Reasoning about Large Taxonomies of Actions

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@cs.ryerson.ca

Abstract

We design a representation based on the situation calculus to facilitate development, maintenance and elaboration of very large taxonomies of actions. This representation leads to more compact and modular basic action theories (BATs) for reasoning about actions than currently possible. We compare our representation with Reiter's BATs and prove that our representation inherits all useful properties of his BATs. Moreover, we show that our axioms can be more succinct, but extended Reiter's regression can still be used to solve the projection problem (this is the problem of whether a given logical expression will hold after executing a sequence of actions). We also show that our representation has significant computational advantages. For taxonomies of actions that can be represented as finitely branching trees, the regression operator can work exponentially faster with our theories than it works with Reiter's BATs. Finally, we propose general guidelines on how a taxonomy of actions can be constructed from the given set of effect axioms in a domain.

Introduction

A long-standing and important problem in AI is the problem of how to represent and reason about effects of actions grouped in a realistically large taxonomy, where some actions can be more generic (or more specialized) than others. While the problem of representing large *semantic networks* of (static) concepts has been addressed in AI research from the 1970s and served as motivation for research on *description logics*, a related problem of representing and reasoning about large taxonomies of *actions* received surprisingly little attention. We would like to address this problem using the *situation calculus*. The situation calculus (SC) is a well known and popular predicate logical theory for reasoning about events and actions. There are several different formulations of the SC. In this paper we would like to concentrate on *basic action theories* (BATs) introduced in (Reiter 2001), in particular, on successor state axioms (SSAs) proposed by Reiter to solve (sometimes) the frame problem (SSAs are part of a BAT). Recall that BATs are more expressive than STRIPS theories: actions specified using BATs can have context-dependent effects. We propose a representation that allows writing more compact and modular BATs than is currently possible. BATs are logical theories of a certain syntactic form that have several desirable theoretical properties. However, BATs have not been designed to support taxonomic reasoning about objects and actions. Essentially, these theories are "flat" and do not provide representation for hierarchies of actions. This can lead to potential difficulties if one intends to use BATs for the purpose of large scale formalization of reasoning about actions on the commonsense level, when potentially arbitrary actions and objects have to be represented. Intuitively, many events and actions have different degrees of generality: the action of driving a car from home to an office is a specialization of the action of transportation using a vehicle, that is in its turn a specialization of the action of moving an object from one location to another. We represent hierarchies of actions explicitly and use them in our new modular SSAs. However, we show that our new modular SSAs can be translated into "flat" Reiter's SSAs and, consequently, we inherit all useful properties of his BATs: formulas entailed from Reiter's BAT remain entailments from a modular BAT; consequently, the projection problem can be solved.

Below, we first review the SC. Then we propose a new representation that helps to design modular BATs and prove that it has the same desirable logical properties as Reiter's BATs. We also discuss the significant computational advantages of using modular BATs in comparison to Reiter's "flat" BAT. Finally, we propose an approach to designing taxonomies of actions and discuss related work.

The Situation Calculus

All dialects of the SC $\mathcal{L}_{sc}$ include three disjoint sorts (*actions*, *situations* and *objects*). **Actions** are first-order (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, the notation $s' \sqsubseteq s$ means that either situation s' is a subsequence of situation s or $s = s'$. **Objects** are FO terms other than actions and situations that depend on the domain of an 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 actions $\alpha_1, \cdots, \alpha_n$ in S_0. For simplicity, we omit all details related to functional fluents below. All free variables are always $\forall$-quantified at the front.

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 mentions only fluents (does not mention $Poss$ or $\sqsubseteq$), it does not quantify over variables of sort situation, it does not use equality on situations, and whenever it mentions a term of sort situation in a fluent, then that term is a variable 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

$$F(\vec{x}, do(a, s)) \equiv \bigvee_i \psi_i^+(\vec{x}, a, s) \vee F(\vec{x}, s) \wedge \neg(\bigvee_j \psi_j^-(\vec{x}, a, s)). \quad (1)$$

Here, each formula $\psi_i^+(\vec{x}, a, s)$ ($\psi_j^-(\vec{x}, a, s)$, respectively) is uniform in s and specifies a positive effect (negative effect, respectively) with certain conditions on fluent F. Each $\psi_i^+(\vec{x}, a, s)$ or $\psi_i^-(\vec{x}, a, s)$ in Eq. (1) has the syntactic form

$$\exists \vec{z}. a = A(\vec{y}) \wedge \gamma(\vec{x}, \vec{z}, s), \quad (2)$$

where $\vec{z} = \vec{y} - \vec{x}$ and $\gamma(\vec{x}, \vec{z}, s)$ is a context where the action $A(\vec{y})$ has the effect. The successor state axiom (SSA) for each fluent F completely characterizes the truth value of F in the next situation $do(a, s)$ in terms of values that fluents have in the current situation s. Notice that, unlike STRIPS, in general these SSA axioms are context-dependent. **Initial theory $\mathcal{D}_{S_0}$:** 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 that 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 $\mathcal{D} = \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{S_0} \cup \Sigma \cup \mathcal{D}_{una}$ 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. 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 reasoning 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 replacements 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 name axioms for actions: $\mathcal{D} \models W$ iff $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models \mathcal{R}[W]$. This fact is the key result in the 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 represent and reason about dynamic systems.

Action Hierarchies

In practice, it is not easy to specify and reason with a logical theory $\mathcal{D}$ if an application domain includes a very large number of actions. To deal with this problem, we propose to represent events and actions using a hierarchy.

Definition 1 *We use the predicate $sp(a_1, a_2)$ to represent that action a_1 is a* direct specialization *of action a_2 (action a_2 is a* direct generalization *of a_1). An* action diagram *is defined by a finite set $\mathcal{H}$ of axioms of the syntactic form*

$$sp(A_1(\vec{x}), A_2(\vec{y})) \equiv \phi_{A_1, A_2}(\vec{x}, \vec{y}) \quad (3)$$

for two action functions $A_1(\vec{x}), A_2(\vec{y})$, where $\phi_{A_1, A_2}(\vec{x}, \vec{y})$ is a satisfiable (i.e., not equivalent to $\perp$) situation-free FO formula with free variables at most among $\vec{x}, \vec{y}$. Also, $\mathcal{H}$ must be such that the following condition hold:

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

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 in $\mathcal{D}$ and a directed edge $A_j \rightarrow A_k$ belongs to the edge set E iff there is an axiom of the form $sp(A_j(\vec{x}), A_k(\vec{y})) \equiv \phi_{A_j, A_k}(\vec{x}, \vec{y})$ in $\mathcal{H}$. From $\mathcal{D}_{una}$ follows that the graph G cannot have multiple edges from one node to another. When the digraph G of $\mathcal{H}$ is acyclic, i.e., there is no directed loop in G, we call $\mathcal{H}$ an *acyclic action diagram*. Below, 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 generally, there can be actions that have several parents (multiple inheritance case), as shown in Examples 1 and 2.

Example 1 Consider actions performed in a kitchen, actions such as washing, cooking, frying, etc. Some can be considered as specializations of others. To simplify the example we assume that water and electricity are always available, ignore some other kitchen activities (such as chopping, mixing, etc) and consider the (simplified) action digraph shown in Fig. 1. Each edge corresponds to one sp axiom in the set $\mathcal{H}$, for example,

$sp(wash(x), kitchenAct)$, $sp(prepFood(x), kitchenAct)$,
$sp(cook(food, vessel), prepFood(food))$,
$sp(oilyCook(food, vessel), cook(food, vessel))$,
$sp(oilyCook(food, vessel), reheat(food))$,
$sp(microwave(food), reheat(food)), \cdots$, etc.

Example 2 We show additional examples where actions in $\mathcal{H}$ can have different numbers of arguments. Consider an action $travel(p, o, d)$: a person p travels from origin o to destination d. It can be regarded as a direct specialization

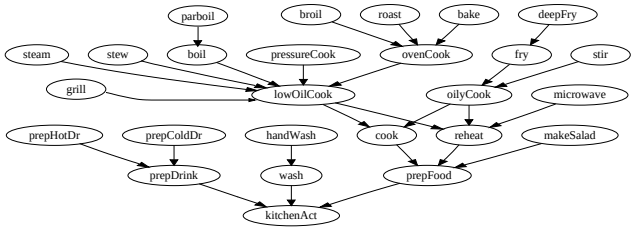

Figure 1: A (Simplified) Action Digraph for Kitchen Activities

of $move$ – person p moves from location o to location d: $sp(\, travel(p,o,d), move(p,o,d)\,)$.

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: $sp(\, drive(p,v,o,d), travel(p,o,d)\,)$. It is also a direct specialization of action $move$ – vehicle v moves from location o to location d: $sp(\, drive(p,v,o,d), move(v,o,d)\,)$.

Consider an action $passDr(p,dr)$: a person p passes through a door dr. It is considered as a direct specialization of $move(p,o,d)$ iff the origin o is the outside of dr and the destination d is the inside of dr, or vice versa:

$sp(\, passDr(p,dr), move(p,o,d)\,) \equiv$
$outside(o,dr) \wedge inside(d,dr) \vee outside(d,dr) \wedge inside(o,dr)$,

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

In this paper, we will only consider action diagrams with *monotonic inheritance* of effects:

$$\mathcal{D} \cup \mathcal{H} \models (\forall F).sp(a_1,a_2) \wedge F(s) \not\equiv F(do(a_2,s)) \supset F(do(a_1,s)) \equiv F(do(a_2,s)).$$

Since there are only finitely many (say, m) fluents in $\mathcal{D}$, the above second-order (SO) formula can be replaced by the finite conjunction (over $j = 1..m$) of FO formulas (where $F_j(\vec{x_j}, s)$ is jth fluent with object arguments $\vec{x_j}$):

$$sp(a_1,a_2) \wedge F_j(\vec{x_j},s) \not\equiv F_j(\vec{x_j}, do(a_2,s)) \supset F_j(\vec{x_j}, do(a_1,s)) \equiv F_j(\vec{x_j}, do(a_2,s)).$$

Because in general we need to reason about a direct specialization of another direct specialization of an action, we define (distant) specializations using the predicate sp^*.

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

$$sp^*(a_1,a_2) \equiv (\forall P).\{(\forall v)[P(v,v)] \wedge (\forall v,v',v'')[sp(v,v') \wedge P(v',v'') \supset P(v,v'')] \wedge (\forall v,v')[sp(v,v') \supset P(v,v')]\} \supset P(a_1,a_2) \quad (4)$$

Axiom (4) requires SO logic, but we will show in Theorem 3 that we can still reduce reasoning about regressable formulas to theorem proving in FOL only. We denote the set of axioms including Axiom (4) and all axioms in an action diagram $\mathcal{H}$ as $\mathcal{H}^\star$ and call it the *action hierarchy (of $\mathcal{H}$)*.

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

One can easily prove that under $\mathcal{D}_{una}$, $\mathcal{H}^*$ is acyclic according to Def. 3 iff the digraph of the action diagram $\mathcal{H}$ is acyclic. Note that the above condition in Def. 3 is more general than the antisymmetry of sp^* (because antisymmetry is not strong enough to assure the acyclicity of $\mathcal{H}$).

The following theorem states that the action hierarchies entail the same intuitively clear taxonomic properties as the predicate sp.

Theorem 1 *Let $\mathcal{H}$ be an acyclic action diagram, whose corresponding action hierarchy is $\mathcal{H}^\star$. Then,*

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

Proof: It follows from Def. 1, Def. 2 and Def. 3 using induction, but details are omitted because of lack of space. □

Moreover, the following lemma will be convenient later.

Lemma 1 *Consider any acyclic action diagram $\mathcal{H}$, whose corresponding action hierarchy is $\mathcal{H}^\star$. For any action functions $A_1(\vec{x})$ and $A_2(\vec{y})$, $A_1(\vec{x})$ is a (distant) specialization of $A_2(\vec{y})$ iff $\phi_{A_1,A_2}(\vec{x},\vec{y})$, for some situation-free FO formula ϕ_{A_1,A_2} (including $\top$ and $\bot$) whose free object variables are at most among $\vec{x}$ and $\vec{y}$. That is,*

$$\mathcal{H}^\star \cup \mathcal{D}_{una} \models sp^*(A_1(\vec{x}), A_2(\vec{y})) \equiv \phi_{A_1,A_2}(\vec{x},\vec{y}).$$

And, ϕ_{A_1,A_2} can be found from $\mathcal{H}$ in finitely many steps.

Proof: Let $G = \langle V, E\rangle$ be the digraph of the given $\mathcal{H}$, 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 $\vec{x},\vec{y}$, $sp^*(A'(\vec{x}), A(\vec{y})) \equiv \phi_{A',A}(\vec{x},\vec{y})$ for some FO formula $\phi_{A',A}$ (including $\top$ and $\bot$) with object variables at most among $\vec{x}$ and $\vec{y}$".

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

Case 1: A=A', since sp^* is reflexive

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

Case 2: $A \neq A'$, and since $max(A',A)$=0, which means there is no sp path between A and A', then $sp^*(A'(\vec{x}), A(\vec{y})) \equiv \bot$.

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 direct generalizations of A' in G, say $\{A_1, \cdots, A_t\}$, which are (distant) specializations of A. Then,

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

For each i, $max(A_i,A) \leq n-1$. By the induction hypothesis, we have $sp^*(A_i(\vec{x_i}), A(\vec{y})) \equiv \phi_{A_i,A}(\vec{x_i},\vec{y})$ for some situation-free FO formula $\phi_{A_i,A}$ whose free variables are at most among $\vec{x_i}$ and $\vec{y}$. In $\mathcal{H}$, for each i we have

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

where ϕ_{A',A_i} is a situation-free FO formula. Let

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

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

Example 3 We continue with Example 2. Most of the FO formulas $\phi_{A_1,A_2}(\vec{x},\vec{y})$ equivalent to $sp^*(A_1(\vec{x}),A_2(\vec{y}))$ are straightforward (either $\top$, $\bot$, or the same as the axioms of sp), except for $sp^*(drive(p,v,o,d),move(obj,orig,dest))$ for any free variable $p,v,o,d,obj,orig,dest$. By using Def. 2 and the axioms given in Example 2, we have

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

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

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

Modular BATs

Our goal is to provide a more compact specification of a BAT based on a given hierarchy of actions. We will call such a modified BAT a *modular BAT* and denote it as $\mathcal{D}^H$, where

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

Here, $\mathcal{D}^H_{S_0} = \mathcal{D}_{S_0} \cup \mathcal{H}^\star$, in which $\mathcal{H}^\star$ is the action hierarchy and $\mathcal{D}_{S_0}$ describes the usual initial state, the same as Reiter's initial theory, and $\mathcal{D}^H_{ss}$ is the new class of SSAs specified based on $\mathcal{H}^\star$. In the sequel, let $sp^*_=(a,a')$ be an abbreviation for either $sp^*(a,a')$ or $a=a'$ in Formula (5) below.

The new syntactic form of SSAs in $\mathcal{D}^H_{ss}$ can be different from Reiter's format in $\mathcal{D}_{ss}$. Intuitively, instead of repeating tediously each individual action in the right-hand side (RHS) of a SSA for a fluent, say $F(\vec{x},do(a,s))$, one can take advantage of the action hierarchies and describe the effect of the whole class of action functions at once. One can say that all those actions which are (distant) specializations of some generic action $A(\vec{y})$ (actions from the branch going out of $A(\vec{y})$), except those (distant) specializations of some other generic actions, say $A(\vec{y}_l)$ for $1 \le l \le h$ (i.e., excluding actions from some branches), can cause the same (positive or negative) effects on F under certain conditions. By doing so, we can represent the effects of actions more compactly. We will see later that this new form of SSAs leads to significant computational advantages as well.

It is convenient to use the following notation related with a fluent $F(\vec{x},s)$: for any variable vector $\vec{y}_l$ ($l \ge 0$), let $\vec{z}_l = \vec{y}_l - \vec{x}$ (i.e., $\vec{z}_l$ are the new variables mentioned in $\vec{y}_l$ but not in $\vec{x}$). Note that, in the RHS of the SSA of $F(\vec{x},s)$, those new variables $\vec{z}_l$ need to be existentially quantified. Formally speaking, the modified SSA of a relational fluent $F(\vec{x},s)$ has the format (1), where each $\psi^+_i(\vec{x},a,s)$ or $\psi^-_i(\vec{x},a,s)$ has either the syntactic form (2) or the following syntactic form:

$$(\exists\, \vec{z}_0)[sp^\star(a,A(\vec{y}_0)) \wedge \gamma(\vec{x},\vec{z}_0,s) \wedge \bigwedge_{l=1}^{h} \neg(\exists\, \vec{z}_l) sp^\star_=(a,A_l(\vec{y}_l))]. \quad (5)$$

In (5), γ is a formula uniform in s that has $\vec{x},\vec{z}_0,s$ at most as its free variables. Notice that whenever $\vec{z}_l$ ($l \ge 0$) is empty, then there is no existential quantifier over $\vec{z}_l$. In addition, in (5), when index $h=0$, the conjunction over l does not exist. One can prove that axiomatizers can always write modified SSAs in $\mathcal{D}^H$ with $h=0$ in (5). However, with negation (when $h>0$), axiomatizers gain flexibility of writing SSAs that can deliver more computational advantages. Details can be found in the next section (see Example 6).

Other classes of axioms such as the initial theory $\mathcal{D}_{S_0}$, the precondition axioms $\mathcal{D}_{ap}$, the foundational axioms Σ and unique name axioms for actions $\mathcal{D}_{una}$ have the same formats as in (Reiter 2001). It is easy to see that a modular BAT $\mathcal{D}^H$ differs from Reiter's BAT $\mathcal{D}$ in the following aspects: $\mathcal{D}^H_{S_0}$ includes the action hierarchy $\mathcal{H}^\star$ and can use sp^* to specify SSAs for *classes* of actions, while $\mathcal{D}_{ss}$ enumerates each action individually. However, according to Lemma 2 (with a constructive proof), theories $\mathcal{D}^H$ and $\mathcal{D}$ are related.

Lemma 2 *For a $\mathcal{D}^H_{ss}$, there exists an equivalent class $\mathcal{D}_{ss}$ including SSAs of the syntactic form given in Reiter's BAT (Reiter 2001): for each relational fluent F*

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

in which ϕ_F may have occurrences of the predicate sp^*, there exists a uniform formula $\phi'_F(\vec{x},a,s)$ that does not mention sp^* and such that $\mathcal{D} \models F(\vec{x},do(a,s)) \equiv \phi'_F(\vec{x},a,s)$.

Proof: Assume that a given BAT $\mathcal{D}$ includes k action functions in total, say $A_1(\vec{v}_1),\cdots,A_k(\vec{v}_k)$. For each relational fluent $F(\vec{x},s)$, assume that its SSA in $\mathcal{D}^H$ is of the form (1) whose positive and negative effect conditions have the syntactic form (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 the SSA of F), and in the RHS obtained by this substitution from the SSA of $F(\vec{x},do(A_i(\vec{v}_i),s))$, replace every occurrence of sp^* (that has two action functions as arguments) with its equivalent FO formula (that exists according to Lemma 1). This replacement results in an axiom of the following 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) are consistent conditions (SC formulas uniform in s), $A_i(\vec{v}_i)$ has a positive effect (a negative effect) on F under such condition. Hence, the following yields the logically equivalent SSA of F in the usual BAT of (Reiter 2001):

$$\begin{aligned} F(\vec{x},do(a,s)) \ \equiv\ & [\textstyle\bigvee_{i=1}^{k}(\exists\vec{v}_i)(a=A_i(\vec{v}_i) \wedge \psi^+_i(\vec{x},\vec{v}_i,s))] \vee \\ & F(\vec{x},s) \wedge \neg[\textstyle\bigvee_{j=1}^{k}(\exists\vec{v}_j)(a=A_j(\vec{v}_j) \wedge \psi^-_j(\vec{x},\vec{v}_j,s))]. \end{aligned}$$

Notice that the above axiom can be simplified by removing inconsistent clauses. 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 that has no occurrences of the predicate sp,* $\mathcal{D}^H \models W$ *iff* $\mathcal{D} \models W$.

Proof: Use Lemma 2. □

Here we provide some examples of the new way of representing SSAs, and compare them with Reiter's format.

Example 4 We continue with Example 1 (recall Figure 1). Consider a fluent $fCooked(x,s)$ (food x is cooked in the situation s), the modular BAT version of its SSA could be:

$$fCooked(x,do(a,s)) \equiv (\exists y) sp^*(a,cook(x,y)) \vee fCooked(x,s).$$

Another example is a SSA for the fluent $dirtyVes(x,s)$ (it will be false after washing a vessel x in some manner, or it will be true when x is used to prepare food or drink):

$dirtyVes(y, do(a, s)) \equiv dirtyVes(y, s) \wedge \neg sp^*(a, wash(y))$
$\vee (\exists x) sp^*(a, cook(x, y)) \vee (\exists x) a = makeSalad(x, y)$
$\vee (\exists x) sp^*(a, prepDrink(x, y))$.

The Reiter's SSA for fluent $fCooked(x, s)$ (with a bigger taxonomy of actions, it will be much longer):

$fCooked(x, do(a, s)) \equiv$
$(\exists y)[a = cook(x, y) \vee a = lowOilCook(x, y) \vee a = steam(x, y)$
$\vee a = boil(x, y) \vee a = stew(x, y) \vee a = broil(x, y)$
$\vee a = bake(x, y) \vee a = roast(x, y) \vee a = ovenCook(x, y)$
$\vee a = pressureCook(x, y) \vee a = oilyCook(x, y)$
$\vee a = fry(x, y) \vee a = deepFry(x, y) \vee a = stir(x, y)$
$\vee a = parboil(x, y) \vee a = grill(x, y)] \vee fCooked(x, s)$.

We can also get a similar longer Reiter's SSA for fluent $dirtyVes(x, s)$ (details are omitted).

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

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

for any regressable sentence W. Let $\mathcal{E}\big[\mathcal{R}[W]\big]$ (called the *extended regression of* W) be the operator that eliminates all occurrences (if any) of the $sp^*(A', A)$ predicate in $\mathcal{R}[W]$ in favor of the corresponding FO formulas $\phi_{A',A}$ that exists according to Lemma 1. Then, we have:

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

This theorem is important because $\mathcal{D}^H_{S_0} \cup \mathcal{D}_{una}$ (and hence $\mathcal{D}^H$) include the SO definition of the predicate sp^*. However, all occurrences of sp^* 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 BATs to reduce projection and executability problems to theorem proving in FOL only.

Advantages of Modular BATs

Using action hierarchies and specifying BATs modularly not only provides a compact way of representing effects of actions, but sometimes leads to a more computationally efficient (than Reiter's) solution of the projection problem.

Example 5 Continuing with Example 4, consider a ground action $\alpha = deepFry(Egg_1, FryingPan_1)$, and the regression of $fCooked(Egg_1, do(\alpha, S_0))$. Using Reiter's SSA for this fluent, regression involves checking 16 equality clauses between actions when regressing on the positive conditions in the SSA of $fCooked$ (see the axiom above). Using the modular BAT, extended regression of the positive conditions involves only 1 step of regression for predicate sp^*, and finally the replacement of $sp^*(\alpha, cook(Egg_1, y))$ with the corresponding FO formula, i.e., the operator $\mathcal{E}$, takes at most 4 steps of recursive computation (see Figure 1).

Apart from specific examples, let us discuss in general the following problems: when we can actually gain computational advantages using action hierarchies and how much we can gain, whether there is any possible computational disadvantage in using action hierarchies alone, and if so, whether it can be avoided.

According to the definition in the previous section, the digraph of an acyclic action diagram is in fact a directed acyclic graph (DAG). Computing the FO formula equivalent to $sp^*(A_1(\vec{x}), A_2(\vec{y}))$ for any pair of action functions $A_1(\vec{x})$ and $A_2(\vec{y})$ is similar to finding all paths from A_1 to A_2 in the corresponding digraph. The latter problem has the computational complexity of $\Theta(p)$ where p is the number of all the distinct edges in the digraph on any path from A_1 to A_2, and therefore has a computational complexity of $O(e)$ where e is the number of all edges in the digraph (i.e., e is the number of sp axioms in $\mathcal{H}$). As a consequence, this yields the following encouraging and important result.

We start with the case $h = 0$ in (5). Let $\phi(a, \vec{x}, s)$ denote $(\exists \vec{z}_0)[sp^*(a, A(\vec{y}_0)) \wedge \gamma(\vec{x}, \vec{z}_0, s)]$. In general, to provide an equivalent SSA of $F(\vec{x}, s)$ in Reiter's representation, $\phi(a, \vec{x}, s)$ has to be replaced by an uniform formula $\psi(a, \vec{x}, s)$ of the form $(\exists \vec{z}_0)[\psi_{sp}(a, \vec{y}_0) \wedge \gamma(\vec{x}, \vec{z}_0, s)]$. Here, $\psi_{sp}(a, \vec{y}_0)$ might have the form $(a = A(\vec{y}_0) \vee \bigvee_{i=1}^{n_A - 1} (\exists \vec{z}_i)(a = A_i(\vec{y}_i) \wedge \psi_i(\vec{y}_0, \vec{y}_i)))$, where each $A_i(\vec{y}_i)$ ($1 \leq i \leq n_A - 1$) is a specialization of $A(\vec{y}_0)$ under the condition $\psi_i(\vec{y}_0, \vec{y}_i)$, n_A is the total number of specializations of A. The formula $\psi_{sp}(a, \vec{y}_0)$ is a logically equivalent replacement of $sp^*(a, A(\vec{y}_0))$ in ϕ (see Lemma 2). Let the action diagram $\mathcal{H}$ in $\mathcal{D}^H$ be acyclic and the corresponding action digraph rooted at A have a tree structure (the most general actions are considered as roots and the most specialized actions are considered as leaves). Then, the computational time of extended regression $\mathcal{E}[\mathcal{R}[sp^*(\alpha, A(\vec{t}', \vec{z}_0))]]$ in the clause $\phi(\alpha, \vec{t}, S)$, for any object terms $\vec{t}$ and any situation term $do(\alpha, S)$, is no worse than and (sometimes) can be exponentially faster than computational time of Reiter's regression on ψ_{sp} in $\psi(\alpha, \vec{t}, S)$.

Theorem 4 *If the sub-tree rooted at A in the digraph of $\mathcal{H}$ is a complete k-ary tree ($k \geq 2$) with n_A action functions as its nodes, the computational complexity of extended regression $\mathcal{E}[\mathcal{R}[sp^*(\alpha, A(\vec{t}', \vec{z}_0))]]$ is $\Theta(\log_k n_A)$, while the computational complexity of Reiter's regression $\mathcal{R}[\psi_{sp}]$ on an equivalent replacement is $\Theta(n_A)$.*

Proof: When a DAG of an action hierarchy has a tree structure or a forest structure, there is at most one path between any two action functions. In particular, assume that the digraph of the action hierarchy is a complete k-ary ($k \geq 2$) tree structure and consider the regression of $F(\vec{t}, do(\alpha, S))$ for any action term α and situation term S. To specify the equivalent SSA in Reiter's format, $\phi(\alpha, \vec{t}, S)$ needs to be replaced by $\psi(\alpha, \vec{t}, S)$. It is easy to see that one-step regression of the above clause in Reiter's format takes $\Theta(n_A)$ steps (subsequently, additional time is required to regress recursively $\mathcal{R}[\gamma(\vec{t}, \vec{z}_0, S)]$). To perform one-step regression using the modular BAT format, it is sufficient to regress $sp^*(\alpha, A(\vec{t}', \vec{z}_0)) \wedge \gamma(\vec{t}, \vec{z}_0, S)$ (which takes $\Theta(1)$ time, excluding again the time that subsequently required to compute recursively $\mathcal{R}[\gamma(\vec{t}, \vec{z}_0, S)]$) and then replace $sp^*(\alpha, A(\vec{t}', \vec{z}_0))$ with the equivalent FO formula. Because this last replacement step can be considered as finding the path from α to A_0 with the computational complexity of $\Theta(\log_k n_A)$. Finally, regression makes the same number of recursive calls in both cases: the formula γ is the same. □

However, if we do not allow the usage of (in)equality between action terms (e.g., $a = A_i$) in $\mathcal{D}^H_{ss}$, we may (some-

times) lose computational advantages when the effects of actions for some fluents only involve very few actions in a large taxonomy or when the structure of a DAG is not a forest, especially if it is close to a complete DAG. Since a dense DAG of n nodes has at most the order of n^2 edges (a complete DAG of n nodes has $n(n-1)/2$ edges in total), computing the FO formula equivalent to the predicate sp^* has complexity $O(n^2)$. To avoid such computational disadvantages, we can easily use both (in)equality between action terms and predicate sp^* in modular BATs. Whenever the (sub)digraph rooted at some action function symbol A has a tree structure (even mostly a tree structure with a few extra edges) and most of its specializations have the same effects under certain common conditions on some fluent F, one can use the sp^* predicate for A to gain both the computational advantage and the advantage of compact representation when writing the SSA for F. Otherwise, one can use the (in)equality format to avoid computational disadvantages.

Now, we illustrate the advantage of using the negated component in clause (5) (i.e., allowing $h>0$).

Example 6 We continue with Example 5. Consider a fluent $nonBBQ(x,s)$: x is cooked without grilling. Its SSA in $\mathcal{D}^H$ can be written without negation in (5), i.e. when $h=0$:

$$nonBBQ(x,do(a,s)) \equiv (\exists y)[sp^*(a,oilyCook(x,y)) \vee sp^*(a,ovenCook(x,y)) \vee a=steam(x,y) \vee a=stew(x,y) \vee sp^*(a,boil(x,y))] \vee nonBBQ(x,s) \wedge \neg(\exists y) a \neq grill(x,y) \quad (6)$$

Alternatively, it can be written with negation (i.e., $h>0$) as:

$$nonBBQ(x,do(a,s)) \equiv (\exists y)[sp^*(a,oilyCook(x,y)) \vee sp^*(a,lowOilCook(x,y)) \wedge (\forall z)(a \neq lowOilCook(x,z) \wedge a \neq grill(x,z))] \vee nonBBQ(x,s) \wedge \neg(\exists y) a \neq grill(x,y) \quad (7)$$

Consider $\mathcal{E}[\mathcal{R}[nonBBQ(Egg_1, do(\alpha, S_0))]]$, extended regression where α is the same as in Example 5. It takes 1 step less using Formula (7) than using Formula (6) during regression (regardless the quantifiers). Clearly, the more branches $lowOilCook(x,y)$ has that have positive effects on $nonBBQ(x,s)$ without extra context conditions, the more computational advantage we can obtain by allowing $h \geq 0$ and using Formula (7) during regession.

How to Construct a Taxonomy of Actions

As we see, hierarchies of actions can lead to important computational advantages. An important practical question remains how an axiomatizer should approach the problem of constructing a hierarchy of actions given only a set of effect axioms which specify for each fluent what actions have a (positive or negative) effect on the fluent. In a somewhat similar vein, (Reiter 2001) starts from effect axioms and demonstrates that under the causal completeness assumption, SSAs can be constructed from effect axioms. We continue to consider only action diagrams $\mathcal{H}$ with monotonic inheritance of effects. In this subsection, we assume that all the variables used below in action functions and fluents are distinct from each other. Consider a BAT $\mathcal{D}$ which includes a set of n action functions, say $\{A_i(\vec{x}_i) \mid i = 1..n\}$, and m fluents, say $\{F_j(\vec{y}_j) \mid j = 1..m\}$, that might be affected by any of the above actions. For any action function $A(\vec{x})$, without loss of generality, we assume that its positive effect axiom for any relational fluent $F(\vec{y},s)$ has the syntactic form

$$\psi^+_{A,F}(\vec{x},\vec{y},s) \supset F(\vec{y}, do(A(\vec{x}),s), \quad (8)$$

and its negative effect axiom for $F(\vec{y},s)$ is of the form

$$\psi^-_{A,F}(\vec{x},\vec{y},s) \supset \neg F(\vec{y}, do(A(\vec{x}),s). \quad (9)$$

Definition 4 *For an action function $A(\vec{x})$ and a fluent $F(\vec{y},s)$, which has effect axioms of the form (8,9) we say that* an action A has effect on a relational fluent F (or F could be affected by A) *iff either* $\not\models \psi^+_{A,F}(\vec{x},\vec{y},s) \equiv F(\vec{y},s)$ *or* $\not\models \psi^-_{A,F}(\vec{x},\vec{y},s) \equiv \neg F(\vec{y},s)$. For any action function $A(\vec{x})$, a special meta-function $N_e(A)$ is used to represent the number of fluents that can be affected by A.

For any two action functions $A_1(\vec{x}_1)$ and $A_2(\vec{x}_2)$, we say that A_1 causes no less effects than A_2 *iff there exists no fluent such that A_1 has no effect on it but A_2 has. We say that* A_1 causes more effects than A_2 *iff A_1 has no less effects than A_2 and there exists at least one fluent such that A_1 has an effect on it but A_2 does not.*

Note that if A_1 causes more effects than A_2, then $N_e(A_1) > N_e(A_2)$; however, it is not necessarily true the other way around: actions might affect different sets of fluents. Given effect axioms, for any pair of actions A_1, A_2, a straightforward linear time $O(m)$ procedure can check whether A_1 causes more effects than A_2.

We would like to provide general guidelines on how an axiomatizer can construct an action diagram $\mathcal{H}$ for $\mathcal{D}$. Under the assumption of monotonic inheritance, if A_1 is a specialization of A_2, then it causes no less effects than A_2 and $N_e(A_1) \geq N_e(A_2)$. Thus, to return a set H that represents an action diagram $\mathcal{H}$, it is enough to start with generic actions A that have the smallest value of $N_e(A)$ and proceed towards more specialized actions checking on each iteration if the next action we consider is a specialization of one of the previously considered actions.

1. Sort the action functions and get the sequence $A_1(\vec{x}_1)$, $\cdots$, $A_n(\vec{x}_n)$ such that $N_e(A_{i_1}) \leq N_e(A_{i_2})$ for $i_1 < i_2$.

2. Initially, let $i=2$ (index $1 \leq i \leq n$) and $H = \emptyset$.

3. If $i > n$, then return H and terminate; else assign $j=i$ and continue: look for A_j's that are generalizations of A_i.

4. Decrement $j=j-1$. If $j=0$ (i.e., all candidates A_j have been already considered), then increment $i=i+1$ and go to step 3 (i.e., take a next action A_{i+1} from the sorted sequence we obtained at step 1); else if $N_e(A_i) = N_e(A_j)$, go to step 4; else continue.

5. For any pair of indices i, j such that $1 \leq j \leq i-1$, if there is a path in H from i to j, then we already know that A_i is a specialization of A_j and because specialization is a transitive relation there is no need to add a new directed edge from A_i to A_j and we can go to step 4; else continue.

6. If $A_i(\vec{x}_i)$ is a specialization of $A_j(\vec{x}_j)$ under FO condition $\phi_{i,j}$, then update $H = H \cup \{(A_i(\vec{x}_i), A_j(\vec{x}_j), \phi_{i,j})\}$ and go to step 4; else do not change H and go to step 4.

To implement the last step for any two action functions $A_i(\vec{x}_i)$ and $A_j(\vec{x}_j)$, provided that axiomatizers are able to write effect axioms of the form (8,9), we formulate the following principles to determine whether or not $A_i(\vec{x}_i)$ is a specialization of $A_j(\vec{x}_j)$ under some condition ϕ.

a. If A_i causes more effects than A_j, "guess" a FO formula ϕ, whose free variables include at most $\vec{x}_j$ and $\vec{x}_i$, such that for any relational fluent $F(\vec{y}, s)$ that could be affected by both A_i and A_j,

$$\mathcal{D} \models \phi \supset (Poss(A_i(\vec{x}_i), s) \supset Poss(A_j(\vec{x}_j), s)),$$
$$\mathcal{D} \models \phi \supset (\psi^+_{A_i,F}(\vec{x}_i, \vec{y}, s) \equiv \psi^+_{A_j,F}(\vec{x}_j, \vec{y}, s)),$$
$$\mathcal{D} \models \phi \supset (\psi^-_{A_i,F}(\vec{x}_i, \vec{y}, s) \equiv \psi^-_{A_j,F}(\vec{x}_j, \vec{y}, s)).$$

If one can find such ϕ, then A_i is a specialization of A_j under the condition ϕ.

b. Otherwise, A_i is not a specialization of A_j.

Note that for any action functions $A_1(\vec{x})$, $A_2(\vec{y})$ and FO formula ϕ, each $(A_1(\vec{x}), A_2(\vec{y}), \phi)$ in the returned set H corresponds to an axiom $sp(A_1(\vec{x}), A_2(\vec{y})) \equiv \phi$, and the collection of all these axioms results in an action diagram $\mathcal{H}$.

In general, to determine whether one action is a specialization of another under certain condition is undecidable. Hence, the axiomatizers have to observe the preconditions and effects of actions, guess formula ϕ (using their intuition), and construct action diagrams manually. In the future, we would like to consider whether it is possible to generate action diagrams automatically in some special cases.

Discussion and Future Work

There are a few papers related to our work that we would like to mention. (Lifschitz & Ren 2006) consider modular theories in the propositional action representation language $\mathcal{C}+$, and address the problem of the development of libraries of reusable, general-purpose knowledge components. In comparison to them, we explore how to manage a large number of actions in the predicate logic using a hierarchical representation for actions in SC. We propose a representation, which not only facilitates writing axioms succinctly, but for realistic taxonomies can also gain computational advantages in solving the projection problem. (Kautz & Allen 1986) and subsequent papers of H.Kautz develop frameworks for plan recognition using hierarchies of plans, in which primitive action and plan instances belong to certain event types represented as unary predicates, and a hierarchy of plans is a collection of restricted-form axioms specifying relationships between various event types. However, their axiomatizations of actions (preconditions and effects of the actions) are limited and they do not address the projection problem. (Kaneiwa & Tojo 2005) give an ontological framework to represent actions/events and their hierarchical relationships in information systems using an order-sorted SO logic. In this framework, events (or actions) are represented as predicates rather than terms, and the authors consider taxonomical reasoning about relationships between events rather than reasoning about effects of actions. The authors do not provide axiomatizations of the dynamic aspects of actions and do not explore computational properties of their framework. (Devanbu & Litman 1996) propose a *plan-based* knowledge representation and reasoning system, called CLASP (CLAssification of Scenarios and Plans). CLASP extends the notions of subsumption from terminological languages to plans by allowing the construction of plans from concepts corresponding to actions and using plan description forming operators for choice, sequencing and looping (similar to propositional dynamic logic). All actions in CLASP are represented in the style of STRIPS, which is less expressive than general Reiter's BATs and our modular BATs. Our formalism is very different from all the papers mentioned above. We use a specialization relation between primitive action functions, and provide a formal axiomatization of the dynamic aspects of actions using full predicate logic (hence, our theory is quite expressive). Also, we gain both representational and computational advantages by using the action hierarchies. The extensive research on Hierarchical Task Networks (HTN), that can be traced to the pioneering work of Sacerdoti on ABSTRIPS, considers a completely different recursive decomposition of complex actions (i.e., plans, or nonprimitve tasks) into constituents, but does not explore large taxonomies of primitive actions and whether these taxonomies can provide any computational advantages when solving the projection problem. Our work is motivated in part by the well-known hierarchies of verbs (full troponym) in WordNet (Fellbaum 1998). Exploring connections with other frameworks (e.g., FrameNet, Levin's taxonomy, VerbNet, etc) in computational linguistics and natural language processing is a possible direction for our future research. (Amir 2000) proposes and studies an object oriented version of the SC with the purpose of developing decomposed theories of actions, but he investigates a representation that is significantly different from our approach and considers neither taxonomies of actions nor BATs. In the future, we will explore how to combine Amir's decomposed SC theories with our action hierarchies. Moreover, currently we consider only hierarchies of primitive actions. In the future, we may also consider hierarchies of complex actions (plans), and explore what criteria should be followed for constructing such hierarchies, whether we can construct them automatically from the existing hierarchies of primitive actions and our BATs.

Acknowledgments

Thanks to the Natural Sciences and Engineering Research Council of Canada (NSERC) for partial financial support of this research. Thanks to Fahiem Bacchus, Hector Levesque, and Sheila McIlraith for comments on preliminary versions.

References

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

Devanbu, P. T., and Litman, D. J. 1996. Taxonomic plan reasoning. *Artif. Intell.* 84(1-2):1–35.

Fellbaum, C. 1998. English verbs as a semantic net. In Fellbaum, C., ed., *WordNet: An Electronic Lexical Database, with a preface by George Miller, Chapter 3*. The MIT Press. 69–104.

Kaneiwa, K., and Tojo, S. 2005. Logical aspects of events: Quantification, sorts,composition and disjointness. In *Proceedings of Australasian Ontology Workshop (AOW 2005)*.

Kautz, H. A., and Allen, J. F. 1986. Generalized plan recognition. In *Proceedings of the fifth National Conference on Artificial Intelligence (AAAI-86)*, 32–37. AAAI Press.

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.

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