) = (\sigma \setminus \{loaded\}) \cup \{nervous\}$, separately for each σ. However, if Φ is such a transition function in a particular model then the result of spinning the gun will be fixed forever regarding a particular state; e.g., it would be impossible to find a model where `initially` *loaded*, *loaded* `after` $[spin]$, and $\overline{loaded}$ `after` $[spin, spin]$ are simultaneously true. This is of course unintended.

For this reason, we adapt a standard concept for dealing with multiple possible successor states in dropping the idea of Φ being a function and using the notion of Φ as a *relation* between a pair of states and a unit action name instead such that $(\sigma, a, \sigma') \in \Phi$ whenever the application of a to σ might yield σ'. The following definition of transition in $\mathcal{A}_N$ formalizes the intended treatment of domains involving nondeterministic actions:

Definition 5 Let D be a domain description in $\mathcal{A}_N$ and let $\Phi \subseteq 2^{F_D} \times A_D \times 2^{F_D}$ be a relation. Then Φ is a *transition relation* for D iff the following conditions are satisfied for each state $\sigma \subseteq F_D$ and each unit action $a \in A_D$:

If

$$\left\{ \begin{array}{llllll} a & \texttt{alternatively} & \texttt{causes} & E_1 & \texttt{if} & C_1 \\ & & \vdots & & & \\ a & \texttt{alternatively} & \texttt{causes} & E_k & \texttt{if} & C_k \end{array} \right\} \tag{9}$$

is the set of all alternative e-propositions in E_D with unit action name a and which are applicable in σ (that is, each fluent literal occurring in C_i holds in σ, for each $1 \leq i \leq k$), then

1. If $k = 0$ then we say that a *causes* a fluent literal e *in* σ iff E_D contains an e-proposition a `causes` e `if` $c_1, \ldots, c_n$ such that each of $c_1, \ldots, c_n$ holds in σ. Define

$$\begin{array}{rcl} B_f(a, \sigma) & := & \{f \in F \mid a \text{ causes } f \text{ in } \sigma\} \\ \overline{B_f}(a, \sigma) & := & \{f \in F \mid a \text{ causes } \overline{f} \text{ in } \sigma\} \end{array}$$

   then $B_f(a, \sigma) \cap \overline{B_f}(a, \sigma)$ must be empty,[4] and we have $(\sigma, a, \sigma') \in \Phi$ iff $\sigma' = (\sigma \setminus \overline{B_f}(a, \sigma)) \cup B_f(a, \sigma)$.

2. If $k > 0$ then for each $\lambda \in \{1, \ldots, k\}$ we say that a *causes* a fluent literal e *wrt* λ *in* σ iff e occurs in E_λ or E_D contains an e-proposition a `causes` e `if` $c_1, \ldots, c_n$ such that each of $c_1, \ldots, c_n$ holds in σ. Define

$$\begin{array}{rcl} B_f(a, \lambda, \sigma) & := & \{f \in F \mid a \text{ causes } f \text{ wrt } \lambda \text{ in } \sigma\} \\ \overline{B_f}(a, \lambda, \sigma) & := & \{f \in F \mid a \text{ causes } \overline{f} \text{ wrt } \lambda \text{ in } \sigma\} \end{array}$$

[4] Otherwise no transition relation for D exists.

then $B_f(a,\lambda,\sigma) \cap \overline{B_f}(a,\lambda,\sigma)$ must be empty for each $\lambda \in \{1,\ldots,k\}$,[5] and we have $(\sigma,a,\sigma') \in \Phi$ iff there exists some $\lambda \in \{1,\ldots,k\}$ such that $\sigma' = (\sigma \setminus \overline{B_f}(a,\lambda,\sigma)) \cup B_f(a,\lambda,\sigma)$.

■

In words, a possible successor state is constructed by accounting for each strict e-proposition; by selecting one collection of effects E_λ (where $\lambda \in \{1,\ldots,k\}$) among the applicable alternatives, (9); and by applying the persistence assumption to all remaining fluents.

Example 3 For the Russian Turkey scenario we obtain the following transition relation Φ:

$$\begin{array}{lll}
(\sigma, spin, \sigma') \in \Phi & \text{iff} & \sigma' = \sigma \cup \{loaded\} \text{ or } \sigma' = (\sigma \setminus \{loaded\}) \cup \{nervous\} \\
(\sigma, wait, \sigma') \in \Phi & \text{iff} & \sigma' = \sigma \\
(\sigma, load, \sigma') \in \Phi & \text{iff} & \sigma' = \sigma \cup \{loaded\} \\
(\sigma, shoot, \sigma') \in \Phi & \text{iff} & \sigma' = \begin{cases} \sigma \setminus \{loaded, alive\}, & \text{if } loaded \in \sigma \\ \sigma, & \text{otherwise.} \end{cases}
\end{array} \tag{10}$$

The reader is invited to verify that this relation satisfies the conditions of Definition 5 wrt the e-propositions (2) and (7). ■

Having defined the notion of transition, we now concentrate on defining the concept of a model in $\mathcal{A}_N$. The purpose of models is, in general, to provide a possible view of the real world according to given knowledge. In $\mathcal{A}$, where no indeterministic and randomized effects are allowed, models differ only in their initial state once a transition function is fixed. Now, however, each concrete model needs additionally state which particular effect occurs whenever alternatives exist. An additional component for each model, namely, a function φ, serves this purpose. More precisely, φ maps action sequences $[a_1,\ldots,a_n]$ to states, stating that the actual outcome of applying $[a_1,\ldots,a_n]$ to the initial state in the model at hand is $\varphi([a_1,\ldots,a_n])$. For instance, if the initial state is known to be $\{alive\}$ and we are interested in the consequences of executing the sequence of unit 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. This is formally captured by requiring that each model have either $\varphi([load,spin]) = \{alive, loaded\}$ or else $\varphi([load,spin]) = \{alive, nervous\}$. Now, 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 and the hunter became nervous after $[load,spin]$. This is illustrated in Figure 1. Note that we would be unable to obtain this conclusion without 'recording,' by using φ, the actual outcome of the nondeterministic action, *spin*.

Definition 6 Let D be a domain description in $\mathcal{A}_N$. A *structure* is a triple (σ_0,Φ,φ) where $\sigma_0 \subseteq F_D$, $\Phi \subseteq 2^{F_D} \times A_D \times 2^{F_D}$ and $\varphi : A_D^* \mapsto 2^{F_D}$ such that[6]

1. $\varphi([\,]) = \sigma_0$ and
2. $(\varphi([a_1,\ldots,a_n]), a_{n+1}, \varphi([a_1,\ldots,a_n,a_{n+1}])) \in \Phi$ for each $a_1,\ldots,a_n,a_{n+1} \in A_D$ ($n \geq 0$).

[5] If not then, as before, no transition relation for D exists.

[6] By A_D^* we denote the set of all finite lists, including the empty one, whose elements are chosen from A_D.

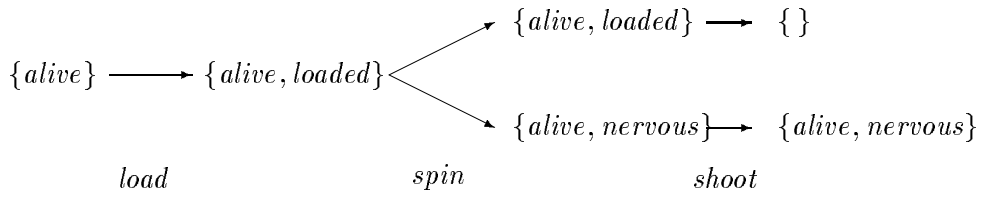

Figure 1: Two possible developments in the Russian Turkey scenario given the initial state $\{alive\}$. On the basis of the additional observation that the turkey is alive after loading, spinning and shooting, we can exclude the upper branch and, thus, safely conclude that the gun was unloaded after $[load, spin]$.

A v-proposition ℓ `after` $[a_1, \ldots, a_n]$ is *true* in such a structure iff ℓ holds in $\varphi([a_1, \ldots, a_n])$ ($n \geq 0$). Such a structure is a *model* of D iff Φ is a transition relation for D and all v-propositions in V_D are true. A v-proposition ν is *entailed* by D iff ν is true in every model of D. ■

In words, the third component φ both respects the transition relation Φ and is now used to validate the given v-propositions. We call a domain description in $\mathcal{A}_N$ *consistent* if it has a model.

Example 4 A structure $(\sigma_0, \Phi, \varphi)$ is a model of (2) and (7) along with the two v-propositions

$$\begin{array}{l} \texttt{initially}\ \ alive \\ alive\ \ \texttt{after}\ \ [load, spin, shoot] \end{array} \tag{11}$$

iff Φ is as in (10), φ satisfies clause 1 and 2 of Definition 6, and $alive \in \sigma_0$ as well as $alive \in \varphi([load, spin, shoot])$. Then $\overline{loaded} \in \varphi([load, spin])$ holds in each such model; hence, $\overline{loaded}$ `after` $[load, spin]$ is entailed (see again Figure 1). ■

As does $\mathcal{A}$, our extended language supports reasoning about so-called *counterfactual* action sequences due to the fact that the model component φ is defined for any sequence of unit actions. To illustrate this, consider the following extension of Example 2, motivated by a scene in a Pierre Richard movie [Richard etal., 1988].[7] An additional fluent name, *broken*, is used to describe the state of a vase. Furthermore, the action *shoot* is replaced by the unit actions *shoot-at-pierre* and *shoot-at-vase*, respectively, along with the e-propositions

$$\begin{array}{llll} shoot\text{-}at\text{-}pierre & \texttt{causes} & \overline{loaded} & \\ shoot\text{-}at\text{-}pierre & \texttt{causes} & \overline{alive} & \texttt{if}\ \ loaded \\ shoot\text{-}at\text{-}vase & \texttt{causes} & \overline{loaded} & \\ shoot\text{-}at\text{-}vase & \texttt{causes} & broken & \texttt{if}\ \ loaded \end{array} \tag{12}$$

[7] The scene is as follows: Pierre Richard pretends to intend to commit suicide with a, as he believes, toy gun. To show it was just a joke, he aims at a vase and pulls the trigger. The vase shatters, and Pierre faints—he obviously drew a conclusion about a counterfactual action sequence.

Now, assume given the v-propositions

$$\begin{array}{l} \texttt{initially}\ \ \mathit{alive} \\ \texttt{initially}\ \ \overline{\mathit{broken}} \\ \mathit{broken}\ \ \texttt{after}\ \ [\mathit{spin}, \mathit{shoot\text{-}at\text{-}vase}] \end{array} \tag{13}$$

According to the e-propositions in (12), each model $(\sigma_0, \Phi, \varphi)$ requires $\mathit{loaded} \in \varphi([\mathit{spin}])$ since otherwise the vase could not have been destroyed. Hence, it is plausible to conclude that had we shot at Pierre instead then he would not have survived this. Definition 6 supports this conclusion formally: The reader is invited to verify that the domain consisting of e-propositions (7) and (12) plus the above v-propositions, (13), entails

$$\overline{\mathit{alive}}\ \ \texttt{after}\ \ [\mathit{spin}, \mathit{shoot\text{-}at\text{-}pierre}] \tag{14}$$

Since the two action sequences used in this v-proposition and in (13), respectively, are incompatible, this example shows that and how reasoning about counterfactuals is supported.

4 Concurrent Actions and Solving Conflicts

Since the problems we discuss in this section become more obvious in the context of the simultaneous execution of actions, we first extend our view to concurrency. We will present different ways of interpreting descriptions of actions which may be executed concurrently. To illustrate our exposition, we use the terms of $\mathcal{A}$ (and, later on, $\mathcal{A}_C$); nevertheless, the differences we identify provide a classification of other languages describing possibly concurrent actions as well.

4.1 Explicit Information about Concurrent Execution

Suppose a rather complex description of a part of the world has to be constructed, where arbitrary unit actions may be executed concurrently. Because of the combinatorial explosion it is obviously impractical to describe the effects of all possible combinations of unit actions. Therefore, it is necessary to infer the effects of compound actions from the descriptions given separately for the various actions involved. Combining these action descriptions may however yield a contradiction among their effects.[8] In terms of the Action Description Language $\mathcal{A}$ this means that for the corresponding sets $B_f \cap \overline{B_f} \neq \{\}$ holds (c.f. Definition 2).

There are several ways of dealing with and inferring the effects of a compound action from descriptions of the involved actions which propose contradictory effects. Therefore, languages describing actions can be classified according to the explicit and implicit methods, respectively, they use to draw these conclusions.

Explicit methods provide further information of the effects of certain compound actions; they are also used to state the difference between the actual effects of a concurrent execution of several actions and the effects of these unit actions when executed alone. In terms of the Action Description Language, additional e-propositions may

1. add a fluent to B_f or $\overline{B_f}$: obviously, the set $B_f \cap \overline{B_f}$ will remain nonempty and, hence, no conflicts will be solved;

[8] This problem might of course occur even without concurrency involved, namely, if several descriptions, i.e., e-propositions, 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 $\mathcal{A}$ and $\mathcal{A}_N$, for instance, define the whole domain description to be inconsistent as it does not admit a proper notion of transition.

2. remove a fluent from B_f or $\overline{B_f}$: this allows to remove predicted conflicts, but not to redefine facts not mentioned by the unit action descriptions (the approach in [Lin and Shoham, 1992] uses this method by "cancelling" effects in specific cases),

3. add or remove a fluent from B_f or $\overline{B_f}$: this enables one to arbitrarily modify B_f and $\overline{B_f}$ (used, for instance, in $\mathcal{A}_C$, our language $\mathcal{A}_{NCC}$, and in State Event Logic [Große, 1994]).

Since an extension of $\mathcal{A}$ to concurrent actions, called $\mathcal{A}_C$, has recently been introduced [Baral and Gelfond, 1993], which uses the latter, most powerful method for stating differing effects of actions as regards their concurrent execution, we use this approach to illustrate our following discussion and adopt it when extending our approach to deal with concurrency.

4.2 The Language $\mathcal{A}_C$

We briefly review the concepts underlying the language $\mathcal{A}_C$ as defined in [Baral and Gelfond, 1993] by pointing out the corresponding extensions of $\mathcal{A}$. In either e- or v-propositions of a domain description, actions are now represented by non-empty, finite subsets of the given set of unit actions A_D, with the intended meaning that all of the elements are executed concurrently. Such actions are called *compound actions* to distinguish them from the unit actions.

Example 3 Assume 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. A 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 $\mathcal{A}_C$, we take the two sets $A_{D_3} = \{activate, pull, run\_into\}$ and $F_{D_3} = \{open, sleeps, hurt\}$. The initial situation shall be partially described by the v-proposition `initially` *sleeps*, and the effects of the actions can be specified by the e-propositions

$$\begin{array}{lllll} \{activate\} & \texttt{causes} & \overline{sleeps} & & \\ \{run\_into\} & \texttt{causes} & hurt & \texttt{if} & \overline{open} \\ \{pull\} & \texttt{causes} & \overline{open} & & \\ \{activate, run\_into\} & \texttt{causes} & open & & \\ \{activate, run\_into\} & \texttt{causes} & \overline{hurt} & \texttt{if} & \overline{hurt} \end{array} \qquad (15)$$

Informally, the last e-proposition is needed to limit the application of the second one (this way of restricting applicability of (less specific) e-propositions is called to *overrule* an e-proposition). Let D_3 denote the domain description given by these propositions. ■

Overruling more general action descriptions by more specific ones is formalized by modifying Definition 2 as follows:[9] If a is an action, ℓ a fluent literal and σ a state then we say that a *causes* ℓ *in* σ iff there is an action b such that a causes ℓ *by* b in σ. We say that a causes ℓ by b in σ iff

1. $b \subseteq a$;

2. there is an e-proposition b `causes` ℓ `if` $c_1, \ldots, c_n$ such that each of $c_1, \ldots, c_n$ holds in σ; and

[9] The following definition differs slightly from the definition given in [Baral and Gelfond, 1993], which is circular; we assume that our's is what the authors actually intended.

3. there is no action c such that $b \subset c$ and a causes $\overline{\ell}$ by c in σ.

If but 3. holds then the e-proposition in 2. is said to be *overruled* (by action c).

Now, if, based on this extended notion, $B_f(a,\sigma)$ and $\overline{B_f}(a,\sigma)$ are defined accordingly (c.f. Definition 2) and share elements then the corresponding transition function Φ is taken to be undefined for the argument (a,σ); otherwise, $\Phi(a,\sigma) = (\sigma \setminus \overline{B_f}(a,\sigma)) \cup B_f(a,\sigma)$, as before.

Example 5 The transition function determined by the e-propositions in our domain description D_3, (15), is defined as follows. Let σ be an arbitrary state then

$$
\begin{array}{lll}
\Phi(\{\},\sigma) & = \sigma & \\
\Phi(\{\mathit{run\_into}\},\sigma) & = \sigma, & \text{if} \quad \mathit{open} \in \sigma \\
\Phi(\{\mathit{run\_into}\},\sigma) & = \sigma \cup \{\mathit{hurt}\}, & \text{if} \quad \mathit{open} \notin \sigma \\
\Phi(\{\mathit{pull}\},\sigma) & = \sigma \setminus \{\mathit{open}\} & \\
\Phi(\{\mathit{activate}\},\sigma) & = \sigma \setminus \{\mathit{sleeps}\} & \\
\Phi(\{\mathit{activate},\mathit{pull}\},\sigma) & = \sigma \setminus \{\mathit{sleeps},\mathit{open}\} & \\
\Phi(\{\mathit{run\_into},\mathit{pull}\},\sigma) & = \sigma, & \text{if} \quad \mathit{open} \in \sigma \\
\Phi(\{\mathit{run\_into},\mathit{pull}\},\sigma) & = \sigma \cup \{\mathit{hurt}\}, & \text{if} \quad \mathit{open} \notin \sigma \\
\Phi(\{\mathit{activate},\mathit{run\_into}\},\sigma) & = \sigma \setminus \{\mathit{sleeps}\} \cup \{\mathit{open}\} & \\
\Phi(\{\mathit{activate},\mathit{run\_into},\mathit{pull}\},\sigma) & \text{is undefined} &
\end{array}
$$

D_3 has four models, viz

$$
\begin{array}{ll}
(\{\mathit{sleeps}\},\Phi) & (\{\mathit{open},\mathit{sleeps}\},\Phi) \\
(\{\mathit{sleeps},\mathit{hurt}\},\Phi) & (\{\mathit{open},\mathit{sleeps},\mathit{hurt}\},\Phi)
\end{array}
\tag{16}
$$

If, for instance, v-proposition $\overline{\mathit{hurt}}$ `after` $\{\mathit{run\_into}\}$ is added to D_3 then the only remaining model is $(\{\mathit{open},\mathit{sleeps}\},\Phi)$ since for all other structures in (16) we find that $\mathit{hurt} \in \Phi(\{\mathit{run\_into}\},\sigma_0)$. Hence, for example, the v-proposition `initially` *open* is entailed by this extended domain. ■

Note that our example domain can be modeled only by adding *and* removing fluents from B_f or $\overline{B_f}$ (c.f. Subsection 4.1):

Example 6 Let σ be an arbitrary state in the domain of our Example 3. The e-proposition $\{\mathit{activate},\mathit{run\_into}\}$ `causes` *open* adds the fluent *open* to the set $B_f(\{\mathit{activate},\mathit{run\_into}\},\sigma)$[10] while the e-proposition $\{\mathit{activate},\mathit{run\_into}\}$ `causes` $\overline{\mathit{hurt}}$ `if` $\overline{\mathit{hurt}}$ removes the fluent *hurt* from $B_f(\{\mathit{activate},\mathit{run\_into}\},\sigma)$ by overruling $\{\mathit{run\_into}\}$ `causes` *hurt* `if` $\overline{\mathit{open}}$. ■

4.3 Implicit Indeterminism: Interpreting Contradictions

After having introduced our basic concept for (explicitly) representing indeterminism in Section 3 and having adopted an adequate formalism for representing concurrent actions, we are now able to discuss and solve the problem of contradictory specifications of dynamic systems. Suppose the effects are not defined explicitly for all possible compound actions. In this case, as argued above, it may happen that certain actions still are proposed to have contradictory effects.

[10] Note that fluent name *open* is not mentioned by either of the descriptions $\{\mathit{activate}\}$ `causes` $\overline{\mathit{sleeps}}$ and $\{\mathit{run\_into}\}$ `causes` *hurt* `if` $\overline{\mathit{open}}$.

From the point of view underlying $\mathcal{A}_C$, this indicates that these actions are not executable in the world. A typical example employed to justify this interpretation of contradictions is as follows: The door is open after it has been opened, and the door is not open after it has been closed; since a door cannot be open and not open at the same time, it is impossible to simultaneously open and close the door. The implicit assumption underlying this argument is that e-propositions do not *describe* concrete actions but *assign* (action-) names to the achievement of effects: "to open" means to do *something* that results the door being open. Hence, "to open" does not refer to a concrete action but to the achievement of a certain effect—then, of course, it is impossible to have both simultaneously.

In contrast, our idea is that e-propositions describe concrete actions, and that all actions (i.e., in the end, the decision to execute an action) can be performed concurrently in any situation, sometimes, maybe, without being successful in achieving a certain effect. From this point of view, the occurrence of actions which are proposed to have contradictory effects when executed simultaneously indicates that the descriptions of their effects are incorrect.[11] Of course, the formal description of an existing complex scenario may be incorrect or incomplete. In many applications, however, it is not desirable that an intelligent agent stops reasoning as soon as he detects an error in the description of the scenario he is acting in (e.g., if he uses the semantics of $\mathcal{A}_C$ and observes a compound action—or is aksed about the effects of it—which is defined to be impossible in the semantics).

To illustrate this, recall our domain description D_3. The e-propositions describing the effects of the elements of $\{activate, pull, run\_into\}$ claim both $open$ and $\overline{open}$. In such cases, depending on the chosen interpretation and the extent of certainty required, one has to regard as unreliable

1. the whole domain description (as in State-Event Logic [Große, 1994]),
2. the whole situation,
3. the effects of the conflicting actions, or
4. the contradictory fluents.

It is the latter, weakest condition which we propose in this paper. This follows the idea of still believing in all information which do not cause the contradiction.

Example 7 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_3 proposes both $open$ and $\overline{open}$ to be an effect of the corresponding compound action, $\{activate, pull, run\_into\}$. Hence, D_3 is incomplete with respect to the world it describes. In fact, without further information we cannot say whether the door will be closed after executing this action or not. However, it is reasonable to assume the dog will not sleep afterwards since we know that $\{activate\}$ `causes` $\overline{sleeps}$ and there is no proposition contradicting this. Using the semantics of $\mathcal{A}_C$ it cannot be inferred that the dog does not sleep after executing $\{activate, pull, run\_into\}$ since no successor state σ' exists. ■

In general, whenever a local inconsistency occurs, this causes the entire set of simultaneously executed actions to be contradictory. As an extreme case, imagine two agents in Germany

[11] 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 $open$; hence, involving (implicit) uncertainty.

executing the above action and, concurrently, another agent in China switching off a light. Again, by $\mathcal{A}_C$ it cannot be inferred that the light is switched off in China because the description used proposes contradictory states of a door somewhere in Germany. Yet it seems reasonable to draw some conclusions about the resulting state instead of declaring it to be totally undefined. Preventing global inconsistency in case of local conflicts is our underlying intention here.

We therefore weaken the basic assumption which says that $\Phi(a,\sigma)$ is undefined whenever the corresponding sets $B_f(a,\sigma)$ and $\overline{B_f}(a,\sigma)$ share one or more elements. To this end, we adopt the concept of nondeterminism developed in the preceding section. 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, that is, if the corresponding intersection $B_f(a,\sigma) \cap \overline{B_f}(a,\sigma)$ is not empty, then each combination of truth values of the controversial fluent names determines exactly one possible successor state.

The following definition of the language $\mathcal{A}_{NCC}$ makes these ideas manifest. Syntactically, domain descriptions in our new language are specified using a combination of $\mathcal{A}_N$ and $\mathcal{A}_C$; that is, we take the syntax of $\mathcal{A}_N$ and extend it by allowing to formalize compound actions. Then the formal definition of transition is as follows:

Definition 7 Let D be a domain description in $\mathcal{A}_{NCC}$ and let $\Phi \subseteq 2^{F_D} \times 2^{A_D} \times 2^{F_D}$ be a relation. Then Φ is a *transition relation* for D if for each $\sigma, \sigma' \subseteq F_D$ and $a \subseteq A_D$ we have $(\sigma, a, \sigma') \in \Phi$ iff the following holds:

For each $b \subseteq a$ let

$$\left\{ \begin{array}{llllll} b & \texttt{alternatively} & \texttt{causes} & E_1^b & \texttt{if} & C_1^b \\ & & \vdots & & & \\ b & \texttt{alternatively} & \texttt{causes} & E_{k_b}^b & \texttt{if} & C_{k_b}^b \end{array} \right\}$$

be the set of all alternative e-propositions in E_D for action b and which are applicable in σ (that is, each fluent literal occurring in C_i^b holds in σ, for each $1 \leq i \leq k_b$). Furthermore, let $b_1, \ldots, b_m$ be all actions $b_j \subseteq a$ which satisfy $k_{b_j} > 0$. Then we can find a selection $\Lambda = \lambda_{b_1}, \ldots, \lambda_{b_m}$, where $\lambda_{b_j} \in \{1, \ldots, k_{b_j}\}$ $(1 \leq j \leq m)$, such that the following holds:

Let e be a fluent literal. We say that a *causes* e *wrt* Λ *in* σ if there is an action $b \subseteq A_D$ such that a causes e *by* b (wrt Λ in σ). We say that a causes e by b wrt Λ in σ iff

1. $b \subseteq a$;
2. e occurs in $E_{\lambda_b}^b$ (in case $k_b > 0$) or E_D contains an e-proposition b `causes` e `if` $c_1, \ldots, c_n$ such that each of $c_1, \ldots, c_n$ holds in σ; and
3. there is no action c such that $b \subset c$ and a causes $\overline{e}$ by c wrt Λ in σ.

Define

$$\begin{array}{rcl} B_f(a,\Lambda,\sigma) & := & \{ f \in F \mid a \text{ causes } f \text{ wrt } \Lambda \text{ in } \sigma \} \\ \overline{B_f}(a,\Lambda,\sigma) & := & \{ f \in F \mid a \text{ causes } \overline{f} \text{ wrt } \Lambda \text{ in } \sigma \}, \end{array}$$

then there exists some $B_f^{\cap}(a,\Lambda,\sigma) \subseteq B_f(a,\Lambda,\sigma) \cap \overline{B_f}(a,\Lambda,\sigma)$ such that

$$\sigma' = (\sigma \setminus \overline{B_f}(a,\Lambda,\sigma)) \cup (B_f(a,\Lambda,\sigma) \setminus B_f^{\cap}(a,\Lambda,\sigma)). \qquad (17)$$

■

In words, a possible successor state is obtained by first randomly selecting among the applicable alternative e-propositions, separately for each compound action $b \subseteq a$. Afterwards, we proceed as in $\mathcal{A}_C$ but in case a conflict occurs, where we take any truth value distribution $B_f^{\cap}(a, \Lambda, \sigma)$ among the disputed fluent names when computing a possible σ' via (17).

The notion of model and entailment are adopted from our language $\mathcal{A}_N$ (c.f. Definition 6):

Definition 8 Let D be a domain description in $\mathcal{A}_{NCC}$. A *structure* is a triple $(\sigma_0, \Phi, \varphi)$ where $\sigma_0 \subseteq F_D$, $\Phi \subseteq 2^{F_D} \times 2^{A_D} \times 2^{F_D}$ and $\varphi : (2^{A_D})^* \mapsto 2^{F_D}$ such that[12]

1. $\varphi([\,]) = \sigma_0$ and
2. $(\varphi([a_1, \ldots, a_n]), a_{n+1}, \varphi([a_1, \ldots, a_n, a_{n+1}])) \in \Phi$ for each $a_1, \ldots, a_n, a_{n+1} \in 2^{A_D}$ $(n \geq 0)$.

A v-proposition ℓ `after` $[a_1, \ldots, a_n]$ is *true* in such a structure iff ℓ holds in $\varphi([a_1, \ldots, a_n])$ $(n \geq 0)$. Such a structure is a *model* of D iff Φ is a transition relation for D and all v-propositions in V_D are true. A v-proposition ν is *entailed* by D iff ν is true in every model of D. ∎

Example 8 If our domain description D_3 is augmented by either one of the v-propositions *open* `after` $\{activate, pull, run\_into\}$ or $\overline{open}$ `after` $\{activate, pull, run\_into\}$ then both extended domains have models (with different functions φ) according to the semantics of $\mathcal{A}_{NCC}$. On the other hand, if D_3 is augmented by *sleeps* `after` $\{activate, pull, run\_into\}$ then there is no model wrt $\mathcal{A}_{NCC}$. Hence, as intended we can conclude our domain entails the v-proposition $\overline{sleeps}$ `after` $\{activate, pull, run\_into\}$. ∎

Note that $\mathcal{A}_{NCC}$ does not distinguish between intentionally expressed nondeterminism of actions and our interpretation of contradictory specified actions. For instance, D_3 could be augmented by the e-propositions $\{activate\}$ `causes` *bark* and $\{activate\}$ `causes` $\overline{bark}$ for describing that the dog possibly starts or stops barking when the door opener is activated. The very same is expressed by the two alternative e-propositions $\{activate\}$ `alternatively causes` *bark* and $\{activate\}$ `alternatively causes` $\overline{bark}$. In fact, for someone reasoning about a domain description it makes no difference whether the designer of this domain description was aware of the uncertainty of the described effects or not.

5 Translating $\mathcal{A}_{NCC}$ into $\mathcal{FC}_{NCC}$

In the second part of this paper, we show how domain descriptions in our new language $\mathcal{A}_{NCC}$ may be encoded as logic programs. While in the preceding sections the sets of elements underlying a domain description were of arbitrary, possibly infinite size, we need to restrict ourselves to finite sets of fluent names, unit actions, and e- and v-propositions in order to obtain a finite logic program. The approach we follow here is based on the reification of complete situation descriptions by treating them as terms [Hölldobler and Schneeberger, 1990]. To this end, each atomic fluent that holds in a state is formally represented by a term (a so-called *fluent term*), and these fluent terms are connected by a special binary function symbol, written $\circ$. For instance, the term $(open \circ sleeps) \circ hurt$ describes the state wrt Example 3 where the door is open, the dog is sleeping, and the protagonist has hurt himself. Intuitively, the order in which

[12] The set $(2^{A_D})^*$ contains all finite lists whose elements are finite, non-empty subsets of A_D.

the various fluent terms are connected is irrelevant as regards the state to be represented. Hence our connection function has some special properties, which are formalized using the following equational theory [Hölldobler and Schneeberger, 1990]:

$$\begin{array}{rrcll} \forall X,Y,Z. & (X \circ Y) \circ Z & = & X \circ (Y \circ Z) & \text{(associativity)} \\ \forall X,Y. & X \circ Y & = & Y \circ X & \text{(commutativity)} \\ \forall X. & X \circ \emptyset & = & X & \text{(unit element)} \end{array}$$

where the constant $\emptyset$ denotes a unit element for $\circ$, which corresponds to an empty collection of fluent terms. These three axioms (AC1, for short) are used as the underlying equational theory for our logic program.[13] Therefore, the special function symbol $\circ$ will be referred to as the *AC1-function*, and a term consisting of fluent terms that are connected by this function will be referred to as an *AC1-term*. In what follows, we use the equality predicate $=_{\mathrm{AC1}}$ in program clauses to illustrate that equality should always be related to the axioms above. Due to the law of associativity, we can omit parenthesis on the level of $\circ$ in any AC1-term.

On the basis of representing a situation by a collection of fluent terms, the execution of actions is modeled by manipulating such collections. For this reason, we call the underlying approach *fluent calculus*. Aside from being closely related, in its basic form, to the Linear Connection Method [Bibel, 1986] and reasoning about actions based on Linear Logic [Girard, 1987; Masseron *et al.*, 1993], the fluent calculus has recently been shown to successfully deal with the ramification problem [Thielscher, 1995a; Thielscher, 1996b].

In the following subsection, 5.1, we describe how to construct a logic program corresponding to a domain description in $\mathcal{A}_{NCC}$. In Subsection 5.2, we discuss the semantics of the resulting program by applying the standard completion procedure [Clark, 1978] augmented by a special treatment of the underlying equational theory [Jaffar *et al.*, 1984; Shepherdson, 1992]. In Subsection 5.3, we then prove soundness and completeness of the equational logic program (by taking the extended completion semantics) wrt the semantics of $\mathcal{A}_{NCC}$. Finally, in Subsection 5.4 we discuss the applicability of a special resolution variant, namely, SLDENF-resolution [Shepherdson, 1992; Thielscher, 1996a], to our logic program. We assume the reader be familiar with the basic concepts of normal logic programs (i.e., logic programs augmented by negation-as-failure) as described, e.g, in the textbook [Lloyd, 1987]. We use a PROLOG-like syntax in denoting constants and predicates by lower case letters and variables 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.

5.1 The Equational Logic Program

Let D be a domain description in $\mathcal{A}_{NCC}$ based on fluent names F_D. For a proper representation of negative fluent literals, we introduce a unary function whose application to a term representing a fluent name indicates the negation of the latter. We will denote this function illustratively by a bar on top of its argument, like negation has been denoted in the action description languages. Formally, we employ a function τ mapping sequences of fluent literals to AC1-terms as follows:

$$\begin{array}{rcl} \tau(f) & := & f \\ \tau(\overline{f}) & := & \overline{\tau(f)} \\ \tau(\ell_1, \ldots, \ell_n) & := & \tau(\ell_1) \circ \ldots \circ \tau(\ell_n) \end{array}$$

[13] While it suffices to consider these axioms in view of a suitable resolution procedure (see Subsection 5.4), the standard axioms of equality plus axioms allowing to derive inequalities are additionally required when discussing an adequate semantics for our program (see Subsection 5.2).

where $f \in F_D$, $\ell_i \in F_D \cup \{\overline{f} \mid f \in F_D\}$, and in case $m = 0$ the function value of τ is the unit element $\emptyset$ of $\circ$.

States over a fixed set of fluent names F_D are represented by an AC1-term as follows:

$$\gamma_D(\{f_1, \ldots, f_m\}) \;:=\; f_1 \circ \ldots \circ f_m \circ \overline{f_{m+1}} \circ \ldots \circ \overline{f_n}\,, \tag{18}$$

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

Finally, we also employ our AC1-function to represent compound actions, viz by simply connecting the unit action names, taken as terms:

$$\mu(\{a_1, \ldots, a_k\}) \;:=\; a_1 \circ \cdots \circ a_k \tag{19}$$

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

We are now prepared for translating domain descriptions in $\mathcal{A}_{NCC}$ into a set of logic program clauses. To begin with, we introduce, for each fluent name $f \in F_D$, a separate unit clause to relate it to its counterpart $\overline{f}$:

$$FLUENT_D \;:=\; \{\, complement(\tau(f, \overline{f})). \mid f \in F_D \,\} \tag{20}$$

Let E_D be a given set of e-propositions then for each strict e-proposition we use an instance of the ternary predicate *eprop* stating the action name, the effect, and the conditions:

$$\begin{aligned} EPROP_D \;:=\; \{\, &eprop(\mu(a), \tau(\ell), \tau(c_1, \ldots, c_n)). \mid \\ &a \;\texttt{causes}\; \ell \;\texttt{if}\; c_1, \ldots, c_n \in E_D \,\} \end{aligned} \tag{21}$$

Analogously, alternative e-propositions describing possible effects of nondeterministic actions are encoded using the ternary predicate *alteprop*:

$$\begin{aligned} ALTEPROP_D \;:=\; \{\, &alteprop(\mu(a), \tau(e_1, \ldots, e_m), \tau(c_1, \ldots, c_n)). \mid \\ &a \;\texttt{alternatively causes}\; e_1, \ldots, e_m \;\texttt{if}\; c_1, \ldots, c_n \in E_D \,\} \end{aligned} \tag{22}$$

Example 4 Let D_4 denote the amalgamation of the two domains described in Example 2 and Example 3, respectively. We then have, e.g., $\tau(sleeps, \overline{open}) = sleeps \circ \overline{open}$, $\gamma_{D_4}(\{alive, sleeps, open\}) = alive \circ sleeps \circ open \circ \overline{loaded} \circ \overline{nervous} \circ \overline{hurt}$, and, for a compound action, $\mu_{D_4}(\{activate, run\_into\}) = activate \circ run\_into$.

The program clauses $FLUENT_{D_4} \cup EPROP_{D_4} \cup ALTEPROP_{D_4}$ are as follows:

$complement(loaded \circ \overline{loaded})$.
$complement(alive \circ \overline{alive})$.
$complement(nervous \circ \overline{nervous})$.
$complement(sleeps \circ awake)$.
$complement(hurt \circ \overline{hurt})$.
$complement(open \circ \overline{open})$.

$eprop(load, loaded, \emptyset)$.
$eprop(shoot, \overline{loaded}, \emptyset)$.
$eprop(shoot, \overline{alive}, loaded)$.
$alteprop(spin, loaded, \emptyset)$.
$alteprop(spin, \overline{loaded} \circ nervous, \emptyset)$.

$$
\begin{array}{l}
\mathit{eprop}(\mathit{activate}, \mathit{awake}, \emptyset)\,. \\
\mathit{eprop}(\mathit{run\_into}, \mathit{hurt}, \overline{\mathit{open}})\,. \\
\mathit{eprop}(\mathit{pull}, \overline{\mathit{open}}, \emptyset)\,. \\
\mathit{eprop}(\mathit{activate} \circ \mathit{run\_into}, \mathit{open}, \emptyset)\,. \\
\mathit{eprop}(\mathit{activate} \circ \mathit{run\_into}, \overline{\mathit{hurt}}, \overline{\mathit{hurt}})\,.
\end{array}
$$

■

To encode the concept of transition of $\mathcal{A}_{NCC}$, we use a ternary predicate $\mathit{action}(i, a, h)$ stating that executing action a in state i possibly yields state h. Following Definition 7, a possible successor state is obtained by taking into account all applicable, strict e-propositions and a set consisting of exactly one element of each set of applicable alternative e-propositions describing an identical subset of a. The *action* predicate is defined as follows:

$$
\begin{array}{lll}
\mathit{action}(I, A, H) & \leftarrow & \mathit{setofalternatives}(A, S, I)\,, \\
& & \neg\,\mathit{impossible}(H, I, A, S)\,, \\
& & \neg\,\mathit{unfounded}(H, I, A, S)\,, \\
& & \neg\,\mathit{inconsistent}(H)\,.
\end{array}
\tag{23}
$$

The intended meaning of this clause is the following: Let i be an AC1-term representing a state (c.f. (18)) and a be an AC1-term representing a (compound) action (c.f. (19)).[14]

1. Let s be an AC1-term of the form $(b_1, e_1, c_1) \circ \ldots \circ (b_m, e_m, c_m)$ where each subterm (b_j, e_j, c_j) represents an alternative e-proposition b_j `alternatively causes` e_j `if` c_j. An instance $\mathit{setofalternatives}(a, s, i)$ is then intended to be true if s represents a complete collection of alternative e-propositions wrt state i and action a as required in Definition 7. To this end, we introduce the following program clause:

$$
\begin{array}{lll}
\mathit{setofalternatives}(A, S, I) & \leftarrow & \neg\,\mathit{overrepresented}(S)\,, \\
& & \neg\,\mathit{underrepresented}(A, S, I)\,.
\end{array}
\tag{24}
$$

   In words, the middle argument of *setofalternatives* contains a representation of not more than and also at least one element of each set of applicable alternative e-propositions. The predicates *overrepresented* and *underrepresented* are defined as follows:

$$
\begin{array}{l}
\mathit{overrepresented}((A, U, X) \circ (A, V, Y) \circ R)\,. \\
\\
\begin{array}{lll}
\mathit{underrepresented}(A \circ B, S, C \circ J) & \leftarrow & \mathit{alteprop}(A, E, C)\,, \\
& & \neg\,\mathit{represented}(A, S)\,.
\end{array} \\
\\
\mathit{represented}(A, (A, E, C) \circ R)\,.
\end{array}
\tag{25}
$$

   In words, $\mathit{overrepresented}(s)$ is true if s contains two (or more) alternatives for the same action, A, while $\mathit{underrepresented}(a \circ b, s, i)$ is true if there exists an applicable

[14] For sake of readability, in the following description we sometimes identify a term t which represents a state (or a sequence of fluent literals or a compound action, respectively) with the state $\gamma_D^{-1}(t)$ itself (or with $\tau^{-1}(t)$ or $\mu^{-1}(t)$, respectively).

(wrt state i)[15] alternative e-proposition for action a but s does not include a triple representing an alternative for this particular action.

2. An instance $impossible(h, i, a, s)$ is intended to be true if h, which is intended to represent a possible successor state of i wrt a, contains a fluent literal which is claimed to be true by some overruled e-proposition (either a strict one, or an alternative one that has been selected via s) while the negation of this fluent literal is claimed by some non-overruled e-proposition. (In case both a fluent literal and its negation are claimed to be true by two non-overruled e-propositions (strict or selected), either of them can be true in a resulting state. The latter encodes our way of solving conflicts, as formalized in Definition 7):

$$\begin{array}{rcl} impossible(F \circ H, I, A, S) & \leftarrow & overruled(F, I, \emptyset, A, S), \\ & & complement(F \circ G), \\ & & \neg overruled(G, I, \emptyset, A, S). \end{array} \tag{26}$$

An instance $overruled(h, i, b, a, s)$ is intended to be true if h contains a fluent literal that is supposed to be true by some applicable (wrt state i) e-proposition (either a strict one, or one that has been selected via s) which is overruled by a (strict or selected) e-proposition for some action c such that $c \supset b$ and $c \subseteq a$. The clauses defining *overruled* follow Definition 7:

$$\begin{array}{l} overruled(F \circ H, C \circ J, A, A \circ B \circ D, S) \\ \quad \leftarrow \quad eprop(A \circ B, G, C), \\ \qquad B \neq_{\mathrm{AC1}} \emptyset, \\ \qquad complement(F \circ G), \\ \qquad \neg overruled(G, C \circ J, A \circ B, A \circ B \circ D, S). \end{array} \tag{27}$$

$$\begin{array}{l} overruled(F \circ H, C \circ J, A, A \circ B \circ D, (A \circ B, G \circ E, C) \circ R) \\ \quad \leftarrow \quad B \neq_{\mathrm{AC1}} \emptyset, \\ \qquad complement(F \circ G), \\ \qquad \neg overruled(G, C \circ J, A \circ B, A \circ B \circ D, (A \circ B, G \circ E, C) \circ R). \end{array}$$

In words, the particular effect F of an action A is overruled by an *eprop* (first clause) or an *alteprop* that has been selected (second clause) postulating the effect $G = \overline{F}$ of an action $A \circ B \supset A$ if this e-proposition is not overruled itself.

3. An instance $unfounded(h, i, a, s)$ is intended to be true if h contains a fluent literal whose negation holds in i but there is no (strict or selected via s) e-proposition wrt state i and action a that induces this change. The definition of this predicate is as follows:

$$\begin{array}{rcl} unfounded(F \circ H, G \circ I, A, S) & \leftarrow & complement(F \circ G), \\ & & \neg overruled(G, G \circ I, \emptyset, A, S). \end{array} \tag{28}$$

In words, a change from fluent literal $G = \overline{F}$ to F is unfounded if there is no (strict or selected) e-proposition that overrules G continue to be true.

4. An instance $inconsistent(h)$ is intended to be true if AC1-term h does not represent a state (c.f. (18)), that is, if h contains some fluent term twice or more, or it contains a

[15] Note that applicability means the conditions c are true in i, which is guaranteed if the terms $c \circ J$ and i are AC1-unifiable (see Lemma 12 below).

fluent name along with its negation, or there is some fluent name $f \in F_D$ such that neither f nor $\overline{f}$ occur as subterm. These three criteria are encoded by the following clauses:

$$\begin{array}{lcl} \mathit{inconsistent}(G \circ G \circ H) & \leftarrow & G \neq_{\mathrm{AC1}} \emptyset\,. \\ \mathit{inconsistent}(F \circ G) & \leftarrow & \mathit{complement}(F)\,. \\ \mathit{inconsistent}(H) & \leftarrow & \mathit{complement}(F \circ G)\,, \\ & & F \neq_{\mathrm{AC1}} \emptyset\,,\ G \neq_{\mathrm{AC1}} \emptyset\,, \\ & & \neg\mathit{holds}(F, H)\,,\ \neg\mathit{holds}(G, H)\,. \\ \mathit{holds}(F, H \circ F)\,. & & \end{array} \tag{29}$$

Having encoded the transition relation, we now show how to model the application of an action sequence $[a_1, \ldots, a_m]$ to some initial state. Since the resulting state, $M^{[a_1,\ldots,a_m]}$, is model-dependent and, in particular, determined by the associated function φ, we need to find a way to encode the latter within the model generation process. To this end, we first introduce the notion of an *action tree* serving as a *(minimal) basis* for the set of v-proposition underlying the domain description at hand:

Definition 9 Let D be a domain description with action names A_D and v-propositions V_D. An *action tree* is a tree $\mathfrak{B}$ whose nodes are finite lists over 2^{A_D} such that

1. the root of $\mathfrak{B}$ is $[\,]$ and
2. if $[a_1, \ldots, a_m, a_{m+1}]$ is a node in $\mathfrak{B}$ then its predecessor is $[a_1, \ldots, a_m]$, where $m \geq 0$.

Such a tree $\mathfrak{B}$ is called *basis* for D iff for each v-proposition ℓ `after` $[a_1, \ldots, a_m]$ in V_D the sequence $[a_1, \ldots, a_m]$ is a node in $\mathfrak{B}$. Moreover, the *minimal basis* for D is the basis with a minimal number of nodes. ■

As an example, assume given the v-propositions $\{(13), (14)\}$, then Figure 2(a) depicts the minimal basis for the corresponding domain. Note that for any finite set of v-propositions a unique minimal basis exists and is finite.

The purpose of the minimal basis is to indicate which arguments of the model component φ are of interest—regarding the underlying v-propositions—when searching for models. Now, to record the actual values of φ in a particular model, we assign variables to the basis, which are intended to be substituted by states, as follows. Let $\mathfrak{B}_D$ be the minimal basis for D containing $\beta + 1$ nodes ($\beta \geq 0$). Furthermore, let $X_1, \ldots, X_\beta$ be pairwise different variables assigned on-to-one to the edges of the tree, then each node $[a_1, \ldots, a_m, a_{m+1}]$ in $\mathfrak{B}_D$ ($m \geq 0$) is replaced by the sequence of pairs $[(a_1, X_\alpha), \ldots, (a_m, X_\delta), (a_{m+1}, X_i)]$ where

1. the predecessor has been replaced by $[(a_1, X_\alpha), \ldots, (a_m, X_\delta)]$ and
2. the edge from the predecessor to the node itself is labeled with X_i.

A possible labeling of our example tree is depicted in Figure 2(b).

We are now in a position to represent the notion of models in our logic program. To this end, we first introduce the ternary predicate *result*, whose intended meaning is that $\mathit{result}(\gamma_D(\sigma_0), [(\mu(a_1), h_1), \ldots, (\mu(a_m), h_m)], \gamma_D(M^{[a_1,\ldots,a_m]}))$ is true iff the application of the sequence $[a_1, \ldots, a_m]$ to σ_0 yields $M^{[a_1,\ldots,a_m]}$ in a model which satisfies $\varphi([a_1, \ldots, a_i]) = \gamma_D^{-1}(h_i)$ for each $1 \leq i \leq m$—where it is required that h_i represents a possible successor state of applying a_i to the preceding state, according to the underlying transition relation. Thus, the clauses

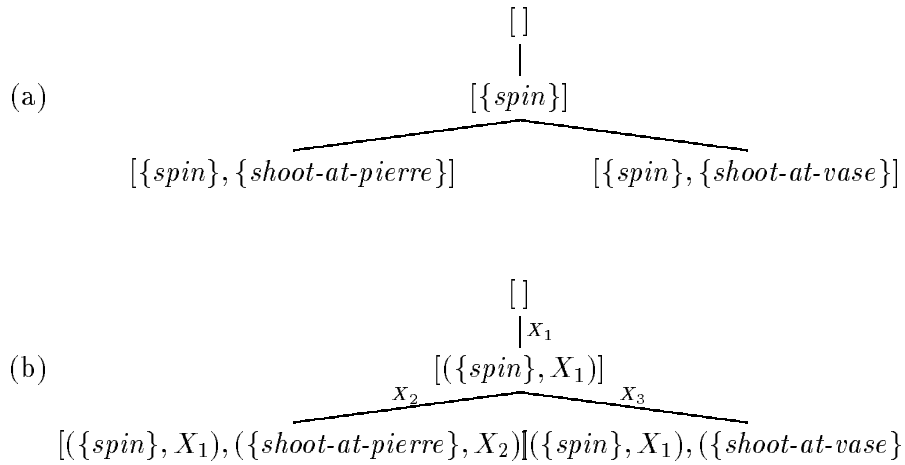

Figure 2: (a) An action tree describing two directions of development, which forms a minimal basis for domains with v-propositions $\{(13), (14)\}$, and (b) the same tree augmented by variables to record the outcomes in a concrete model.

---

defining *result* are as follows:

$$\begin{array}{lll} result(I, [\,], I)\,. & & \\ result(I, [(A, H)|P], G) & \leftarrow & action(I, A, H)\,, \\ & & result(H, P, G)\,. \end{array} \tag{30}$$

In words, $M^{[\,]}$ is σ_0, and in case $m > 0$, $M^{[a_1,\ldots,a_m]}$ is obtained by computing a successor state $H = \gamma_D^{-1}(h_1)$ of executing a_1 in σ_0 and applying the remaining sequence $[a_2, \ldots, a_m]$ to this state.

Finally, to encode the given v-propositions, $V_D = \{\ell_i \,\texttt{after}\, [a_{i1}, \ldots, a_{im_i}] \mid 1 \leq i \leq n\}$, we use the following clause defining the predicate *model*. The construction of this clause is grounded on a given minimal basis $\mathfrak{B}_D$ augmented by variables for the domain under consideration:

$$\begin{array}{l} model(I, X_1, \ldots, X_\beta) \\ \quad \leftarrow \quad \neg inconsistent(I)\,, \\ \qquad\quad result(I, [(\mu(a_{11}), X_{\alpha_1}), \ldots, (\mu(a_{1m_1}), X_{\delta_1})], \tau(\ell_1) \circ G_1)\,, \\ \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ \qquad\quad result(I, [(\mu(a_{n1}), X_{\alpha_n}), \ldots, (\mu(a_{nm_n}), X_{\delta_n})], \tau(\ell_n) \circ G_n)\,. \end{array} \tag{31}$$

where $X_1, \ldots, X_\beta$ are the variables assigned to $\mathfrak{B}_D$ and, for each $1 \leq i \leq n$, the variables $X_{\alpha_i}, \ldots, X_{\delta_i}$ are chosen according to the corresponding node in the labeled basis $\mathfrak{B}_D$. Hence the intended meaning is that $model(i, h_1, \ldots, h_\beta)$ is true if i represents a consistent initial state such that all v-propositions in V_D are satisfied wrt the execution results $h_1, \ldots, h_\beta$.

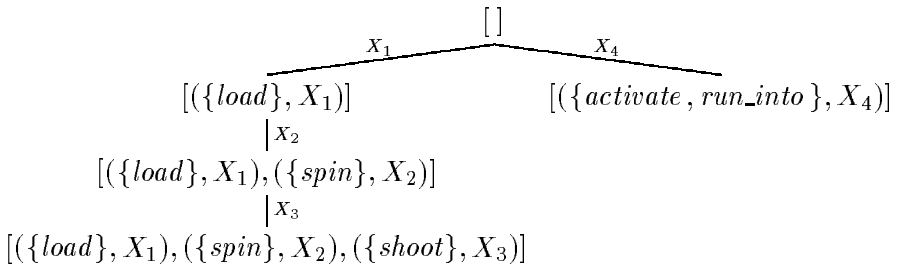

Figure 3: A suitable labeled action tree for encoding the v-propositions in (32).

Example 9 Given the four v-propositions

$$\begin{array}{lll} alive & \texttt{after} & [\{load\}] \\ alive & \texttt{after} & [\{load\}, \{spin\}, \{shoot\}] \\ \texttt{initially} & sleeps & \\ \overline{hurt} & \texttt{after} & [\{activate, run\_into\}] \end{array} \tag{32}$$

these are encoded by the following program clause if we take the suitably labeled basis shown in Figure 3:

$$\begin{array}{l} model(I, X_1, X_2, X_3) \\ \quad \leftarrow \quad \neg inconsistent(I), \\ \qquad result(I, [(load, X_1)], alive \circ G_1), \\ \qquad result(I, [(load, X_1), (spin, X_2), (shoot, X_3)], alive \circ G_2), \\ \qquad result(I, [\,], sleeps \circ G_3), \\ \qquad result(I, [(activate \circ run\_into, X_4)], \overline{hurt} \circ G_4). \end{array} \tag{33}$$

■

To summarize, a domain description D in $\mathcal{A}_{NCC}$ is translated into the set of clauses $P_D = FLUENT_D \cup EPROP_D \cup ALTEPROP_D \cup \{(23)\text{–}(31)\}$. The resulting equational logic program is denoted by $(P_D, \mathrm{AC1})$, and the class of resulting equational logic programs is denoted by $\mathcal{FC}_{NCC}$. One should observe that the major part of this program, clauses (23)–(30), is domain independent and constitutes an intuitive and direct translations of Definition 7 and 8.

5.2 The Completion Semantics

The equational logic program developed in the previous subsection contains negative literals in the body of some clauses. These negative literals are intended to be treated by the (non-monotonic) negation-as-failure principle. An adequate semantics for such programs which is based on classical first-order logic is obtained by applying an extension of Clark's completion procedure [Clark, 1978] to the program: The idea is to consider the set of program clauses which define a predicate p as a complete description of the positive information regarding p. Formally, the completion procedure applied to a set of clauses P is as follows:

Definition 10 Let $p(t_1,\ldots,t_n) \leftarrow L_1,\ldots,L_m$ be a program clause in P, and let $\overline{Y}$ denote a sequence of all variables which occur in this clause. Let $X_1,\ldots,X_n$ be pairwise different variables not in $\overline{Y}$ then the *rectified form* of this clause is the formula $p(X_1,\ldots,X_n) \leftarrow \exists\overline{Y}\,(X_1 = t_1 \wedge \ldots \wedge X_n = t_n \wedge L_1 \wedge \ldots \wedge L_m)$. Let p be an arbitrary predicate symbol and

$$\begin{array}{rcl} p(X_1,\ldots,X_n) & \leftarrow & D_1 \\ & \vdots & \\ p(X_1,\ldots,X_n) & \leftarrow & D_k \end{array}$$

be all clauses in P defining p in rectified form ($k \geq 0$). The *completed definition* of p in P is the formula

$$\forall X_1,\ldots,X_n\ (\ p(X_1,\ldots,X_n) \leftrightarrow D_1 \vee \ldots \vee D_k\ )$$

(In case $k = 0$ this reduces to $\forall(\neg p(X_1,\ldots,X_n))$). The *completion* P^* of P is the conjunction of the completed definitions of all predicate symbols occurring in the underlying alphabet except for the equality predicate $=$. ■

Given a domain description D in $\mathcal{A}_{NCC}$, the entire completion of the corresponding program clauses P_D, written P_D^*, is shown in the appendix.

Aside from completing the program clauses, a logic program with an underlying equational theory requires a special kind of completion for the equality predicate since axioms are needed which allow for proving inequalities in order to derive negative information. K. Clark added some axiom schemata to the completed formula which allow for proving inequality of two terms whenever these are not syntactically unifiable [Clark, 1978]. The concept of *unification completeness* [Jaffar *et al.*, 1984; Shepherdson, 1992] generalizes these axiom schemata for arbitrary equational theories.

Prior to stating the formal definition, we need to introduce some notions and notations related to unification theory, taken from the survey article [Baader and Siekmann, 1993]. The *standard axioms of equality* are $\forall(X = X)$ (*reflexivity*), $\forall(X = Y \rightarrow Y = X)$ (*symmetry*), $\forall(X = Y \wedge Y = Z \rightarrow X = Z)$ (*transitivity*), $\forall(X_i = Y \rightarrow f(X_1,\ldots,X_i,\ldots,X_n) = f(X_1,\ldots,Y,\ldots,X_n))$ (*substitutivity for functions*), $\forall(X_i = Y \rightarrow [\,p(X_1,\ldots,X_i,\ldots,X_n) \leftrightarrow p(X_1,\ldots,Y,\ldots,X_n)\,])$ (*substitutivity for predicates*). If E is an equational theory then two terms s,t are called *E-equal* if the formula $s = t$ is a logical consequence of E and the standard equality axioms. Two terms s,t are said to be *E-unifiable* if there exists a substitution θ such that $s\theta$ and $t\theta$ are E-equal; in which case θ is called an *E-unifier* for s,t. A *complete set of E-unifiers* $cU_E(s,t)$ for two terms s,t is a set of E-unifiers for s,t such that each E-unifier for s,t is subsumed by at least one element in $cU_E(s,t)$.

As in [Shepherdson, 1992], given a substitution $\theta = \{X_1 \mapsto t_1,\ldots,X_n \mapsto t_n\}$ we use $eqn(\theta)$ to denote the formula $X_1 = t_1 \wedge \ldots \wedge X_n = t_n$:

Definition 11 Let E be an equational theory. A consistent set of first-order formulas E^* is called *unification complete* wrt E if it consists of the axioms in E, the standard equality axioms, and a number of equational formulas, i.e., formulas with $=$ as the only predicate, such that for any two terms s and t with variables $\overline{X}$ the following holds:

1. If s and t are not E-unifiable then $E^* \models \neg\exists\overline{X}.\ s = t$.
2. If s and t are E-unifiable then for each complete set of E-unifiers $cU_E(s,t)$

$$E^* \models \forall\overline{X}\ \left(\ s = t \rightarrow \bigvee_{\theta \in cU_E(s,t)} \exists\overline{Y}.\ eqn(\theta)\ \right) \qquad (34)$$

where $\overline{Y}$ denotes the variables which occur in $eqn(\theta)$ but not in $\overline{X}$.

■

In [Hölldobler and Thielscher, 1995], we have proved the existence of such a unification complete theory AC1* for the equational theory AC1 used in $\mathcal{FC}_{NCC}$. Since we do not intend to compute with AC1*, we are only interested in the properties of this theory as given by Definition 11; its actual design is irrelevant for our analysis. Given a domain description D in $\mathcal{A}_{NCC}$, we take the formulas $P_D^* \cup \text{AC1}^*$ as the semantics of the corresponding logic program $(P_D, \text{AC1})$.

5.3 Soundness and Completeness of the Translation

Based on the completion semantics, we now prove soundness and completeness of our equational logic program wrt the entailment relation defined for $\mathcal{A}_{NCC}$. We start with a number of lemmas concerning specific parts of our program.

Lemma 12 *Let D be a domain description with fluent names F_D, and let then $c_1, \ldots, c_m$ be a sequence of fluent literals over F_D ($m \geq 0$) and $\sigma \subseteq F_D$ be a state. Then each c_i ($1 \leq i \leq m$) holds in σ iff $\tau(c_1, \ldots, c_m) \circ V$ and $\gamma_D(\sigma)$ are AC1-unifiable (where V is an arbitrary variable).*

Analogously, let A_D be the set of unit actions and $a, b \subseteq A_D$ two actions then $b \subseteq a$ iff $\mu(b) \circ V$ and $\mu(a)$ are AC1-unifiable.

> **Proof:** In case $n = 0$, $\emptyset \circ V$ and $\gamma_D(\sigma)$ are always AC1-unifiable using the substitution $\{V \mapsto \gamma_D(\sigma)\}$. Otherwise, associativity and commutativity of $\circ$ imply that the two term are AC1-unifiable iff each subterm $\tau(c_i)$ occurs in $\gamma_D(\sigma)$, which contains exactly the fluent literals that hold in σ. The second claim follows analogously.

■

Moreover, unification completeness of AC1* ensures

$$\text{AC1}^* \models \forall V.\ \tau(c_1, \ldots, c_m) \circ V \neq \gamma_D(\sigma)$$

whenever some c_i does not hold in σ.

In what follows, a notation like (20*) refers to the completed definition(s) of the clause(s) in (20), which are all listed in the appendix.

Lemma 13 *Let D be a domain description in $\mathcal{A}_{NCC}$. Furthermore, let $\sigma \subseteq F_D$ be a state and $a \subseteq A_D$ an action. If $b'_1, \ldots, b'_n$ are actions and $e'_1, \ldots, e'_n$ and $c'_1, \ldots, c'_n$ both are sequences of fluent literals ($n \geq 0$) then*

$$P_D^* \cup AC1^* \models$$
$$setofalternatives(\mu(a), (\mu(b'_1), \tau(e'_1), \tau(c'_1)) \circ \ldots \circ (\mu(b'_n), \tau(e'_n), \tau(c'_n)), \gamma_D(\sigma))$$

iff the following holds:

For each $b \subseteq a$ let $\{\, b$ `alternatively causes` E_l^b `if` $C_l^b \mid 1 \leq l \leq k_b \,\}$ be the set of all e-propositions in E_D for action b such that each element occurring in C_i^b holds in σ, for each $1 \leq i \leq k_b$; and let $b_1, \ldots, b_m$ be all actions $b_j \subseteq a$ which satisfy $k_{b_j} > 0$. Then there exists a selection $\Lambda = \lambda_{b_1}, \ldots, \lambda_{b_m}$ (where $\lambda_{b_j} \in \{1, \ldots, k_{b_j}\}$) such that there is a one-to-one correspondence between the triples $(\mu(b'_1), \tau(e'_1), \tau(c'_1)), \ldots, (\mu(b'_n), \tau(e'_n), \tau(c'_n))$ and the

elements of Λ *such that if* $(\mu(b_i'), \tau(e_i'), \tau(c_i'))$ *corresponds to* λ_{b_j} *then* $b_i' = b_j$, $\tau(e_i') = E^{b_j}_{\lambda_{b_j}}$ *and* $\tau(c_i) = C^{b_j}_{\lambda_{b_j}}$.

Proof: Let $s = (\mu(b_1), \tau(e_1), \tau(c_1)) \circ \ldots \circ (\mu(b_n), \tau(e_n), \tau(c_n))$. From (25^*) and (22^*) in conjunction with Lemma 12 it follows that $\mathit{underrepresented}(\mu(a), s, \gamma_D(i))$ is entailed iff there exists some $b \subseteq a$ such that $k_b > 0$ but no subterm $(\mu(b), t_1, t_2)$ occurs in s. Conversely, according to (25^*), $\mathit{overrepresented}(s)$ is entailed iff s includes two subterms $(\mu(b), t_1, t_2)$ and $(\mu(b), t_3, t_4)$. Thus, for each subterm $(\mu(b_i'), \tau(e_i'), \tau(c_i'))$ in s we can find some $b_j \in \{b_1, \ldots, b_m\}$ and, hence, some λ_{b_j} in Λ such that $b_i' = b_j$—and vice versa. Moreover, (25^*) in conjunction with (22^*) ensures that $\tau(e_i') = E^{b_j}_{\lambda_{b_j}}$ and $\tau(c_i) = C^{b_j}_{\lambda_{b_j}}$. The claim then follows from $(24^*) \in P^*_D$.

■

The following lemma describes the connection between our definition of predicate *overruled* and the notion of *causes* in Definition 7:

Lemma 14 *Let* D *be a domain description in* $\mathcal{A}_{NCC}$. *Furthermore, let* $e_1, \ldots, e_n$ *be a sequence of fluent literals (*$n \geq 1$*),* σ *a state,* a, b *actions such that* $b \subseteq a$, *and* s *an AC1-term representing a complete selection of applicable alternative e-propositions wrt* σ *and* a, *as defined in Lemma 13. Then*

$$P^*_D \cup AC1^* \models \neg \mathit{overruled}(\tau(e_1, \ldots, e_n), \gamma_D(\sigma), b, a, s) \tag{35}$$

iff there is no e_i *(*$1 \leq i \leq n$*) such that* a *causes* $\overline{e_i}$ *by some* c *such that* $c \supset b$ *and* $c \subseteq a$.

Proof: From $b \subseteq a$ we know $|b| \leq |a|$. The proof is by induction on $n = |a| - |b|$. In case $n = 0$ (i.e., $a = b$), no such c can possibly exist. Correspondingly, the literal $B \neq_{\mathrm{AC1}} \emptyset$ in both disjuncts of (27^*) guarantees (35) to hold.

In case $n > 0$, from (27^*) in conjunction with (20^*) and (21^*) and Lemma 12 we know that (35) is *false* iff there exists some e_i $(1 \leq i \leq n)$ such that

$$P^*_D \cup AC1^* \models \neg \mathit{overruled}(\tau(\overline{e_i}), \gamma_D(\sigma), c, a, s) \tag{36}$$

where $c \supset b$ and $c \subseteq a$. Since $|b| < |c| \leq |a|$, we have $0 \leq |a| - |c| < n$. Hence, the induction hypothesis is applicable and ensures (36) be true iff a causes $\overline{e_i}$ by some c' such that $c' \supset c$ and $c' \subseteq a$. Due to $c' \supset b$, this proves the claim.

■

Finally, we need an adequate definition of consistency of AC1-terms that are intended to represent states:

Lemma 15 *Let* D *be a domain description with fluents* F_D *and* i *an AC1-term then*

$$P^*_D \cup AC1^* \models \neg \mathit{inconsistent}(i)$$

iff for each fluent name $f \in F_D$ *either* f *or else* $\overline{f}$ *occurs in* i, *and* i *does not contain any fluent term more than once.*

Proof: In conjunction with Lemma 12, the first disjunct in (29*) ensures that no fluent term occurs twice or more in i, the second disjunct ensures that i does not contain a fluent name along with its negation, and the third disjunct ensures that each fluent name is represented affirmatively or negatively.

■

The following theorem concerning transition of states forms the basis of our soundness and completeness result:

Theorem 16 *Let D be a domain description in $\mathcal{A}_{NCC}$. If Φ is a transition relation for E_D then for each $\sigma \subseteq F_D$, $a \subseteq A_D$, and each AC1-term h*

$$P_D^* \cup AC1^* \models action(\gamma_D(\sigma), \mu(a), h) \tag{37}$$

iff $(\sigma, a, \sigma') \in \Phi$ where h represents state σ'.

Proof: From Lemma 15 and from $(23^*) \in P_D^*$ and Lemma 13 it follows that (37) holds iff h represents a state and there exists some term s that represents a complete selection of alternative e-propositions (wrt σ and a) and

$$P_D^* \cup \text{AC1}^* \models \neg unfounded(h, \gamma_D(\sigma), \mu(a), s) \wedge \neg impossible(h, \gamma_D(\sigma), \mu(a), s)$$

holds. Following (28*), (26*) and Lemma 14 this is true iff

1. for each fluent literal ℓ that is true in σ but false in σ' there is some applicable (strict or selected in s) e-proposition postulating $\overline{\ell}$ and
2. there is no fluent literal ℓ true in σ' such that a causes $\overline{\ell}$ in σ but not a causes ℓ in σ.[16]

Following Definition 7, this is equivalent to $(\sigma, a, \sigma') \in \Phi$.

■

Based on this theorem, we can prove soundness and completeness of our equational logic program wrt the semantics of $\mathcal{A}_{NCC}$ as given by Definition 8. More precisely, we will prove that a v-proposition ℓ `after` $[a_1, \ldots, a_m]$ is entailed iff no model can be found—based on (31*)—that contradicts this v-proposition. To this end, recall Definition 9, where we introduced the concept of an action tree to encode the model component φ. In order to test entailment of a v-proposition using the literal $result(i, [(a_1, X_1), \ldots, (a_m, X_m)], \tau(\ell) \circ G)$, we have to take into account the underlying labeled action tree that has been used to construct clause (31). Let k be maximal in $\{0, \ldots, m\}$ such that $[a_1, \ldots, a_k]$ occurs in this tree then we use the k variables $X_\alpha, \ldots, X_\delta$ assigned to the actions in this node plus pairwise different, new variables $X'_1, \ldots, X'_{m-k}$ for the tail $[a_{k+1}, \ldots, a_m]$ of the action sequence. As before, $X_1, \ldots, X_\beta$ denotes the entire sequence of variables assigned to the underlying action tree:

Theorem 17 *Let D be a domain description in $\mathcal{A}_{NCC}$. If $\nu = \ell$ `after` $[a_1, \ldots, a_m]$ is a v-proposition then ν is entailed by D iff*

$$\begin{aligned} P_D^* \cup AC1^* \models \neg\exists I, \widetilde{X} \; (\; & model(I, X_1, \ldots, X_\beta) \wedge \\ & result(I, [(\mu(a_1), X_\alpha), \ldots, (\mu(a_k), X_\delta), \\ & \qquad (\mu(a_{k+1}), X'_1), \ldots, (\mu(a_m), X'_{m-k})], \tau(\overline{\ell}) \circ G) \;) \end{aligned}$$

where $\widetilde{X} = X_1, \ldots, X_\beta, X'_1, \ldots, X'_{m-k}$.

[16] Note that if a causes both ℓ and $\overline{\ell}$ then the corresponding fluent name belongs to $B_f^\cap(a, \sigma)$, that is, either value can be true in σ'.

Proof: Let Φ be a transition relation for D. From Lemma 15, (31*), (30)*and repeated application of Theorem 16 it follows that $model(i, h_1, \ldots, h_\beta)$ is entailed iff $(\sigma_0, \Phi, \varphi)$ satisfies every v-proposition in D, that is, if it is model for D — where i represents state σ_0 and $h_1, \ldots, h_\beta$ correspond to φ in the sense that each h_j ($1 \leq j \leq \beta$) represents the state $\varphi([a'_1, \ldots, a'_l])$ where variable X_j has been assigned to the vertex ending in node $[a'_1, \ldots, a'_l]$ in the underlying labeled action tree. The claim then follows from (30*) and the fact that ν is entailed iff there is no model $(\sigma_0, \Phi, \varphi)$ for D in which $\overline{\ell}$ holds in $\varphi([a_1, \ldots, a_m])$.

■

5.4 SLDENF-Resolution

Our equational logic programs $\mathcal{FC}_{NCC}$ are based on a special equational theory, viz AC1, and they also contain negation in the body of some program clauses. In the preceding subsection, we have taken the completion of these programs as an adequate semantics when negative literals are to be treated by negation-as-failure. An adequate computation mechanism for programs including equality and (nonmonotonic) negation is *SLDENF-resolution*, which is based on SLD-resolution (see, e.g., [Lloyd, 1987]) but with the standard unification procedure replaced by an E-unification algorithm and negation-as-failure used to treat negative subgoals. A formal introduction to this resolution principle can be found in [Shepherdson, 1992; Thielscher, 1996a]. In [Shepherdson, 1992], soundness of SLDENF-resolution wrt the completion semantics (including the use of unification complete theories) has been proved for arbitrary equational logic programs with negation. More precisely, let P be a set of normal program clauses, E an equational theory, and $\leftarrow L_1, \ldots, L_n$ a query. If there exists an SLDENF-refutation for this query wrt P with computed answer substitution θ then

$$P^* \cup E^* \models \forall (L_1 \wedge \ldots \wedge L_n)\theta$$

Combining this result with Theorem 17 proves that SLDENF-resolution can be applied as a sound proof procedure for the entailment relation defined in $\mathcal{A}_{NCC}$.

In [Thielscher, 1996a], we have discussed completeness of SLDENF-resolution. As already known from the special case of programs with negation and the empty equational theory, completeness cannot be guaranteed in general [Clark, 1978; Apt *et al.*, 1987]. The classical completeness result for SLDNF-resolution is restricted to so-called *hierarchical* and *allowed* programs [Clark, 1978; Apt *et al.*, 1987]. In a hierarchical program, every SLDNF-derivation is guaranteed to be finite, and the allowedness criterion prevents so-called *floundering*: Since by definition of SLD(E)NF-resolution negative subgoals can be selected only if they are ground, a derivation might end up with only non-ground negative subgoals. In such cases, the proof procedure does not come to a conclusion.

In [Thielscher, 1996a], we have lifted the aforementioned classical result to logic programs with equational theories. We have shown that this is possible only in case the underlying equational theory E meets two further restrictions: It should be *finitary* (that is, for each two terms s and t there exists a finite complete set of E-unifiers) to ensure finiteness of derivations in hierarchical programs, and it should also be *regular* (that is, for each equation $s = t \in E$ the set of variables occurring in s equals the set of variables occurring in t) to avoid the problem of floundering in allowed programs. See [Thielscher, 1996a] for a more detailed and formal discussion.

The equational theory used in this paper, AC1, is known to be both finitary [Stickel, 1975] and, obviously, regular. Nonetheless the result presented in [Thielscher, 1996a] cannot be applied since the program developed in Subsection 5.1 is neither hierarchical nor allowed. We therefore have to perform a more detailed and specific analysis of our program.

Although the programs $\mathcal{FC}_{NCC}$ are not hierarchical, it can be shown that all SLDENF-derivations we are interested in to decide entailment wrt $\mathcal{A}_{NCC}$ are necessarily finite. Since no mutual recursion involving two or more program clauses occurs, the only crucial clauses are direct recursive ones, that is, where the predicate in the head also occurs in the body.

There are three clauses of this kind, shown in (27) and (30), respectively. As regards the two definitions of *overruled* in (27), it is easy to see that each recursive call increases the size (viz the number of subterms) of the third argument, A. Moreover, the body of the first (resp. second) clause can only be satisfied if there exists a strict e-proposition (resp. a selected alternative e-proposition) for an action that includes A. Since there are only a limited number of such proposition in a concrete domain, the recursive calls eventually stop, provided a fair selection rule is used.

Analogously, the number of recursive calls of *result*, c.f. (30), is limited by the size of the second argument, provided it is (partially) instantiated. This is indeed the case in both clause (31) and the query used to decide entailment of an additional v-proposition (c.f. Theorem 17).

While it is easy to show finiteness of derivations, we cannot in general prove non-floundering. In fact, whenever we try to decide entailment of a v-proposition $\ell\,\mathtt{after}\,[a, \ldots, a_m]$ by creating the clause

$$\begin{array}{lll} \mathit{satisfiable} & \leftarrow & \mathit{model}(I, \widetilde{X}), \\ & & \mathit{result}(I, [(\mu(a_1), X_1), \ldots, (\mu(a_m), X_m)], \tau(\overline{\ell}) \circ G). \end{array} \tag{38}$$

(see Theorem 17) and we use the query $\leftarrow \mathit{satisfiable}$ then the derivation flounders after clause (31) has been applied—to solve the subgoal $\mathit{model}(I, \widetilde{X})$—since $\neg\mathit{inconsistent}(I)$ cannot be selected as I is a variable. Analogously, whenever clause (30) has been applied such that a subgoal of the form $\mathit{action}(I, A, H)$ occurs then this can only be resolved using clause (23). This, however, requires I and H be fully instantiated if the derivation is not to flounder.[17] Moreover, whenever a suitable set S of alternative e-propositions is selected via clause (24) then this additionally requires S to be instantiated.

There are two ways of solving this problem. First, one could define an extension of the SLDENF-resolution principle that supports a proper treatment of non-ground, negative subgoals, such as the concept of *constructive negation*, which is a well-known technique to avoid floundering in case of non-equational logic programs [Chan, 1988; Przymusinski, 1989]. This, however, requires a new formal definition of an extended calculus, and then soundness and completeness have to be proved again.

Here, we follow a simpler and more straightforward way. It is possible to rewrite some program clauses such that the crucial variables become instantiated early enough during the derivation to avoid floundering. To this end, we provide all possible collections of fluent literals of the length $|F_D|$, where F_D denotes the underlying set of fluent names of domain D. These combinations are obtained by the following clause:

[17] The variable A always becomes instantiated when deciding entailment wrt $\mathcal{A}_{NCC}$ since action sequences in v-propositions are fixed.

$$
\begin{array}{lll}
sterm(I) & \leftarrow & I =_{AC1} H_1 \circ \ldots \circ H_{|F_D|}\,, \\
 & & complement(H_1 \circ J_1)\,, \\
 & & \vdots \\
 & & complement(H_{|F_D|} \circ J_{|F_D|})\,.
\end{array}
$$

where $H_1, J_1, \ldots, H_{|F_D|}, J_{|F_D|}$ are pairwise distinct variables.

The new predicate *sterm* can then be used to instantiate the crucial arguments in advance and, thus, helps to avoid floundering. The following clause is a modification of (30):

$$
\begin{array}{lll}
result(I, [\,], I)\,. & & \\
result(I, [(A, H)|P], G) & \leftarrow & sterm(H)\,, \\
 & & action(I, A, H)\,, \\
 & & result(H, P, G)\,.
\end{array}
$$

This clause can be used instead of (31) given a set of v-propositions:

$$
\begin{array}{l}
model(I, X_1, \ldots, X_\beta) \\
\quad \leftarrow \; sterm(I)\,, \\
\quad\quad \neg inconsistent(I)\,, \\
\quad\quad result(I, [(\mu(a_{11}), X_{\alpha_1}), \ldots, (\mu(a_{1m_1}), X_{\delta_1})], \tau(\ell_1) \circ G_1)\,, \\
\quad\quad\quad\quad \vdots \\
\quad\quad result(I, [(\mu(a_{n1}), X_{\alpha_n}), \ldots, (\mu(a_{nm_n}), X_{\delta_n})], \tau(\ell_n) \circ G_n)\,.
\end{array}
$$

Finally, to avoid floundering after clause (24) has been applied, by the following clause we provide all suitable instances for a variable that contains a collection of triples each of which represents an alternative e-proposition:

$$
\begin{array}{lll}
eterm(\emptyset)\,. & & \\
eterm((A, E, C) \circ S) & \leftarrow & alteprop(A, E, C)\,, \\
 & & eterm(S)\,.
\end{array}
$$

Clause (24) is then modified as follows:

$$
\begin{array}{lll}
setofalternatives(A, S, I) & \leftarrow & eterm(S)\,, \\
 & & \neg overrepresented(S)\,, \\
 & & \neg underrepresented(A, S, I)\,.
\end{array}
$$

To summarize, employing the modified equational logic program avoids the problem of derivations that flounder. Moreover, all derivations that occur when deciding entailment of a v-proposition via (38) are guaranteed to terminate. Hence, SLDENF-resolution can now be applied as a sound and complete calculus for the equational logic program encoding domains represented in $\mathcal{A}_{NCC}$. Furthermore, finiteness of derivations shows that we have obtained a decision procedure for entailment in our high-level action semantics.

6 Summary

We have presented formalisms for intentionally specifying actions so as to have nondeterministic effects, and for extracting as much as possible consistent and reasonable information from

contradictory representations of dynamic systems by interpreting them so as to be implicitly nondeterministic. Our resulting language $\mathcal{A}_{NCC}$ allows the representation of nondeterministic concurrent actions in dynamic systems and the resolution of conflicts. Furthermore, we have developed a sound and complete encoding of representations in $\mathcal{A}_{NCC}$ in terms of equational logic programming and, thereby, have provided an instrument for automated reasoning about such representations.

Our formalisms are based on the Action Description Language $\mathcal{A}$. Two recent extensions of $\mathcal{A}$, namely, first-order-fluents [Dung, 1993] and indirect effects [Kartha and Lifschitz, 1994], are not subsumed by our approach. Yet the fluent calculus, which we used here, has already been extended to successfully cope with the ramification problem [Thielscher, 1995a; Thielscher, 1996b]. Hence we have good reasons to assume that the logic program presented in this paper can be extended to form an adequate encoding of a high-level action semantics including $\mathcal{A}_{NCC}$ and indirect effects.

Acknowledgments. The first author acknowledges support from the German Research Community (DFG) within project MPS under grant no. Ho 1294/3-3.

A The Completed Equational Logic Program

Let D be a domain description in $\mathcal{A}_{NCC}$, and let $(P_D, \mathrm{AC1})$ be the corresponding equational logic program. The completion P_D^* of P_D consists of the following first-order formulas.

Let F_D be the underlying set of fluent names then

$$\forall X \ \Big(\ \mathit{complement}(X) \leftrightarrow \bigvee_{f \in F_D} X = f \circ \overline{f} \ \Big) \tag{5.20*}$$

completes $FLUENT_D$.

Let $\{ a_i \ \texttt{causes} \ e_i \ \texttt{if} \ C_i \mid 1 \leq i \leq k \}$ be the set of all strict e-propositions in E_D $(k \geq 0)$ then

$$\forall A, E, C \ \Big(\ \mathit{eprop}(A, E, C) \leftrightarrow \bigvee_{i=1}^{k} \big(\begin{array}{l} A = \mu(a_i) \wedge \\ E = \tau(e_i) \wedge \\ C = \tau(C_i) \ ) \ \Big) \end{array} \tag{5.21*}$$

completes $EPROP_D$.

Let $\{ a_i \ \texttt{alternatively causes} \ E_i \ \texttt{if} \ C_i \mid 1 \leq i \leq k \}$ be the set of all alternative e-propositions in E_D $(k \geq 0)$ then

$$\forall A, E, C \ \Big(\ \mathit{alteprop}(A, E, C) \leftrightarrow \bigvee_{i=1}^{k} \big(\begin{array}{l} A = \mu(a_i) \wedge \\ E = \tau(E_i) \wedge \\ C = \tau(C_i) \ ) \ \Big) \end{array} \tag{5.22*}$$

completes $ALTEPROP_D$.

The completion of clause (23) and (24) is as follows:

$$\forall I, A, H \ \Big(\ \mathit{action}(I, A, H) \leftrightarrow \exists S \ \big(\begin{array}{l} \mathit{setofalternatives}(A, S, I) \wedge \\ \neg \mathit{impossible}(H, I, A, S) \wedge \\ \neg \mathit{unfounded}(H, I, A, S) \wedge \\ \neg \mathit{inconsistent}(H) \ ) \ \Big) \end{array} \tag{5.23*}$$

$$\forall A,S,I\ (\ setofalternatives(A,S,I) \leftrightarrow \begin{array}{l} \neg overrepresented(S) \wedge \\ \neg underrepresented(A,S,I)\ ) \end{array} \tag{5.24*}$$

The three clauses shown in (25) are completed by

$$\forall S\ (\ overrepresented(S) \leftrightarrow \exists A,R,U,V,X,Y.\ S = (A,U,X) \circ (A,V,Y) \circ R\ )$$

$$\forall X,S,Y\ \Big(\ underrepresented(X,S,Y) \leftrightarrow \exists A,B,C,J,E\ \Big(\begin{array}{l} X = A \circ B \wedge \\ Y = C \circ J \wedge \\ alteprop(A,E,C) \wedge \\ \neg represented(A,S)\ )\ \Big) \end{array} \tag{5.25*}$$

$$\forall A,S\ (\ represented(A,S) \leftrightarrow \exists E,C,R.\ S = (A,E,C) \circ R\ )$$

The completion of *impossible* is

$$\forall X,I,A,S\ \Big(\ impossible(X,I,A,S) \leftrightarrow \exists F,G,H\ \Big(\begin{array}{l} X = F \circ H \wedge \\ overruled(F,I,\emptyset,A,S) \wedge \\ complement(F \circ G) \wedge \\ \neg overruled(G,I,\emptyset,A,S)\ )\ \Big) \end{array} \tag{5.26*}$$

and the completion of the two clauses for *overruled* is the formula

$$\begin{array}{l} \forall A,S,X,Y,Z\ \Big(\ overruled(X,Y,A,Z,S) \leftrightarrow \\ \quad \exists B,C,D,F,G,H,J\ \Big(\begin{array}{l} X = F \circ H \wedge \\ Y = C \circ J \wedge \\ Z = A \circ B \circ D \wedge \\ eprop(A \circ B,G,C) \wedge \\ B \neq \emptyset \wedge \\ complement(F \circ G) \wedge \\ \neg overruled(G,C \circ J,A \circ B,A \circ B \circ D,S)\ ) \end{array} \\ \quad \vee \\ \quad \exists B,C,D,E,F,G,H,J,R\ \Big(\begin{array}{l} X = F \circ H \wedge \\ Y = C \circ J \wedge \\ Z = A \circ B \circ D \wedge \\ S = (A \circ B, G \circ E, C) \circ R \wedge \\ B \neq \emptyset \wedge \\ complement(F \circ G) \wedge \\ \neg overruled(G,C \circ J,A \circ B,A \circ B \circ D,S)\ )\ \Big) \end{array} \end{array} \tag{5.27*}$$

Completing (28) yields

$$\forall X,Y,A,S\ \Big(\ unfounded(X,Y,A,S) \leftrightarrow \exists F,G,H,I\ \Big(\begin{array}{l} X = F \circ H \wedge \\ Y = G \circ I \wedge \\ complement(F \circ G) \wedge \\ \neg overruled(G,G \circ I,\emptyset,A,S)\ )\ \Big) \end{array} \tag{5.28*}$$

The four clauses used to express consistency are completed as follows:

$$\begin{aligned}\forall I \Big(\ \mathit{inconsistent}(I) \ \leftrightarrow \quad & \exists G, H \ (\ \ I = G \circ G \circ H \ \wedge \ G \neq \emptyset \ ) \\ \vee \quad & \exists F, G \ (\ \ I = F \circ G \ \wedge \\ & \qquad \mathit{complement}(F) \ ) \\ \vee \quad & \exists F, G \ (\ \ \mathit{complement}(F \circ G) \ \wedge \\ & \qquad F \neq \emptyset \ \wedge \ G \neq \emptyset \ \wedge \\ & \qquad \neg \mathit{holds}(F, I) \ \wedge \ \neg \mathit{holds}(G, I) \ ) \ \Big)\end{aligned} \tag{5.29*}$$

$$\forall F, I \ (\ \mathit{holds}(F, I) \ \leftrightarrow \ \exists H .\ I = H \circ F \ )$$

Clause (30) is completed by

$$\begin{aligned}\forall I, L, G \Big(\ \mathit{result}(I, L, G) \ \leftrightarrow \quad & L = [\,] \ \wedge \ I = G \\ \vee \quad & \exists A, H, P \ (\ \ L = [(A, H) | P] \ \wedge \\ & \qquad \mathit{action}(I, A, H) \ \wedge \\ & \qquad \mathit{result}(H, P, G) \ ) \ \Big)\end{aligned} \tag{5.30*}$$

Finally, let $\{\, \ell_i \ \texttt{after} \ [a_{i1}, \ldots, a_{im_i}] \mid \ 1 \leq i \leq n \,\}$ be the set of all v-propositions in V_D ($n \geq 0$) then

$$\begin{aligned}\forall I, X_1, \ldots, X_\beta \ (\ & \mathit{model}(I, X_1, \ldots, X_\beta) \ \leftrightarrow \\ & \neg \mathit{inconsistent}(I) \ \wedge \\ & \textstyle\bigwedge_{i=1}^{n} \mathit{result}(I, [(\mu(a_{i1}), X_{\alpha_i}), \ldots, (\mu(a_{im_i}), X_{\delta_i})], \tau(\ell_i) \circ G_i) \ )\end{aligned} \tag{5.31*}$$

completes (31).

References

[Apt *et al.*, 1987] Krzysztof R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann, 1987.

[Baader and Siekmann, 1993] Franz Baader and Jörg H. Siekmann. Unification theory. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford University Press, 1993.

[Baker, 1989] A. B. Baker. A simple solution to the Yale Shooting problem. In *Proceedings of the International Conference on Knowledge Representation and Reasoning*, pages 11–20, 1989.

[Baral and Gelfond, 1993] Chitta Baral and Michael Gelfond. Representing concurrent actions in extended logic programming. In R. Bajcsy, editor, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 866–871, Chambéry, France, August 1993. Morgan Kaufmann.

[Baral, 1995] Chitta Baral. Reasoning about actions: Non-deterministic effects, constraints and qualification. In C. S. Mellish, editor, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2017–2023, Montreal, Canada, August 1995. Morgan Kaufmann.

[Bibel, 1986] Wolfgang Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.

[Brüning *et al.*, 1993] Stefan Brüning, Steffen Hölldobler, Josef Schneeberger, Ute Sigmund, and Michael Thielscher. Disjunction in resource-oriented deductive planning. In D. Miller, editor, *Proceedings of the International Logic Programming Symposium (ILPS)*, page 670, Vancouver, Canada, October 1993. MIT Press. (Poster).

[Chan, 1988] David Chan. Constructive negation based on the completed database. In R. Kowalski and K. Bowen, editors, *Proceedings of the International Joint Conference and Symposium on Logic Programming (IJCSLP)*, pages 111–125. MIT Press, 1988.

[Clark, 1978] Keith L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[Denecker and de Schreye, 1993] Marc Denecker and Danny de Schreye. Representing incomplete knowledge in abductive logic programming. In D. Miller, editor, *Proceedings of the International Logic Programming Symposium (ILPS)*, pages 147–163, Vancouver, October 1993. MIT Press.

[Dung, 1993] Phan Minh Dung. Representing actions in logic programming and its applications in database updates. In D. S. Warren, editor, *Proceedings of the International Conference on Logic Programming (ICLP)*, pages 222–238, Budapest, June 1993. MIT Press.

[Gelfond and Lifschitz, 1993] Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–321, 1993.

[Girard, 1987] Jean-Yves Girard. Linear Logic. *Journal of Theoretical Computer Science*, 50(1):1–102, 1987.

[Große, 1994] Gerd Große. Propositional State-Event Logic. In C. MacNish, D. Peirce, and L. M. Peireira, editors, *Proceedings of the European Workshop on Logics in AI (JELIA)*, volume 838 of *LNAI*, pages 316–331. Springer, 1994.

[Hanks and McDermott, 1987] Steve Hanks and Drew McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence Journal*, 33(3):379–412, 1987.

[Hölldobler and Schneeberger, 1990] Steffen Hölldobler and Josef Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8:225–244, 1990.

[Hölldobler and Thielscher, 1995] Steffen Hölldobler and Michael Thielscher. Computing change and specificity with equational logic programs. *Annals of Mathematics and Artificial Intelligence*, 14(1):99–133, 1995.

[Jaffar *et al.*, 1984] Joxan Jaffar, Jean-Louis Lassez, and Michael J. Maher. A theory of complete logic programs with equality. *Journal of Logic Programming*, 1(3):211–223, 1984.

[Kartha and Lifschitz, 1994] G. Neelakantan Kartha and Vladimir Lifschitz. Actions with indirect effects. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 341–350, Bonn, Germany, May 1994. Morgan Kaufmann.

[Kartha, 1993] G. Neelakantan Kartha. Soundness and completeness theorems for three formalizations of actions. In R. Bajcsy, editor, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 724–729, Chambéry, France, August 1993. Morgan Kaufmann.

[Kartha, 1994] G. Neelakantan Kartha. Two counterexamples related to Baker's approach to the frame problem. *Artificial Intelligence Journal*, 69(1–2):379–391, 1994.

[Lin and Shoham, 1992] Fangzhen Lin and Yoav Shoham. Concurrent actions in the situation calculus. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 590–595, San Jose, California, 1992. MIT Press.

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

[Łukaszewicz and Madalińska-Bugaj, 1995] Witold Łukaszewicz and Ewa Madalińska-Bugaj. Reasoning about action and change using Dijkstra's semantics for programming languages: Preliminary report. In C. S. Mellish, editor, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1950–1955, Montreal, Canada, August 1995. Morgan Kaufmann.

[Masseron *et al.*, 1993] M. Masseron, Christophe Tollu, and Jacqueline Vauzielles. Generating plans in linear logic I. Actions as proofs. *Journal of Theoretical Computer Science*, 113:349–370, 1993.

[McCarthy and Hayes, 1969] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[McCarthy, 1963] John McCarthy. *Situations and Actions and Causal Laws.* Stanford Artificial Intelligence Project, Memo 2, 1963.

[Przymusinski, 1989] Teodor C. Przymusinski. On constructive negation in logic programming. In E. L. Lusk and R. A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming (NACLP)*, Cleveland, October 1989. (Insertion).

[Richard etal., 1988] Pierre Richard etal. À gauche en sortant de l'ascenseur. Renn Productions, 1988.

[Sandewall, 1993] Erik Sandewall. The range of applicability of nonmonotonic logics for the inertia problem. In R. Bajcsy, editor, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 738–743, Chambéry, France, August 1993. Morgan Kaufmann.

[Sandewall, 1994] Erik Sandewall. *Features and Fluents.* Oxford University Press, 1994.

[Shepherdson, 1992] John C. Shepherdson. SLDNF-resolution with equality. *Journal of Automated Reasoning*, 8:297–306, 1992.

[Stickel, 1975] Mark E. Stickel. A complete unification algorithm for associative-commutative functions. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 71–76, Tbilisi, USSR, 1975.

[Thielscher, 1994] Michael Thielscher. An analysis of systematic approaches to reasoning about actions and change. In P. Jorrand and V. Sgurev, editors, *International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA)*, pages 195–204, Sofia, Bulgaria, September 1994. World Scientific.

[Thielscher, 1995a] Michael Thielscher. Computing ramifications by postprocessing. In C. S. Mellish, editor, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1994–2000, Montreal, Canada, August 1995. Morgan Kaufmann.

[Thielscher, 1995b] Michael Thielscher. The logic of dynamic systems. In C. S. Mellish, editor, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1956–1962, Montreal, Canada, August 1995. Morgan Kaufmann.

[Thielscher, 1996a] Michael Thielscher. On the completeness of SLDENF-resolution. *Journal of Automated Reasoning*, 1996. (To appear Fall '96).

[Thielscher, 1996b] Michael Thielscher. Ramification and Causality. Technical Report TR-96-003, ICSI, Berkeley, CA, 1996.