Annals of Mathematics and Artificial Intelligence 14(1995)99–133

Computing change and specificity with equational logic programs

Steffen Hölldobler

Wissensverarbeitung, Informatik, TU Dresden, 01062 Dresden, Germany

e-mail: sh@inf.tu-dresden.de

Michael Thielscher

Intellektik, Informatik, TH Darmstadt, Alexanderstrasse 10, 64283 Darmstadt, Germany

e-mail: mit@intellektik.informatik.th-darmstadt.de

Recent deductive approaches to reasoning about action and chance allow us to model objects and methods in a deductive framework. In these approaches, inheritance of methods comes for free, whereas overriding of methods is unsupported. In this paper, we present an equational logic framework for objects, methods, inheritance and overriding of methods. Overriding is achieved via the concept of specificity, which states that more specific methods are preferred to less specific ones. Specificity is computed with the help of negation as failure. We specify equational logic programs and show that their completed versions behave as intended. Furthermore, we prove that SLDENF-resolution is complete if the equational theory is finitary, the completed programs are consistent and no derivation flounders or is infinite. Moreover, we give syntactic conditions which guarantee that no derivation flounders or is infinite. Finally, we discuss how the approach can be extended to reasoning about the past in the context of incompletely specified objects or situations. It will turn out that constructive negation is needed to solve these problems.

1. Introduction

Logic plays an important role for human thinking. 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. For instance, the examination of the ability of humans to reason about actions and change is one of the major parts of interest in Intellectics, i.e. Artificial Intelligence and Cognitive Science [7]. A particular description of the world consists of facts (or fluents) which are believed to hold at a certain instant. An important

© J.C. Baltzer AG, Science Publishers

property of these facts is that they are time dependent, i.e. the truth value of a proposition may change in the course of time. In contrast, classical logic seems to have difficulties in modelling non-static 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 an implication like *execute*(*load*) $\rightarrow$ *loaded*. Then, unfortunately, classical logic tells us that the action *load* can never be executed since otherwise the world becomes inconsistent.[1)]

The problem of classical logic is that propositions are not treated as resources which could be produced and consumed in the course of time [24]. To overcome these difficulties, McCarthy and Hayes [39,36] defined the situation calculus which mainly consists of using an additional argument to state that a particular fact only holds in a particular situation. In our example, this can be done by using the symbols s_0 and s_1 to denote the situation where the gun is unloaded and the situation after executing the *load* action, respectively, i.e. $\neg$*loaded*(s_0) and *execute*(*load*) $\rightarrow$ *loaded*(s_1). Now, the action can be executed without causing an inconsistency problem. However, this formalization leads to the well-known technical frame problem. Suppose that in an old Asiatic antique shop a Chinese vase is standing on a table in situation s_0, formalized by *on*(*chinese-vase*, *table*, s_0). Does the Chinese vase still stand on the table after loading the gun? In the absence of any other information, we assume that it does. But our formalization does not reflect this since the proposition *on*(*chinese-vase*, *table*, s_1) is not entailed by the axioms.

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 [39] solved this problem by employing 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. Kowalski reduced the number of frame axioms to become linear with respect to the number of different actions [32]. He introduced a special predicate named *Holds*(f, s), with the intended meaning that the reified fact f holds in the situation s. Some years later, it was again McCarthy who proposed the use of non-monotonic inference rules to tackle the frame problem [38]. He used 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, Reiter presented another solution to the technical frame problem within the situation calculus by introducing a so-called successor state axiom for each fluent [41].

Over the last years, three new deductive approaches to deal with situations, actions, and change were proposed, each of them without the need to state frame axioms explicitly. The linear connection method restricts proofs such that each literal is connected at most once [6]. Thus, connecting a literal during the inference process

[1)] One might suggest to use a literal like *unloaded* instead of the negation $\neg$*loaded*. However, after executing the *load* action, both literals *unloaded* and *loaded* are entailed, which is no improvement.

simulated consumption of the corresponding fact. Conversely, if the conditions of an implication are satisfied, 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 [18] 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 [35]. In the equational logic programming approach [25], facts about a situation are represented as multisets of facts on the term level. An action α with condition C and effect $\mathcal{E}$ is applicable in a situation S if C is contained in S, and, if α is applied in S, then C is deleted from S and $\mathcal{E}$ is added. Thus, planning in the equational logic approach is closely related to planning in STRIPS [14,33] except that multisets are used instead of sets and that planning is performed in a purely deductive system. As argued in [20], multisets represent resources more adequately than sets and, moreover, it is more efficient to compute with multisets instead of sets. Furthermore, the concept of resources and multisets are more adequate solutions to the frame problem [27].

The three recent approaches [6,35,25] are equivalent for planning problems, where the condition and effect of actions are multisets of facts [46,19]. In [20], it is also shown that the equational logic approach can handle database transactions as well as objects and methods in much the same way as database transactions as well as objects and methods are handled in [42] and [1], respectively. It has turned out that the inheritance of methods comes for free in the approaches of [1] and [20], but that neither approach allows overriding of methods. As an example, consider a scenario with a top class of objects, its subclass of fragile objects, and a method *drop*. *drop* may be defined for objects belonging to the top class such that if an object is dropped, then it will be on the floor but, otherwise, remains as it was. This works perfectly well for objects representing silver bars. Chinese vases, however, have the additional property that they are fragile and, hence, are broken after they have been dropped. In an object-oriented framework, this is typically modelled by defining a (refined) method *drop* for the subclass of fragile objects which – as it is more specific – overrides the method *drop* inherited from the top class. Unfortunately, both approaches, [42] as well as [1], do not provide a mechanism to suppress the application of the less specific method and, hence, nothing can be concluded about the state of the vase after being dropped.

In this paper, we extend the equational logic programming approach of [25,20] such that the concept of overriding can be integrated. After a brief description of the basic notions and notations concerning unification, logic programs, and multisets in section 2, we formalize the notions of objects, hierarchies of classes, and inheriting as well as overriding methods in section 3. Furthermore, we define the concept of specificity and argue that the application of – possibly inherited or overridden – methods can be adequately modelled by the equational logic programming approach [25,20] augmented by specificity. We also illustrate the approach by examples motivated

by the Broken Item and the Yale Shooting domain. Equational logic programs for computing change and specificity as well as their completion in the sense of [9] and [29,48] are presented in section 4. Section 5 focuses on models which interpret terms representing situations as multisets. We also show how these models are related to the intended meaning of causality and specificity. Section 6 introduces SLDENF-resolution as SLDNF-resolution extended by a unification algorithm for equational theories. In section 6, the soundness result of [48] for SLDENF-resolution is recapitulated and a completeness result for SLDENF-resolution is established, which can be applied to the equational logic programs specified in section 4. In section 7, we show how our approach can be used to perform backward reasoning, i.e. drawing conclusions about the past. This kind of use of our logic program does not fall into the class of programs for which SLDENF-resolution is complete, which is the reason for our suggestion in section 8 to use constructive negation to obtain answers in the case of non-ground negative goals. Finally, section 9 summarizes and discusses the results and outlines possible future extensions.

2. Preliminaries

We briefly review the notions and notations concerning logic programming and unification under an equational theory (see e.g. [3,49]). We also give a formal introduction to the concept of multisets, which play an important role for our semantics of actions, change, and specificity.

Terms and substitutions

A *term alphabet* is a pair $(\mathcal{V}, \mathcal{F})$ of disjoint sets. The elements of $\mathcal{V}$ are *variables* and the elements of $\mathcal{F}$ are *function symbols*. Each function symbol has a unique *arity* which is a natural number or 0. A function symbol with arity 0 is also called a *constant*. A *term* is either a variable or an expression of the form $f(t_1,\ldots,t_n)$, where $f \in \mathcal{F}$ with arity $n \geq 0$ and $t_1,\ldots,t_n$ are terms. The set of variables occurring in a term t is denoted by $\mathcal{V}ar(t)$. A term which does not contain variables is called *ground*. Throughout this paper, capital letters such as Z, Y,... denote variables, lower case letters such as a, b,... constant, f, g,... function symbols, and s, t,... denote terms.

A *substitution* is a mapping from the set of variables into the set of terms, which is equal to the identity almost everywhere. Hence, a substitution σ can be represented as the finite set of pairs $\{X_1 \mapsto t_1,\ldots,X_n \mapsto t_n\}$, where $X_i \neq t_i$. Substitutions are denoted by lower case Greek letters such as σ, θ,... . The identity substitution is denoted by ε. The *application* of a substitution to a term is defined by $t\sigma = \sigma(t)$ if t is a variable and $t\sigma = f(t_1\sigma,\ldots,t_n\sigma)$ iff $t = f(t_1,\ldots,t_n)$. This definition is extended to other syntactical objects like atoms, sets, or multisets in the usual way. The *domain* of σ – written $\mathcal{D}om(\sigma)$ – is the set $\{X_i | X_i \in \mathcal{V} \text{ and } X_i\sigma \neq X_i\}$. The *codomain* of σ

– written $Cod(\sigma)$ – is the set $\{t_i | t_i = X_i\sigma$ and $X_i \in \mathcal{D}om(\sigma)\}$. The variables in the codomain of σ are denoted by $\mathcal{V}Cod(\sigma)$, i.e. $\mathcal{V}Cod(\sigma) = \bigcup_{t_i \in Cod(\sigma)} \mathcal{V}ar(t_i)$. The *composition* of two substitutions σ and θ is a substitution – written $\sigma\theta$ – which is defined by $X(\sigma\theta) = (X\sigma)\theta$ for all variables X.[2] The *restriction* of a substitution σ to a set $\mathcal{W}$ of variables is a substitution – written $\sigma|_{\mathcal{W}}$ – which is defined by $X(\sigma|_{\mathcal{W}}) = X\sigma$ if $X \in \mathcal{W}$ and $X\sigma|_{\mathcal{W}} = X$ otherwise. Using the binary equality predicate $=$, a substitution $\sigma = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ has a corresponding formula – written $\sigma_=$ – which is the conjunction $X_1 = t_1 \wedge \ldots \wedge X_n = t_n$, and $\sigma_=|_{\mathcal{W}}$ means $(\sigma|_{\mathcal{W}})_=$.

An *equational theory* E consists of a set of universally closed expressions of the form $s = t$. Two terms s and t are called *E-equivalent* – written $s =_E t$ – if they are equal w.r.t E. If this relation is decidable for any two terms s, t, then E is said to be *decidable*. Two substitutions σ_1 and σ_2 are called *E-equivalent* w.r.t. a set $\mathcal{W}$ of variables – written $\sigma_1 =_E \sigma_2|_{\mathcal{W}}$ – iff $\forall X \in \mathcal{W}.\ X\sigma_1 =_E X\sigma_2$. A substitution σ_1 is called *more general* than a substitution σ_2 w.r.t. a set $\mathcal{W}$ of variables – written $\sigma_1 \leq_E \sigma_2|_{\mathcal{W}}$ – iff there is a substitution θ such that $\sigma_1\theta =_E \sigma_2|_{\mathcal{W}}$. An equational theory E is *regular* if $\mathcal{V}ar(s) = \mathcal{V}ar(t)$ holds for any equation $s = t \in E$ [49]. If E is regular, then $s =_E t$ implies $\mathcal{V}ar(s) = \mathcal{V}ar(t)$ for any two terms s, t.

Logic programs

An *atom* is an expression of the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate symbol ($n \geq 0$) and $t_1, \ldots, t_n$ are terms. An atom is called *ground* if its arguments are ground terms. A *literal* is either an atom or a negated atom. A *normal equational logic program* (or *normal E-program*) is a pair (P, E), where E is an equational theory and P is a finite set of clauses of the form $A \leftarrow L_1, \ldots, L_n$, where A is an atom and $L_1, \ldots, L_n$ are literals ($n \geq 0$). A *normal goal* (or *normal query*) is a clause of the form $\leftarrow L_1, \ldots, L_n$, where $L_1, \ldots, L_n$ are again literals ($n \geq 0$), and in the case $n = 0$, the *empty goal* is also denoted by $\square$.

Unification

A substitution σ is called an *E-unifier* of two terms s and t iff $\mathcal{D}om(\sigma) \subseteq \mathcal{V}ar(s) \cup \mathcal{V}ar(t)$ and $s\sigma =_E t\sigma$.[3] An *E-unification problem* consists of two terms s and t and is the problem whether there exists an E-unifier of s and t. The set of all E-unifiers for two terms s and t is denoted by $U_E(s, t)$. A set $cU_E(s, t)$ is called a *complete* set of E-unifiers iff $cU_E(s, t) \subseteq U_E(s, t)$ (correctness) and $\forall \sigma_1 \in U_E(s, t).\ \exists \sigma_2 \in cU_E(s, t).\ \sigma_2 \leq_E \sigma_1|_{\mathcal{V}ar(t)}$ (completeness). A set $\mu U_E(s, t)$ is called a complete and *minimal* set of E-unifiers iff it is complete and $\forall \sigma_1, \sigma_2 \in \mu U_E(s, t).\ \sigma_1 \leq_E \sigma_2|_{\mathcal{V}ar(s) \cup \mathcal{V}ar(t)} \Rightarrow \sigma_1 = \sigma_2$ (minimality).

[2] An equivalent definition is given by $\sigma\theta = \{X \mapsto t \in \theta | X \notin \mathcal{D}om(\sigma)\} \cup \{X \mapsto t\theta | X \mapsto t \in \sigma$ and $X \neq t\theta\}$.

[3] One should observe that the domain of an E-unifier is restricted to the variables occurring in s and t. This restriction is needed in definition 4.

The unification problem w.r.t. an equational theory E is called *finitary* if for all terms s and t a set $\mu U_E(s, t)$ exists which contains at most finitely many elements. It is called *infinitary* if for all terms s and t a set $\mu U_E(s, t)$ exists and there are two terms s and t such that there is no finite $\mu U_E(s, t)$.

An *E-unification procedure* is a procedure which takes two terms s and t as input and generates a subset of $U_E(s, t)$. An E-unification procedure is *complete* iff it generates a complete set of unifiers. A complete E-unification procedure is *minimal* iff it generates a minimal set of unifiers whenever this set exists. An equational theory E is said to be *decidable* iff the E-unification problem is decidable for each pair of terms.

The notions of E-unifier, set of E-unifiers, etc. are extended to atoms in the obvious way.

A *matching problem* consists of two terms s and t and is the problem whether there exists a substitution σ such that $s = t\sigma$. The notions of a *matcher*, *matching algorithm*, etc., can be defined in analogy to the notions of a unifier, unification algorithm, etc.

Multisets

A *multiset* is a collection of elements where, in contrast to classical sets, elements may occur more than once. Multisets are depicted using the brackets $\{\!\{ \cdots \}\!\}$. For instance, in $\mathcal{M} = \{\!\{a, b, a\}\!\}$, the element a occurs twice (written $a \in_2 \mathcal{M}$), whereas the element b occurs once (written $b \in_1 \mathcal{M}$), and the element c does not occur (written $c \in_0 \mathcal{M}$ or, equivalently, $c \notin \mathcal{M}$). The *union* $\mathcal{M} \cup \mathcal{N}$, *difference* $\mathcal{M} \dot{-} \mathcal{N}$, and *subset* $\mathcal{M} \dot{\subseteq} \mathcal{N}$ on multisets $\mathcal{M}$ and $\mathcal{N}$ are defined as follows:

$$e \in_m \mathcal{M} \wedge e \in_n \mathcal{N} \Rightarrow e \in_k \mathcal{M} \cup \mathcal{N} \wedge k = m + n.$$

$$e \in_m \mathcal{M} \wedge e \in_n \mathcal{N} \Rightarrow e \in_k \mathcal{M} \dot{-} \mathcal{N} \wedge k = \begin{cases} m - n, & \text{if } m > n, \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathcal{M} \dot{\subseteq} \mathcal{N} \Leftrightarrow \forall e[e \in_m \mathcal{M} \wedge e \in_n \mathcal{N} \Rightarrow n \geq m)].$$

Futhermore, $\mathcal{M} \doteq \mathcal{N}$ iff $\mathcal{M} \dot{\subseteq} \mathcal{N}$ and $\mathcal{M} \dot{\supseteq} \mathcal{N}$, whereas $\mathcal{M} \dot{\subset} \mathcal{N}$ iff both $\mathcal{M} \dot{\subseteq} \mathcal{N}$ and $\mathcal{M} \neq \mathcal{N}$. Throughout this paper, we use calligraphic letters such as $\mathcal{M}$, $\mathcal{N}$,... to denote multisets.

3. Objects and specificity

In this section, we approach the main notions of this paper, viz. objects and specificity. Let f be an n-ary function symbol and t_i, $1 \leq i \leq n$, be terms. In the terminology of frame-based or production systems like OPS5 [13], f denotes a slot and the t_i denote fillers. For example, when modelling a movable object we may wish

to specify its location with the help of a unary function symbol *at* such that $at(3)$ denotes that the object is at location 3. When modelling a Chinese vase, we may wish to specify that it is fragile by the nullary function symbol *fragile*.

DEFINITION 1

An *object* is a multiset of facts of the form $f(t_1,\ldots,t_n)$, where the terms t_i, $1 > i \leq n$, are ground. A *class* is a multiset of facts of the form $f(t_1,\ldots,t_n)$. An object $\mathcal{O}$ *belongs to* a class C, in symbols $\mathcal{O} \in C$, iff $\exists\sigma.\ C\sigma \doteq \mathcal{O}$. A *method* named by the constant a for the class C consists of a multiset of facts $\mathcal{E}$ with $\mathcal{V}ar(\mathcal{E}) \subseteq \mathcal{V}ar(C)$; it is written $\langle C, a, \mathcal{E}\rangle$. The *application* of the method $\langle C, a, \mathcal{E}\rangle$ to an object $\mathcal{O} \in C$ yields $\mathcal{E}\sigma$, where σ is a substitution such that $C\sigma \doteq \mathcal{O}$.

For example, a green item at location 3 may be denoted by $\{\!|at(3), color(green)|\!\}$ [4], and a fragile and broken item is denoted by $\{\!|fragile, broken|\!\}$. Let $\{\!|at(X)|\!\}$ denote the class of movable items and $\{\!|at(X), color(Y)|\!\}$ denote the class of colored movable items, then the object $\{\!|at(3), color(green)|\!\}$ belongs to the class $\{\!|at(X), color(Y)|\!\}$. For the class $\{\!|at(X)|\!\}$ of movable objects, a method *move* may be defined by the multiset $\{\!|at(X+1)|\!\}$. Hence, the application of the method $\langle\{\!|at(X)|\!\}, move, \{\!|at(X+1)|\!\}\rangle$ to the object $\{\!|at(3)|\!\}$ yields $\{\!|at(4)|\!\}$. Observe that there might be several instances of one and the same method regarding a particular object since the matching problem $C\sigma \doteq \mathcal{O}$ might admit more than one solution.

DEFINITION 2

A class C_1 is a *subclass* of class C_2 iff $\exists\sigma.\ C_1 \dot{\supset} C_2\sigma$. If C_1 is a subclass of C_2, then C_1 *inherits* each method for C_2. The *application* of an inherited method $\langle C_2, a, \mathcal{E}\rangle$ to an object $\mathcal{O} \in C_1$ yields $(\mathcal{O} \dot{-} C_2\sigma) \dot{\cup} \mathcal{E}\sigma$, where C_1 is a subclass of C_2 and σ is a substitution such that $\mathcal{O} \dot{\supseteq} C_2\sigma$.

For example, the class $\{\!|at(X), color(Y)|\!\}$ of colored movable items is a subclass of the class $\{\!|at(X)|\!\}$ of movable items. Thus, the method $\langle\{\!|at(X)|\!\}, move, \{\!|at(X+1)|\!\}\rangle$ is inherited by the former. The application of this inherited method to the object $\{\!|at(3), color(green)|\!\}$ yields the object $\{\!|at(4), color(green)|\!\}$. One should observe that the application of a method as defined in definition 1 is subsumed by the definition of how to apply an inherited method. Henceforth, whenever talking about the application of a method we implicitly refer to the latter definition.

Figure 1 shows the class hierarchy for the Broken Item domain, where *fragile* denotes that an item is fragile, *broken* denotes that an item is broken, and *intact* denotes that an item is intact. Intuitively, intact is the opposite of broken and, thus,

[4] Throughout this paper, we distinguish between items and objects. Items are entities in the real world which are represented by objects.

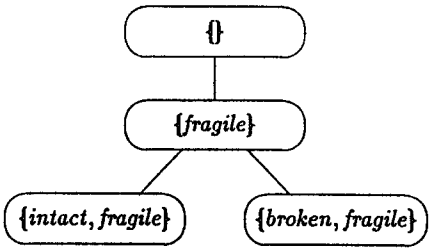

Fig. 1. A hierarchy of classes representing items $\{\!\{\ \}\!\}$, fragile items $\{\!\{fragile\}\!\}$, intact items $\{\!\{intact, fragile\}\!\}$, and broken items $\{\!\{broken, fragile\}\!\}$.

an item cannot be broken and intact at the same time. Since we do not want to allow explicit negation in classes and objects,[5] we have to explicitly specify inconsistencies for the domain of discourse. For example, in the Broken Item domain and object $\mathcal{O}$ with $\{\!\{intact, broken\}\!\} \dot{\subseteq} \mathcal{O}$ does not represent an item in the real world and, hence, is called *inconsistent*. Furthermore, since 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 item is broken into a few pieces, whereas several occurrences of *broken* may be interpreted as the fact that the item is shattered into many pieces. Thus, fuzziness may be expressed. On the other hand, if we model quarters, nickels, and dimes, then it is quite adequate to represent the contents of a purse as a multiset. However, for the sake of simplicity we assume for the Broken Item domain that no fact should occur more than once in a consistent object. Altogether, an object $\mathcal{O}$ is said to be inconsistent iff

$$\{\!\{broken, intact\}\!\} \dot{\subseteq} \mathcal{O} \vee \exists X \in \{fragile, broken, intact\}.\ \{\!\{X, X\}\!\} \dot{\subseteq} \mathcal{O}. \qquad (1)$$

Having specified inconsistency and defining that an object is *consistent* iff it is not *inconsistent*, we can now concentrate on specificity and overriding.

Following definition 2, methods are automatically inherited. However, one often might wish to refine the definition of a method for a subclass and, then, the automatic inheritance should be suppressed. In other words, a definition of a method shall *override* the definition which belongs to some superclass. To this end, we formally introduce the notion of *specificity*.

DEFINITION 3

A method $\alpha_1 = \langle C_1, a_1, \mathcal{E}_1 \rangle$ is *more specific* than a method $\alpha_2 = \langle C_2, a_2, \mathcal{E}_2 \rangle$ (written $\alpha_1 < \alpha_2$) iff $a_1 = a_2$ and C_1 is a subclass of C_2. A method $\alpha = \langle C, a, \mathcal{E} \rangle$ is

[5] 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.

applicable to an object $\mathcal{O}$ iff $\mathcal{O}$ belongs to C or a subclass of C and there is no more specific method applicable to $\mathcal{O}$.

In other words, more specific methods *override* less specific inherited ones. As an example, consider the method[6]

$$\langle \{\!\{\,\}\!\}, drop, \{\!\{\,\}\!\} \rangle, \tag{2}$$

which is defined for the topmost class $\{\!\{\,\}\!\}$ of any class hierarchy and, hence, is inherited by all other classes. In particular, it is inherited by the class of fragile objects and, if applied to the object $\{\!\{fragile\}\!\}$, yields again the object $\{\!\{fragile\}\!\}$. For the class of fragile objects, however, the more specific method

$$\langle \{\!\{fragile\}\!\}, drop, \{\!\{fragile, broken\}\!\} \rangle, \tag{3}$$

may have been defined. This method overrides the preceding one and is applicable to the object $\{\!\{fragile\}\!\}$. In the sequel, we call a method *most specific* whenever it is applicable in the sense of definition 3.

It is noteworthy that our definition allows more than one applicable description of one and the same method – a phenomenon which is usually called *multiple inheritance*. For instance, both $\langle \{\!\{fragile\}\!\}, drop, \{\!\{broken\}\!\} \rangle$ and $\langle \{\!\{broken\}\!\}, drop, \{\!\{broken\}\!\} \rangle$ are applicable in $\{\!\{fragile, broken\}\!\}$. If applied, only the latter method description leads to a consistent situation (cf. (1)) and, thus, the former should not be considered. Ideally, if we do not consider indeterminism or uncertainty, i.e. the existence of alternative and coincidental effects, there should be a unique applicable description for a method to each consistent object leading to a consistent object. Such an ideal set of object descriptions can be obtained in the Broken Item domain (see also [52]): We have already considered the methods (2) and (3). These two methods, however, do not yet completely specify the Broken Item domain. For instance, (3) is inherited by the class of intact objects. Its application would lead to the inconsistency $\{\!\{intact, fragile, broken\}\!\}$, where the represented item is intact and broken at the same time. To avoid such a behavior, we define the additional method

$$\langle \{\!\{intact, fragile\}\!\}, drop, \{\!\{fragile, broken\}\!\} \rangle. \tag{4}$$

As (4) < (3), it will be preferred whenever the dropped item is known to be intact. Similarly, if the dropped item is already broken, i.e. if the object is $\{\!\{broken, fragile\}\!\}$, then the execution of (3) would lead to the inconsistency $\{\!\{broken, broken, fragile\}\!\}$. This can be avoided by defining a more specific method for the class of broken items:

[6] Later in this section, we argue that the application of methods to objects can be identified with reasoning about dynamically changing worlds. This is why we do not distinguish between a method *drop* which shall be applied to some object, and an action *drop* which shall be executed in some state of the world by a robot, say.

$$\langle \{\!\{broken, fragile\}\!\}, drop, \{\!\{fragile, broken\}\!\} \rangle. \quad (5)$$

Altogether, we obtain a set of methods, viz. {(2), (3), (4), (5)}, such that whenever an applicable and most specific method is applied to a consistent object, then the resulting object is also consistent. A formal proof for this set of actions being consistency preserving is analogous to the respective proofs for the Blocksworld domain presented in [25] and can be extracted from a general result presented in [52].

So far, we have formally defined objects and methods. However, reasoning about change usually deals with situations and actions (e.g. [39]). It would be an interesting paper in itself to discuss the similarities and differences of situation-based planning and object-oriented programming. As far as this paper is concerned, the difference can be neglected. We will not distinguish between a green item at location 3 and the situation where there is a green item at location 3. Similarly, we will not distinguish between a method which is applied to an object and an action which is executed in a certain situation. Rather, we will use the notions object and situation as well as methods and action interchangeably.

As a second, more expressive example, we consider the famous Yake Shooting domain (cf. [21,4]). The Yale Shooting environment consists of a gun which is unloaded or loaded, a turkey which is alive or dead, and three actions, viz. loading the gun (*load*), shooting (*shoot*), and waiting (*wait*). Figure 2 depicts the hierarchical

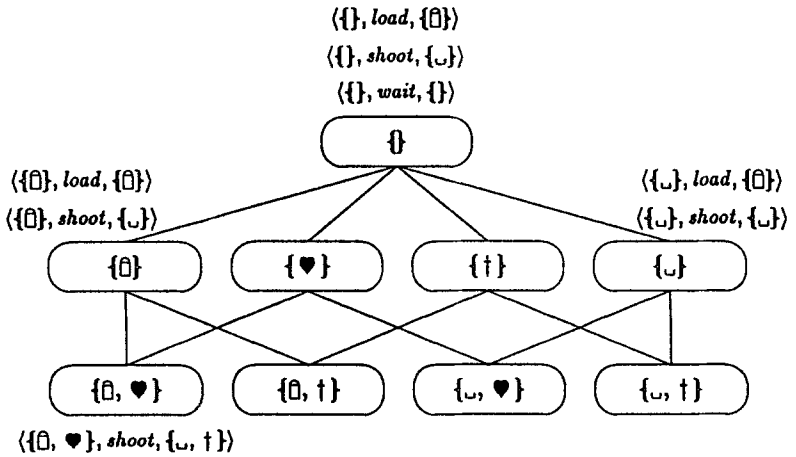

Fig. 2. The hierachy of classes for the consistent situation description in the Yale Shooting scenario together with all action descriptions of the three actions *load*, *shoot*, and *wait*, where ⍓, ␣, ♥, and † represent the four symbols *loaded*, *unloaded*, *alive*, and *dead*, respectively.

structure of all consistent situations in this domain (provided again that no fact is allowed to occur more than once). The topmost class represents the case where nothing is known about the gun or the turkey. A class C_1 is a subclass of a class C_2

iff it contains more information than C_2. Moreover, in fig. 2, some of the classes are labelled with (different) action descriptions which are used to prevent inconsistencies as in the Broken Item domain. According to definition 3, the applicable action description of a certain action is given via the following criterion: The action description is either defined for this class or it is inherited from the nearest superclass which is labelled with an action description. Note that this defines a single description for each action and each situation, and that this corresponds exactly to the specificity criterion as described above. The reader is invited to verify that under this assumption the application of an action to a consistent situation always yields the expected result. For instance, $\{\!|\mathit{unloaded}, \mathit{alive}|\!\}$ inherits the action description $\langle \{\!|\mathit{unloaded}|\!\}, \mathit{load}, \{\!|\mathit{loaded}|\!\}\rangle$, i.e. loading the gun transforms this situation into $\{\!|\mathit{loaded}, \mathit{alive}|\!\}$.

The Yale Shooting 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 ($\{\!|\mathit{unloaded}|\!\}$), loaded with one ($\{\!|\mathit{loaded}|\!\}$) or with two bullets ($\{\!|\mathit{loaded}, \mathit{loaded}|\!\}$). Modelling such a gun requires to add several classes in the hierarchy depicted in fig. 2 and to modify the consistency criteria, which are analogous to (1), such that a situation S is inconsistent if *loaded* occurs more than twice in S. It also requires to modify the action descriptions of *load* and *shoot* in the obvious way, e.g. $\langle \{\!|\mathit{loaded}|\!\}, \mathit{load}, \{\!|\mathit{loaded}|\!\}\rangle$ should no longer occur in the set of action descriptions. Rather, the description $\langle \{\!|\ |\!\}, \mathit{load}, \{\!|\mathit{loaded}|\!\}\rangle$ should be inherited in the case of $\{\!|\mathit{loaded}|\!\}$. The complete hierarchy representing this extension is illustrated in fig. 3.

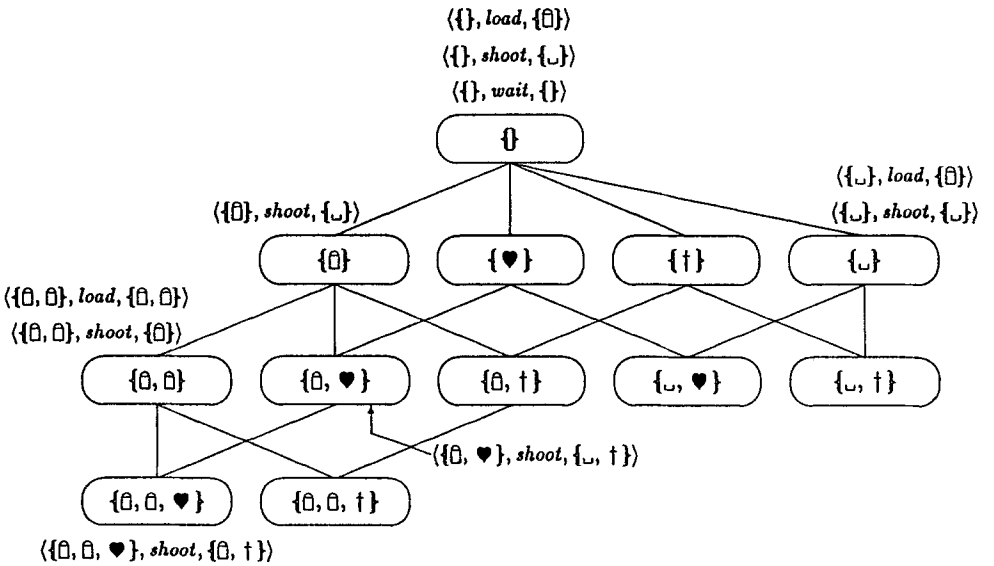

Fig. 3. The hierarchy of classes for the consistent situation descriptions in the extended Yale Shooting scenario together with all action descriptions of the three actions *load*, *shoot*, and *wait*.

4. Equational logic programs

In the preceding section, we have considered objects (or situations) and specificity, but we have said nothing about a logical formalism to represent situations, to compute specificity, and to determine that the application of a sequence of actions to a given initial situation yields a certain goal situation. This will be the subject of this section. We will present a completed logic program in the sense of [9] together with a unification complete equational theory in the sense of [29] which can be used to represent objects or situations and to define actions, change, and specificity.

Recall that a situation is a multiset of facts. Facts themselves are represented by terms. In order to represent multisets, a binary function symbol $\circ$ is introduced such that $\circ$ is associative, commutative, and admits a unit element denoted by the constant $\emptyset$. Hence, we need the following axioms of equality.[7)]

$$
\begin{aligned}
(X \circ Y) \circ Z &= X \circ (Y \circ Z) && \text{(associativity)} \\
X \circ Y &= Y \circ X && \text{(commutativity)} \\
X \circ \emptyset &= X && \text{(unit element)} \\
X &= X && \text{(reflexivity)} \qquad \text{(AC1)} \\
X = Y &\rightarrow Y = X && \text{(symmetry)} \\
X = Y \wedge Y = Z &\rightarrow X = Z && \text{(transitivity)} \\
X = Y &\rightarrow (W[X] \leftrightarrow W[Y]) && \text{(substitutivity)}
\end{aligned}
$$

Here, $W[X] \leftrightarrow W[Y]$ denotes the equivalence between a first-order formula W which contains an occurrence of the variable X and the formula which is obtained from W by replacing this occurrence with Y.

The relation between these axioms and the concept of a multiset is as follows. Let s_i, $1 \leq i \leq n$, be *elementary terms*, i.e. terms that are not built up from the special symbols $\circ$ and $\emptyset$. We can define two mappings $\cdot^{\mathcal{I}}$ and $\cdot^{\mathcal{I}^{-1}}$ such that the following equations hold:

$$
\begin{aligned}
\emptyset^{\mathcal{I}} &= \{\!\{\,\}\!\} \\
(s_1 \circ \ldots \circ s_n)^{\mathcal{I}} &= \{\!\{s_1\}\!\} \dot{\cup} (s_2 \circ \ldots \circ s_n)^{\mathcal{I}} \\
\{\!\{\,\}\!\}^{\mathcal{I}^{-1}} &= \emptyset \\
\{\!\{s_1, \ldots, s_n\}\!\}^{\mathcal{I}^{-1}} &= s_1 \circ \{\!\{s_2, \ldots, s_n\}\!\}^{\mathcal{I}^{-1}}.
\end{aligned}
$$

One should observe that $(s^{\mathcal{I}})^{\mathcal{I}^{-1}} =_{\mathrm{AC1}} s$ and $(S^{\mathcal{I}^{-1}})^{\mathcal{I}} \doteq S$, where s is a term and S is a multiset. Moreover, $s^{\mathcal{I}} \doteq S$ iff $s =_{\mathrm{AC1}} S^{\mathcal{I}^{-1}}$. Henceforth, we will usually not distinguish

[7)] All variables in the following formulas are implicitly universally quantified.

between terms built up from the function $\circ$ and the corresponding multisets of elementary terms. As a consequence, we are allowed to identify, for instance, the expressions $C \circ \mathcal{V}$ and $C \dot{\cup} \mathcal{V}$.

Based on this representation, we can easily extend the equational logic approach of [25,20] to capture specificity. Since each action or method $\langle C, a, \mathcal{E} \rangle$ is defined by its condition C, its name a, and its effect $\mathcal{E}$, we can specify these actions by a unit clause based on a ternary predicate as follows:

$$action(C, a, \mathcal{E}). \tag{6}$$

A predicate $causes(S, [a_1,\ldots,a_n], S')$ can be used to express that the sequence $[a_1,\ldots,a_n]$ of actions causes a situation S to become situation S'. This is expressed via the following two clauses. Note that the predicate = below denotes equality modulo our equational theory AC1.[8)]

$$\begin{aligned} causes(I, [\,], G) &\leftarrow I = G. \\ causes(I, [A|P], G) &\leftarrow action(C, A, E), \\ &\quad C \circ V = I, \\ &\quad \neg\, non\_specific(A, C, V), \\ &\quad causes(V \circ E, P, G). \end{aligned} \tag{7}$$

The intended interpretation of the first clause of (7) is that the empty sequence does not change anything. The second clause of (7) should be interpreted as follows. An action named A followed by a sequence of actions P transforms the situation I into the situation G if there is an action definition $\langle C, A, E \rangle$ with name A along with the condition C and effect E such that the condition C is contained in $I = C \circ V$, this particular action is the most specific one applicable to $C \circ V$, and the application of the sequence P to the new situation $V \circ E$ – obtained from I by deleting C and adding E – yields G. We intend to determine whether the chosen action definition with name A is most specific in situation $C \circ V$ via negation-as-failure, where C denotes the condition of the particular action definition. Informally, according to definition 3 $non\_specific(A, C, V)$ shall be true if there is another action definition with condition C', name A, and effect E such that this definition is also applicable in $C \circ V$ and C' is a super-multiset of C.

$$\begin{aligned} non\_specific(A, C, V) \leftarrow\ & action(C', A, E'), \\ & C' \circ V' = C \circ V, \\ & C \circ W = C', \\ & W \neq \emptyset. \end{aligned} \tag{8}$$

[8)] As usual, [ ] and $[H|T]$ denote the empty list and a list with head H and tail T, respectively.

One should note that we do not employ additional axioms for solving the technical frame problem since each elementary term which is not affected by an action is automatically contained in the resulting situation $V \circ E$ after having applied the second clause of (7).

Finally, since in our program equations occur in the bodies of some clauses, we have to add the axiom of reflexivity

$$X = X, \tag{9}$$

(see [22]). Altogether, for each set of action descriptions A we obtain the normal E-program $(P_A, \mathrm{AC1})$ consisting of the clauses (7)–(9) along with all clauses of the form (6).[9]

To deal with negation in the body of clauses, we will consider completed equational logic programs. In such a completion, we have to be able to prove inequalities like $W \neq \emptyset$ occurring in (8). Unfortunately, the equational axioms (6) are insufficient for such a task. In the case of non-equational logic programs, Clark used some axiom schemata which allow for proving inequality of two terms whenever these are not unifiable [9]. In the case of equational theories, the original method has to be generalized by the concept of *unification completeness*. This concept was firstly used in [29] and improved in [48].

DEFINITION 4

Let E be an equational theory. A consistent set of formulas E^* is called *unification complete* w.r.t. 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 $\overline{X} = \mathcal{V}ar(s) \cup \mathcal{V}ar(t)$ 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 unifiers $cU_E(s, t)$

$$E^* \models \forall \overline{X}.(s = t \rightarrow \bigvee_{\theta \in cU_E(s,t)} \exists \overline{Y_\theta}.\, \theta_=), \tag{10}$$

where $\overline{Y_\theta}$ denotes the variables which occur in $\theta_=$ but not in $\overline{X}$.

As pointed out in [48], the use of $\overline{Y_\theta}$, which was missing in [29], is necessary due to the fact that E-unifiers might introduce new variables. Note that in the case of infinitary equational theories, the disjunct in (10) may contain infinitely many elements.

As we intend to compute with an equational theory E rather than with its completion E^*, it suffices to know that there is a unification complete theory for E.

[9] Note that, for instance, (8) could be contracted to $non\_specific(A, C, W \circ V') \leftarrow action(C \circ W, A, E')$, $W \neq \emptyset$. However, this would disguise the intended meaning of this clause, which is why we kept the more redundant formulations.

The following proposition 2 guarantees this property for the AC1 theory considered in this paper, since this theory is decidable and a complete AC1 unification algorithm is known (see e.g. [3]). The proof of this proposition is based on the following observation which relates the notion of subsuming substitutions to a logical relation between their corresponding equational formulas. For instance, $\{X \mapsto a \circ Z\} \leq_{\mathrm{AC1}} \{X \mapsto a\}|_{\{X\}}$ and, hence, $(6) \models \forall X\ (X = a \rightarrow \exists Z.\ X = a \circ Z)$.[10)]

LEMMA 1

Let E be an equational theory, σ and θ two substitutions, and $\mathcal{W} \supseteq \mathcal{D}om(\sigma) \cup \mathcal{D}om(\theta)$ a set of variables. If $\sigma \leq_E \theta|_{\mathcal{W}}$, then

$$E \models \forall \mathcal{W}(\exists \overline{Y_\theta}.\, \theta_= \rightarrow \exists \overline{Y_\sigma}.\, \sigma_=),$$

where $\overline{Y_\sigma}$ and $\overline{Y_\theta}$ denote sequences of variables which occur in $\sigma_=$ and $\theta_=$ but not in $\mathcal{W}$, respectively.

Proof

$\sigma \leq_E \theta|_{\mathcal{W}}$ implies that we can find a substitution τ such that $X\theta =_E X\sigma\tau$ for each $X \in \mathcal{W}$. Let $X \in \mathcal{W}$ and, then, $t = X\theta$ and $s = X\sigma$, which implies $t =_E s\tau$; hence,

$$E \models \forall X(\exists \overline{Y_\theta}.\, X = t \rightarrow \exists \overline{Y_\sigma}.\, \exists \overline{Y_\tau}.\, X = s\tau), \tag{11}$$

where $\overline{Y_\tau}$ denotes a sequence of variables which occur in τ but neither in W nor in $\overline{Y_\sigma}$. From (11), it follows that $E \models \forall X(\exists \overline{Y_\theta}.\, X = t \rightarrow \exists \overline{Y_\sigma}.\, X = s)$. This holds for any $X \in \mathcal{W}$, which proves the claim. □

This lemma enables us to show that decidable equational theories admit a complete theory.

PROPOSITION 2

If E is a decidable equational theory and $\mathcal{P}$ is complete E-unification procedure, then there is a unification complete theory w.r.t. E.

Proof

E^* can be constructed from the set of axioms in E and the standard equality axioms as follows. For each pair of terms s and t, do the following, where $\overline{X}$ and $\overline{Y_\theta}$ are as in definition 4: If s and t are not E-unifiable, then add $\neg \exists \overline{X}(s = t)$ to E^*; otherwise, use $\mathcal{P}$ to compute a complete set $cU_E(s, t)$ of E-unifiers of s and t and add $\forall \overline{X}(s = t \rightarrow \bigvee_{\theta \in cU_E(s,t)} \exists \overline{Y_\theta}.\, \theta_=)$ to E^*. E^* is consistent since it admits a model whose

[10)] This holds since Z can be replaced by the constant $\emptyset$.

universe consists of the congruence classes w.r.t. E. Furthermore, lemma 1 guarantees that condition (2) of definition 4 does not only hold for the particular complete set of unifiers being computed, but also for arbitrary complete sets $\widetilde{cU_E}(s,t)$: For each $\theta \in cU_E(s,t)$ there is a more general $\sigma \in \widetilde{cU_E}(s,t)$ due to the completness of $\widetilde{cU_E}(s,t)$. Hence, lemma 1 implies

$$E^* \models \forall \overline{X} \left(\bigvee_{\theta \in cU_E(s,t)} \exists \overline{Y_\theta}.\, \theta_= \rightarrow \bigvee_{\sigma \in \widetilde{cU_E}(s,t)} \exists \overline{Y_\sigma}.\, \sigma_= \right),$$

where $\overline{Y_\sigma}$ denotes the variables occurring in σ but not in $\overline{X}$. The claim follows immediately from the transitivity of the implication. □

Having defined the notion of a unification complete equational theory for AC1, we can now turn to the definition of a completed logic program for modelling actions and specificity. One should note that all equations in this completed logic program below are meant with respect to the unification complete theory AC1*. Let $\alpha_i = \langle C_i, a_i.\, \mathcal{E}_i \rangle$, $1 \leq i \leq n$, be the actions considered in a given scenario. As usual (see [9]), the completion of a set of clauses of the form (6) is the formula

$$\forall C, A, E[action(C, A, E) \leftrightarrow \bigvee_{i=1}^{n} (C = C_i \wedge A = a_i \wedge E = \mathcal{E}_i)]. \tag{12}$$

Analogously, the completed definition of *causes* is obtained from (7):

$$\begin{aligned} \forall I, P, G[causes(I, P, G) \leftrightarrow\ & (P = [\,] \wedge I = G) \\ & \vee \\ & \exists A, C, E, P', V(P = [A|P'] \wedge \\ & \qquad action(C, A, E) \wedge \\ & \qquad C \circ V = I \wedge \\ & \qquad \neg non\_specific(A, C, V) \wedge \\ & \qquad causes(V \circ E, P', G))]. \end{aligned} \tag{13}$$

Similarly, the completed definition of *non_specific* is obtained from (8):

$$\begin{aligned} \forall A, C, V[non\_specific(A, C, V) \leftrightarrow\ & \exists C', E', V', W(action(C', A, E') \wedge \\ & \qquad C' \circ V' = C \circ V \wedge \\ & \qquad C \circ W = C' \wedge \\ & \qquad W \neq \emptyset)]. \end{aligned} \tag{14}$$

Finally, let C be the conjunction of all clauses 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 $\{=, causes, non\_specific, action\}$.

Let $P^* = (12) \wedge (13) \wedge (14) \wedge C$. Then, $(P^*, \mathrm{AC1}^*)$ is the completed equational logic program specifying actions and specificity. One should observe that various scenarios like the Broken Item or the Yale Shooting domain differ only in the definition of the predicate *action*. In what follows, we use $(P_A^*, \mathrm{AC1}^*)$ to denote this completed equational logic program encoding a particular set A of action definitions of the form $\langle C, a, \mathcal{E} \rangle$.

5. Models

Completed equational logic programs of the form $(P^*, AC1^*)$ differ from the equational programs considered in [29] since negative literals occur in clause bodies. They also differ from programs considered in e.g. [2], since they contain the equational theory AC1. Hence, we cannot simply apply the model theoretic semantics of [29] or [2]. As usual, we consider only Herbrand interpretations, i.e. interpretations of the form $(\mathcal{U}', \cdot^{\mathcal{I}'})$, where $\mathcal{U}'$ is the set of ground terms. In this section, we show that we can restrict $\cdot^{\mathcal{I}}$ such that AC1 terms are interpretated as multiset expressions. Furthermore, we show that $(P_A^*, \mathrm{AC1}^*)$ is consistent and models actions, change, and specificity as intended. We use the notation $(P^*, E^*) \models F$ to denote that the formula F is a logical consequence of the set of formulas $P^* \cup E^*$.

THEOREM 3

Let A be a set of action descriptions; then $(P_A^*, \mathrm{AC1}^*)$ is consistent.

Proof

A model of $(P_A^*, \mathrm{AC1}^*)$ can be constructed as follows. From proposition 2 and from the fact that AC1-unification is decidable, we conclude that $\mathrm{AC1}^*$ is consistent. To be more precise, let

$$M_{\mathrm{AC1}^*} = \{s = t \mid \mathrm{AC1}^* \models s = t \wedge \{s, t\} \subseteq \mathcal{U}'\}.$$

where $\mathcal{U}'$ is the set of all ground terms, then M_{AC1^*} is a model of $\mathrm{AC1}^*$. Let

$$M_{action} = M_{\mathrm{AC1}^*} \cup \{action(c, a, e) \in A \mid \exists \sigma, \langle C, a, \mathcal{E} \rangle \in A. \{c = C^{\mathcal{I}^{-1}}\sigma, a = a, e = \mathcal{E}^{\mathcal{I}^{-1}}\sigma\} \subseteq M_{\mathrm{AC1}^*}\},$$

then M_{action} is a model of $\{(12)\} \cup \mathrm{AC1}^*$. Let

$$M_{non\_specific} = M_{action} \cup \{non\_specific(a, c, v) \mid \exists c', e' v', w. \{action(c', a, e'), c' \circ v' = c \circ v, c \circ w = c'\} \subseteq M_{action} \wedge w = \emptyset \notin M_{action}\}$$

then $M_{non\_specific}$ is a model of $\{(14), (12)\} \cup \mathrm{AC1}^*$. Let

$$M_0 = M_{non\_specific} \cup \{causes(i, p, g) \mid \{p = [\,], i = g\} \subseteq M_{non\_specific}\},$$

and for all $k > 0$ let

$$M_k = M_{k-1} \cup \{causes(i, p, g) \mid \exists a, c, e, p', v.\ \{p = [a \mid p'], action(c, a, e), c \circ v = i, causes(v \circ e, p', g)\} \subseteq M_{k-1} \wedge non\_specific(a, c, v) \notin M_{k-1}\}.$$

Then, by a straightforward induction on the length of the second argument of *causes*, we learn that $M = \bigcup_{k \in \mathbb{N}_0} M_k$ is a model of of (13) and, consequently, M is a model for $P_A^* \cup \mathrm{AC1}^*$. □

It should be possible to develop a model and fixpoint theory for $(P^*, \mathrm{AC1}^*)$ in the spirit of [29] and [2] and, in particular, to show that $(P^*, \mathrm{AC1}^*)$ admits a least model. However, this is beyond the scope of this paper. One should observe that if we assign the level 0 to the predicate symbols = and *action*, 1 to *non_specific*, and 2 to *causes*, then the program $(P_A, \mathrm{AC1})$ is stratified (see e.g. [34]) since the level of the predicate symbol of every positive literal occurring in the body of a clause is less than or equal to the level of the predicate symbol of the head of the clause, and the level of the predicate symbol of every negative literal occurring in the body of a clause is less than the level of the predicate symbol of the head of the clause. But $(P_A, \mathrm{AC1})$ is not hierarchical (see e.g. [34]) since *causes* is recursive.

To prove that the completed logic program $(P^*, \mathrm{AC1}^*)$ models actions, change, and specificity as intended, we start with a formal justification of the already mentioned correspondence between the unification complete theory $\mathrm{AC1}^*$ and multiset equality.

PROPOSITION 4

Let $(\mathcal{U}', \cdot^{\mathcal{I}'})$ be a model of $(P^*, \mathrm{AC1}^*)$ and let $(\mathcal{U}, \cdot^{\mathcal{I}})$ be an interpretation such that

(1) $\mathcal{U} = \{s \mid s \text{ is the ground instance of an elementary term}\} \cup \{\{\!\{\,\}\!\}\} \cup \{\{\!\{t_1, \ldots, t_n\}\!\} \mid t_1 \circ \ldots \circ t_n \in \mathcal{U}'\}$.

(2) $\cdot^{\mathcal{I}}$ *is as* $\cdot^{\mathcal{I}'}$ *except that* $\emptyset^{\mathcal{I}} = \{\!\{\,\}\!\}$ *and* $(s \circ t)^{\mathcal{I}} = s^{\mathcal{I}} \dot{\cup} t^{\mathcal{I}}$.

Then, $(\mathcal{U}, \cdot^{\mathcal{I}})$ is also a model of $(P^*, \mathrm{AC1}^*)$ and for any two ground terms s and t, we find that

$$\mathrm{AC1}^* \models s = t \text{ iff } s^{\mathcal{I}} \doteq t^{\mathcal{I}}$$

and

$$\mathrm{AC1}^* \models s \neq t \text{ iff } s^{\mathcal{I}} \not\doteq t^{\mathcal{I}}$$

Proof

$(\mathcal{U}', \cdot^{\mathcal{I}})$ and $(\mathcal{U}, \cdot^{\mathcal{I}})$ differ only in the interpretation of the constant $\emptyset$ and the binary function symbol $\circ$ such that each element of the AC1-congruence class of an

AC1 term $t_1 \circ \ldots \circ t_n$ in $\mathcal{U}'$ is mapped to $\{\!\{t_1,\ldots,t_n\}\!\} \in \mathcal{U}$. Hence, we find for two AC1 terms s and t that $s^{\mathcal{I}'} = t^{\mathcal{I}'}$ iff $s^{\mathcal{I}} \doteq t^{\mathcal{I}}$. The proposition follows immediately. □

We now proceed by showing that specificity and causality are interpreted by the models of $(P^*, \text{AC1}^*)$ as intended. The following lemma is concerned with specificity. One should observe that the literal $\neg \mathit{non\_specific}(A, C, V)$ occurs in (14) in conjunction with the literal $\mathit{action}(C, A, E)$. Hence, the condition of the following lemma is always satisfied whenever it is applied in the context of (14).

LEMMA 5

If A is a set of action descriptions and $\langle C, a, \mathcal{E}\rangle \in A$, then

$$(P_A^*, \text{AC1}^*) \models \mathit{non\_specific}(a, C, \mathcal{V})$$

iff $\langle C, a, \mathcal{E}\rangle$ is not a most specific action description in A w.r.t. the situation $C \dot\cup \mathcal{V}$.

Proof

Following clause (14), the formula $\mathit{non\_specific}(a, c, v)$ is entailed by $(P_A^*, \text{AC1}^*)$ iff we can find terms C', $\mathcal{E}'$, $\mathcal{V}'$, $\mathcal{W}$ such that

$$\mathit{action}(C', a, \mathcal{E}') \wedge C' \circ \mathcal{V}' = C \circ \mathcal{V} \wedge C \circ \mathcal{W} = C' \wedge \mathcal{W} \neq \emptyset$$

is entailed. This conjunction is entailed iff all its literals are entailed. Following proposition 4, this is true iff $\langle C', a, \mathcal{E}'\rangle \in A$ (due to clause (12)), $C' \dot\cup \mathcal{V}' \doteq C \dot\cup \mathcal{V}$, $C' \dot\cup \mathcal{W} \doteq C'$, and $\mathcal{W} \not\doteq \{\!\}\!\}$. Hence, $(P_A^*, \text{AC1}^*) \models \mathit{non\_specific}(a, C, \mathcal{V})$ iff we can find a $\langle C', a, \mathcal{E}'\rangle \in A$ which is applicable in $C \dot\cup \mathcal{V}$ and for which $C' \dot\supset C$ holds, which is equivalent to $\langle C, a, \mathcal{E}\rangle$ not being a most specific description. □

We now consider the interpretation of the predicate *causes*.

PROPOSITION 6

If A is a set of action descriptions, then

$$(P_A^*, \text{AC1}^*) \models \mathit{causes}(\mathcal{I}, [a_1,\ldots,a_n], \mathcal{G})$$

iff there are multisets $S_0,\ldots,S_n$ such that $S_j \doteq (S_{j-1} \dot- C_j) \dot\cup \mathcal{E}_j$, where $\langle C_j, a_j, \mathcal{E}_j\rangle \in A$ is a most specific action description of a_j w.r.t. S_{j-1} $(1 \leq j \leq n)$, and $\mathcal{I} \doteq S_0$ and $\mathcal{G} \doteq S_n$ and $n \geq 0$.

Proof

The proof is by induction on the length n of the sequence of actions.

In the case $n = 0$, the formula $causes(\mathcal{I}, [\,], \mathcal{G})$ is entailed iff the disjunction on the right-hand side of (13) is entailed. The second part of this disjunction contains the subformula $[\,] = [A|P']$, which is false in any model of $(P_A^*, \mathrm{AC1}^*)$ due to the unification completeness of $\mathrm{AC1}^*$. Consequently, $causes(\mathcal{I}, [\,], \mathcal{G})$ is entailed iff $[\,] = [\,] \wedge \mathcal{I} = \mathcal{G}$ is entailed. Using proposition 4, this is true iff $\mathcal{I}$ and $\mathcal{G}$ denote that same multiset.

Turning to the induction step $n > 0$, we assume that the result holds for all instances of the *causes* predicate with an action sequence of length $< n$. As above, $causes(\mathcal{I}, [a_1,\ldots,a_n], \mathcal{G})$ is entailed iff the disjunction on the right-hand side of (13) is entailed. The first part of this disjunction contains the subformula $[a_1,\ldots,a_n] = [\,]$, which is false in any model of $(P_A^*, \mathrm{AC1}^*)$ due to the unification completeness of $\mathrm{AC1}^*$. Consequently, $causes(\mathcal{I}, [a_1,\ldots,a_n], \mathcal{G})$ is entailed iff we can find terms $a, C, \mathcal{E}, p', \mathcal{V}$ such that

$$[a_1,\ldots,a_n] = [a|p'] \wedge action(C, a, \mathcal{E}) \wedge C \circ \mathcal{V} = \mathcal{I} \wedge$$
$$\neg non\_specific(a, C, \mathcal{V}) \wedge causes(\mathcal{V} \circ \mathcal{E}, p', \mathcal{G})$$

is entailed. This conjunction is entailed iff all its literals are entailed. Following proposition 4, this is true iff $a = a_1$, $p' = [a_2,\ldots,a_n]$, $\langle C, a_1, \mathcal{E}\rangle \in A$ (due to clause (12)), $C \dot{\cup} \mathcal{V} \doteq \mathcal{I}$, $\langle C, a_1, \mathcal{E}\rangle$ is most specific w.r.t. $\mathcal{I}$ (due to the preceding lemma 5) and $causes(\mathcal{V} \circ \mathcal{E}, [a_2,\ldots,a_n], \mathcal{G})$ is entailed. Using the induction hypothesis, the latter is true iff there are multisets $S_1,\ldots S_n$ such that $S_j \doteq (S_{j-1} \dot{-} C_j) \dot{\cup} \mathcal{E}_j$, where $\langle C_j, a_j, \mathcal{E}_j\rangle \in A$ is the most specific action description of a_j w.r.t. S_{j-1} $(2 \leq j \leq n)$ and $\mathcal{V} \dot{\cup} \mathcal{E} \doteq S_1$ and $\mathcal{G} \doteq S_n$. The result follows immediately. □

The completeness of $(P^*, \mathrm{AC1}^*)$ ensures that the analogous proposition concerning negative statements about causality holds in the very same way.

PROPOSITION 7

If A is a set of action descriptions, then

$$(P_A^*, \mathrm{AC1}^*) \models \neg causes(\mathcal{I}, [a_1,\ldots,a_n], \mathcal{G})$$

iff there are no multisets $S_0,\ldots,S_n$ such that $S_j \doteq (S_{j-1} \dot{-} C_j) \dot{\cup} \mathcal{E}_j$, where $\langle C_j, a_j, \mathcal{E}_j\rangle \in A$ is a most specific action description w.r.t. S_{j-1} $(1 \leq j \leq n)$ and $\mathcal{I} \doteq S_0$ and $\mathcal{G} \doteq S_n$.

Proof

The proof follows from the completeness of $(P_A^*, \mathrm{AC1}^*)$ and is again straightforward by induction on n. □

The last two propositions ensure that *causes* behaves as intended. We can now turn to the question of how to compute answers to queries posed to equational logic programs of the form $(P, \mathrm{AC1})$, for which we will use SLDNF-resolution extended by a complete and minimal AC1-unification algorithm.

6. SLDENF-resolution

Proving with completed logic programs is known to be quite inefficient. We therefore do not want to compute with $(P_A^*, \mathrm{AC1}^*)$, rather we would like to compute with $(P_A, \mathrm{AC1})$, i.e. the if-halves of the definitions in P^* only and to use negation-as-failure for deriving negative information.

SLDENF-resolution is like SLDNF-resolution [9] but standard unification is replaced by an appropriate E-unification procedure. We follow the idea of [29] and adopt the definition used in [48]. The selection rule is constrained such that negative literals are only selected if they are ground. If the selected literal is positive, then the derivation step is done as for SLDE-resolution [29,15,23]. If the selected literal is negative and if the SLDENF-evaluation of the corresponding positive literal succeeds, then the derivation fails; otherwise, the derivation continues with the selected literals removed from the actual set of goal literals. The following two definitions are similar to the definitions given in [48] and extend the various definitions concerning SLDNF-resolution which can be found in [34].

DEFINITION 5

If (P, E) is a normal E-program and G a normal goal, then an *SLDENF-refutation of rank* 0 for $P \cup \{G\}$ consists of a sequence $G_0,\ldots,G_n$ of normal goals such that $G = G_0$ and $G_n = \square$ and for each $i = 1,\ldots,n$, the selected literal L_k of G_{i-1}, which is the goal $\leftarrow L_1,\ldots,L_k,\ldots,L_m$, is positive and there is a new variant $A \leftarrow B_1,\ldots,B_l$ of a program clause such that L_k E-unifies with A using the E-unifier θ and G_i is the goal $\leftarrow(L_1,\ldots,L_{k-1}, B_1,\ldots,B_l, L_{k+1},\ldots,L_m)\theta$. The number n is called the *length* of the refutation. The composition of the substitutions $\theta_1,\ldots,\theta_n$ restricted to the variables in the first goal, i.e. $(\theta_1 \ldots \theta_n)|_{\mathcal{V}ar(G)}$ is called *computed answer substitution.*[11)]

A *finitely failed SLDENF-tree of rank* 0 for $P \cup \{G\}$ via a selection function R is a finite tree such that

(1) Each node is labelled with a non-empty normal goal and the root is labelled with G.

(2) For each node, R selects a positive literal.

(3) For each leaf node $\leftarrow L_1,\ldots,L_m$, the selected literal L_k does not E-unify with the head of a new variant of any program clause.

(4) If $\leftarrow L_1,\ldots,L_k,\ldots,L_m$ is an inner node such that L_k is selected, then for each program clause let $A \leftarrow B_1,\ldots,B_l$ be a new variant and let $cU_E(L_k, A)$ be a complete set of E-unifiers of L_k and A. Then, for each $\theta \in cU_E(L_k, A)$, the node labelled with the goal $\leftarrow L_1,\ldots,L_{k-1}, B_1,\ldots,B_l, L_{k+1},\ldots,L_m)\theta$ is a child of this inner node.

[11)] $\mathcal{V}ar(\leftarrow L_1,\ldots,L_m) := \bigcup_{i=1\ldots m} \mathcal{V}ar(L_i)$.

As usual, the *depth* of an SLDENF-tree is defined as the length of the longest path from the root node to a leaf.

DEFINITION 6

An *SLDENF-refutation of rank* r ($r \geq 1$) for $P \cup \{G\}$ consists of a sequence $G_0,\ldots,G_n$ of normal goals such that $G = G_0$ and $G_n = \square$ and for each $i = 1,\ldots,n$

(1) If the selected literal L_k of G_{i-1} is positive, then there is a new variant of a program clause such that L_k E-unifies with the head of this clause and G_i is the resulting goal (which is constructed as in definition 5).

(2) If the selected literal L_k of G_{i-1} is a negative ground literal $\neg A$, then there exists a finitely failed SLDENF-tree of rank less than r for $P \cup \{\leftarrow A\}$ and G_i is as G_{i-1} except that it does not contain L_k.

A *finitely failed SLDENF-tree of rank* r ($r \geq 1$) for $P \cup \{G\}$ via a selection function R is a finite tree such that

(1) Each node is labelled with a non-empty normal goal and the root is labelled with G.

(2) For each leaf node $\leftarrow L_1\ldots,L_m$ such that L_k is the selected literal

  (a) If L_k is a positive literal, then it does not E-unify with the head of a new variant of any program clause.

  (b) If L_k is a negative ground literal $\neg A$, then there exists an SLDENF-refutation of rank less than r for $P \cup \{\leftarrow A\}$ via R.

(3) If $\leftarrow L_1,\ldots,L_k,\ldots,L_m$ is an inner node such that L_k is selected, then

  (a) If L_k is a positive literal, then for each program clause let $A \leftarrow B_1,\ldots,B_l$ be a new variant and let $cU_E(L_k, A)$ be a complete and minimal set of E-unifiers of L_k and A. Then, for each $\theta \in cU_E(L_k, A)$, the node labelled with the goal $\leftarrow L_1,\ldots,L_{k-1}, B_1,\ldots,B_l, L_{k+1},\ldots,L_m)\theta$ is a child of this inner node.

  (b) If L_k is a negative ground literal $\neg A$, then there exists a finitely failed SLDENF-tree of rank less than r for $P \cup \{\leftarrow A\}$ via R and the only child of the inner node is labelled with the goal $\leftarrow L_1,\ldots,L_{k-1}, L_{k+1},\ldots,L_m$.

The length of a refutation, the computed answer substitution, and the depth of a tree are defined as above.

It is noteworthy that all SLDENF-refutations and finitely failed SLDENF-trees of rank r are at the same time of rank greater than r. As for the selection function, we assume that it is fair, i.e. that each literal occurring in a goal is selected after finitely many steps. As usual, a derivation is said to *flounder* if the derivation yields a goal which contains only non-ground negative literals [34].

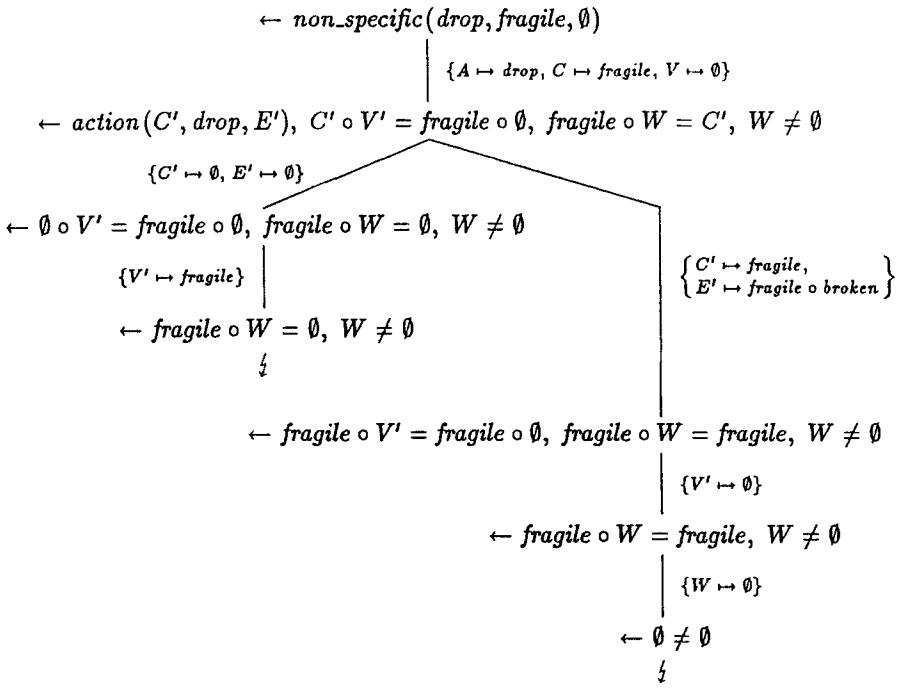

Fig. 4. A finitely failed SLDENF-tree of rank 1 for $P_A \cup \{\leftarrow non\_specific(drop, fragile, \emptyset)\}$, where A consists of the two action descriptions (2) and (3). The last node of the leftmost branch being a leaf is justified by the fact that there is no AC1-unifier of $fragile \circ W$ and $\emptyset$, whereas the last node in the rightmost branch is justified to be a leaf by the existence of an SLDENF-refutation of rank 0 for $P_A \cup \{\leftarrow \emptyset = \emptyset\}$.

To illustrate these definitions, fig. 4 shows a finitely failed SLDENF-tree of rank 1 for $P_A \cup \{\leftarrow non\_specific(drop, fragile, \emptyset)\}$, where $A = \{(2), (3)\}$, i.e. the action definition $\langle \{\!\{ fragile \}\!\}, drop, \{\!\{ fragile, broken \}\!\} \rangle$ is indeed the most specific one w.r.t. the situation $fragile \circ \emptyset =_{AC1} fragile$. Figure 5 shows an SLDENF-refutation of rank 2 for $P_A \cup \{\leftarrow causes(fragile, [drop], X)\}$ which uses the SLDENF-tree of fig. 4 and yields the computed answer substitution $\{X \mapsto fragile \circ broken\}$. In both examples, subgoals are selected from left to right and unifiers are computed under the equational theory AC1.

Recently, Shepherdson could prove the soundness of SLDENF-resolution w.r.t. the completion semantics [48]:

THEOREM 8

Let (P, E) be a normal E-program and G a normal goal $\leftarrow L_1, \ldots, L_m$.

(1) If there is an SLDENF-refutation for $P \cup \{G\}$ with computed answer substitution θ, then $(P^*, E^*) \models \forall((L_1 \wedge \ldots \wedge L_m)\theta)$.

(2) If $P \cup \{G\}$ has a finitely failed SLDENF-tree, then $(P^*, E^*) \models \neg\exists(L_1 \wedge \ldots \wedge L_m)$.

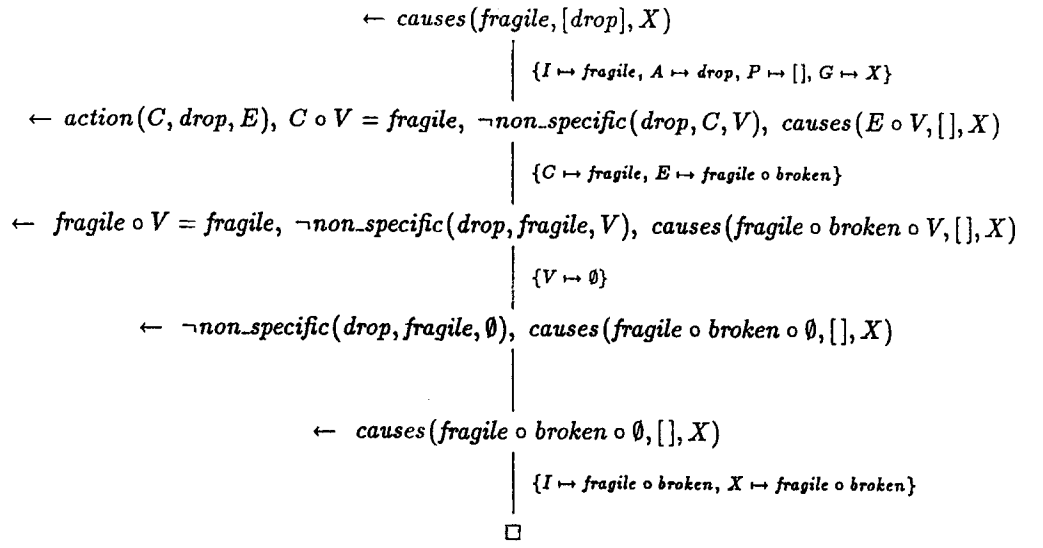

Fig. 5. An SLDENF-refutation of rank 2 for $P_A \cup \{\leftarrow causes(fragile, [drop], X)\}$, where A consists of the two action descriptions (2) and (3). The fourth derivation step is justified by the exitence of a finitely failed SLDENF-tree of rank 1 for $P_A \cup \{\leftarrow non\_specific(drop, fragile, \emptyset)\}$ (see fig. 4).

The question whether SLDENF-resolution is complete is more difficult to answer. Negation-as-failure is incomplete in general [9], i.e. completeness results for SLDNF-resolution, where no equational theory is considered, are obtained only for restricted classes of programs. For instance, it is complete for definite programs and normal ground goals [28] and for hierarchical normal programs and normal goals, if they are allowed [28,47]. In the latter case, the conditions attached to programs and goals ensure that the SLDNF-tree of a program and a goal exists and is finite, and that a derivation of a goal with respect to a program never flounders.

We need similar conditions for the E-programs considered in this paper. But there is an additional difficulty. As the axioms of equality are built into the unification computation and E-unification problems may be infinitary, a node occurring in an SLDENF-tree may have infinitely many successor nodes. Consider, for instance, the normal E-program (P, E) which consists of the clauses

$$\begin{array}{lll} p(b). & & \\ q(X, X) & \leftarrow & p(X). \\ r & \leftarrow & q(a \cdot Y, Y \cdot a). \end{array} \tag{15}$$

along with the law of associativity for the binary function $\cdot$. Clearly, the completion of this program is consistent, each derivation of $P \cup \{\leftarrow r\}$ is finite, and no derivation of $P \cup \{\leftarrow r\}$ flounders. $\neg r$ is a consequence of the completion of (P, E) since q is

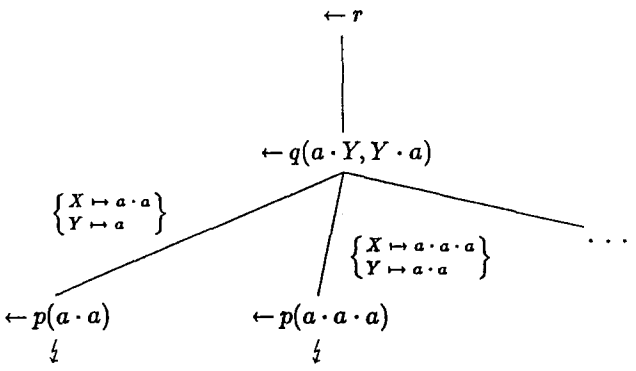

Fig. 6. An infinite SLDENF-tree for $P \cup \{\leftarrow r\}$, where P consists of the clauses (15) and $\cdot$ is an associative function.

true just in the case of $q(b, b)$. Searching for an SLDENF-refutation for the goal $\leftarrow r$ using any most general E-unifier of $q(a \cdot Y, Y \cdot a)$ and $q(X, X)$ leads to a binding of the form $\{X \mapsto a \cdot \ldots \cdot a\}$ and, hence, to a subgoal of the form $p(a \cdot \ldots \cdot a)$ after two steps which fails to unify with $p(b)$. Thus, every derivation of r fails within exactly three steps. However, there is no finitely failed SLDENF-tree for $P \cup \{\leftarrow r\}$ since there is an infinite number of most general E-unifiers of $q(a \cdot Y, Y \cdot a)$ and $q(X, X)$ w.r.t. associativity (see fig. 6). This phenomenon cannot be observed in the case of finitary equational theories – provided the employed unification algorithm always computes a finite set – which leads to the following completeness result.

THEOREM 9

Let (P, E) be a normal E-program, where E is finitary and (P^*, E^*) is consistent, and G be a normal goal $\leftarrow L_1, \ldots, L_m$ such that no SLDENF-derivation of $P \cup \{G\}$ flounders or is infinite. If $(P^*, E^*) \models \exists(L_1 \wedge \ldots \wedge L_m)$, then there exists an SLDENF-refutation for $P \cup \{G\}$.

Proof

The proof is by contradiction. Assume that $(P^*, E^*) \models \exists(L_1 \wedge \ldots \wedge L_m)$ and there does not exist an SLDENF-refutation for $P \cup \{G\}$. Since no derivation of $P \cup \{G\}$ is infinite and because E is finitary, we find that the SLDENF-tree of $P \cup \{G\}$ is finite. Since no derivation of $P \cup \{G\}$ flounders, this tree is finitely failed. By theorem 8(2), we learn that in this case $(P^*, E^*) \models \neg\exists(L_1 \wedge \ldots \wedge L_m)$. □

The previous theorem is based on the premise that no derivation of $P \cup \{G\}$ flounders or is infinite. We would like to have a syntactical condition which guarantees this premise. Unfortunately, as already mentioned in section 5, the E-programs $(P_A, \text{AC1})$ are not hierarchical as *causes* is recursive. The number of recursive calls to *causes*

depends on the second argument of *causes*, which is a list. If the length of this list is known in advance, then each derivation of a query of the form $\leftarrow causes(i, p, g)$ is finite, where i and g are (possibly uninstantiated) AC1 terms denoting the initial and goal situation, respectively, and p is a (possibly uninstantiated) list of a given length, e.g. $p = [A_1,\ldots,A_n]$, where A_j, $1 \leq j \leq n$, are variables. Hence, if the program $(P_A, \text{AC1})$ is applied to generate a plan, i.e. sequence of actions, for transforming an initial situation i into a goal situation g and completeness is important, then the number of actions in the plan have to be fixed in advance. If such a plan does not exist, then a kind of *iterative deepening* procedure should be applied which seeks for a longer plan.

However, knowing the length of the plan does not guarantee that an SLDENF-derivation does not flounder. In what follows, we prove the claim that no derivation of some $\leftarrow causes(i, p, g)$ flounders if the initial situation i is ground. These two observations regarding finiteness and non-floundering, respectively, are combined in the following proposition, which allows us to apply theorem 9 to our program. In the sequel, let $(P_A, \text{AC1})$ be an equational logic program encoding a set A of action definitions. Consequently, SLDENF-derivations are computed w.r.t. the equational theory AC1.

PROPOSITION 10

Let G be the goal $\leftarrow causes(i, [a_1,\ldots,a_n], g)$, where i is ground and $a_1,\ldots,a_n, g$ are arbitrary terms ($n \geq 0$). Then, no SLDENF-derivation of $P_A \cup \{G\}$ flounders or is infinite.

Proof

The proof is by induction on n.

In the case $n = 0$, $causes(i, [\,], g)$ can only be AC1-unified with the first clause of (7) and only if i and g are AC1-unifiable. Hence, either we obtain the empty clause, or the derivation fails immediately.

The the case $n > 0$, G can only be AC1-unified with the head of the second clause of (7). Applying an SLDENF-resolution step yields the goal

$$\leftarrow action(C, a_1, E), C \circ V = i, \neg non\_specific(a_1, C, V), causes(E \circ V, [a_2,\ldots,a_n], g).$$

Without loss of generality, it is assumed that the literals are selected from left to right. Note that the third literal cannot be selected until its arguments are instantiated to ground terms. If the leftmost literal is selected, then either the derivation finitely fails, or the literal can be solved using a new variant $action(c, a, e)$ of one of the clauses (6), yielding a substitution θ_1 such that $C\theta_1 = c\theta_1$, $a_1\theta_1 = a\theta_1$, and $E\theta_1 = e\theta_1$. According to definition 1, we know that a is a constant and, hence, $\mathcal{V}ar(a\theta_1) = \mathcal{V}ar(a) = \emptyset$, and $\mathcal{V}ar(e\theta_1) \subseteq \mathcal{V}ar(c\theta_1)$. The resulting goal is

$$\leftarrow c\theta_1 \circ V = i, \neg non\_specific(a, c\theta_1, V), causes(e \circ V, [a_2,\ldots,a_n], g)\theta_1.$$

Now, if again the leftmost literal is selected, then either the derivation finitely fails or the literal can be solved via AC1-unification, i.e. by applying clauses (9), with substitution θ_2. Note that $(c\theta_1 \circ V)\theta_2$ and $a\theta_2 = a$ as well as $(e \circ V)\theta_1\theta_2$ are necessarily ground because i is a ground term and the equational theory AC1 is regular. Then, the remaining goal is

$$\leftarrow \neg non\_specific(a, c, V)\theta_1\theta_2, causes(e \circ V, [a_2,\ldots,a_n], g)\theta_1\theta_2.$$

According to the preceding discussion, the negative literal can now be selected. The following lemma 11 proves that the corresponding SLDENF-tree is finite. Thus, either the derivation finitely fails, or continues with

$$\leftarrow causes(e \circ V, [a_2,\ldots,a_n], g)\theta_1\theta_2.$$

Due to the fact that θ_2 is a ground substitution for $c\theta_1$ and V, and since $Var(e\theta_1) \subseteq Var(c\theta_1)$, it follows that the first argument of the remaining literal is ground. Hence, we can apply the induction hypothesis to show that the derivation neither flounders nor is infinite. □

LEMMA 11

Let G be the goal $\leftarrow non\_specific(a, c, v)$, where c, a, v are ground terms; then no SLDENF-derivation of $P_A \cup \{G\}$ flounders or is infinite.

Proof

There is only one possibility to start the derivation, namely to apply clause (8), which yields

$$\leftarrow action(C', a, E'), C' \circ V' = c \circ v, c \circ W = C', W \neq \emptyset.$$

Without loss of generality, it is assumed that the literals are selected from left to right. Note that the rightmost literal cannot be selected until W is instantiated to a ground term. If the leftmost literal is selected, then either the derivation finitely fails, or the literal can be solved using one of the clauses (6), yielding a substitution θ_1 such that $C'\theta_1 = c'\theta_1$ and $E'\theta_1 = e'\theta_1$ for some terms c', e'. The resulting goal is

$$\leftarrow c'\theta_1 \circ V' = c \circ v, c \circ W = c'\theta_1, W \neq \emptyset.$$

Now, if the two leftmost literals are selected, then either the derivation finitely fails or they can be solved via AC1-unification, i.e. by applying clause (9), with substitutions θ_2 and θ_3, respectively. Note that $c \circ v$ being ground implies, as in the preceding proof, $c'\theta_1\theta_2\theta_3$ being ground, which therefore holds for $W\theta_2\theta_3$ as well. Hence, the remaining goal

$$\leftarrow W\theta_2\theta_3 \neq \emptyset$$

does not flounder and either fails or reduces to the empty goal. □

Requiring the initial situation to be fully instantiated is acceptable in the case of modelling the application of methods to given objects, or in the case of planning problems. On the other hand, our approach is also suitable for more general kinds of reasoning about actions and change. This is illustrated in the following section, where it is also argued that, to this end, it is necessary to consider incompletely specified, i.e. only partially instantiated, initial situations. A solution which allows one to overcome the restriction given via proposition 10 is then briefly discussed in section 8.

7. Reasoning about the past

Beside temporal projections, which is to predict what happens given an initial situation or object and a sequence of actions or methods, and planning, which is to find an appropriate sequence of actions which transforms a given initial situation into a given goal situation, one is often interested in reasoning about situations in the past, depending on what can be observed in the actual situation. This causes no problems in the original equational logic programming approach [25], but is more difficult for actions and specificity since negative literals occur in clause bodies and incomplete information about former situations is modelled via partially instantiated situations which may lead to uninstantiated negative subgoals. These problems are investigated in this and the following section.

As an example, recall the Broken Item domain from section 3 in a variant which we call Broken Item Mystery. Assume that a previously intact item is broken after the execution of a *drop* action. In this case, we want to conclude that the item must have been fragile at the beginning. Can we derive such a conclusion with our method?

The Broken Item Mystery is as the Broken Item domain discussed in section 3, except that a new fluent, *solid*, is introduced. We will see later on that a fundamental operation for reasoning about the past is the decision whether a situation or a collection of situations is consistent. The new fluent causes a reformulation of the inconsistency criterion (1): A situation S in the Broken Item Mystery domain is defined to be inconsistent iff

$$\begin{aligned} &\{\!\{broken, intact\}\!\} \dot{\subseteq} S \vee \{\!\{solid, fragile\}\!\} \dot{\subseteq} S \vee \\ &\exists X \in \{broken, intact, solid, fragile\}.\ \{\!\{X, X\}\!\} \dot{\subseteq} S. \end{aligned} \tag{16}$$

As a completed clause, this can be formalized for the Broken Item domain by

$$\begin{aligned} \forall X[inconsistent(X) \leftrightarrow\ & \exists Y.\ broken \circ intact \circ Y = X \\ \vee\ & \exists Y.\ solid \circ fragile \circ Y = X \\ \vee\ & \exists Y.\ broken \circ broken \circ Y = X \\ \vee\ & \exists Y.\ intact \circ intact \circ Y = X \\ \vee\ & \exists Y.\ solid \circ solid \circ Y = X \\ \vee\ & \exists Y.\ fragile \circ fragile \circ Y = X]. \end{aligned} \tag{17}$$

Recall that the actions or methods were defined such that their application to a consistent situation yields again a consistent situation (see section 3). Since in the temporal projection problems considered so far, the initial – and presumably consistent – situation was given, there was no need to test the consistency of derived situations. Reasoning about the past, however, is mainly based on finding consistent explanations for observations, and the initial as well as the final situation are only partially defined. For example, in the Broken Item Mystery, one might ask whether the formula

$$\exists V, W[causes(intact \circ V, [drop], broken \circ W)] \tag{18}$$

is entailed by the completed program. Since the initial situation $intact \circ V$ as well as the final situation $broken \circ W$ contains a variable, we have to check the consistency of the situations as soon as the variables are instantiated. Fortunately, however, since the application of an action preserves the consistency of a situation, we have to perform a consistency check only once. Hence, it suffices to add to the query (18) a single consistency check:

$$\exists V, W[causes(intact \circ V, [drop], broken \circ W) \wedge \neg inconsistent(intact \circ V)]. \tag{19}$$

In other words, can we find a consistent initial situation $\mathcal{I} \dot{\supseteq} \{\!\{intact\}\!\}$ such that the sequence of actions $[drop]$ transforms $\mathcal{I}$ into a situation $\mathcal{G}$ such that $\mathcal{G} \dot{\supseteq} \{\!\{broken\}\!\}$? It should be observed that requiring consistency is inevitably necessary since otherwise a simple solution to (18) would consist of the bindings $\{V \mapsto broken, W \mapsto intact\}$.

Expression (19) is in fact entailed, yielding e.g. the bindings $V \mapsto fragile$ and $W \mapsto fragile$. Thus, it is consistent to assume that the object was fragile. However, this result is not sufficient if we want to be ensured that the object must have been fragile. To test whether *fragile* holds necessarily in the initial situation, we should ask whether it is impossible to assume the contrary, i.e. to assume that the object was solid. This could be achieved by the additional question whether the formula

$$\begin{aligned} \neg \exists V', W'[&causes(intact \circ solid \circ V', [drop], broken \circ W') \wedge \\ &\neg inconsistent(intact \circ solid \circ V')] \end{aligned} \tag{20}$$

is entailed, which should be interpreted as there is no way to satisfy the observation that the item is broken provided it was intact and solid at the beginning. Expression (20) is indeed entailed. Informally, the only possibility for unifying the goal situation $broken \circ W'$ with the result of applying an action description of *drop* to the initial situation $intact \circ solid \circ V'$ requires V' to be substituted by a term like $fragile \circ Z$. This, however, can be shown to be inconsistent for any Z since *solid* and *fragile* must not occur in a situation due to the consistency criterion (17). Thus, the consistency check and the completeness of AC1* provide the desired (negative) answer. Obviously, such conclusions are mainly based on the unification completeness of AC1*. Otherwise,

it is impossible to find proofs for statements as, for example, $\forall Z[\mathit{intact} \circ \mathit{solid} \circ V = Z \rightarrow (\forall Z'.\, \mathit{fragile} \circ Z' \neq Z) \vee \mathit{inconsistent}(Z)]$, i.e. if situation Z contains the fluents *intact* and *solid*, then it either does not contain the fluent *fragile* or it is inconsistent.

The final proposition is a justification of the method described above. Whenever backward reasoning should be performed, then we start with searching for fluents which are consistent to assume by investigating the various instances which make formulas as e.g. (19) true. Then, to test whether these fluents must have been true at the beginning, we try to assume the contrary of each fluent separately using formulas as e.g. (20). The following proposition claims that whenever there are no two multisets $\mathcal{V}$ and $\mathcal{W}$ such that the consistent initial situation $i^{\mathcal{I}} \dot{\cup} \mathcal{V}$ can be transformed into the goal situation $g^{\mathcal{I}} \dot{\cup} \mathcal{W}$ via the sequence $[a_1,\ldots,a_n]$ w.r.t. a set of action descriptions A, then the formula $\neg \exists V, W[\mathit{causes}(i \circ V, [a_1,\ldots,a_n], g \circ W) \wedge \neg \mathit{inconsistent}(i \circ V)]$ is entailed by $(P_A^*, \mathrm{AC1}^*)$.

PROPOSITION 12

If A is a set of action descriptions, then

$$(P_A^*, \mathrm{AC1}^*) \models \neg \exists V, W[\mathit{causes}(i \circ V, [a_1,\ldots,a_n], g \circ V) \wedge \neg \mathit{inconsistent}(i \circ V)]$$

iff there are no consistent multisets $S_0,\ldots,S_n$ such that $S_j \doteq (S_{j-1} \dot{-} C_j) \dot{\cup} \mathcal{E}_j$, where $\langle C_j, a_j, \mathcal{E}_j \rangle \in A$ is a most specific action description of a_j w.r.t. S_{j-1} $(1 \leq j \leq n)$ and $i^{\mathcal{I}} \dot{\subseteq} S_0$ and $g^{\mathcal{I}} \dot{\subseteq} S_n$.

Proof

The claim follows from propositions 7 and 4. □

A semantics for methods to reason about actions and change was recently proposed by Gelfond and Lifschitz [17]. Their *Action Description Language* $\mathcal{A}$ allows the specification of simple action scenarios and is capable of performing the kind of reasoning described here, i.e. reasoning about the past and handling partially specified situations are supported. As the above analysis already indicates, our equational logic programming based approach includes the expressiveness of the Action Description Language. The formal soundness and completeness result of a formalization in terms of equational programs including the concept of specificity w.r.t. the semantics of $\mathcal{A}$ is established in [52]. This result shows that our method is provably equivalent to a variety of other systems designed for reasoning about actions and change, most of them based on the situation calculus, which were also related to $\mathcal{A}$ recently, such as Baker's method [5] based on circumscription [37], Pednault's [40] and Reiter's [41] approach based on classical logic (all three adequateness result regarding $\mathcal{A}$ were established in [31]), or the application of abductive logic programming ([12] and, independently, [11]).

Moreover, in [51], the Action Description Language is related to Sandewall's so-called *Ego-World-Semantics* [43,44], which provides another semantical framework

to reason about dynamically changing worlds. The equivalence of a slightly restricted version of $\mathcal{A}$ to a particular ontological problem class in Sandewall's methodology has been proved. This justifies the claim that our approach forms just as well a sound and complete encoding of this problem class within the frame of the Ego-World-Semantics.

8. Constructive negation

A problem arises when trying to solve goals of the form used in the previous section as, for example, (18) with SLDENF-resolution. The problem is a consequence of the fact that a derivation flounders if the actual set of subgoals contains only negative non-ground literals. Consider, for instance, the formula $\exists V[causes(intact \circ V, [], intact \circ V) \wedge \neg inconsistent(intact \circ V)]$. Clearly, this is entailed by the completed clauses (13) and (17) by using the binding $\{V \mapsto \emptyset\}$, say. But as a goal to the logic E-program defined by (13) and (17), there is no refutation because this requires to solve the goal $\leftarrow \neg inconsistent(intact \circ V)$, which flounders immediately.

To solve this problem, we suggest to use the idea of constructive negation [8]. Informally, constructive negation handles negative non-ground literals in the way that a complete set of answer substitutions to the corresponding positive literals is computed, and afterwards these answers are used to define restrictions on the variables occurring in the original negative goal literal. A special case occurs when the disjunction of these answers evaluates to true – then the negative goal fails. For instance, the set of answers to the positive goal $\leftarrow inconsistent(intact \circ V)$ determined by clause (17) is

$$\begin{aligned}
&\{V = broken \circ Y,\\
&\ V = solid \circ fragile \circ Z \wedge Y = intact \circ Z,\\
&\ V = broken \circ broken \circ Z \wedge Y = intact \circ Z,\\
&\ V = intact \circ Y,\\
&\ V = solid \circ solid \circ Z \wedge Y = intact \circ Z,\\
&\ V = fragile \circ fragile \circ Z \wedge Y = intact \circ Z\}.
\end{aligned} \tag{21}$$

Thus, the constructive answer to the negative goal $\leftarrow \neg inconsistent(intact \circ V)$ is the conjunction

$$\begin{aligned}
&V \neq broken \circ Y \wedge V \neq solid \circ fragile \circ Z \wedge V \neq broken \circ broken \circ Z \wedge\\
&V \neq intact \circ Y \wedge V \neq solid \circ solid \circ Z \wedge V \neq fragile \circ fragile \circ Z,
\end{aligned} \tag{22}$$

which is satisfiable via, for example, $V = \emptyset$. Note that the restrictions to the variable Y in (21) do not occur in (22) since Y was not a free variable in the goal literal $\neg inconsistent(intact \circ V)$.

What happens in the case of a formula like e.g. $\exists V. \neg inconsistent(intact \circ broken \circ V)$, which should be unsatisfiable? The answer set to the positive goal $inconsistent(intact \circ broken \circ V)$ is

$$\begin{aligned}
&\{V = Y,\\
&V = solid \circ fragile \circ Z \wedge Y = intact \circ broken \circ Z,\\
&V = broken \circ Z \wedge Y = intact \circ Z,\\
&V = intact \circ Z \wedge Y = broken \circ Z,\\
&V = solid \circ solid \circ Z \wedge Y = intact \circ broken \circ Z,\\
&V = fragile \circ fragile \circ Z \wedge Y = intact \circ broken \circ Z\}.
\end{aligned} \tag{23}$$

This set, which is interpreted as a disjunction of the elements, evaluates to true due to the first element. Thus, the goal $\neg inconsistent(intact \circ broken \circ V)$ fails, which is exactly the desired behavior.

Constructive negation is known to be sound and complete w.r.t. the completed program in the case of finite derivation trees (cf. [8]) if the standard equational theory is assumed. Whether this also holds in the case of an additional equation theory – which is finitary – is an open problem. If this question can be positively answered, then we are able to solve goals of the form determined by formulas as, for example, (18) or (20).

9. Discussion

In this paper, 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 S by deleting its condition from and adding its effect to 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 [1,20] or [42].

The specificity relation is computed syntactically by comparing the conditions of different action descriptions for the same action and preferring the description whose condition is a superset of the conditions of the other descriptions. At first glance, this seems to be a weaker criterion compared to the definition of specificity in non-monotonic logics such as conditional logics as given in [10] or [16]. In these logics, specificity is not only defined w.r.t. the antecedent of conditional implications (i.e. defaults), but also w.r.t. a set of strict formulae. We believe that our approach can be extended to consider such a set of strict formulae as well. However, only a rigorous comparison between the various approaches to handle specificity will show whether they are really identical.

The equational logic programs considered in this paper are normal programs together with an equational theory. Such programs are queried by normal goals. As mentioned before, this class of problems is a combination of the class of problems considered in [29] consisting of definite programs with equality and normal goals and the class of problems considered in [2] consisting of normal programs and normal goals, but without a general equational theory. We have shown that the models for the equational logic programs considered herein interpret causality and specificity as

intended. However, we have not yet investigated the class of normal programs with equality and normal goals in general. We believe that the model intersection property holds also for this class and that a fixpoint theory can be developed which relates least fixpoints and least models. The question which of the various results concerning consistency and completeness of SLDNF-resolution such as [45, 30, 50] can be extended to SLDENF-resolution as well is part of future work.

We have mainly restricted our presentation to actions, whose conditions and effects are ground, and to situations, which are ground as well. However, as already discussed at the end of section 6, 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. It causes also no problems as long as negative literals do not occur in clause bodies (cf. [20]). However, as soon as negative literals in clause bodies are allowed and negation-as-failure is applied, we have to be more careful. Currently, we must ensure that negative subgoals are fully instantiated before they are called. However, this condition seems to be too strong. For the particular application in mind, viz. specifying actions, causality and specificity, we hope to find weaker conditions. For instance, a subgoal of the form $W \neq \emptyset$ can be decided iff W is either instantiated to $\emptyset$ or to a term of the form $s_1 \circ \ldots \circ s_n$, where $n \geq 1$ and s_i, $1 \leq i \leq n$ are elementary terms. We could go even one step further and use constructive negation as indicated in section 8. Such an extension would greatly enhance the expressive power of our system, as the example given in section 7 shows.

References

[1] J-M. Andreoli and R. Pareschi, Linear objects: Logical processes with built-in inheritance, New Generation Comp. 9(3,4) (1991).

[2] K.R. Apt, H.A. Blair and A. Walker, Towards a theory of declarative knowledge, in: *Foundations of Deductive Databases and Logic Programming*, chap. 2, ed. J. Minker (Kaufmann, 1987) pp. 89–148.

[3] F. Baader and J.H. Siekmann, Unification theory, in: *Handbook of Logic in Artificial Intelligence and Logic Programming*, eds. D.M. Gabbey, C.J. Hogger and J.A. Robinson (Oxford University Press, 1993).

[4] A.B. Baker, A simple solution to the Yale shooting problem, *Proc. Int. Conf. on Knowledge Representation and Reasoning*, 1989, pp. 11–20.

[5] A.B. Baker, Nonmonotonic reasoning in the framework of situation calculus, Artificial Intelligence Journal 49(1991)5–23.

[6] W. Bibel, A deductive solution for plan generation, New Generation Comp. 4(1986)115–132.

[7] W. Bibel, Intellectics, in: *Encyclopedia of Artificial Intelligence*, ed. S.C. Shapiro (Wiley, New York, 1992) pp. 705–706.

[8] D. Chan, Constructive negation based on the completed database, *Proc. Int. Joint Conf. and Symp. on Logic Programming (IJCSLP)*, 1988, pp. 111–125.

[9] K.L. Clark, Negation as failure, in: *Workshop Logic and Data Bases*, eds. H. Gallaire and J. Minker (Plenum Press, 1978) pp. 293–322.

[10] J.P. Delgrande, A semantically-based account of nonmonotonic reasoning in Horn-clause and logic programming (1992), submitted J. Logic. Progr.

[11] M. Denecker and D. de Schreye, Representing incomplete knowledge in abductive logic programming, *Proc. Int. Logic Programming Symposium* (*ILPS*), Vancouver, 1993, ed. D. Miller (MIT Press) pp. 147–163.
[12] P.M. Dung, Representing actions in logic programming and its applications in database updates, *Proc. Int. Conf. on Logic Programming* (*ICLP*), Budapest, 1993, ed. D.S. Warren (MIT Press) pp. 222–238.
[13] L. Brownston et al., *Programming Expert Systems in OPS5* (Addison–Wesley, Reading, MA, 1985).
[14] R.E. Fikes and N.J. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving, Artificial Intelligence Journal 5(1971)189–208.
[15] J.H. Gallier and S. Raatz, Extending SLD-resolution to equational Horn clauses using *E*-unification, J. Logic Progr. 6(1989)3–44.
[16] H. Geffner and J. Pearl, Conditional entailment: Bridging two approaches to default reasoning, Artificial Intelligence 53(1992)209–244.
[17] M. Gelfond and V. Lifschitz, Representing action and change by logic programs, J. Logic Progr. 17(1993)301–321.
[18] J.Y. Girard, Linear logic, J. Theor. Comp. Sci. 50(1987)1–102.
[19] G. Große, S. Hölldobler and J. Schneeberger, Linear deductive planning, J. Logic and Comput. (1995), to appear.
[20] G. Große, S. Hölldobler, J. Schneeberger, U. Sigmund and M. Thielscher, Equational logic programming, actions, and change, *Proc. Int. Joint Conf. and Symp. on Logic Programming* (*IJCSLP*), Washington, 1992, ed. K. Apt (MIT Press) pp. 177–191.
[21] S. Hanks and D. McDermott, Nonmonotonic logic and temporal projection, Artificial Intelligence Journal 33(1987)379–412.
[22] P. Hoddinott and E.W. Elcock, Prolog: Subsumption of equality axioms by the homogeneous form. *Proc. Symp. on Logic Programming* (1986) pp. 115–126.
[23] S. Hölldobler, *Foundations of Equational Logic Programming*, Lecture Notes in Artificial Intelligence 353 (Springer 1989).
[24] S. Hölldobler, On deductive planning and the frame problem, *Proc. Int. Conf. Logic Programming and Automated Reasoning* (*LPAR*), Lecture Notes in Artificial Intellegence 624 (Springer, 1993) pp. 13–29.
[25] S. Hölldobler and J. Schneeberger, Deductive approach to planning, New Generation Comp. 8(1990)225–244.
[26] S. Hölldobler and M. Thielscher, Actions and specificity, *Proc. Int. Logic Programming Symposium* (*ILPS*), Vancouver, 1993 (MIT Press) pp. 164–180.
[27] S. Hölldobler and M. Thielscher, Properties vs. resources – solving simple frame problems, Technical Report AIDA-94-15, Intellektik, Informatik, TH Darmstadt (1994).
[28] J. Jaffar, J.-L. Lassez and J. Lloyd, Completeness of the negation as failure rule, *Proc. Int. Joint. Conf. on Artificial Intelligence* (*IJCAI*), 1983, pp. 500–506.
[29] J. Jaffar, J.-L. Lassez and M.J. Maher, A theory of complete logic programs with equality, J. Logic Progr. 1(1984)211–223.
[30] G. Jäger and R.F. Stärk, The defining powers of stratified and hierarchical logic programs, J. Logic Progr. 15(1993)55–77.
[31] G.N. Kartha, Soundness and completeness theorems for three formalizations of actions, *Proc. Int. Joint Conf. on Artificial Intelligence* (*IJCAI*), Chambéry, France, 1993 (Kaufmann) pp. 724–729.
[32] R. Kowalski, *Logic for Problem solving*, Vol. 7 of Artificial Intelligence Series (Elsevier, 1979).
[33] V. Lifschitz, On the semantics of STRIPS, *Proc. Workshop on Reasoning about Actions and Plans*, eds. M.P. Georgeff and A.L. Lansky (Kaufmann, 1986).
[34] J.W. Lloyd, *Foundations of Logic Programming*, Symbolic Computation Series, 2nd extended ed. (Springer, 1987).

[35] M. Masseron, C. Tollu and J. Vauzielles, Generating plans in linear logic, in: *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science 472 (Springer, 1990) pp. 63–75.

[36] J. McCarthy, Situations and actions and causal laws, Memo 2, Stanford Artificial Intelligence Project (1963).

[37] J. McCarthy, Circumscription – A form of non-monotonic reasoning, Artificial Intelligence Journal 13(1980)27–39.

[38] J. McCarthy, Applications of circumscription to formalizing common-sense knowledge, Artificial Intelligence Journal 28(1986)89–116.

[39] J. McCarthy and P.J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, Machine Intelligence 4(1969)463–502.

[40] E. Pednault, ADL: Exploring the middle ground between STRIPS and the situation calculus, *Proc. Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR*), 1989, eds. R. Brachman, H. Levesque and R. Reiter, pp. 324–332.

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

[42] R. Reiter, Formalizing database evolution in the situation calculus, *Proc. Int. Conf. on Fifth Generation Computer Systems*, 1992.

[43] E. Sandewall, Features and fluents, Technical Report LiTH-IDA-R-92-30, Institutionen för datavetenskap, Univeritetet och Tekniska högskolan i Linköping, Schweden (1992).

[44] E. Sandewall, The range of applicability of nonmonotonic logics for the inertia problem, *Proc. Int. Joint Conf. on Artificial Intelligence* (*IJCAI*), Chambéry, France, 1993 (Kaufmann) pp. 738–743.

[45] T. Sato, Completed logic programs and their consistency, J. Logic Progr. 9(1990)33–44.

[46] J. Schneeberger, Plan generation by linear deduction, Ph.D. Thesis, FG Intellektik, TH Darmstadt (1992).

[47] J.C. Shepherdson, Negation in logic programming for general logic programs, in: *Foundations of Deductive Databases and Logic Programming*, chap. 1, ed. J. Minker (Kaufmann, 1987) pp. 19–88.

[48] J.C. Shepherdson, SLDNF-resolution with equality, J. Autom. Reasoning 8(1992)297–306.

[49] J.H. Siekmann, Unification theory, J. Symb. Comput. 7(1989)207–274. Special issue on unification.

[50] K. Stroetmann, A completeness result for SLDNF-resolution, J. Logic Progr. 15(1993)337–355.

[51] M. Thielscher, An analysis of systematic approaches to reasoning about actions and change, *Int. Conf. on Artificial Intelligence: Methodology, Systems, Applications* (*AIMSA*), Sofia, Bulgaria, 1994, ed. P. Jorrand (World Scientific, Singapore). Available by anonymous ftp from 130.83.26.1 in /pub/AIDA/Tech-Reports/OTHER.

[52] M. Thielscher, Representing actions in equational logic programming, *Proc. Int. Conf. on Logic Programming* (*ICLP*), Santa Margherita Ligure, Italy, 1994, ed. P. Van Hentenryck (MIT Press) pp. 207–225.