Modular Basic Action Theories

Yilan Gu
Dept. of Computer Science
University of Toronto
10 King's College Road
Toronto, ON, M5S 3G4, Canada
Email: yilan@cs.toronto.edu

Mikhail Soutchanski
Department of Computer Science
Ryerson University
245 Church Street, ENG281
Toronto, ON, M5B 2K3, Canada
Email: mes@scs.ryerson.ca

Abstract

In this paper we design a representation that allows writing more compact and modular basic action theories, than it is currently possible. Moreover, such representation also provides formal foundations for reasoning about actions in OpenCyc by using Reiter's basic action theory formalism.

Keywords: common sense knowledge, reasoning about actions, situation calculus

Introduction

The situation calculus (SC) is a well known and popular logical theory for reasoning about events and actions. There are several different formulations of SC. According to John McCarthy the history is the following: "(McCarthy 1959) proposed mathematical logic as a tool for representing facts about the consequences of actions and using logical reasoning to plan sequences of actions that would achieve goals. Situation calculus as a formalism was proposed in (McCarthy 1963) and elaborated in (McCarthy & Hayes 1969). The name situation calculus was first used in (McCarthy & Hayes 1969) but wasn't defined there. (McCarthy 1986) proposed to solve the frame and qualification problems by circumscription, but the proposed solution to the frame problem was incorrect. (Shanahan 1997) and (Reiter 2001) describe several situation calculus formalisms and give references." (See the footnote 4 in (McCarthy 2002).) In this paper we would like to concentrate on *basic action theories* (BAT) introduced in (Reiter 2001), in particular, on precondition and successor state axioms proposed by Reiter to solve (sometimes) the frame problem. The motivation for our paper is twofold. Our first motivation comes from the intention to design a representation that allows writing more compact and modular BAT, than it is currently possible, and also BAT that will be easier to elaborate and expand to new domains. BAT are logical theories of a certain syntactic form that have several desirable theoretical properties. For example, two important problems, the task of reasoning whether a given sequence of actions can be executed (the executability problem) and the task of determining whether a certain SC formula is true in the situation resulting from performing an executable sequence of actions (the projection problem), can be conveniently formulated using BAT and reduced using regression to the first-order logic (FOL) entailment problem in the theory of the initial situation (together with unique name axioms). However, BAT have not been designed to support other forms of reasoning, e.g., taxonomic reasoning about objects and actions. Essentially, these theories are "flat" and do not provide representation for hierarchies of actions or objects. This can lead to potential difficulties if one intends to use BAT for the purposes of large scale formalization of reasoning about actions on the commonsense level, when potentially arbitrary actions and objects have to be represented. To the best of our knowledge, all BAT constructed so far are relatively small, include less than 100 axioms, and it is not clear how the task of scaling up the axiomatizations using BAT can be undertaken. Intuitively, many events and actions have different degrees of generality: the action of driving a car from home to an office is specialization of the action of transportation using a vehicle, that is in its turn specialization of the action of moving a thing from one location to another. Our intention in this paper is to represent hierarchies between actions and objects explicitly and use them in new hierarchical BAT. However, we would like to show that our new hierarchical BAT can be potentially expanded into "flat" Reiter's BAT and consequently, they are conservative extensions of Reiter's BAT: every entailment from Reiter's BAT remains an entailment in hierarchical BAT. Our second motivation comes from experience with OpenCyc (Matuszek *et al.* 2006; Cycorp 2002): the open source version of the Cyc technology, the world's largest general knowledge base and commonsense reasoning engine. OpenCyc has extensive taxonomies for things, and elaborated, but unsystematic taxonomies for events and actions. In addition, (Parmar 2001) observes that `Cyc` does not follow the tradition of representing precondition and effects of actions using SC. In particular, there are some predicates in `Cyc` to represent preconditions (e.g., `preconditionFor-Props`) and effects of actions (e.g., `effectOfAction-Props` and `effectOfActionIf-Props`), there are also a few planning-related predicates (e.g., `methodForAction`, `sufficientFor-Props`, `planForTask`) but there is no explicit mechanism for solving the frame problem, for solving executability and projection problems using regression, and, to the best of our knowledge, there is no formal specification for implementing reasoning about actions in OpenCyc. The second goal of this paper is to provide formal semantics for reasoning about actions in OpenCyc by using Reiter's BAT.

The rest of this paper is structured as follows. First, we review background (SC and representation of events and actions in OpenCyc). Then we propose a new representation that helps to design modular BAT. We prove that this new representation has the same desirable logical properties as Reiter's BAT. Then, we briefly discuss related work and future research.

The Situation Calculus

All dialects of the SC $\mathcal{L}_{sc}$ include three disjoint sorts (*actions*, *situations* and *objects*). **Actions** are FO terms consisting of an action function symbol and its arguments. Actions change the world. **Situations** are FO terms which denote

possible world histories. A distinguished constant S_0 is used to denote the *initial situation*, and function $do(a, s)$ denotes the situation that results from performing action a in situation s. Every situation corresponds uniquely to a sequence of actions. Moreover, notation $s' \preceq s$ means that either situation s' is a subsequence of situation s or $s = s'$.[1] **Objects** are FO terms other than actions and situations that depend on the domain of application. **Fluents** are relations or functions whose values may vary from one situation to the next. Normally, a fluent is denoted by a predicate or function symbol whose last argument has the sort situation. For example, $F(\vec{x}, do([\alpha_1, \cdots, \alpha_n], S_0))$ represents a relational fluent in the situation $do(\alpha_n, do(\cdots, do(\alpha_1, S_0) \cdots))$ resulting from execution of ground action terms $\alpha_1, \cdots, \alpha_n$ in S_0. We do not consider functional fluents in this paper.

The SC includes the distinguished predicate $Poss(a, s)$ to characterize actions a that are possible to execute in s. For any first order SC formula ϕ and a term s of sort situation, we say ϕ is a formula *uniform* in s iff it does not mention the predicates $Poss$ or $\prec$, it does not quantify over situations, it does not mention equality on situations, and whenever it mentions a term of sort situation in the situation argument position of a fluent, then that term is s (see (Reiter 2001)).

A *basic action theory* (BAT) $\mathcal{D}$ in the SC is a set of axioms written in $\mathcal{L}_{sc}$ with the following five classes of axioms to model actions and their effects (Reiter 2001). **Action precondition axioms $\mathcal{D}_{ap}$:** For each action function $A(\vec{x})$, there is one axiom of the form $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$. $\Pi_A(\vec{x}, s)$ is a formula uniform in s with free variables among $\vec{x}$ and s, which characterizes the preconditions of action A. **Successor state axioms $\mathcal{D}_{ss}$:** For each relational fluent $F(\vec{x}, s)$, there is one axiom of the form $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$, where $\Phi_F(\vec{x}, a, s)$ is a formula uniform in s with free variables among $\vec{x}, a$ and s. The successor state axiom (SSA) for fluent F completely characterizes the value of fluent F in the next situation $do(a, s)$ in terms of the current situation s. **Initial theory $\mathcal{D}_{S_0}$:** It is a set of FO formulas whose only situation term is S_0. It specifies the values of all fluents in the initial state. It also describes all the facts that are not changeable by any actions in the domain. **Unique name axioms for actions $\mathcal{D}_{una}$:** Includes axioms specifying that two actions are different if their action names are different, and identical actions have identical arguments. **Foundational axioms for situations Σ:** The axioms for situations which characterize the basic properties of situations. These axioms are domain independent. They are included in the axiomatization of any dynamic system in the SC (see (Reiter 2001) for details).

Suppose that $\mathcal{D} = \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \Sigma$ is a BAT, $\alpha_1, \cdots, \alpha_n$ is a sequence of ground action terms, and $G(s)$ is a uniform formula with one free variable s. One of the most important reasoning tasks in the SC is the projection problem, that is, to determine whether $\mathcal{D} \models G(do([\alpha_1, \cdots, \alpha_n], S_0))$. Another basic reasoning task is the executability problem. Let $executable(do([\alpha_1, \cdots, \alpha_n], S_0))$ be an abbreviation of the formula $Poss(\alpha_1, S_0) \wedge \bigwedge_{i=2}^{n} Poss(\alpha_i, do([\alpha_1, \cdots, \alpha_{i-1}], S_0))$. Then, the executability problem is to determine whether $\mathcal{D} \models executable(do([\alpha_1, \cdots, \alpha_n], S_0))$. Planning and high-level program execution are two important settings where the executability and projection problems arise naturally. *Regression* is a central computational mechanism that forms the basis for automated solution to the executability and projection tasks in the SC (Reiter 2001). A recursive definition of the regression operator $\mathcal{R}$ on any *regressable formula* ϕ is given in (Reiter 2001); we use notation $\mathcal{R}[\phi]$ to denote the formula that results from eliminating $Poss$ atoms in favor of their definitions as given by action precondition axioms and replacing fluent atoms about $do(\alpha, s)$ by logically equivalent expressions about s as given by SSAs repeatedly until it cannot make such replacement any further. The regression theorem (Reiter 2001) shows that one can reduce the evaluation of a regressable formula W to a FO theorem proving task in the initial theory together with unique names axioms for actions:

$$\mathcal{D} \models W \text{ iff } \mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models \mathcal{R}[W].$$

This fact is the key result in SC. It demonstrates that an executability or a projection task can be reduced to a theorem proving task that does not use precondition, successor state, and foundational axioms. This is one of the reasons why the SC provides a natural and easy way to representation and reasoning about dynamic systems.

Open Cyc

In Cyc, there is the collection `Action` that is a subclass of `Event`. Actions are events that are required to have doers. Cyc uses around 37,000 different event types to describe what happens in the world. Many events in Cyc are represented using Davidsonian representations (Davidson 1967): events are described by the assertion that there exists a particular event of a given type, by the roles that other individual things play in that event, and by the fillers of those roles. `ActorSlot` in CYC is a collection of binary predicates to express relationships between an event and the participants in it. They are also known in other systems as participants roles, deep cases or thematic relations. Actor slots are arranged in a hierarchy.

Modular BAT

The SC and BAT are able to specify the evolution of dynamic systems in a very natural way. However, in practice it is not easy to specify a dynamic system with very large number of actions. If a system involves hundreds or even thousands of actions, it will be difficult to specify BAT.

To deal with this problem, we propose to classify events and actions hierarchically similar to the representations in Cyc. In the following, each action function symbol $A(x_1, \cdots, x_n)$ is used to represent an *event* that could possibly change states of a dynamic system. For example, action $move(o, l_1, l_2)$ represents the event that object o was moved from l_1 to l_2. We introduce the following predicates.

Notation 1 (a) *The predicate $argType(a, x, t)$ represents that the argument x of action a has type t, where types are object variables or constants. For any types t_1 and t_2,*

$$t_1 \sqsubseteq t_2 \stackrel{def}{=} (\forall x, a).\ argType(a, x, t_2) \supset argType(a, x, t_1).$$

[1] Reiter (Reiter 2001) uses the notation $s' \sqsubseteq s$, but we use $s' \preceq s$ to avoid confusion with the inclusion relation $\sqsubset$.

(b) *The predicate* $actorSlot(slotName, a, y)$ *represents that argument* y *in action* a *has a relation* $slotName$ *with action* a*. For any* $actorSlot$ *names* l_1 *and* l_2*,*

$$l_1 \sqsubseteq l_2 \stackrel{def}{=} (\forall a, x).\ actorSlot(l_2, a, x) \supset actorSlot(l_1, a, x).$$

Intuitively, we can say that t_1 (l_1, respectively) is a specialization of t_2 (l_2, respectively). The class of axioms $\mathcal{T}$ includes the set of sentences $T_1 \sqsubseteq T_2$ and $L_1 \sqsubseteq L_2$ that characterize taxonomy of argument types and actor slots.

For each action function $A(\vec{x})$, every argument x_i has a certain type T_i and has a certain relationship L_i with the action that is expressed using $actorSlot$:

$$\bigwedge_{i=1}^{m}(actorSlot(L_i, A(\vec{x}), x_i) \wedge argType(A(\vec{x}), x_i, T_i)),$$

where variable vector $\vec{x} = \langle x_1, \cdots, x_m \rangle$, and the axiomatizer has to ensure that each type T_i is the most specific type (in terms of the $\sqsubseteq$ relation) that can be used to characterize x_i in action $A(\vec{x})$ and L_i is the most specific slot name that can characterize the relation between $A(\vec{x})$ and x_i. For each action function there is an axiom of this syntactic form; we denote this class of axioms as $\mathcal{A}$. Since the axioms in $\mathcal{A}$ and in $\mathcal{T}$ are static facts, that is, they are situation independent, they will be included in the initial KB $\mathcal{D}_{S_0}$. Moreover, the axioms in $\mathcal{A}, \mathcal{T}$ should entail that a specialization relation between actor slots of two actions implies a specialization relation between the corresponding types: [2]

$$\mathcal{A} \cup \mathcal{T} \models (\forall).actorSlot(l_1, a_1, x_1) \wedge actorSlot(l_2, a_2, x_2) \wedge argType(a_1, x_1, t_1) \wedge argType(a_2, x_2, t_2) \wedge l_1 \sqsubseteq l_2 \supset t_1 \sqsubseteq t_2.$$

Example 1 Consider an action $move(o, l_1, l_2)$ which represents an event of moving object o from l_1 to l_2, then

$actorSlot(ObjectMoving, move(o, l_1, l_2), o) \wedge$
$argType(move(o, l_1, l_2), o, MovableObject) \wedge$
$actorSlot(Origin, move(o, l_1, l_2), l_1) \wedge$
$argType(move(o, l_1, l_2), l_1, Location) \wedge$
$actorSlot(Destination, move(o, l_1, l_2), l_2) \wedge$
$argType(move(o, l_1, l_2), l_2, Location)$

will be included in an initial KB.

Definition 1 *An* action diagram *is defined by a finite set of axioms* $\mathcal{H}$ *in which each axiom is of the form*

$$specialization(A_1(\vec{x}), A_2(\vec{y})) \equiv \phi(\vec{x}, \vec{y}) \qquad (1)$$

for any vectors of object variables $\vec{x}$ *and* $\vec{y}$*, where* A_1, A_2 *are action function symbols, the predicate* $specialization(a_1, a_2)$ *represents that action* a_1 *is a* direct specialization *of action* a_2 *(action* a_2 *is a* direct generalization *of* a_1*), and* $\phi(\vec{x}, \vec{y})$ *is a situation-free FO formula (that can be* $\top$*, but that cannot be equivalent to* $\bot$*) with all free variables (if any) among* $\langle \vec{x}, \vec{y} \rangle$*. Also,* $\mathcal{H}$ *must be such that the following conditions hold:*

$$\mathcal{H} \cup \mathcal{T} \models specialization(a_1, a_2) \wedge actorSlot(l_1, a_1, x) \wedge actorSlot(l_2, a_2, x) \supset l_1 \sqsubseteq l_2$$

$$\mathcal{H} \cup \mathcal{D} \models specialization(a_1, a_2) \supset (\forall s.Poss(a_1, s) \supset Poss(a_2, s)),$$

where $\mathcal{D}$ *is a domain specific BAT that includes* $\mathcal{D}_{ap}$*.*

[2] All free variables are universally quantified at front.

Given any action diagram $\mathcal{H}$, we say that a directed graph $G = \langle V, E \rangle$ is a *digraph of* $\mathcal{H}$ when $V = \{A_1, \cdots, A_n\}$, where all A_i's are distinct action function symbols that occur in at least one axiom in $\mathcal{H}$ and a directed edge $A_k \rightarrow A_j$ belongs to edge set E iff there is an axiom of the form $specialization(A_j(\vec{x}), A_k(\vec{y})) \equiv \phi(\vec{x}, \vec{y})$ in $\mathcal{H}$. When the digraph G of $\mathcal{H}$ is acyclic, i.e., there is no directed loop in G, then we call $\mathcal{H}$ an *acyclic action diagram*. In this paper, we will only consider acyclic action diagrams. Note that if each action in the digraph of an acyclic action diagram has only one parent (single inheritance case), then the digraph is actually a forest, but in a more general case, there are actions that have several parents (multiple inheritance case), as shown in Example 2.

Example 2 In this example we show that actions in $\mathcal{H}$ may have different number of arguments. Consider an action $travel(p, o, d)$, representing an event that person p travels from origin o to destination d. It can be considered as a direct specialization of $move$ – person p moves from location o to location d:

$$specialization(\ travel(p, o, d), move(p, o, d)\ ).$$

Consider an action $drive(p, v, o, d)$, representing that a person p drives a vehicle v from origin o to destination d. It can be considered as a direct specialization of action $travel$ – person p travels from location o to location d:

$$specialization(\ drive(p, v, o, d), travel(p, o, d)\ ).$$

It is also a direct specialization of action $move$ – vehicle v moves from location o to location d:

$$specialization(\ drive(p, v, o, d), move(v, o, d)\ ).$$

Consider an action $enter(p, door)$, representing that person p enters a door $door$. It can be considered as a direct specialization of action $move(p, o, d)$ iff the origin o is the outside of the door and the destination d is the inside of the door, or vice versa:

$$specialization(\ enter(p, door), move(p, o, d)\ ) \equiv (\ outside(o, door) \wedge inside(d, door) \vee outside(d, door) \wedge inside(o, door)\ ),$$

where predicate $outside(o, r)$ ($inside(d, r)$, respectively) is true iff o (d, respectively) is the location that is outside (inside, respectively) of r.

Because in general case we need to consider a direct specialization of a direct *specialization* of an action, we define (distant) specializations using the predicate isA.

Definition 2 *The predicate* $isA(a_1, a_2)$ *represents that action* a_1 *is a specialization of action* a_2 *and is defined as a reflexive-transitive closure of* $specialization$*:*

$$isA(a_1, a_2) \equiv (\forall P).\{(\forall v)[P(v, v)] \wedge (\forall v, v')[specialization(v, v') \supset P(v, v')] \wedge (\forall v, v', v'')[specialization(v, v') \wedge P(v', v'') \supset P(v, v'')]\} \supset P(a_1, a_2) \qquad (2)$$

The above definition requires second-order logic, but we will show in Theorem 3 that we still can reduce reasoning about regressable formulas to theorem proving in FOL only. We denote the set of axioms including Axiom (2)and all axioms in an acyclic action diagram $\mathcal{H}$ as $\mathcal{H}^\star$ and call it the *action hierarchy (of $\mathcal{H}$).*

Definition 3 *An action hierarchy H^* is acyclic iff it entails the following conditions: $H^* \models isA(A_1(\vec{x}_1), A_2(\vec{y}_1)) \wedge isA(A_2(\vec{y}_2), A_1(\vec{x}_2)) \supset A_1(\vec{x}_3) = A_2(\vec{y}_3)$ for all action functions A_1, A_2.*

In the sequel, we consider only acyclic hierarchies H^*. Note that this condition is more general than antisymmetry of isA predicate. One can easily prove that H^* is acyclic according to Def. 3 if the digraph of the action diagram H is acyclic.

The following theorem specifies that the action hierarchies entails the same intuitively clear taxonomic properties as the predicate $specialization$.

Theorem 1 *Let $\mathcal{H}^\star$ be an acyclic action hierarchy. Then*

$$\mathcal{H}^\star \cup \mathcal{T} \models (\forall).actorSlot(l_1, a_1, x) \wedge actorSlot(l_2, a_2, x) \wedge isA(a_1, a_2) \supset l_1 \sqsubseteq l_2;$$

$$\mathcal{H}^\star \cup \mathcal{D} \models (\forall).isA(a_1, a_2) \supset (Poss(a_1, s) \supset Poss(a_2, s)).$$

Proof: Follows from Def. 1, Def. 2 and Def. 3 using induction, but proof is omitted because of lack of space.

Moreover, the following lemma will be convenient later.

Lemma 1 *Given any acyclic action hierarchy $\mathcal{H}^\star$, for any action functions $A'(\vec{x})$ and $A(\vec{y})$ with distinct free variables $\vec{x}$ and $\vec{y}$, $A'(\vec{x})$ is a specialization of $A(\vec{y})$ iff $\phi_{A',A}(\vec{x}, \vec{y})$, for some situation-free FO formula $\phi_{A',A}$ (including $\top$ and $\bot$) over object variables whose free variables are at most among $\langle \vec{x}, \vec{y} \rangle$. That is,*

$$H^* \models isA(A'(\vec{x}), A(\vec{y})) \equiv \phi_{A',A}(\vec{x}, \vec{y}).$$

Moreover, such ϕ_{A_1,A_2} can be found in finitely many steps.

Proof: Let $G = \langle V, E \rangle$ be the digraph of the given acyclic diagram $\mathcal{H}$, and let $max(A, A')$ be the maximum of the lengths of all the distinct paths from A to A' in G. We prove the following property $P(n)$ for any natural number n: "For any action function symbol A, A' such that $max(A, A') = n$, $n \leq |V|$, and for any distinct free variables $\vec{x}, \vec{y}$, $isA(A'(\vec{x}), A(\vec{y})) \equiv \phi_{A',A}(\vec{x}, \vec{y})$ for some FO formula $\phi_{A',A}$ (including $\top$ and $\bot$) with object variables (if any) at most among $\langle \vec{x}, \vec{y} \rangle$". We prove $P(n)$ by using complete induction.

Base case: $P(0)$, $max(A, A') = 0$, two sub-cases.

Case 1: $A = A'$, since isA is reflexive

$$isA(A'(\vec{x}), A(\vec{y})) \equiv A'(\vec{x}) = A(\vec{y}) \equiv \bigwedge_{i=1}^{|\vec{x}|} x_i = y_i \text{ (by UNA).}$$

Case 2: $A \neq A'$, and since $max(A, A') = 0$ this means there is no $specialization$ path between A and A' at all, then $isA(A'(\vec{x}), A(\vec{y})) \equiv \bot$.

Inductive step: Assume that $P(j)$ is true for all $j < n$, we prove $P(n)$, where $n > 0$. Consider any action function symbols A, A' such that $max(A, A') = n$, where $n \leq |V|$. Since G is acyclic, hence each path from A to A' has no repetitions of the action nodes. Since $n > 0$, collect all those parents of A' in G, say $\{A_1, \cdots, A_t\}$, which are specializations of A and $A_i \neq A$ for each i (i.e., there is a path from A to each of A_i). Then we have

$$isA(A'(\vec{x}), A(\vec{y})) \equiv \bigvee_{i=1}^{t} (\exists \vec{x_i}) [specialization(A'(\vec{x}), A_i(\vec{x_i})) \wedge isA(A_i(\vec{x_i}), A(\vec{y}))]$$

By the induction hypothesis, since $max(A, A_i)$=n–1, for each i, we have $isA(A_i(\vec{x_i}), A(\vec{y})) \equiv \phi_{A_i,A}(\vec{x_i}, \vec{y})$ for some FO formula $\phi_{A_i,A}$. In $\mathcal{H}$, we have axioms such that for each i there is a FO formula ϕ_{A',A_i}

$$specialization(A'(\vec{x}), A_i(\vec{x_i})) \equiv \phi_{A',A_i}(\vec{x}, \vec{x_i}).$$

Therefore, let a required FO formula be

$$\phi_{A',A}(\vec{x}, \vec{y}) = \bigvee_{i=1}^{t} (\exists \vec{x_i}) [\phi_{A',A_i}(\vec{x}, \vec{x_i}) \wedge \phi_{A_i,A}(\vec{x_i}, \vec{y})],$$

then $P(n)$ is proved. Notice that $n \leq |V|$; hence such FO formula can always be obtained in finitely many steps.

Example 3 Continue with Example 2, for any action functions $A_1(\vec{x})$ and $A_2(\vec{y})$ in the example, most of the equivalent FO formula $\phi_{A_1,A_2}(\vec{x}, \vec{y})$ of $isA(A_1(\vec{x}), A_2(\vec{y}))$ are straightforward (either $\top$, or $\bot$ or the same as the axioms of $specialization$), expect for $isA(drive(p, v, o, d), move(obj, orig, dest))$ for any free variable $p, v, o, d, obj, orig, dest$. By using Def. 2 and the axioms given in Example 2, we have

$$specialization(drive(p, v, o, d), move(obj, orig, dest)) \supset isA(drive(p, v, o, d), move(obj, orig, dest)), \text{ and}$$
$$specialization(drive(p, v, o, d), travel(p, o, d)) \wedge specialization(travel(p, o, d), move(obj, orig, dest)) \supset isA(drive(p, v, o, d), move(obj, orig, dest)),$$

and there are no other non-equivalent axioms for $drive$ and $move$. Hence,

$$\begin{aligned} & isA(drive(p, v, o, d), move(obj, orig, dest)) \\ \equiv\ & specialization(drive(p, v, o, d), move(obj, orig, dest)) \vee \\ & specialization(drive(p, v, o, d), travel(p, o, d)) \wedge \\ & specialization(travel(p, o, d), move(obj, orig, dest)) \\ \equiv\ & v = obj \wedge o = orig \wedge d = dest \vee \\ & p = obj \wedge o = orig \wedge d = dest, \end{aligned}$$

which can be simplified as: for any variables p, v, o, d, obj,

$$isA(drive(p, v, o, d), move(obj, o, d)) \equiv p = obj \vee v = obj.$$

Now, we provide a more compact way to specify the basic action theory (BAT) based on a given hierarchy of actions. For later convenience, we will call such a modified BAT as a *hierarchical BAT* and denote it as $\mathcal{D}^H$. Moreover,

$$\mathcal{D}^H = \mathcal{H}^\star \cup \mathcal{D}^H_{ap} \cup \mathcal{D}^H_{ss} \cup \mathcal{D}_{S_0} \cup \Sigma \cup \mathcal{D}_{una},$$

where $\mathcal{H}^\star$ is the action hierarchy specified for the given dynamic system, $\mathcal{D}_{S_0}$ includes $\mathcal{A} \cup \mathcal{T}$, $\mathcal{D}^H_{ap}$ ($\mathcal{D}^H_{ss}$, respectively) is the new class of action precondition axioms (the new class of successor state axioms, respectively) specified based on the action hierarchy. First, $\mathcal{D}^H_{ap}$ includes an action precondition axiom represented as follows.

$$Poss(a, s) \equiv \bigvee_{i=0}^{h} \pi_i(a, s) \tag{3}$$

where $h \geq 0$ and each $\pi_i(a,s)$ is of the form

$$\exists \vec{x_i}.\, isA(a, A_i(\vec{x_i})) \wedge (\bigwedge_{j=0}^{n_i} \neg(\exists \vec{y_{ij}}) isA(a, A_{ij}(\vec{y_{ij}}))) \wedge \phi_i(\vec{x_i}, s) \quad (4)$$

When $n_i = 0$, the conjunction does not exist. All clauses in the RHS of Eq. (3) are exclusive to each other and each clause of the form (4) has the meaning that whenever a is a specialization of action $A_i(\vec{x_i})$ and it is not a specialization of some other actions $A_{ij}(\vec{y_{ij}})$, then the precondition of a is an uniform formula $\phi_i(\vec{x_i}, s)$.

The syntactic form of successor state axioms in $\mathcal{D}_{ss}^H$ is also different. For each fluent, the change of its truth value is now determined not by each action individually, as it is in Reiter's $\mathcal{D}_{ss}$, but it is determined by whole classes of actions. In general, the new syntactic form of SSA of a relational fluent F is as follows.

$$F(\vec{x}, do(a,s)) \equiv \bigvee_i \psi_i^+(\vec{x}, a, s) \vee F(\vec{x}, s) \wedge \neg(\bigvee_i \psi_i^-(\vec{x}, a, s)), \quad (5)$$

where each formula $\psi_i^+(\vec{x}, a, s)$ ($\psi_i^-(\vec{x}, a, s)$, respectively) specifies a positive effect (negative effect, respectively) with certain conditions on fluent F and has the following format

$$\exists \vec{z'}.isA(a, A_i(\vec{y_i})) \wedge (\bigwedge_{l=0}^{n_i} \neg(\exists \vec{y_{il}}) isA(a, A_{il}(\vec{y_{il}}))) \wedge (\bigwedge_{j=0}^{m_i} actorSlot(L_j, a, z_j) \wedge argType(a, z_j, T_j)) \wedge \gamma(\vec{x}, \vec{z'}, s) \quad (6)$$

where $\vec{y_i}$ ($\vec{y_{il}}$, respectively) are all variables in action term A_i (in action term A_{il}, respectively) and some of these variables can be the same as variables in $\vec{x}$; each variable z_j is an object argument of an action term that can be substituted for a, and z_j is either the same as an object variable in $\vec{x}$ or $\vec{y}$, or is a new variable, $\vec{z'} = \vec{y_i} \cup \{z_1, \cdots, z_{m_i}\} - \vec{x}$, and $\gamma(\vec{x}, \vec{z'}, s)$ is an uniform formula that has $\vec{x}, \vec{z'}, s$ at most as its free variables. According to this axiom, every action term that is a specialization of A_i, but is not a specialization of some other action terms A_{il} will have (either positive or negative) effect on a fluent F if this action term has actor slots L_j and the corresponding object arguments have types T_j explicitly mentioned in a SSA. Note that action terms can be underspecified in the sense that only required actor slots and types can be mentioned in the axiom, but actions that have effect on F might have also other actior slots (and their object arguments can have also other types), in addition to those which occur in the SSA above. Thus, (6) specifies classes of action terms that can have effect on a fluent.

Similarly, we can specify the SSA for each functional fluent, say f, using classes of actions instead of individual actions. Let $\mathcal{D}^H$ be the collection of the SSAs for all relational and function fluents.

Other classes of axioms such as initial KB $\mathcal{D}_{S_0}$ (which includes the class of axioms $\mathcal{A}$ as discussed above), the foundational axioms Σ and unique name axioms for actions $\mathcal{D}_{una}$ have the same formats as in (Reiter 2001).

We can prove the following lemmas.

Lemma 2 *For the precondition axiom class $\mathcal{D}_{ap}^H$, there exists a set of precondition axioms of the form of Reiter's basic action theory (Reiter 2001), say $\mathcal{D}_{ap}$, such that $\mathcal{D}_{ap}$ specifies the precondition for each action function symbol and $\mathcal{D}_{ap}$ is equivalent to $\mathcal{D}_{ap}^H$: for each action term a and any situation term s,*

$$\mathcal{D}^H \models Poss(a,s) \textit{ iff } \mathcal{D} \models Poss(a,s).$$

Proof: Notice that $\mathcal{D}_{ap}^H$ only includes one axiom and it is of the form Eq. (3), it specifies the precondition for any action a. Consider each action function symbol A with free variable $\vec{x}$ as its argument, i.e., $A(\vec{x})$, without loss of generality, we assume that $\vec{x}$ are new variables never used in Eq. (3). We substitute a with $A(\vec{x})$ and get

$$Poss(A(\vec{x}), s) \equiv \bigvee_{i=0}^{h} \exists \vec{x_i}.isA(A(\vec{x}), A_i(\vec{x_i})) \wedge \big(\bigwedge_{j=0}^{n_i} \neg(\exists \vec{y_{ij}}) isA(A(\vec{x}), A_{ij}(\vec{y_{ij}}))\big) \wedge \phi_i(\vec{x_i}, s).$$

By Lemma 1, for each A_i (A_{ij}, respectively), there exists a FO formula, say ϕ_{A,A_i} ($\phi_{A,A_{ij}}$, respectively) such that

$$isA(A(\vec{x}), A_i(\vec{x_i})) \equiv \phi_{A,A_i}(\vec{x}, \vec{x_i}) \quad (7)$$

$$isA(A(\vec{x}), A_{ij}(\vec{y_{ij}})) \equiv \phi_{A,A_{ij}}(\vec{x}, \vec{y_{ij}}), \quad (8)$$

respectively. Therefore we have

$$Poss(A(\vec{x}), s) \equiv \bigvee_{i=0}^{h} \exists \vec{x_i}.\phi_{A,A_i}(\vec{x}, \vec{x_i}) \wedge [\bigwedge_{j=0}^{n_i} \neg(\exists \vec{y_{ij}}) \phi_{A,A_{ij}}(\vec{x}, \vec{y_{ij}})] \wedge \phi_i(\vec{x_i}, s).$$

Let $\mathcal{D}_{ap}$ be the collection of all the axioms obtained above for each action symbol, and it is easy to see that $\mathcal{D}_{ap}$ is of the form described in (Reiter 2001). To prove the statement in the opposite direction, note that $\mathcal{D}_{ap}^H$ specifies the preconditions for all actions and no more, where the worst case happens when all A_i ($i = 1..h$) in the RHS of Eq. (3) enumerate each and every action function symbol. Therefore, it is easy to see $\mathcal{D}_{ap}^H$ is equivalent to $\mathcal{D}_{ap}$ according to the way how we obtain $\mathcal{D}_{ap}$ as the above.

Note that $\mathcal{D}_{ap}^H$ can be more succinct than $\mathcal{D}_{ap}$, because they deal with classes of actions instead of individual actions.

Lemma 3 *For a $\mathcal{D}_{ss}^H$, there exists an equivalent class $\mathcal{D}_{ss}$ including SSAs of the forms given in Reiter's basic action theory (Reiter 2001): for each predicate fluent F*

$$\mathcal{D}^H \models F(\vec{x}, do(a,s)) \equiv \phi_F(\vec{x}, a, s)$$

(for each function fluent f, $\mathcal{D}^H \models f(\vec{x}, do(a,s)) = y \equiv \phi_f(\vec{x}, y, a, s)$, respectively), there exists an uniform formula $\phi'_F(\vec{x}, a, s)$ ($\phi'_f(\vec{x}, y, a, s)$, respectively) such that

$$\mathcal{D} \models F(\vec{x}, do(a,s)) \equiv \phi'_F(\vec{x}, a, s)$$

($\mathcal{D} \models f(\vec{x}, do(a,s)) = y \equiv \phi'_f(\vec{x}, y, a, s)$, respectively).

Proof: Since the proof for predicate fluents and functional fluents are almost the same, we will provide the proof for predicate fluents only.

Assume that the given dynamic system includes totally k action functions, say $A_1(\vec{v}_1), \cdots, A_k(\vec{v}_k)$. For each predicate fluent F, assume that the SSA of $F(\vec{x})$ in $\mathcal{D}^H$ is of the form Eq. (5), then we substitute a with each action function, say $A_i(\vec{v}_i)$ (without loss of generality, we assume that variables in $\vec{v}_i$ are all new variables never used in Eq. (5)). By using Lemma 1, similar to the proof of Lemma 2 (replacing isA for any two action functions with an equivalent FO formula), we obtain an axiom of the form

$$F(\vec{x}, do(A_i(\vec{v}_i), s)) \equiv \psi_i^+(\vec{x}, \vec{v}_i, s) \vee F(\vec{x}, s) \wedge \neg\psi_i^-(\vec{x}, \vec{v}_i, s).$$

Whenever $\psi_i^+(\vec{x},\vec{v}_i,s)$ ($\psi_i^-(\vec{x},\vec{v}_i,s)$, respectively) is consistent, $A_i(\vec{v}_i)$ has positive effect (negative effect) on F under such condition. Therefore, the equivalent SSA of F in the classical basic action theory of (Reiter 2001) can be constructed as

$$F(\vec{x},do(a,s)) \equiv [\bigvee_{i=0}^{k^+}(\exists\vec{v}_i)(a{=}A_i(\vec{v}_i)\wedge\psi_i^+(\vec{x},\vec{v}_i,s))]\vee F(\vec{x},s)\wedge\neg[\bigvee_{i=0}^{k^-}(\exists\vec{v}_i)(a{=}A_i(\vec{v}_i)\wedge\psi_i^-(\vec{x},\vec{v}_i,s))].$$

Notice that the above equation can be simplified by removing inconsistent formulas. Hence the lemma is proved.

We then have the following important property:

Theorem 2 *For each $\mathcal{D}^H$, there exists an equivalent $\mathcal{D}$ of the format given in (Reiter 2001), where equivalence means that for any FO regressable sentence W,*

$$\mathcal{D}^H \models W \text{ iff } \mathcal{D} \models W.$$

Proof: Use Lemma 2 and Lemma 3.

The definition of the regression operator and the regressable sentences in $\mathcal{D}^H$ are all the same as in (Reiter 2001), and similar to the regression theorem (Reiter 2001), we have

$$\mathcal{D}^H \models W \text{ iff } \mathcal{D}_{S_0}\cup\mathcal{H}^*\cup\mathcal{D}_{una} \models \mathcal{R}[W]$$

for any regressable sentence W. Let $\mathcal{E}\big[\mathcal{R}[\mathcal{W}]\big]$ be the operator that eliminates all occurrences (if any) of the $isA(A',A)$ predicate in $\mathcal{R}[W]$ in favor of the corresponding FO formulas $\phi_{A',A}$ that exist according to Lemma 1.

Theorem 3 *For each $\mathcal{D}^H$ and for any FO regressable sentence W, $\mathcal{D}^H \models W$ iff $\mathcal{D}_{S_0}\cup\mathcal{H}\cup\mathcal{D}_{una} \models \mathcal{E}\big[\mathcal{R}[W]\big]$.*

The theorem is important because $\mathcal{D}_{S_0}\cup\mathcal{H}^*\cup\mathcal{D}_{una}$ (and hence $\mathcal{D}^H$) include second-order axioms of the definition of the predicate isA. However, all occurrences of isA in sentence $\mathcal{R}[W]$ can be replaced by FO sentences in finitely many steps according to the Lemma 1. Consequently, one can use regression in our modular BAT to reduce projection and executability problems to theorem proving in FOL only.

Discussion and Future Work

There are a few papers with similar motivation that we would like to mention. (Erdogan & Lifschitz 2006) and (Lifschitz & Ren 2006) consider modular theories in the action representation language $\mathcal{C}+$, and address the problem of the development of libraries of reusable, general-purpose knowledge components (that is formulated in (Barker, Porter, & Clark 2001)), but do not consider the situation calculus. It also remains unclear whether their approach can handle hierarchies of actions. Eyal Amir (Amir 2000) proposes and studies an object oriented version of the situation calculus, but he investigates representation that is significantly different from our approach and considers neither taxonomies of actions nor BAT. Yolanda Gil (Gil 2005) reviews implemented systems that benefited from integration of planning and description logics (DL) for the purposes of representing taxonomies of objects, actions, plans and goals. However, in our paper we would like to take advantage of the well know fact that many description logics (in particular, all DL systems consider in her paper) are syntactic fragment of FOL (Borgida 1996), and for this reason we use FOL syntax only. (Aitken 2005) discuss possible integration of the situation calculus based Process Specification Language (PSL) and CYC, but his paper concentrates on implementation issues and does not explore formal foundations for reasoning about actions in CYC.

References

Aitken, S. 2005. An ontological account of action in processes and plans. *Knowledge-Based Systems* 18(6):295–305.

Amir, E. 2000. (De)Composition of Situation Calculus Theories. In *Proceedings of the Seventienth National Conference on Artificial Intelligence (AAAI-00)*. AAAI.

Barker, K.; Porter, B.; and Clark, P. 2001. A library of generic concepts for composing knowledge bases. In *First International Conference on Knowledge Capture*. http://www.cs.utexas.edu/users/mfkb/papers/.

Borgida, A. 1996. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence* 82(1-2):353–367.

Cycorp. 2002? Opencyc 1.0: www.opencyc.org.

Davidson, D. 1967. The logical form of action sentences. In Rescher, N., ed., *The Logic of Decision and Action*. University of Pittsburgh Press. 81–95.

Erdogan, S., and Lifschitz, V. 2006. Actions as special cases. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR-06)*.

Gil, Y. 2005. Description logics and planning. *AI Magazine* 26(2):73–84.

Lifschitz, V., and Ren, W. 2006. A modular action description language. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*. AAAI.

Matuszek, C.; Cabral, J.; Witbrock, M.; and DeOliveira, J. 2006. An introduction to the syntax and content of cyc. In *Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, AAAI Spring Symposium. Stanford, CA: http://www.cyc.com/cyc/technology/pubs.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence*, volume 4. Edinburgh University Press, Reprinted in (McCarthy 1990). 463–502.

McCarthy, J. 1959. Programs with common sense. In *Mechanisation of Thought Processes, Proceedings of the Symposium of the National Physics Laboratory*, 77–84. London, U.K.: Her Majesty's Stationery Office. Reprinted in (McCarthy 1990).

McCarthy, J. 1963. Situations, actions and causal laws. Technical Report Technical Report Memo 2, Stanford University Artificial Intelligence Laboratory, Stanford, CA. Reprinted in Marvin Minsky, editor, S̈emantic information processing, MIT Press, 1968.

McCarthy, J. 1986. Applications of circumscription to formalizing common sense knowledge. *Artifficial Intelligence* 28:89–116.

McCarthy, J. 1990. *Formalization of common sense: papers by John McCarthy edited by V. Lifschitz*. Norwood, N.J.: Ablex.

McCarthy, J. 2002. Actions and other events in situation calculus. In *Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, http://www–formal.stanford.edu/jmc/sitcalc.html.

Parmar, A. 2001. The representation of actions in KM and CYC. Technical Report FRG 1, Dept. of Computer Science, Stanford University, Stanford, CA. http://www-formal.stanford.edu/aarati/techreports/action-reps-frg-techreport.ps.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. The MIT Press.

Shanahan, M. 1997. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. The MIT Press, 410 pages.