Two Components of An Action Language

Vladimir Lifschitz
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA

Abstract

Some of the recent work on representing action makes use of high-level action languages. In this paper we show that an action language can be represented as the sum of two distinct parts: an "action description language" and an "action query language." A set of propositions in an action description language describes the effects of actions on states. Mathematically, it defines a transition system of the kind familiar from the theory of finite automata. An action query language serves for expressing properties of paths in a given transition system. We define the general concepts of a transition system, of an action description language and of an action query language, give a series of examples of languages of both kinds, and show how to combine a description language and a query language into one. This construction makes it possible to design the two components of an action language independently, which leads to the simplification and clarification of the theory of actions.

1 Introduction

Properties of actions have been the subject of research in the theory of commonsense knowledge since its early days. Even if we restrict attention to the representations based on the ontology of the situation calculus [McCarthy and Hayes, 1969], we will need to distinguish between proposals of several kinds, differing by the choice of the underlying logic. Some of them combine the situation calculus with classical logic (for instance, [Schubert, 1990], [Reiter, 1991], [Elkan, 1992]), some with logic programming (for instance, [Eshghi and Kowalski, 1989], [Evans, 1989], [Apt and Bezem, 1990]), some with circumscription (for instance, [Haugh, 1987], [Lifschitz, 1987], [Baker, 1989]) or with other nonmonotonic formalisms (for instance, [Morris, 1988] and [Gelfond, 1989]).

Some of the more recent work on representing action makes use of high-level action languages, such as the language $\mathcal{A}$ from [Gelfond and Lifschitz, 1993].[1] Action languages can be thought of as formal models of the part of natural language that is used for describing the effects of actions.

Originally, action languages were meant to play an auxiliary role. The primary goal was to represent properties of actions in the less specialized formalisms mentioned above, and the idea was to present methods for doing that as translations from action languages. Later on, it became clear that such languages, with their limited but concise syntax, can be of interest in their own right. Defining action languages, comparing them and studying their properties helps us improve our understanding of the aspects of commonsense knowledge and reasoning related to action.

[1] Two related approaches to the study of action are associated with the ideas of a casual theory [Lin and Shoham, 1991] and of a dynamical system [Sandewall, 1995].

An examination of the existing action languages shows that each of them can be represented as the sum of two distinct parts: an "action description language" and an "action query language." A set of propositions in an action description language describes the effects of actions on states. Mathematically, it defines a transition system of the kind familiar from the theory of finite automata. An action query language serves for expressing properties of paths in a given transition system. In $\mathcal{A}$, for instance, "effect propositions" are the description part of the language, and "value propositions" are the query part.

In this paper, we define the general concepts of a transition system, of an action description language and of an action query language. We give a series of examples of languages of both kinds, and show how to combine a description language and a query language into one. This construction makes it possible to design the two components of an action language independently, which leads to the simplification and clarification of the theory of actions.

The most expressive of the action description languages defined here, DL_{is}, is based on the semantics of static causal laws proposed recently by Norman McCain and Hudson Turner, which extends and modifies the theory developed in [McCain and Turner, 1995].

Our two examples of action query languages present alternatives to "value propositions" of the language $\mathcal{A}$. Value propositions mimic the syntax of the ground atoms of the situation calculus, with its advantages (the possibility of expressing hypothetical reasoning) and disadvantages (the inability to distinguish between actual and hypothetical situations).[2]

The distinction between the actual and the hypothetical becomes particularly important when nondeterministic actions are involved. The value proposition

$$F \textbf{ after } A$$

can be taken to express any of several different things:

- that A has been executed, and F was true afterward,
- that A will be executed, and F will be true afterward,
- that no matter how A is executed, F will be true afterward,
- that A can be executed in such a way that F will be true afterward,
- that A can be executed, and if it were, F would be true afterward.

The use of value propositions in [Kartha and Lifschitz, 1994] corresponds most closely to the last, "counterfactual" interpretation. But expressing the other ideas is important also.[3]

One of the query languages described in this paper is the language of temporal projection. It allows us to talk about what will be necessarily true after executing a sequence of actions. The other is the language of actual events. It can be used for expressing conclusions about the past.

[2]An extension of the situation calculus in which one can make such a distinction is described in [Pinto and Reiter, 1993]. This difference is also made explicit in the action language proposed in [Baral *et al.*, 1997].

[3]See [Boutilier and Friedman, 1995].

2 Transition Systems

Thinking of action in terms of states and transitions is an old idea in AI. For some authors, a state is completely described by all relevant facts about it ("For a Blocks World with three blocks, there are 13 possible states" [Genesereth and Nilsson, 1987]). Others operate with states of the whole universe, or "situations," that can never be described completely [McCarthy and Hayes, 1969]. The treatment of states in this section is compatible with both views.

An *action signature* consists of three nonempty sets of symbols: a set $\mathbf{V}$ of *values*, a set $\mathbf{F}$ of *fluent names*, and a set $\mathbf{A}$ of *action names*. A *transition system* of an action signature $\langle \mathbf{V}, \mathbf{F}, \mathbf{A} \rangle$ consists of

- a set S of objects called *states*,
- a function V from $\mathbf{F} \times S$ into $\mathbf{V}$, and
- a function R from $\mathbf{A} \times S$ into the powerset of S.

The value $V(F, s)$ is the *value of F in state s*. The states that belong to $R(A, s)$ are the possible *results of executing A in state s*.

If the values of the underlying action signature are the truth values of classical logic, that is,

$$\mathbf{V} = \{\mathsf{f}, \mathsf{t}\},$$

then transition systems are identical to what Boutilier and Friedman [1995] call "MPL-models."

A *valuation* of an action signature σ is a function from fluent names of σ to values of σ. In this paper, the set of states of a transition system is always defined in such a way that every state is a valuation, and the values of fluents are defined by the equation

$$V(F, s) = s(F). \tag{1}$$

But there are cases when states have to be defined as valuations with some additional information [Giunchiglia and Lifschitz, 1995].

A transition system can be viewed as a directed graph whose vertices are states. With every vertex s we associate the valuation $F \mapsto V(F, s)$. For any action name A and any states s, s^* such that $s^* \in R(A, s)$, the graph has an edge that leads from s to s^* and is labeled A.

A sequence of consecutively executed actions can be viewed as a path in this graph, and reasoning about action can be viewed as reasoning about such paths.

3 Action Description Languages

The syntax of an *action description language* of an action signature σ is defined by a class of syntactic expressions, called *propositions*. A *system description* is a set of propositions. The semantics of the language is defined by specifying, for every system description D, the transition system of the signature σ that is *described* by D.

The action description languages introduced in this paper use the auxiliary syntactic objects, called "formulas," that are defined as follows. Let σ be an action signature. *Atomic formulas* of σ are expressions of the form

$$(F \ \mathbf{is} \ W) \tag{2}$$

where F is a fluent name of σ and W is a value of σ. *Formulas* are propositional combinations of atomic formulas.

A valuation v *satisfies* an atomic formula (2) if $v(F) = W$. The satisfaction relation is extended to arbitrary formulas according to the truth tables of propositional logic. A valuation *satisfies* a set Γ of formulas if it satisfies all formulas in Γ.

4 Basic Description Language: *DL*

In the basic description language, denoted by DL (or $DL(\sigma)$, if dependence on the signature needs to be made explicit), propositions are expressions of the form

$$A \textbf{ causes } \varphi \textbf{ if } \psi \tag{3}$$

where A is an action name and φ, ψ are formulas. We will drop **if** ψ when ψ is $\alpha \vee \neg\alpha$ for some fixed formula α.

The semantics of DL is defined as follows. For any system description D in this language, a state of the corresponding transition system $\langle S, V, R\rangle$ is an arbitrary valuation, and the values of fluents are defined by (1). For any action name A and any state s, let $D_{A,s}$ be the set of formulas φ such that, for some proposition (3) in D, s satisfies ψ. Then $R(A, s)$ is defined as the set of states that satisfy all formulas in $D_{A,s}$.

The language DL is simpler than $\mathcal{A}$ in that its semantics does not incorporate the idea of inertia—in other words, it does not address the frame problem. Whenever a fluent is not affected by an action, this fact needs to be included in the system description. In simple cases, this can be done by means of a set of propositions of the form

$$A \textbf{ causes } (F \textbf{ is } W) \textbf{ if } (F \textbf{ is } W)$$

for all $W \in \mathbf{V}$. We will denote this set by

$$A \textbf{ preserves } F.$$

As an example, consider m objects $B_1, \ldots, B_m$ that can be located in n places $L_1, \ldots, L_n$. The action $Move(B_i)$ ("change the location of B_i") can be described by the propositions

$$\begin{array}{l} Move(b) \textbf{ causes } \neg(Location(b) \textbf{ is } l) \textbf{ if } (Location(b) \textbf{ is } l), \\ Move(b) \textbf{ preserves } Location(b') \qquad (b' \neq b) \end{array}$$

for all $b, b' \in \{B_1, \ldots, B_m\}$ and $l \in \{L_1, \ldots, L_n\}$. In the transition system $\langle S, V, R\rangle$ described by these propositions, $R(Move(b), s)$ is the set of states s^* such that

$$\begin{array}{l} V(Location(b), s^*) \neq V(Location(b), s), \\ V(Location(b'), s^*) = V(Location(b'), s) \qquad (b' \neq b). \end{array} \tag{4}$$

We will abbreviate the proposition

$$A \textbf{ causes } \alpha \wedge \neg\alpha \textbf{ if } \psi$$

by

$$\textbf{impossible } A \textbf{ if } \psi.$$

Such propositions allow us to say that an action is not executable. Imagine, for instance, that location L_1 is the bottom of a well; we can express that no object can be recovered from L_1 by writing

$$\textbf{impossible } \mathit{Move}(b) \textbf{ if } \mathit{Location}(b) \textbf{ is } L_1.$$

With these propositions added to the system description, R will be characterized by the equation

$$R(\mathit{Move}(b), s) = \emptyset$$

if $V(\mathit{Location}(b), s) = L_1$, and by conditions (4) otherwise.

5 A Language with Inertia: DL_i

Consider an action signature σ and assume that the elements of a certain subset of its fluent names are designated as *inertial*. The syntax of the language DL_i is the same as that of DL. In the definition of the transition system $\langle S, V, R\rangle$ corresponding to a system description D, the functions S and V are defined as before; the only difference is in the definition of R.

Recall that, in DL, a state s^* is a possible result of executing A in a state s whenever s^* satisfies $D_{A,s}$. In DL_i, such a state s^* will be counted as a possible result only if, in addition, the changes in the values of the inertial fluents that occur when s turns into s^* are "minimal." In order to make this precise, introduce the following notation: for any valuations v_1, v_2, $\mathit{New}(v_1, v_2)$ is the set of atomic formulas of the form

$$F \textbf{ is } v_2(F) \tag{5}$$

such that F is inertial and $v_2(F) \neq v_1(F)$. The last formula expresses that (5) is a "new fact" that becomes true when an action turns v_1 into v_2 [Giunchiglia *et al.*, 1995].

For any action name A and any state s, we define $R(A, s)$ to be the set of all states s^* such that

- s^* satisfies $D_{A,s}$, and
- for every state s' that satisfies $D_{A,s}$, if

  $$\mathit{New}(s, s') \subseteq \mathit{New}(s, s^*)$$

  then

  $$\mathit{New}(s, s') = \mathit{New}(s, s^*).$$

It is clear that DL_i turns into DL in the special case when all fluent names are noninertial. In this sense, DL is a sublanguage of DL_i. The opposite special case, when all fluent names are inertial, is similar to the action description part of the original action language $\mathcal{A}$, except that $\mathcal{A}$ is less general in two ways. First, all fluents in $\mathcal{A}$ are truth-valued. Second, the syntax of $\mathcal{A}$ requires that φ in a proposition (3) be a literal, and that ψ be a conjunction of literals; in DL_i they are arbitrary propositional combinations of atomic formulas.

In the example from the previous section, if all fluent names are declared inertial then the propositions expressing persistence can be dropped, and we can simply write

$$\begin{array}{l} \mathit{Move}(b) \textbf{ causes } \neg(\mathit{Location}(b) \textbf{ is } l) \textbf{ if } (\mathit{Location}(b) \textbf{ is } l), \\ \textbf{impossible } \mathit{Move}(b) \textbf{ if } \mathit{Location}(b) \textbf{ is } L_1. \end{array} \tag{6}$$

6 Adding Ramification Constraints: DL_{ir}

The language DL_{ir} is the extension of DL_i that has a new kind of proposition:

$$\textbf{always } \varphi \tag{7}$$

where φ is a formula. Propositions (3) will be called *dynamic causal laws*, and propositions (7) will be called *ramification constraints* ("static" causal laws and "qualification" constraints will be introduced later).

In the transition system $\langle S, V, R\rangle$ described by a system description D in DL_{ir}, S is the set of all valuations that satisfy φ for every constraint (7) in D; V is defined by (1), and R is defined in terms of $D_{A,s}$ and *New* as before.

Adding ramification constraints to action languages was proposed in [Kartha and Lifschitz, 1994]. The semantics of these propositions follows [Winslett, 1988]. McCain and Turner [1995] convincingly argue that many examples usually expressed in terms of ramification constraints can be better described by more general "static causal laws." Still, there are cases when ramification constraints are useful.

First, they can represent "sort information." For any fluent name F and any values $W_1, \dots, W_n$ $(n > 0)$, let

$$F \textbf{ in } W_1, \dots, W_n$$

stand for the constraint

$$\textbf{always } (F \textbf{ is } W_1) \vee \cdots \vee (F \textbf{ is } W_n).$$

To see how such propositions are used, let us go back to the example discussed above, and imagine that we want to enhance it by including some truth-valued fluents. Then f and t will need to be added to the set of values of the underlying action signature, and we will need to postulate that these new values are not among the possible locations of blocks:

$$\mathit{Location}(b) \textbf{ in } L_1, \dots, L_n.$$

Similarly, for each of the new fluent names F, we will need to postulate

$$F \textbf{ in } \mathsf{f}, \mathsf{t}. \tag{8}$$

If a system description includes a proposition of this form, it is convenient to write F for the atomic formula (F **is** t).

Another use of ramification constraints is for representing explicit definitions (see [Kartha and Lifschitz, 1994], Section 5.2). To introduce a fluent name F as an abbreviation for a formula φ, we declare F noninertial and include propositions (8) and

$$\textbf{always } F \equiv \varphi.$$

7 Adding Qualification Constraints: DL_{irq}

Consider now what happens if we further extend DL_{ir} by adding *qualification constraints*

$$\textbf{never } \varphi \tag{9}$$

where φ is a formula. The semantics of the new language DL_{irq} is defined as follows. The set S of states of the transition system described by a system description D in this language is the set of all valuations that satisfy φ for every ramification constraint (7) in D, and satisfy $\neg\varphi$ for every qualification constraint (9) in D. As before, the values of fluents are defined by (1). To define R, consider the system description obtained from D by dropping all qualification constraints. This description D^- is in the language DL_{ir}, so that the corresponding function R^- is already defined. We define R as the result of restricting the arguments and values of R^- to set S:

$$R(A, s) = R^-(A, s) \cap S \qquad (s \in S).$$

The distinction between ramification and qualification constraints was introduced by Ginsberg and Smith [1988] and by Lin and Reiter [1994]. An action language that includes constraints of both kinds was outlined in an unpublished draft by Arcot Rajasekar.

To see how qualification constraints are used, consider the enhancement of the example discussed above in which L_2 is a box of limited capacity: it has enough room for at most k objects. The corresponding system description is obtained from (6) by adding the constraints

$$\textbf{never} \bigwedge_{b \in X} (\mathit{Location}(b) \textbf{ is } L_2)$$

for all subsets X of $\{B_1, \ldots, B_m\}$ of cardinality $k+1$. These constraints make it impossible to move an object into the box when the box is full.

8 A Language with Static Causal Laws: DL_{is}

A dynamic causal law (3) expresses "event causality": it tells us which changes are caused by performing an action. The language DL_{is} is the extension of DL_i that includes, as the second kind of proposition, *static causal laws* of the form

$$\varphi \textbf{ causes}_s\ \psi \tag{10}$$

where φ and ψ are formulas. Such a proposition expresses "fact causality"—a causal relationship between two facts obtaining in the same state. For instance, being broken causes a watch to be of no use for determining the time of the day:

$$\mathit{Broken} \textbf{ causes}_s \neg \mathit{Useful}.$$

From this proposition we conclude that any action which causes *Broken* to become true has also an indirect effect—it makes *Useful* false.

In the language DL_{is}, constraints (7) and (9) can be viewed as abbreviations. A ramification constraint (7) stands for

$$\alpha \vee \neg\alpha \textbf{ causes}_s\ \varphi. \tag{11}$$

A qualification constraint (9) stands for

$$\varphi \textbf{ causes}_s\ \alpha \wedge \neg\alpha. \tag{12}$$

In the next section we show that this translation faithfully represents the semantics of the language DL_{irq} (Proposition 1).

The idea of a static causal law, in relation to the ramification problem, was developed in [Geffner, 1990], [Elkan, 1992], [Brewka and Hertzberg, 1993], [Baral, 1995], [Lin, 1995], [McCain and Turner, 1995], [Thielscher, 1995] and [Turner, 1997]. More recent research was described in the talks given by some of these authors at the workshop on causality held at the University of Texas at Austin in May of 1996. McCain and Turner discussed, in particular, the difference between the "foundationalist" theory from [McCain and Turner, 1995] and alternative "nonfoundationalist" approaches.

The papers [Baral, 1995] and [Turner, 1997] incorporate fact causality in action languages. They are restricted, however, to the case when all fluents are truth-valued, and they prohibit or restrict the use of noninertial fluent names. Our goal here is to define a "nonfoundationalist" semantics of static causal laws in the more general context of the language DL_{is}.

The transition system $\langle S, V, R\rangle$ described by a system description D in the language DL_{is} is defined as follows. The states of the system are the valuations v such that, for every static causal law (10) in D, v satisfies ψ whenever v satisfies φ. As usual, V is defined by (1). To describe R, we need the following definitions and notation.

A *literal* is an atomic formula or the negation of an atomic formula:

$$(F \textbf{ is } W) \text{ or } \neg(F \textbf{ is } W).$$

A literal is *inertial* if the fluent name F occurring in it is inertial. For any valuation v, by $\widehat{v}$ we denote the set of inertial literals that are satisfied by v. It is clear that the set $New(v_1, v_2)$ can be characterized as the set of positive literals in $\widehat{v_2} \setminus \widehat{v_1}$. Note that, for every valuation v, all negative literals in $\widehat{v}$ are entailed by the set of positive literals in $\widehat{v}$.

For any state s, by D_s we denote the set of formulas ψ such that, for some static causal law (10) in D, s satisfies φ. This notation is similar to the symbol $D_{A,s}$ introduced in Sec. 4 in connection with dynamic causal laws.

Finally, we say that a formula φ is *entailed* by a set Γ of formulas if every valuation satisfying Γ satisfies φ also.

We define $R(A, s)$ to be the set of all states s^* such that

(i) s^* satisfies $D_{A,s}$, and

(ii) every formula in $\widehat{s^*} \setminus \widehat{s}$ is entailed by $D_{A,s} \cup D_{s^*} \cup (\widehat{s} \cap \widehat{s^*})$.

Condition (i), familiar from the semantics of the basic action description language DL, expresses that the resulting state satisfies the applicable dynamic causal laws. Condition (ii) requires that any violation of inertia—any element of $\widehat{s^*} \setminus \widehat{s}$—be "justified" on the basis of

- the dynamic causal laws of D: $D_{A,s}$,
- the static causal laws of D: D_{s^*},
- the facts about the resulting state that comply with inertia: $\widehat{s} \cap \widehat{s^*}$.

Condition (ii) can be equivalently replaced by

(ii′) every formula in $\widehat{s^*}$ is entailed by $D_{A,s} \cup D_{s^*} \cup (\widehat{s} \cap \widehat{s^*})$, because

$$\widehat{s^*} = (\widehat{s^*} \setminus \widehat{s}) \cup (\widehat{s} \cap \widehat{s^*}).$$

If all fluent names in the underlying action signature are inertial then the definition of R can be stated as a fixpoint condition (see Proposition 2 below). This condition is a nonfoundationalist counterpart of Definition 4 from [McCain and Turner, 1995], along the lines of more recent unpublished work by these authors.

9 Two Theorems about DL_{is}

The following fact is similar to Proposition 3 from [McCain and Turner, 1995]:

Proposition 1. *The semantics of the language DL_{is}, restricted to system descriptions in the language DL_{irq}, is equivalent to the semantics of DL_{irq}.*

Lemma 1. *For any valuations v, v_1, v_2, if*

$$\widehat{v} \cap \widehat{v_1} = \widehat{v} \cap \widehat{v_2}$$

then $\widehat{v_1} = \widehat{v_2}$.

Proof. Assume that $\widehat{v_1} \neq \widehat{v_2}$, and take an inertial fluent name F such that $v_1(F) \neq v_2(F)$. Then $v(F)$ is different from at least one of the values $v_1(F)$, $v_2(F)$. Assume, for instance, that $v(F) \neq v_1(F)$. Then the literal $\neg(F$ **is** $v_1(F))$ belongs to $\widehat{v} \cap \widehat{v_2}$ but does not belong to $\widehat{v} \cap \widehat{v_1}$.

Lemma 2. *For any valuations v, v_1, v_2,*

$$New(v, v_1) \subseteq New(v, v_2)$$

iff

$$\widehat{v} \cap \widehat{v_2} \subseteq \widehat{v} \cap \widehat{v_1}.$$

Proof. We will prove the equivalence of the negations of the two inclusions. Assume that the formula F **is** W belongs to $New(v, v_1)$ but not to $New(v, v_2)$. Then $v_1(F) = W$, $v(F) \neq W$ and $v_2(F) \neq W$. It follows that $\neg(F$ **is** $v_1(F))$ belongs to $\widehat{v} \cap \widehat{v_2}$ but not to $\widehat{v} \cap \widehat{v_1}$. Now assume that F **is** W belongs to $\widehat{v} \cap \widehat{v_2}$ but not to $\widehat{v} \cap \widehat{v_1}$. Then $v(F) = v_2(F) = W$ and $v_1(F) \neq W$. It follows that F **is** $v_1(F)$ belongs to $New(v, v_1)$ but not to $New(v, v_2)$. Finally, assume that $\neg(F$ **is** $W)$ belongs to $\widehat{v} \cap \widehat{v_2}$ but not to $\widehat{v} \cap \widehat{v_1}$. Then $v(F) \neq W$, $v_2(F) \neq W$ and $v_1(F) = W$, and we can again conclude that F **is** $v_1(F)$ belongs to $New(v, v_1)$ but not to $New(v, v_2)$.

The following lemma is similar to Proposition 4 from [McCain and Turner, 1995]:

Lemma 3. *Let D be a system description in the language DL_{is}, and let $\langle S, V, R \rangle$ be the system described by D. Let Q be a set of qualification constraints, and let $\langle S', V', R' \rangle$ be the system described by $D \cup Q$. For any action name A and any s in S',*

$$R'(A, s) = R(A, s) \cap S'.$$

Proof. Take any $s, s^* \in S'$. We need to check that $s^* \in R'(A, s)$ iff $s^* \in R(A, s)$. Since Q consists of static causal laws, $(D \cup Q)_{A,s} = D_{A,s}$. Since s^* belongs to S', it does not satisfy any of the formulas φ from constraints (12) in Q, so that $Q_{s^*} = \emptyset$. Consequently

$$(D \cup Q)_{A,s} \cup (D \cup Q)_{s^*} \cup (\widehat{s} \cap \widehat{s^*}) = D_{A,s} \cup D_{s^*} \cup (\widehat{s} \cap \widehat{s^*}).$$

Proof of Proposition 1. Lemma 3 shows that it is sufficient to prove the assertion of Proposition 2 for system descriptions in the language DL_{ir}. Let D be a system description in this language, let A be an action name, and let s, s^* be states such that $s^* \in D_{A,s}$. We need to verify that condition (ii$'$) is equivalent to the minimality of $New(s, s^*)$. Lemmas 1 and 2 show that the minimality of $New(s, s^*)$ can be equivalently stated as follows:

(iii) for every state s' that satisfies $D_{A,s}$, if $\widehat{s} \cap \widehat{s^*} \subseteq \widehat{s} \cap \widehat{s'}$ then $\widehat{s^*} = \widehat{s'}$.

Assume (ii$'$), and consider a state s' satisfying $D_{A,s}$ such that $\widehat{s} \cap \widehat{s^*} \subseteq \widehat{s} \cap \widehat{s'}$. Since each static causal law in DL_{ir} has the form (11) and s' is a state, s' satisfies D_{s^*}. It follows that s' satisfies $D_{A,s} \cup D_{s^*} \cup (\widehat{s} \cap \widehat{s^*})$. Then, by (ii$'$), s' satisfies $\widehat{s^*}$. Consequently, for every inertial fluent name F, $s'(F) = s^*(F)$, which means that $\widehat{s^*} = \widehat{s'}$.

Now assume (iii). We need to show that all formulas in $\widehat{s^*}$ are entailed by $D_{A,s} \cup D_{s^*} \cup (\widehat{s} \cap \widehat{s^*})$. Take any valuation v that satisfies these formulas. Since v satisfies D_{s^*}, it is a state. From $\widehat{s} \cap \widehat{s^*} \subseteq \widehat{v}$ we conclude that $\widehat{s} \cap \widehat{s^*} \subseteq \widehat{s} \cap \widehat{v}$. Then, by (iii), $\widehat{s^*} = \widehat{v}$. Thus $v(F) = s^*(F)$ for every inertial fluent name F, so that v satisfies all formulas in $\widehat{s^*}$.

Proposition 2. *Let D be a system description in the language DL_{is} without noninertial fluent names. For any action name A and any states s, s^* of D, s^* belongs to $R(A, s)$ iff $\widehat{s^*}$ equals the set of literals that are entailed by $D_{A,s} \cup D_{s^*} \cup (\widehat{s} \cap \widehat{s^*})$.*

Proof. Denote the set of formulas entailed by $D_{A,s} \cup D_{s^*} \cup (\widehat{s} \cap \widehat{s^*})$ by Γ. Assume that s^* belongs to $R(A, s)$. By (ii$'$), $\widehat{s^*}$ is a subset of the set of literals in Γ. To prove the inclusion in the opposite direction, consider a literal $\varphi \in \Gamma$. Since s^* satisfies $D_{A,s} \cup D_{s^*} \cup (\widehat{s} \cap \widehat{s^*})$, it satisfies φ also. It follows that $\varphi \in \widehat{s^*}$. Assume now that $\widehat{s^*}$ is the set of all literals in Γ. Then Γ is a proper subset of the set of formulas. Consequently, there exists a valuation v that satisfies all formulas in Γ. In particular, v satisfies both $D_{A,s}$ and $\widehat{s^*}$. The last condition implies that $v = s^*$, so that s^* satisfies $D_{A,s}$.

10 Action Query Languages

The syntax of an *action query language* of an action signature σ is defined by two classes of syntactic expressions: *axioms* and *queries*. The semantics of the language is defined by specifying, for every transition system T of the signature σ, every set Γ of axioms, and every query Q, whether Q is a *consequence* of Γ in T; in symbols, whether $\Gamma \models_T Q$.

We will define how an action description language and an action query language of the same signature can be combined in a single language. First, consider two examples.

11 A Query Language for Projection

In a "temporal projection problem," axioms are observations about the current state of the world, and a query is an assertion about executing a sequence of actions. When some of the actions involved are nondeterministic, the resulting state is not completely determined by the initial state, and we are interested in what will be *necessarily* true after executing the actions.

Consider an action signature σ. In the corresponding action query language QL^p, axioms are expressions of the form

$$\textbf{now}\ \varphi \tag{13}$$

where φ is a formula. Queries are expressions of the form

$$\textbf{necessarily}\ \varphi\ \textbf{after}\ \overline{A} \tag{14}$$

where $\overline{A}$ is a string of action names, and φ is a formula. Notationally, it is sometimes convenient to separate the members of $\overline{A}$ by semicolons. To define the semantics of QL^p, we need the following auxiliary definition. A *history* of a transition system $\langle S, V, R\rangle$ is a path in the corresponding graph, that is, a sequence

$$s_0, A_1, s_1, \ldots, A_n, s_n \tag{15}$$

$(n \geq 0)$ such that

$$s_i \in R(A_i, s_{i-1}) \qquad (1 \leq i \leq n).$$

The consequence relation of the language QL^p is defined as follows: A query

$$\textbf{necessarily}\ \psi\ \textbf{after}\ A_1 \ldots A_n$$

is a consequence of Γ in a transition system T if, for every history of T of the form $s_0, A_1, s_1, \ldots, A_n, s_n$ such that s_0 satisfies φ for all axioms (13) in Γ, s_n satisfies ψ.

Consider, for instance, the axioms

$$\textbf{now}\ \mathit{Location}(B_1)\ \textbf{is}\ L_5,$$
$$\textbf{now}\ \mathit{Location}(B_2)\ \textbf{is}\ L_5.$$

The query

$$\textbf{necessarily}\ \neg(\mathit{Location}(B_1)\ \textbf{is}\ L_5)\ \textbf{after}\ \mathit{Move}(B_1); \mathit{Move}(B_2)$$

is a consequence of these axioms in the transition system T described in the language DL_i by propositions (6).

We have seen that the same system T can be described in a different way, in the basic language DL. This fact illustrates the independence of two issues—deciding which language to use for *describing* a transition system, and deciding which language to use to talk about its *properties*.

12 A Query Language for Actual Events

In the language QL^a, both axioms and queries have the form

$$\textbf{actually}\ \varphi\ \textbf{after}\ \overline{A}. \tag{16}$$

The intuitive meaning of this expression is that the actions $\overline{A}$ have been sequentially executed at some initial instant of time, and in the resulting state φ was true.[4] Thus a pair of axioms (16) whose action strings $\overline{A}$ diverge (in the sense that neither is a prefix of the other one) would make the axiom set inconsistent.

If the string $\overline{A}$ is empty, (16) can be written as

$$\textbf{initially } \varphi.$$

About an expression (16) we say that it *holds* in a history (15) if $\overline{A}$ is a substring $A_1 \ldots A_k$ of $A_1 \ldots A_n$ for some k $(0 \leq k \leq n)$, and s_k satisfies φ. The consequence relation of the language QL^a is defined as follows: A query Q is a consequence of a set Γ of axioms if, for every history h such that all axioms in Γ hold in h, Q holds in h also.

For instance,

$$\begin{array}{l} \textbf{initially } \neg(\mathit{Location}(B_1) \textbf{ is } L_1), \\ \textbf{initially } \neg(\mathit{Location}(B_1) \textbf{ is } L_5) \end{array} \tag{17}$$

are consequences of

$$\textbf{actually } \mathit{Location}(B_1) \textbf{ is } L_5 \textbf{ after } \mathit{Move}(B_1)$$

in the system described by (6).

13 Combining a Description Language with a Query Language

Let L_1 be an action description language, and let L_2 be an action query language of the same signature such that propositions of L_1 are different from axioms of L_2. A *domain description* in the combined language $L_1 + L_2$ is a set $D \cup \Gamma$ consisting of propositions of L_1 and axioms of L_2. A query Q of L_2 is a *consequence* of $D \cup \Gamma$ if it is a consequence of Γ in the transition system described by D.

This construction can be applied, in particular, to any of the description languages and query languages defined above. These combinations

$$\begin{array}{l} DL + QL^p,\ DL_i + QL^p,\ DL_{ir} + QL^p,\ \ldots, \\ DL + QL^a,\ DL_i + QL^a,\ DL_{ir} + QL^a,\ \ldots \end{array}$$

can be conveniently denoted by L^p,L^p_i,L^p_{ir},...,L^a,L^a_i, L^a_{ir},....

For instance, the expressions

$$\begin{array}{l} \mathit{Move}(b) \textbf{ causes } \neg(\mathit{Location}(b) \textbf{ is } l) \textbf{ if } (\mathit{Location}(b) \textbf{ is } l), \\ \textbf{impossible } \mathit{Move}(b) \textbf{ if } \mathit{Location}(b) \textbf{ is } L_1, \\ \textbf{actually } \mathit{Location}(B_1) \textbf{ is } L_5 \textbf{ after } \mathit{Move}(B_1) \end{array}$$

form a domain description in the language L^a_i. Queries (17) are consequences of this domain description.

These examples illustrate the "modular" approach to the design of action languages—combining an action description language with an action query language described independently.

[4] This is similar to the meaning of "true-at" propositions from [Baral *et al.*, 1997].

14 Conclusions

We have defined, for a given action signature, the general concepts of an action description language and of an action query language, and gave examples of languages of both kinds. An action description language serves for describing transition systems; in an action query language one can express properties of paths in a given transition system. The most expressive action description language introduced here includes static causal laws, which subsume both ramification constraints and qualification constraints. One of the two examples of action query languages is designed for temporal projection, and the other is for reasoning about past events.

A description language and a query language for the same action signature can be combined into a single action language. Action languages of this kind are suitable for formalizing commonsense facts about scenarios involving actions.

Acknowledgements

I am grateful to Charles Elkan, Michael Gelfond, Enrico Giunchiglia, Neelakantan Kartha, Norman McCain, Hudson Turner and to the anonymous referees for their comments and constructive criticisms. This work was partially supported by National Science Foundation under grant IRI-9306751.

References

[Apt and Bezem, 1990] Krzysztof Apt and Marc Bezem. Acyclic programs. In David Warren and Peter Szeredi, editors, *Logic Programming: Proc. of the Seventh Int'l Conf.*, pages 617–633, 1990.

[Baker, 1989] Andrew Baker. A simple solution to the Yale Shooting Problem. In Ronald Brachman, Hector Levesque, and Raymond Reiter, editors, *Proc. of the First Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 11–20, 1989.

[Baral *et al.*, 1997] Chitta Baral, Michael Gelfond, and Alessandro Provetti. Reasoning about actions: laws, observations and hypotheses. *Journal of Logic Programming*, 1997. To appear.

[Baral, 1995] Chitta Baral. Reasoning about actions: Non-deterministic effects, constraints, and qualifications. In *Proc. of IJCAI-95*, pages 2017–2023, 1995.

[Boutilier and Friedman, 1995] Craig Boutilier and Nir Friedman. Nondeterministic actions and the frame problem. In *Working Notes of the Symposium on Extending Theories of Actions*, 1995.

[Brewka and Hertzberg, 1993] Gerhard Brewka and Joachim Hertzberg. How to do things with worlds: On formalizing actions and plans. *Journal of Logic and Computation*, 3(5), 1993.

[Elkan, 1992] Charles Elkan. Reasoning about action in first-order logic. In *Proc. of the 1992 Canadian Conf. on Artificial Intelligence*, 1992.

[Eshghi and Kowalski, 1989] Kave Eshghi and Robert Kowalski. Abduction compared with negation as failure. In Giorgio Levi and Maurizio Martelli, editors, *Logic Programming: Proc. of the Sixth Int'l Conf.*, pages 234–255, 1989.

[Evans, 1989] Chris Evans. Negation-as-failure as an approach to the Hanks and McDermott problem. In *Proc. of the Second Int'l Symp. on Artificial Intelligence*, 1989.

[Geffner, 1990] Hector Geffner. Causal theories for nonmonotonic reasoning. In *Proc. AAAI-90*, pages 524–530, 1990.

[Gelfond and Lifschitz, 1993] Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.

[Gelfond, 1989] Michael Gelfond. Autoepistemic logic and formalization of commonsense reasoning. In Michael Reinfrank, Johan de Kleer, Matthew Ginsberg, and Erik Sandewall, editors, *Non-Monotonic Reasoning: 2nd International Workshop (Lecture Notes in Artificial Intelligence 346)*, pages 176–186. Springer-Verlag, 1989.

[Genesereth and Nilsson, 1987] Michael R. Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA, 1987.

[Ginsberg and Smith, 1988] Matthew Ginsberg and David Smith. Reasoning about action II: the qualification problem. *Artificial Intelligence*, 35:311–342, 1988.

[Giunchiglia and Lifschitz, 1995] Enrico Giunchiglia and Vladimir Lifschitz. Dependent fluents. In *Proc. IJCAI-95*, pages 1964–1969, 1995.

[Giunchiglia *et al.*, 1995] Enrico Giunchiglia, G. Neelakantan Kartha, and Vladimir Lifschitz. Actions with indirect effects (extended abstract). In *Working Notes of the Symposium on Extending Theories of Actions*, 1995.

[Haugh, 1987] Brian Haugh. Simple causal minimizations for temporal persistence and projection. In *Proc. AAAI-87*, pages 218–223, 1987.

[Kartha and Lifschitz, 1994] G. Neelakantan Kartha and Vladimir Lifschitz. Actions with indirect effects (preliminary report). In *Proc. of the Fourth Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 341–350, 1994.

[Lifschitz, 1987] Vladimir Lifschitz. Formal theories of action (preliminary report). In *Proc. of IJCAI-87*, pages 966–972, 1987.

[Lin and Reiter, 1994] Fangzhen Lin and Raymond Reiter. State constraints revisited. *Journal of Logic and Computation*, 4:655–678, 1994.

[Lin and Shoham, 1991] Fangzhen Lin and Yoav Shoham. Provably correct theories of action (preliminary report). In *Proc. AAAI-91*, pages 349–354, 1991.

[Lin, 1995] Fangzhen Lin. Embracing causality in specifying the indirect effects of actions. In *Proc. of IJCAI-95*, pages 1985–1991, 1995.

[McCain and Turner, 1995] Norman McCain and Hudson Turner. A causal theory of ramifications and qualifications. In *Proc. of IJCAI-95*, pages 1978–1984, 1995.

[McCarthy and Hayes, 1969] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, 1969. Reproduced in [McCarthy, 1990].

[McCarthy, 1990] John McCarthy. *Formalizing common sense: papers by John McCarthy.* Ablex, Norwood, NJ, 1990.

[Morris, 1988] Paul Morris. The anomalous extension problem in default reasoning. *Artificial Intelligence*, 35(3):383–399, 1988.

[Pinto and Reiter, 1993] Javier Pinto and Raymond Reiter. Temporal reasoning in logic programming: a case for the situation calculus. In *Proc. of ICLP-93*, pages 203–221, 1993.

[Reiter, 1991] Raymond Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1991.

[Sandewall, 1995] Erik Sandewall. *Features and fluents*, volume 1. Oxford University Press, 1995.

[Schubert, 1990] Lenhart Schubert. Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In H.E. Kyburg, R. Loui, and G. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer, 1990.

[Thielscher, 1995] Michael Thielscher. Computing ramifications by postprocessing. In *Proc. of IJCAI-95*, pages 1994–2000, 1995.

[Turner, 1997] Hudson Turner. Representing actions in logic programs and default theories: a situation calculus approach. *Journal of Logic Programming*, 1997. To appear.

[Winslett, 1988] Marianne Winslett. Reasoning about action using a possible models approach. In *Proc. AAAI-88*, pages 89–93, 1988.