In D. Miller, ed., Proc. of the Int. Symp. on Log. Prog., 164–180. Vancouver, 1993.

Actions and Specificity

Steffen Hölldobler and **Michael Thielscher**
Intellektik, Informatik, TH Darmstadt
Alexanderstraße 10, D–64283 Darmstadt, Germany
E-mail: {steffen,thielscher}@intellektik.informatik.th-darmstadt.de

Abstract

A solution to the problem of specificity in a resource–oriented deductive approach to actions and change is presented. Specificity originates in the problem of overloading methods in object oriented frameworks but can be observed in general applications of actions and change in logic. We give a uniform solution to the problem of specificity culminating in a completed equational logic program with an equational theory. We show the soundness and completeness of SLDENF–resolution, ie. SLD–resolution augmented by negation–as–failure and by an equational theory, wrt the completed program. Finally, the expressiveness of our approach for performing general reasoning about actions, change, and causality is demonstrated.

1 Introduction

Logic plays a fundamental role for intelligent behavior. Especially the pioneers in Artificial Intelligence realized the importance of logic and deduction for their field. However, classical logic seems to lack some properties to adequately represent human thinking. The examination of the ability of humans to reason about actions and change is one of the major parts of interest in Intellectics, ie. Artificial Intelligence and Cognitive Science [4]. A particular description of the world consists of facts (or fluents) which are believed to be true. An important property of these facts is that they are time–dependent, ie. the truth value of a proposition may change from time to time. In contrast, classical logic seems to have difficulties in changing logical values. For example, one may try to formalize the situation where a gun is unloaded by the negative literal $\neg loaded$ and the execution of an action called *load* which loads the gun, by the implication $execute(load) \rightarrow loaded$. Then, unfortunately, classical logic tells us that the action *load* can never be executed since otherwise the world becomes inconsistent.

The problem of classical logic is that propositions are not treated as resources [15]. A proposition cannot be produced and consumed in the course of time. To overcome these difficulties, John McCarthy and Pat Hayes [25, 23] defined the situation calculus which mainly consists of using an additional argument to state that a particular fact only holds in a particular situation. However, this formalization leads to the well–known frame problem. In general, the technical frame problem is the question of how to express that a particular fact which is not affected by an action continues

to hold after executing the action. McCarthy and Hayes [25] solved this problem by adding additional frame axioms; one for each action and each fact. The obvious problem with this solution is that the number of frame axioms rapidly increases when many actions and many facts occur. Robert Kowalski reduced the number of frame axioms to become linear with respect to the number of different actions [20]. Some years later, it was again John McCarthy who proposed the use of nonmonotonic inference rules to tackle the frame problem [24]. He uses a default rule called *law of inertia* which states that a proposition does not change its value when executing an action unless the contrary is known.

Recently, three new deductive approaches to deal with situations, actions, and change were proposed, each of them without the need to state frame axioms explicitly. In the linear connection method [3] proofs are restricted such that each literal is connected at most once. Thus, connecting a literal during the inference process simulates consumption of the corresponding fact. Conversely, if the conditions of an implication are fulfilled then the conclusion can be used and, thus, the literals occurring in the conclusion are produced. This treatment of literals resembles the concept of resources. In a similar way, linear logic [12] can be used, which is a Gentzen–style proof system without weakening and contraction rules. In the multiplicative fragment of the linear logic, literals and formulas cannot be copied or erased which also provides the idea of resources [22].

In [16], classical logic along with an equational theory is used as a planning formalism. Facts describing a situation in the world are reified and represented as terms which are connected via a binary function symbol $\circ$ which is associative (A), commutative (C), and admits a unit element (1), viz. the constant $\emptyset$. For instance, the situation where the gun is unloaded and the victim, a turkey, is alive is represented by the term $\mathit{unloaded} \circ \mathit{alive}$. The load action can be specified by the clause[1]

$$\mathit{action}(C, \mathit{load}, E) \leftarrow C =_{\mathrm{AC1}} \mathit{unloaded} \wedge E =_{\mathrm{AC1}} \mathit{loaded}, \tag{1}$$

where $=_{\mathrm{AC1}}$ denotes equality modulo AC1. Then, a ternary predicate $\mathit{causes}(i, [a_1, \ldots, a_n], g)$ is used to express that the sequence $[a_1, \ldots, a_n]$ of actions causes a situation i to become situation g:

$$\mathit{causes}(I, [\,], G) \leftarrow I =_{\mathrm{AC1}} G. \tag{2}$$

$$\mathit{causes}(I, [A|P], G) \leftarrow \mathit{action}(C, A, E) \wedge C \circ V =_{\mathrm{AC1}} I \wedge \mathit{causes}(E \circ V, P, G). \tag{3}$$

Let us illustrate these two rules by investigating the goal

$$?- \mathit{causes}(\mathit{unloaded} \circ \mathit{alive}, [\mathit{load}], X) \tag{4}$$

to examine what happens if we execute the *load* action in the situation above. (4) can be resolved to $?- \mathit{causes}(\mathit{loaded} \circ \mathit{alive}, [\,], X)$ using (3),

(1), and solving the AC1–unification problems. In particular, the respective instances of $C \circ V$ and I, viz. $\mathit{unloaded} \circ V$ and $\mathit{unloaded} \circ \mathit{alive}$, are AC1–unified using the substitution $\{V \mapsto \mathit{alive}\}$. This illustrates that the technical frame problem is solved by the variable V which carries over all those facts which are not affected by the action. It also illustrates how the concept of dealing with facts as resources is handled in this approach. Via clause (3), the fact *unloaded* is removed from the actual situation, ie. it is consumed, whereas the fact *loaded* is added to the situation, ie. it is produced. Finally, the goal $?- \mathit{causes}(\mathit{loaded} \circ \mathit{alive}, [\,], X)$ can be solved with computed answer substitution $\{X \mapsto \mathit{loaded} \circ \mathit{alive}\}$ by applying (2) and performing an AC1–unification step.

The three recent approaches [3, 22, 16] turned out to be equivalent for planning problems, where situations as well as the conditions and effects of actions are conjunctions of atomic facts [27, 13]. This result does not only provide a standard semantics for fragments of the linear logic and the linear connection method, it also suggests that resources can be treated within classical logic — viz. by using equational logic — without loosing expressive power.

Let us examine the equational logic approach of [16] more closely. The axioms AC1 for the function symbol $\circ$ essentially define the data structure multisets. Hence, a situation $\mathcal{S}$ is a multiset of facts where consumption means removing elements and production means adding elements [13]. As each description of an action is defined by its name a, the multiset $\mathcal{C}$ of its conditions, and the multiset $\mathcal{E}$ of its effects, $\alpha = \langle \mathcal{C}, a, \mathcal{E} \rangle$ is applicable in $\mathcal{S}$ if $\mathcal{C} \dot{\subseteq} \mathcal{S}$ and, if executed, transforms $\mathcal{S}$ into $\mathcal{S}' = (\mathcal{S} \dot{-} \mathcal{C}) \dot{\cup} \mathcal{E}$.[2] Thus, planning in this approach is closely related to planning in STRIPS [9, 21] except that multisets are used instead of sets and that planning is performed in a purely deductive system. As argued in [13] multisets represent resources more adequately than sets and, moreover, it is more efficient to compute with multisets instead of sets.

In [13] it was also shown that the equational logic approach can handle database transaction and objects in much the same way as database transactions and objects are handled in [26] and [1], respectively. As an example consider the class hierarchy depicted in Figure 1 and suppose that an action $\mathit{move}(O, L_1, L_2)$ has been defined for the class *object* which moves an object O from location L_1 to location L_2. The conditions and effects of this action are $\{\!|\mathit{on}(O, L_1)|\!\}$ and $\{\!|\mathit{on}(O, L_2)|\!\}$, respectively. Such an action can be represented in the equational logic approach by

$$\begin{aligned} \mathit{action}(C, \mathit{move}(O, L_1, L_2), E) \;\leftarrow\; & C =_{\mathrm{AC1}} \mathit{on}(O, L_1) \;\wedge \\ & E =_{\mathrm{AC1}} \mathit{on}(O, L_2). \end{aligned} \tag{5}$$

Using clause (3) it can be applied to move objects. But it can also be applied to move fragile objects like eg. vases. For instance, the goal

$$?- \;\; \mathit{causes}(\mathit{on}(\mathit{vase}, \mathit{table}) \circ \mathit{fragile}(\mathit{vase}), [\mathit{move}(\mathit{vase}, \mathit{table}, \mathit{board})], X).$$

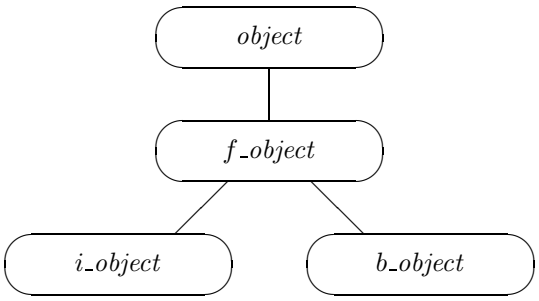

Figure 1: A hierarchy of classes representing objects $(object)$, fragile objects (f_object), intact objects (i_object), and broken objects (b_object). An object belongs to the class f_object if it is known to be fragile. A fragile object belongs to the classes i_object or b_object if it is known that it is intact or broken, respectively.

can be solved with computed answer substitution $\{X \mapsto on(vase, board) \circ fragile(vase)\}$ by applying (3), (5), (2), and performing the required AC1–unification computations. This demonstrates that actions defined for a certain class are automatically inherited by its subclasses. It also demonstrates that variables such as O, L_1, and L_2 may occur within the conditions, the effects, and the name of an action.

In an object oriented framework, however, we do not only want to inherit descriptions of actions, but we also want to override action descriptions if a more specific description is defined for a particular subclass. For example, if a robot holding a solid object like a silver bar drops the object then the object will remain as it was, ie. nothing is changed:

$$action(C, drop, E) \leftarrow C =_{\mathrm{AC1}} \emptyset \wedge E =_{\mathrm{AC1}} \emptyset. \tag{6}$$

However, if a robot holding a fragile object like a vase drops the object, then the object will be broken afterwards (to keep this example small we omit the argument of the *fragile* function and use it as a constant):

$$action(C, drop, E) \leftarrow C =_{\mathrm{AC1}} fragile \wedge E =_{\mathrm{AC1}} fragile \circ broken. \tag{7}$$

Now, in a situation where a robot holds a fragile object, ie. solving the query $?- causes(fragile, [drop], X)$, both action descriptions (6) and (7) are applicable using (3). (6) is inherited from the class $object$. However, (7) is defined for the class f_object and, thus, is more specific than (6) and should be preferred.[3] The approaches of [1] or [13] do not model the concept of overriding through specificity in class hierarchies. Neither does the approach of [26] allow to specify more specific transactions in deductive databases.

In this paper we extend the equational logic approach of [16, 13] such that the most specific action description is preferred. Specificity and the

use of multisets to represent situations are discussed more thoroughly in the following Section 2, where we also illustrate the approach by an additional example from the Yale Shooting domain. Section 3 shows how completed logic programs in the sense of [6] along with a unification–complete (see [18]) AC1–theory capture specificity. Informally, the most specific action description is preferred by adding a negative literal of the form $\neg \mathit{non\_specific}(A, C, V)$ to the body of (3), which guarantees that the selected description of action A with conditions C is the most specific applicable action description in the situation $C \circ V$. One should observe that such a logic program does not belong to the class of programs investigated in [18] as a negative literal occurs in the body of a clause. On the other hand, it also does not belong to the class of programs investigated in eg. [2] as our program contains the equational theory AC1. Thus, our program falls into a class which has recently been investigated in [28]. Section 4 focuses on models which assign to $\circ$ and $\emptyset$ operators which can be used to build up multisets. Section 5 introduces SLDENF–resolution as SLDNF–resolution extended by a unification algorithm for an equational theory. Section 6 shows how SLDENF–resolution can be used as a sound and complete computation procedure for logic programs modelling actions and specificity. Finally, Section 7 discusses these results and outlines possible extensions. In particular, our results are related to a semantical approach to reason about actions developed by Gelfond and Lifschitz [11].

2 Specificity

In the introduction we have informally motivated specificity with the help of the Fragile Object example. In this section we will formally define specificity. To ease our presentation we restrict actions to those whose conditions and effect are ground.

As we do not want to allow explicit negation in situations[4] we have to specify inconsistency for the domain of discourse. Intuitively, in the Fragile Object domain an object cannot be intact and broken at the same time. Furthermore, as we are dealing with multisets, we may want to consider the number of occurrences of a fact in a multiset. For example, a single occurrence of *broken* may be interpreted as the fact that the object is broken into a few pieces, whereas several occurrences of *broken* may be interpreted as the fact that the object is shattered into many pieces. Thus, fuzziness may be expressed. However, for the sake of simplicity we assume that no fact should occur more than once in a consistent situation, ie. for the Fragile Object domain we specify that a situation $\mathcal{S}$ is inconsistent iff

$$\{\mathit{broken}, \mathit{intact}\} \dot{\subseteq} \mathcal{S} \ \vee \ \exists X \in \{\mathit{fragile}, \mathit{broken}, \mathit{intact}\}.\ \{X, X\} \dot{\subseteq} \mathcal{S}. \quad (8)$$

Having specified inconsistency and defining that a situation is consistent iff it is not inconsistent we can now concentrate on specificity.

Recall that an action description $\alpha = \langle \mathcal{C}, a, \mathcal{E} \rangle$ is defined by the multiset $\mathcal{C}$ of its conditions, its name a, and by the multiset $\mathcal{E}$ of its effects. Given a finite set $\mathcal{A}$ of action descriptions, we define a partial order with regard to specificity as follows. An action description $\alpha_1 = \langle \mathcal{C}_1, a_1, \mathcal{E}_1 \rangle$ is said to be *more specific* than an action description $\alpha_2 = \langle \mathcal{C}_2, a_2, \mathcal{E}_2 \rangle$ (written $\alpha_1 < \alpha_2$) iff $a_1 = a_2$ and $\mathcal{C}_1 \dot{\supset} \mathcal{C}_2$.

In our example we have already considered the two action descriptions

$$\langle \{\}, drop, \{\} \rangle \tag{9}$$

$$\text{and} \quad \langle \{fragile\}, drop, \{fragile, broken\} \rangle, \tag{10}$$

which were represented by (6) and (7), respectively. Clearly, (10) < (9). These two action descriptions do not yet completely specify the Fragile Object domain: If the robot drops an object belonging to the class *i_object*, then the description of *drop* inherited from the class *f_object* could be applied. This, however, would lead to an inconsistent situation, where the object is intact and broken at the same time. To avoid such a behavior we define the additional action description

$$\langle \{intact, fragile\}, drop, \{fragile, broken\} \rangle. \tag{11}$$

As (11) is more specific than (10) it will be preferred if the dropped object is known to be intact. Similarly, if the dropped object is already broken, ie. if the current situation is $\{broken, fragile\}$, then the execution of (10) would lead to the situation $\{broken, broken, fragile\}$. This can be avoided by defining a more specific action description for the class *b_object*, namely

$$\langle \{broken, fragile\}, drop, \{fragile, broken\} \rangle. \tag{12}$$

Altogether we obtain a set of action descriptions such that whenever an applicable and most specific description is executed in a consistent situation the resulting situation is also consistent. A formal proof for this set of action descriptions being consistency preserving is analogous to the respective proofs for the Blocksworld domain in [16].

Hierarchies of classes can be used in general to describe scenarios of actions and change. Eg., the Yale Shooting environment ([14]) consists of a gun which might be unloaded or loaded, a turkey which is alive or dead, and three actions, viz. loading the gun (*load*), shooting (*shoot*), and waiting (*wait*). The reader is invited to prove that the following set of action descriptions forms a complete set for the various consistent situations — where consistency is defined analogously to (8) — in the Yale Shooting domain:

$$\begin{array}{rr}
\langle \{\}, wait, \{\} \rangle & \langle \{\}, shoot, \{unloaded\} \rangle \\
\langle \{\}, load, \{loaded\} \rangle & \langle \{loaded\}, shoot, \{dead, unloaded\} \rangle \\
\langle \{loaded\}, load, \{loaded\} \rangle & \langle \{unloaded\}, shoot, \{unloaded\} \rangle \\
\langle \{unloaded\}, load, \{loaded\} \rangle & \langle \{loaded, alive\}, shoot, \{dead, unloaded\} \rangle \\
 & \langle \{loaded, dead\}, shoot, \{dead, unloaded\} \rangle
\end{array}$$

This example can be enriched in many directions using the expressive power of the concept of multisets. For instance, a double–barrelled gun can be modelled as having three different states: It is either unloaded ($\{\!\{unloaded\}\!\}$), loaded with one ($\{\!\{loaded\}\!\}$) or two pellets ($\{\!\{loaded, loaded\}\!\}$). Modelling such a gun requires to modify the consistency criteria such that a situation is inconsistent if *loaded* occurs more than twice. It also requires to modify the action descriptions of *load* and *shoot* in the obvious way, eg. $\langle \{\!\{loaded\}\!\}, load, \{\!\{loaded, loaded\}\!\} \rangle$ should occur in the new set of action descriptions.

3 Completed Equational Logic Programs

In this section we present a completed logic program in the sense of [6] along with a unification complete equational theory in the sense of [18] which can be used to define actions and specificity.

As already argued in the introduction, multisets can be represented using a constant $\emptyset$ denoting the empty multiset and a binary AC1–function symbol $\circ$ such that $\emptyset$ is a unit element of $\circ$. More formally, let $\cdot^{\mathcal{I}}$ and $\cdot^{\mathcal{I}^{-1}}$ be two mappings such that $\emptyset^{\mathcal{I}} = \{\!\}$, $(s_1 \circ s_2 \circ \ldots \circ s_n)^{\mathcal{I}} = \{\!\{s_1\}\!\} \dot\cup (s_2 \circ \ldots \circ s_n)^{\mathcal{I}}$, $\{\!\}^{\mathcal{I}^{-1}} = \emptyset$ and $\{\!\{s_1, s_2, \ldots, s_n\}\!\}^{\mathcal{I}^{-1}} = s_1 \circ \{\!\{s_2, \ldots, s_n\}\!\}^{\mathcal{I}^{-1}}$. One should observe that $(s^{\mathcal{I}})^{\mathcal{I}^{-1}} =_{\mathrm{AC1}} s$ and $(\mathcal{S}^{\mathcal{I}^{-1}})^{\mathcal{I}} \doteq \mathcal{S}$. For the sake of simplicity we assume that terms are well–sorted in the following sense. *Elementary terms* are non–variable terms built up from a given alphabet not containing $\emptyset$ and $\circ$ as usual. $\circ$*–terms* are defined inductively: $\emptyset$ and each elementary term is a $\circ$–term, and if s and t are $\circ$–terms then $s \circ t$ is a $\circ$–term. Finally, AC1*–terms* are either $\circ$–terms or of the form $s \circ X$, where s is a $\circ$–term and X is a variable called AC1*–variable* which must not occur elsewhere.

Proving equality of two AC1–terms under the equational theory AC1 requires the axioms of associativity, commutativity, and existence of a unit element $\emptyset$ for $\circ$ together with the general axioms of equality, ie. the axioms of reflexivity, symmetry, and transitivity along with the substitution schema. However, these axioms are not sufficient for completed logic programs where also inequalities must be provable. In other words, $s \neq_{\mathrm{AC1}} t$ has to be derivable whenever s and t are not AC1–unifiable.

Inequality axiom schemata for the empty equational theory are well–known and contain, among others, the unique name assumption. We use the appropriate axiom schemata described in [6] with the restriction that they must not be instantiated with the function symbol $\circ$. This symbol requires special axioms. As we intend to restrict equalities to equalities between AC1–terms the following two axiom schemata are sufficient. Informally, these schemata express the fact that two AC1–terms s and t are equal only if for each subterm of s there is a corresponding subterm in t and vice versa. The expression $\Pi(n)$ below denotes the set of all permutations over

the natural numbers $\{1, \ldots, n\}$:

$$\forall X, \overline{V}.\ [\, X \circ s_1 \circ \cdots \circ s_m \ =_{\mathrm{AC1}}\ t_1 \circ \cdots \circ t_n \ \rightarrow \bigvee_{\pi \in \Pi(n)} s_1 = t_{\pi(1)} \ \wedge\ \ldots\ \wedge\ s_m = t_{\pi(m)} \ \wedge\ X = t_{\pi(m+1)} \circ \cdots \circ t_{\pi(n)}\,] \tag{13}$$

for any $n \geq m \geq 1$ and any elementary terms $s_1, \ldots, s_m, t_1, \ldots, t_n$ with free variables $\overline{V}$, and

$$\forall X, \overline{V}.\ [\, X \circ s_1 \circ \cdots \circ s_m \ \neq_{\mathrm{AC1}}\ t_1 \circ \cdots \circ t_n\,] \tag{14}$$

for any $m > n \geq 0$ and any elementary terms $s_1, \ldots, s_m, t_1, \ldots, t_n$ with free variables $\overline{V}$. For instance, to derive that $X \circ \mathit{intact} \neq_{\mathrm{AC1}} \mathit{fragile} \circ \mathit{broken}$ via the contraposition of (13) we have to prove that $(\mathit{intact} \neq \mathit{fragile} \vee X \neq \mathit{broken}) \wedge (\mathit{intact} \neq \mathit{broken} \vee X \neq \mathit{fragile})$, which is a consequence from the unique name axioms. Note that neither in (13) nor in (14) both AC1–terms contain an AC1–variable since otherwise two terms are always unifiable, as it is required that each AC1–variable in an AC1–term has to be unique. In what follows, we abbreviate the union of AC1, (13), (14), and the general axioms for equality along with the (restricted) completion axioms of [6] by AC1*. AC1* can be shown to be complete in the sense that for any two AC1–terms s and t with variables $\overline{V}$ either $\mathrm{AC1}^* \models \exists \overline{V}.\, s =_{\mathrm{AC1}} t$ if s and t are AC1–unifiable, or $\mathrm{AC1}^* \models \forall \overline{V}.\, s \neq_{\mathrm{AC1}} t$, otherwise.[5]

To select the most specific, applicable action description in a given situation, the body of clause (3) is extended by the literal $\neg \mathit{non\_specific}(A, C, V)$, and we obtain the following completed definition.

$$\begin{aligned} \forall I, P, G.\ [\ \mathit{causes}(I, P, G) \ \leftrightarrow\ & (P = [\,] \ \wedge\ I =_{\mathrm{AC1}} G) \vee \\ & \exists A, P', C, E, V.\ (\ P = [A|P'] \ \wedge\ \mathit{action}(C, A, E) \wedge \\ & \qquad C \circ V =_{\mathrm{AC1}} I \ \wedge\ \neg \mathit{non\_specific}(A, C, V) \wedge \\ & \qquad \mathit{causes}(E \circ V, P', G)\ )\ ]. \end{aligned} \tag{15}$$

An action description of A with conditions C is non–specific in the situation $C \circ V$ iff there is a more specific, applicable action description:

$$\begin{aligned} \forall A, C, V.\, [\mathit{non\_specific}(A, C, V) \leftrightarrow \exists C', E', V', W.\ (& \mathit{action}(C', A, E') \wedge \\ & C' \circ V' =_{\mathrm{AC1}} C \circ V \ \wedge \\ & C \circ W =_{\mathrm{AC1}} C' \ \wedge\ W \neq_{\mathrm{AC1}} \emptyset)]. \end{aligned} \tag{16}$$

The elements $\alpha_i = \langle \mathcal{C}_i, a_i, \mathcal{E}_i \rangle$ of a set containing m action descriptions are represented by the following completed clause.

$$\forall C, A, E.\, [\mathit{action}(C, A, E) \ \leftrightarrow\ \bigvee_{i=1}^{m} (C = \mathcal{C}_i^{\mathcal{I}^{-1}} \ \wedge\ A = a_i \ \wedge\ E = \mathcal{E}_i^{\mathcal{I}^{-1}})]. \tag{17}$$

Finally, let C be the conjunction of all formulas of the form $\forall X_1, \ldots, X_n.\ \neg p(X_1, \ldots, X_n)$, where p is an n–ary predicate symbol not occurring in the set of predicates $\{=,\ =_{\mathrm{AC1}},\ \mathit{causes},\ \mathit{non\_specific},\ \mathit{action}\}$.

Let $P^* = (15) \wedge (16) \wedge (17) \wedge C$. Then, $(P^*, \mathrm{AC1}^*)$ is the completed equational logic program specifying actions and specificity. One should observe that various scenarios like the Fragile Object or the Yale Shooting domain differ only in the definition of the predicate *action*.

4 Models

For the purpose of this paper it suffices to consider standard first–order models $\mathcal{M} = (\cdot^{\mathcal{I}}, \mathcal{D})$ for the first–order formula $(P^*, \mathrm{AC1}^*)$, where $\mathcal{D}$ and $\cdot^{\mathcal{I}}$ denote the domain and the mapping of the model, respectively. In particular, we consider interpretations which interpret $\emptyset$ and $\circ$ as multiset–building operators, ie. $\cdot^{\mathcal{I}}$ is defined on these symbols as in Section 3. In addition, list expressions of the form $[h \,|\, t]$ are interpreted as usual. This is not a restriction as any other models for $\mathrm{AC1}^*$ can be mapped onto such a $(\cdot^{\mathcal{I}}, \mathcal{D})$ (see [17]).

Now, let $(P^*, \mathrm{AC1}^*)$ contain the definition of a set of action descriptions $\mathcal{A}$ and let $\mathcal{M} = (\cdot^{\mathcal{I}}, \mathcal{D})$ be a model for $(P^*, \mathrm{AC1}^*)$. Furthermore, let i and g be ground AC1–terms denoting the initial situation $i^{\mathcal{I}} \doteq \mathcal{S}_0$ and the goal situation $g^{\mathcal{I}} \doteq \mathcal{S}_n$, respectively, and let $a_1, \ldots, a_n$ denote action names. It is not difficult to see that $\mathcal{M} \models \mathit{causes}(i, [a_1, \ldots, a_n], g)^{\mathcal{I}}$ iff there is a sequence of multisets $(\mathcal{S}_j \mid 0 \leq j \leq n)$ and a sequence of action descriptions $(\alpha_j = \langle \mathcal{C}_j, a_j, \mathcal{E}_j \rangle \mid 1 \leq j \leq n)$ from $\mathcal{A}$ such that for all j, $1 \leq j \leq n$, we find that $\mathcal{S}_j = (\mathcal{S}_{j-1} \dot{-} C_j) \dot{\cup} \mathcal{E}_j$ and α_j is the most specific applicable action description of the action a_j in $\mathcal{S}_{j-1}$ wrt $\mathcal{A}$.

5 SLDENF–resolution

Proving in completed logic programs is known to be quite inefficient. We therefore do not want to compute with $(P^*, \mathrm{AC1}^*)$, rather we would like to compute with the if–halves of the definitions in P^*, to use AC1–unification instead of the unification complete theory $\mathrm{AC1}^*$, and to use negation–as–failure for deriving negative information. Let P be the following normal logic program:

$$\mathit{causes}(I, [\,], G) \leftarrow I =_{\mathrm{AC1}} G. \tag{18}$$

$$\mathit{causes}(I, [A|P], G) \leftarrow \mathit{action}(C, A, E) \wedge C \circ V =_{\mathrm{AC1}} I \wedge \neg \mathit{non\_specific}(A, C, V) \wedge \mathit{causes}(E \circ C, P, G). \tag{19}$$

$$\mathit{non\_specific}(A, C, V) \leftarrow \mathit{action}(C', A, E') \wedge C' \circ V' =_{\mathrm{AC1}} C \circ V \wedge C \circ W =_{\mathrm{AC1}} C' \wedge W \neq_{\mathrm{AC1}} \emptyset. \tag{20}$$

$$\mathit{action}(C, a_i, E) \leftarrow C =_{\mathrm{AC1}} c_i \wedge E =_{\mathrm{AC1}} e_i. \quad \text{for all actions } \langle c_i^{\mathcal{I}}, a_i, e_i^{\mathcal{I}} \rangle \in \mathcal{A}. \tag{21}$$

As our programs contain an equational theory we intend to build this theory into the unification computation. In particular, we are interested in

the AC1–unification of two AC1–terms, which is decidable, finitary, and for which a complete and minimal unification algorithm is known [13, 29]. The programs are carefully specified such that the need for AC1–unification is localized within calls to subgoals of the form $s =_{\mathrm{AC1}} t$ or $s \neq_{\mathrm{AC1}} t$, whereas all other subgoals can be solved by applying the usual unification procedure.

Following the ideas of [28], *SLDENF–resolution* is like SLDNF–resolution [6] if the selected literal is not of the form $s =_{\mathrm{AC1}} t$ or $s \neq_{\mathrm{AC1}} t$. If the selected literal is of the form $s =_{\mathrm{AC1}} t$ and s and t are AC1–terms, then the AC1–unification algorithm in [13, 29] is called, which either returns a minimal complete set of AC1–unifiers for s and t if both terms are AC1–unifiable, or returns a failure message otherwise. In the former case, the literal $s =_{\mathrm{AC1}} t$ is removed from the goal and one of the AC1–unifiers is applied to the remaining literals occurring in the goal. Thus, we obtain a finitely branching derivation tree. In the latter case the derivation fails. Conversely, if the selected literal is of the form $s \neq_{\mathrm{AC1}} t$ and s and t are AC1–terms, then the AC1–unification algorithm is called to unify s and t. If s and t are not AC1–unifiable, then the literal $s \neq_{\mathrm{AC1}} t$ is removed from the goal. Otherwise, the derivation fails.

As for the selection function we assume that it is fair, ie. that each literal occurring in a goal is selected after finitely many steps, that negative literals are selected only if they are ground, and that literals of the form $s =_{\mathrm{AC1}} t$ and $s \neq_{\mathrm{AC1}} t$ are selected only if s and t are AC1–terms. In the following section we show that the application of SLDENF–resolution to our equational logic program yields the intended results. General soundness and completeness results concerning SLDENF–resolution, which extend the results of [28], can be found in [17].

6 Soundness and Completeness of SLDENF–Resolution

In this section we assume that i and g are ground AC1–terms denoting an initial situation $i^{\mathcal{I}}$ and a goal situation $g^{\mathcal{I}}$, respectively. Furthermore, p denotes a list of action names $[a_1, \ldots, a_n]$. Note that SLDENF–refutations are defined with respect to an equational logic program like $(P, \mathrm{AC1})$, whereas models $\mathcal{M} = (\cdot^I, \mathcal{D})$ are defined for the corresponding completed equational logic program $(P^*, \mathrm{AC1}^*)$.

Theorem 6.1 *(Soundness.) If there exists an SLDENF–refutation of* $? \leftarrow \mathit{causes}(i, p, g)$ *then for each model* $\mathcal{M}$ *we find that* $\mathcal{M} \models \mathit{causes}(i, p, g)^{\mathcal{I}}$.

Proof (sketch): The proof is a straightforward induction on the length of the sequence p of actions. The interesting parts of this proof are the two cases where negation–as–failure is used. The first one is concerned with the subgoal $W \neq_{\mathrm{AC1}} \emptyset$, where W is bound to a ground AC1–term. As AC1–unification is decidable and a correct and complete unification algorithm is

known, this subgoal will be evaluated to true whenever an attempt to unify the respective instance of W and $\emptyset$ fails. The second case is concerned with the subgoal $\neg non\_specific(A, C, V)$, where A is bound to an action name a and C and V are bound to ground AC1–terms c and v, respectively. Such a subgoal is called if an action description $\langle c^{\mathcal{I}}, a, e^{\mathcal{I}} \rangle$ has been selected and it has to be checked whether this is the most specific, applicable description in the situation $(c \circ v)^{\mathcal{I}}$. In this case we apply the following Lemma 6.2. ■

Lemma 6.2 *An SLDENF–derivation of* $?\!- non\_specific(a, c, v)$ *finitely fails iff there is no action description* $\langle \mathcal{C}', a, \mathcal{E}' \rangle \in \mathcal{A}$ *such that* $\mathcal{C}' \dot{\subseteq} (c \circ v)^{\mathcal{I}}$ *and* $\mathcal{C}' \dot{\supset} c^{\mathcal{I}}$.

Proof: To prove the only–if–half we assume that an SLDENF–derivation of $?\!- non\_specific(a, c, v)$ finitely fails. Using (20) this goal is replaced by

$$?\!- action(C', a, E') \wedge C' \circ V' =_{\mathrm{AC1}} c \circ v \wedge c \circ W =_{\mathrm{AC1}} C' \wedge W \neq_{\mathrm{AC1}} \emptyset. \quad (22)$$

Without loss of generality we assume that this goal is evaluated from left to right. As $\mathcal{A}$ contains only finitely many action descriptions, the program P contains only finitely many rules of the form (21). For each action description $\langle \mathcal{C}', a', \mathcal{E}' \rangle \in \mathcal{A}$ we find that if $a' \neq a$ then the derivation fails; the action has another name. Otherwise, the derivation continues with C' and E' bound to $c' = \mathcal{C}'^{\mathcal{I}^{-1}}$ and $e' = \mathcal{E}'^{\mathcal{I}^{-1}}$, respectively. The unification attempt for the literal $c' \circ V' =_{\mathrm{AC1}} c \circ v$ fails iff $\mathcal{C}' \dot{\not\subseteq} (c \circ v)^{\mathcal{I}}$. Otherwise, $\mathcal{C}' \dot{\subseteq} (c \circ v)^{\mathcal{I}}$ and (22) reduces to

$$?\!- c \circ W =_{\mathrm{AC1}} c' \ \wedge \ W \neq_{\mathrm{AC1}} \emptyset.$$

The subgoal $c \circ W =_{\mathrm{AC1}} c'$ fails iff $\mathcal{C}' \dot{\not\supseteq} c^{\mathcal{I}}$. Otherwise, $\mathcal{C}' \dot{\supseteq} c^{\mathcal{I}}$ and W will be bound to $w = (\mathcal{C}' \dot{-} c^{\mathcal{I}})^{\mathcal{I}^{-1}}$. In this case, if w is equal to $\emptyset$ modulo AC1, ie. if the derivation of $?\!- w \neq_{\mathrm{AC1}} \emptyset$ fails, then $\mathcal{C}' \doteq c^{\mathcal{I}}$. Altogether, whenever a derivation of (22) fails, then either $a \neq a'$ or $\mathcal{C}' \dot{\not\subseteq} (c \circ v)^{\mathcal{I}}$ or $\mathcal{C}' \dot{\not\supset} c^{\mathcal{I}}$. The result follows immediately from the observation that the derivation tree of (22) is finite.

To prove the if–half, assume that there is no action description $\langle \mathcal{C}', a, \mathcal{E}' \rangle \in \mathcal{A}$ such that $\mathcal{C}' \dot{\subseteq} (c \circ v)^{\mathcal{I}}$ and $\mathcal{C}' \dot{\supset} c^{\mathcal{I}}$. Suppose there is an SLDENF–refutation of $?\!- non\_specific(a, c, v)$. Then, as (20) is the only rule in the program P for *non_specific*, we find a refutation of (22). Hence, we find an action description $\langle \mathcal{C}', a, \mathcal{E}' \rangle \in \mathcal{A}$ and a ground AC1–term w such that $\mathcal{C}' \dot{\subseteq} (c \circ v)^{\mathcal{I}}$, $(c \circ w)^{\mathcal{I}} \doteq \mathcal{C}'$, and $w^{\mathcal{I}} \not\doteq \{\}$. The last two facts translate to $c^{\mathcal{I}} \dot{\subset} \mathcal{C}'$, which contradicts the initial premise. ■

Theorem 6.3 *(Completeness.) If we find that* $\mathcal{M} \models causes(i, p, g)^{\mathcal{I}}$ *for each model* $\mathcal{M}$ *then there exists an SLDENF–refutation of* $?\!- causes(i, p, g)$.

Proof (sketch): The proof is again a straightforward induction on the length of the expression $p^{\mathcal{I}}$ using Lemma 6.2 and the fact that AC1* is complete. ■

7 Discussion

We have presented an equational logic approach to reasoning about situations, actions, and change, where situations are multisets of facts and an action is applied to a situation $\mathcal{S}$ by deleting its conditions from and adding its effects to $\mathcal{S}$. In particular, we have focused on specificity such that more specific action descriptions are preferred. This solves an open problem in the approaches of [1, 13] or [26].

Aside from checking whether a given sequence of actions transforms a given situation into a goal situation, our approach can as well be used to generate a plan, ie. a sequence of actions, as a computed answer substitution to a query of the form $?- \mathit{causes}(i, P, g)$ (cf. [16, 13]).

We have restricted our presentation to actions whose conditions and effects are ground, and to situations which are ground as well. As already indicated by an example in the introduction, we may lift this restriction and allow variables to occur in situations as well as in the conditions and effects of actions. This causes no problems as long as we focus on the completed equational logic program and lift the definition of specificity as well. However, as soon as negation–as–failure is applied, we have to be more careful. Not only must we ensure that negative subgoals are fully instantiated before they are called, but inconsistencies may be derived if the initial situation is only partially specified. This will be discussed in the next paragraph.

Reasoning about the past

Aside from temporal projection and planning, we are also interested in the ability of deriving information about former situations. As an example, recall the Fragile Object domain from Section 1. If we know that a previously intact object is broken after it was dropped, then we want to conclude that the object is fragile. Can we derive such a conclusion with our equational logic program? To answer this question, we relate our approach to the action description language developed by Gelfond and Lifschitz.

The language described in [11] consists of action names and fluent names, which might occur negated, along with expressions such as

$$\mathit{drop} \ \texttt{causes} \ \mathit{broken} \ \texttt{if} \ \mathit{fragile}. \tag{23}$$

These so–called *e–propositions* describe the effect of an action (*drop*) on a single fluent (*broken*) provided a number of conditions (*fragile*) hold. The set of e–propositions describing a domain is used to define a transition function which maps situations into situations given a particular action name. A situation $\mathcal{S}$ is a set of fluent names and it is assumed that f holds in $\mathcal{S}$ iff $f \in \mathcal{S}$ and that $\neg f$ holds in $\mathcal{S}$ iff $f \notin \mathcal{S}$. Eg., in the Fragile Object domain the set $\{\mathit{fragile}\}$ describes a situation where *fragile* and $\neg \mathit{broken}$

hold, and the transition function Φ determined by (23) is

$$\Phi(\mathit{drop}, \mathcal{S}) \;=\; \begin{cases} \mathcal{S} \cup \{\mathit{broken}\}, & \text{if } \mathit{fragile} \in \mathcal{S} \\ \mathcal{S}, & \text{otherwise.} \end{cases} \tag{24}$$

So–called *v–propositions* are used to describe the value of a single fluent in a particular situation. For example, the two v–propositions

$$\texttt{initially}\ \neg \mathit{broken} \quad \text{and} \quad \mathit{broken}\ \texttt{after}\ \mathit{drop} \tag{25}$$

describe the fact that the object is not broken in the initial situation and is broken after executing the *drop* action, respectively. A *model* for a set of e–propositions and v–propositions consists of a transition function Φ along with a particular situation $\mathcal{S}_0$ — called the initial situation — such that Φ is determined by the e–propositions, and Φ and $\mathcal{S}_0$ satisfy the v–propositions. In our example, Φ is defined as in (24) and, due to (25), is required that $\mathit{broken} \notin \mathcal{S}_0$ and $\mathit{broken} \in \Phi(\mathit{drop}, \mathcal{S}_0)$. It is easy to verify that $\mathit{fragile} \in \mathcal{S}_0$ holds in each such model, ie. we are allowed to conclude that the object was fragile before having dropped it. In other words, the v–proposition `initially` *fragile* is a consequence of the domain description.

Gelfond and Lifschitz give a translation for this language into an extended logic program with two kinds of negation (cf. [10]). The soundness of this translation is proved, ie. each conclusion drawn by the logic program holds in every model of the domain description. However, they have also shown by a simple counterexample that their translation is incomplete. In the sequel, we briefly sketch how to encode the action description language in our approach, ie. in terms of a completed equational logic program for actions and specificity.

As we do not support explicit negation, we need two different fluents for each fluent name of the domain description. In case of the Fragile Object domain we use the set $\{\mathit{fragile}, \overline{\mathit{fragile}}, \mathit{broken}, \overline{\mathit{broken}}\}$ where $\overline{f}$ should be interpreted as an independent fluent denoting the negation of the fluent name f. Since situations in the action description language are assumed to be sets of fluents, we do not allow any fluent to occur more than once in an AC1–term. In addition, a fluent must not occur together with its negation and each fluent is required to occur either affirmatively or negatively. To create an appropriate set of action descriptions, we use a simple and straightforwardly automated transformation, which creates exactly the four action descriptions (9)–(12) of Section 2 given the e–proposition (23).[6] Due to lack of space, we omit a formal description of this transformation here. It can be found in [30].

Reasoning about the past is mainly based on finding consistent explanations for observations. Recall that the actions for the Fragile Object domain were designed such that the application of an action to a consistent situation yields a consistent situation. Now, however, we have to ensure that additional derived facts of a situation are consistent with the known facts,

which is why the completed clause (15) has to be modified such that the initial situation is tested for consistency.

Now, given the completed logic program of a domain description, how to decide which v–propositions are entailed, eg. given the v–propositions (25), how to decide whether the object was necessarily fragile? The key idea is to ask how the various v–propositions can be satisfied if the initial situation is left unspecified. Unspecified means that a variable is used to denote that more facts may hold in the initial situation, ie.

$$\begin{array}{l} \exists X, Y.\ \mathit{causes}\,(\overline{\mathit{broken}} \circ \mathit{fragile} \circ X, [\mathit{drop}], \mathit{broken} \circ Y) \wedge \\ \forall V, W.\ \neg\, \mathit{causes}\,(\overline{\mathit{broken}} \circ \overline{\mathit{fragile}} \circ V, [\mathit{drop}], \mathit{broken} \circ W) \end{array} \tag{26}$$

can be used to ask whether the object was necessarily fragile at the beginning if we observe that it is broken after having dropped it and that it was intact before. The negative part of this conjunction is used to ensure that *fragile* is not an irrelevant fact when trying to satisfy the positive part of the query. The entailment of (26) could be interpreted as `initially` *fragile* which would be exactly the desired result. (26) is in fact entailed by our modified completed program. The crucial point is the second part of this conjunction. Informally, the only possibility for unifying the goal situation $\mathit{broken} \circ W$ with the result of applying an action description of *drop* to the initial situation $\overline{\mathit{broken}} \circ \overline{\mathit{fragile}} \circ V$ requires V to be substituted by a term like $\mathit{fragile} \circ Z$. This, however, can be shown to be inconsistent for any Z since *fragile* and $\overline{\mathit{fragile}}$ must not occur in a situation due to the consistency criterion. Note that by asking (26) we have guessed that *fragile* might have been true in the initial situation. This is not necessary in general of course, since leaving the initial situation totally unspecified, the various answer substitutions to a query such as $\exists X, Y.\ \mathit{causes}\,(X, [\mathit{drop}], \mathit{broken} \circ Y)$ should give us a hint of what might have been true at the beginning.

In [30] it is shown that the ideas sketched above can be used as a sound and complete implementation of the action description language of Gelfond and Lifschitz. This makes our approach to reason about actions and change comparable to various other approaches which were also related to this language recently, such as [8, 19, 7].

A problem arises when the SLDENF–approach of Section 5 is used to pose queries as eg. the negative part of (26). The problem is a consequence from the fact that negation–as–failure does not allow for solving negative non–ground literals. In [17] we propose to use the concept of *constructive negation* [5] to be able to compute queries such as (26).

Reasoning about the past often requires also to find out the sequence of actions which has been performed. This is what detectives must do when solving a crime or natural scientists who have to explain observations. In general, problems like this show an enormous search space and require a large number of efficient heuristics in practical applications, but in principle they can be formulated in our approach as it stands.

Sequences of Actions

Hitherto we have investigated specificity with respect to the conditions of action descriptions. Another way to receive — at first glance — inconsistent descriptions of causality occurs when a sequence of actions has another effect than the application of each element of this sequence one after another.

Assume we have parked a car without closing the door. Further, assume the only action to be a *wait* action. Normally waiting has no effects. However, if we wait too long then we must expect that the car is stolen. With an extended notion of action descriptions, this example can be encoded by $\langle \{\}, [wait], \{\} \rangle$ and $\langle \{parked\}, [wait, wait, wait], \{stolen\} \rangle$. Fortunately, as before, we find a syntactic criterion for preferring the — in some sense — more specific derivation if both are applicable, viz. the sequence of actions $[wait, wait, wait]$ is a superlist of $[wait]$. By slightly modifying our program, this can be straightforwardly encoded within the definition of *non_specific*.

Acknowledgements

This work was supported in part by ESPRIT within basic research action MEDLAR-II under grant no. 6471.

Notes

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

2. Multisets are depicted using the brackets $\{\ \}$. Furthermore, $\dot\cup$, $\dot-$, $\dot\subseteq$, $\doteq$, etc. denote the multiset extensions of the usual set operations and relations $\cup$, $-$, $\subseteq$, $=$, etc. More formally, if an element occurs m–times in a multiset $\mathcal{M}$ and n–times in a multiset $\mathcal{N}$, then it occurs $m+n$–times in $\mathcal{M} \dot\cup \mathcal{N}$ and $m-n$–times in $\mathcal{M} \dot- \mathcal{N}$ if $m > n$ and not in $\mathcal{M} \dot- \mathcal{N}$ if $m \leq n$. If each element occurring m–times in $\mathcal{M}$ occurs $n \leq m$–times in $\mathcal{N}$ then $\mathcal{N} \dot\subseteq \mathcal{M}$, and if $\mathcal{M} \dot\subseteq \mathcal{N}$ as well as $\mathcal{N} \dot\subseteq \mathcal{M}$ then $\mathcal{M} \doteq \mathcal{N}$.

3. Although the owner of the vase may think different in this particular case.

4. It is an interesting philosophical question whether rational agents have a general concept of negation comparable to the concept of negation in first–order logic. This was brought to our attention by J. A. Robinson.

5. Note that AC1* is not a complete unification theory for AC1 in exactly the sense of [18] or [28], rather it is restricted to AC1–terms and to this completeness criterion. However, this is sufficient in our case.

6. Thereby *intact* must be substituted by $\overline{broken}$. Observe that the introduction of $\overline{fragile}$ does not force to define more action descriptions.

References

[1] J.-M. Andreoli and R. Pareschi. Linear Objects: Logical Processes with built-in Inheritance. *New Generation Computing*, 9(3+4), 1991.

[2] K. 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*, 89–148. Morgan Kaufmann, 1987.

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

[4] W. Bibel. Intellectics. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, 705–706, New York, 1992.

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

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

[7] M. Denecker and D. de Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. In D. Miller, editor, *Proc. of ILPS*. 1993.

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

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

[10] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

[11] M. Gelfond and V. Lifschitz. Representing Actions in Extended Logic Programming. In K. Apt, editor, *Proc. of IJCSLP*, 559–573. 1992.

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

[13] G. Große, S. Hölldobler, J. Schneeberger, U. Sigmund, and M. Thielscher. Equational Logic Programming, Actions, and Change. In K. Apt, editor, *Proc. of IJCSLP*, 177–191. 1992.

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

[15] S. Hölldobler. On Deductive Planning and the Frame Problem. In A. Voronkov, editor, *Proc. of the Int. Conf. on Logic Programming and Automated Reasoning (LPAR)*, 13–29. Volume 624 of LNAI, 1992.

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

[17] S. Hölldobler and M. Thielscher. Computing Change and Specificity with Equational Logic Programs. 1993 (submitted).

[18] J. Jaffar, J-L. Lassez, and M. J. Maher. A theory of complete logic programs with equality. In *Proc. of the Int. Conf. on FGCS*, 175–184. ICOT, 1984.

[19] G. N. Kartha. Soundness and Completeness Theorems for Three Formalizations of Actions. In *Proc. of IJCAI*, 1993. (to appear).

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

[21] V. Lifschitz. On the Semantics of STRIPS. In *Proc. of the Workshop on Reasoning about Actions and Plans*. Morgan Kaufmann, 1986.

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

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

[24] J. McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

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

[26] R. Reiter. Formalizing database evolution in the situation calculus. In *Proc. of the Int. Conf. on FGCS*, 1992.

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

[28] J. S. Sheperdson. SLDNF–Resolution with Equality. *Journal of Automated Reasoning*, 8:297–306, 1992.

[29] M. Thielscher. AC1-Unifikation in der linearen logischen Programmierung. Diplomarbeit, Intellektik, TH Darmstadt, 1992.

[30] M. Thielscher. Representing Actions in Equational Logic Programming. 1993. (forthcoming paper).