(De)Composition of Situation Calculus Theories

Eyal Amir

Department of Computer Science,
Gates Building, 2A wing
Stanford University, Stanford, CA 94305-9020, USA
eyal.amir@cs.stanford.edu

Abstract

We show that designing large situation calculus theories can be simplified by using object-oriented techniques and tools together with established solutions to the frame problem. Situation calculus (McCarthy & Hayes 1969) is one of the leading logical representations for action and change, but large situation calculus theories are not easy to design and maintain, nor are they flexible for extension or reuse. However, we wish to use it to represent large, complex domains.

To address this problem, we apply our proposed methodology to situation calculus theories and analyze the composition of theories in this light. The object-oriented tools that we use do not change the semantics of situation calculus, so all the original situation calculus results apply in our setting and vice versa. We get two additional results from this approach. First, we offer a new treatment of loosely interacting agents that uses situation calculus without abandoning the *result* formalism. This treatment allows a theory-builder to construct a theory without considering its potential inclusion in a multiple-agents setup. Second, theories that we build in this way admit specialized reasoning algorithms.

Introduction

Our work has two related motivations: making large situation calculus theories easier to design and maintain, and giving a partitioned representation to situation calculus theories. Situation calculus theories have a restricted form and a clear ontology that together help modeling. However, using situation calculus in large, multi-domain settings is not easy. It is hard to think of all the parts of a large theory in advance, reuse is not simple, and modifications can have unintended consequences. Giving situation calculus theories a partitioned representation allows us to use the algorithms of (Amir & McIlraith 2000) and opens the way for merging situation calculus theories into reactive frameworks such as (Amir & Maynard-Reid 1999). This paper focuses on the first motivation and addresses the second as a by-product.

We present one object-oriented reformulation of situation calculus, following some traditional decompositions of theories of action (Shanahan 1997). We create a class for situations and a class for actions (roughly speaking, *classes* are theory fragments). *Subclasses* of situations (roughly, these

are supersets of the class of situations) contain domain constraints. Subclasses of actions contain effect axioms. We describe one theory constructed using objects of these and other classes. The combined theory has semantics that is equivalent to situation calculus, after translation to First-Order Logic (FOL). We apply Reiter's monotonic solution to the frame problem (Reiter 1991) to this system of objects and classes.

Object-oriented situation calculus theories can be combined easily and are elaboration-tolerant in that sense. We demonstrate this by combining domain theories and by giving a new treatment of agents in the situation calculus. Domain theories are connected by linking the corresponding objects with only a small vocabulary, even when the frame solution is used. This result extends to our proposed representation of agents. Our treatment of agents allows each agent's theory to be written as a simple theory, with no regard to the existence of other agents. Only new subtheories in which the agents interact need to consider the explicit existence of agents. We do this without giving up the *result* formalism (which is planning-friendly), as in (McCarthy & Costello 1998), or the independence of the agents (which allows efficient computation), as in (Reiter 1996), (Pinto 1998). We point out the applications and influence of the algorithms of (Amir & McIlraith 2000) on these theories. We conclude by proposing some guidelines for the knowledge engineering of situation calculus theories.

Our running example is a simplification of the *Daddy and Junior* scenario, taken from (McCarthy & Costello 1998).

Objects for Situations and Actions

Situation Calculus

Situation calculus (McCarthy & Hayes 1969) is a knowledge representation formalism for representing temporal information. The language consists of four sorts: *situations*, for situations in the world; *actions*, for events and actions; *fluents*, for situation-dependent properties; and *objects*, for simple other FOL objects.

s, a, f (or subscripted versions thereof) are variables for situations, actions and fluents, respectively. All other variables are of sort "object". The predicate $Holds(f, s)$ asserts that a fluent f holds in the situation s ($f(s)$ is used as a shorthand for $Holds(f, s)$ when such a shorthand causes

no confusion). The function $result(a, s)$ returns the situation that results from performing a in s (we use $res(a, s)$ as a shorthand). $S0$ is a situation constant. Figure 1 displays A_{bw}, a sample situation calculus theory for the blocks-world domain. Capitalized symbols are constants. Free variables in axioms are universally quantified with maximum scope.

$$on(A, B, S0) \land on(B, Table, S0) \land on(C, Table, S0)$$
$$on(x, y, s) \Rightarrow above(x, y, s)$$
$$above(x, y, s) \land above(y, z, s) \Rightarrow above(x, z, s)$$
$$clear(b, s) \iff \forall b' \neg on(b', b, s)$$
$$handEmpty(s) \iff \forall b \neg inHand(b, s)$$
$$clear(b, s) \land handEmpty(s) \Rightarrow inHand(b, res(pickUp(b), s))$$
$$(y = Table \lor clear(y, s)) \land inHand(x, s) \Rightarrow$$
$$on(x, y, res(putOn(x, y), s))$$
$$on(x, y, s) \Rightarrow \neg inHand(x, s) \land \neg inHand(y, s)$$
$$inHand(x, s) \Rightarrow \neg on(x, y, s) \land \neg on(y, x, s)$$
$$on(x, y, s) \land y \neq z \Rightarrow \neg on(x, z, s)$$

Figure 1: Blocks-world in the situation calculus: A_{bw}.

Object-Oriented FOL

We wish to make the design of situation calculus theories more structured. To this end, we adopt some object-oriented design principles and tools. In the rest of this section, we build an object-oriented situation calculus theory for the blocks-world. At the same time, we provide a few meta-logical tools and notations that are needed for this description, borrowing from (Amir 1999). Readers familiar with applications of context (e.g., (Ghidini & Serafini 1998)) can view the following as a special case having the semantics of FOL, object-oriented tools, and no explicit contexts.

An *object-oriented first-order logic (OOFOL) theory* is divided into subtheories associated with *objects*. A subtheory associated with an object has its own first-order language and axioms. We distinguish a subset of the vocabulary of each object and call it the object's *interface*. There are interface *links* between objects. Each interface link specifies equality/equivalence between symbols in two objects. Only symbols from these interfaces may participate in the link.

A situation calculus theory can be broken into an object (subtheory) associated with situations and an object associated with actions (additional other objects are possible). In the first, we put domain constraints (sentences that mention no action term). In the second, we put effect axioms (axioms of the form $Holds(\Phi, s) \Rightarrow Holds(\Psi, res(a, s))$). This decomposition for our blocks-world theory A_{bw} is diagrammatically presented in Figure 2. Here, the symbols $on, inHand, Holds$ are on the link between the objects and thus have the same semantics in both objects.

An OOFOL theory T is a set of object declarations (see below). The semantics is given by a translation to FOL: We replace every symbol in an axiom with the same symbol appended to the name of the object in which this axiom appears (making symbols in different objects distinct). For example, the symbol on used in a situation object $\mathcal{S}$ is translated to $\mathcal{S}.on$ in all the axioms of $\mathcal{S}$. Then, we add equality (or equivalence) axioms of the form $\forall \overrightarrow{x} (\mathcal{S}.P(\overrightarrow{x}) \equiv \mathcal{A}.P'(\overrightarrow{x}))$ for
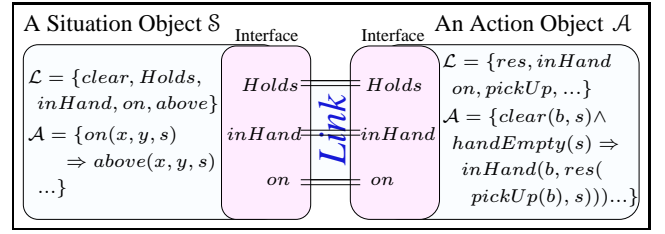

Figure 2: A situation object connected to an action object.

all linked symbols P, P' between objects $\mathcal{S}, \mathcal{A}$. The result of this translations is denoted $\widetilde{T}$.

Definition 1 (FOL Semantics for OOFOL) $\mathcal{M}$ *is a* model of T *iff* $\mathcal{M} \models \widetilde{T}$.

Situation Calculus Classes

We can now represent a theory as a set of linked objects. Presenting situation calculus theories in this form gives a designer a bird's-eye view of the theory and allows the combination of several theories by *linking* them. We also wish to reuse theories. For this, the notions of *classes* and *inheritance*[1] become handy. A class is a template for its objects, specified with no interface links, but otherwise identical to them. This allows the application of identical axiom sets in different positions in the theory by specifying the class to which the object belongs.

Inheritance extends this reuse by allowing some extensions to a copy of an existing class. For engineering purposes, inheritance in our system is treated outside the logic (unlike nonmonotonic inheritance, such as in (Morgenstern 1998)). In principle, a *subclass* (child) inherits all the axioms, vocabulary and interface vocabulary of its parent class. We can make the subclass different from its parent by: (1) adding to the inherited vocabulary and interface vocabulary; (2) adding axioms; and (3) explicitly replacing some non-logical symbols in all of the inherited axioms, with some new symbols in the child's language.

We use standard object-oriented methodology and define a hierarchy of classes for the blocks-world situation calculus. Figure 3 display our classes. We explain its contents together with the notation (most of the notation is similar to that used in object-oriented programming languages).

There are two root classes: **Situation** and **Action**. Both include interface declarations (to be inherited by subclasses) and no axioms. The notation *class1:class2* is used to declare that class1 is a subclass of class2. Classes **BWSit**,**BWAct** are subclasses of **Situation**,**Action**, respectively, specialized for the blocks-world. They inherit the respective interfaces, and add it to their explicitly declared interface. Since **BWS0** is a subclass of **BWSit**, it includes all the interface and axioms of its superclass, adding some observations about a particular $S0$.

---

[1]*Objects*, *classes* and *inheritance* are meta-logical notions.

```
class Situation {
    interface:    Holds }

class Action {
    interface:    res, Holds }

class BWSit : Situation {
    interface:    on, inHand
    axioms:       clear(b, s) ⟺ ∀b'¬on(b', b, s)
                  on(x, y, s) ⇒ above(x, y, s)
                  above(x, y, s) ∧ above(y, z, s) ⇒ above(x, z, s)
                  on(x, y, s) ⇒ ¬inHand(x, s) ∧ ¬inHand(y, s)
                  inHand(x, s) ⇒ ¬on(x, y, s) ∧ ¬on(y, x, s)
                  on(x, y, s) ∧ y ≠ z ⇒ ¬on(x, z, s) }

class BWAct : Action {
    interface:    on, inHand, pickUp, putOn, Table
    axioms:       clear(b, s) ∧ handEmpty(s) ⇒
                        inHand(b, res(pickUp(b), s)))
                  (y = Table ∨ clear(y, s)) ∧ inHand(x, s) ⇒
                        on(x, y, res(putOn(x, y), s))
                  clear(b, s) ⟺ ∀b'¬on(b', b, s)
                  handEmpty(s) ⟺ ∀b¬inHand(b, s) }

class BWS0 : BWSit {
    interface:    on, S0, A, B, C, Table
    axioms:       on(A, B, S0)∧on(B, Table, S0)∧on(C, Table, S0) }
```

Figure 3: Blocks-world situation calculus classes.

Modeling With One Object per Class

Using these classes, we create the theory $\mathcal{A}_{ob}$ shown in Figure 4. In this theory, $S0_{obj}$ is an object (instantiation) of class **BWS0**, S is an object of class **BWSit** and $\mathcal{A}$ is an object of class **BWAct**. The statement $Object(\mathbf{Cl}, \mathcal{O}, \{(P, \mathcal{O}', Q), ...\})$ declares an object named $\mathcal{O}$ of class **Cl** and specifies interface links with other objects: P in $\mathcal{O}$ is equal/equivalent to Q in $\mathcal{O}'$ (e.g., $\mathcal{O}.P \equiv \mathcal{O}'.Q$). $S0_{obj}, S$ are linked on $on, Holds$; $S, \mathcal{A}$ are linked on $on, inHand, Holds$; and $\mathcal{A}, S0_{obj}$ are linked on $Table$.

Our modeling can be summarized by the following two rules: (1) link all those symbols that should have the same intended meaning in two objects; and (2) omit links of symbols that follow from transitivity of equality (e.g., we omit $Holds, on$ from the link between $\mathcal{A}$ and $S0_{obj}$).

```
Object(BWS0, S0_obj, {(on, S, on), (Holds, S, Holds)}).
Object(BWSit, S, {(inHand, A, inHand),
    (on, A, on), (Holds, A, Holds)}).
Object(BWAct, A, {(Table, S0_obj, Table)}).
```

Figure 4: The OOFOL theory $\mathcal{A}_{ob}$

Figure 5 is a diagrammatic view of the structure of $\mathcal{A}_{ob}$ showing the three objects and the links between them. Each link is labeled with the symbols that are pronounced equal between the two objects (e.g., for the fluent $on(x, y)$, $\forall xy S.on(x, y) = \mathcal{A}.on(x, y)$). Almost all the examples we will see have identical symbols on both ends of a link (e.g., on in S and on in $\mathcal{A}$), so this convention is clear.

The OOFOL semantics (Definition 1) given to $\mathcal{A}_{ob}$ is equivalent to the semantics given to A_{bw} from Figure 1. The translation of $\mathcal{A}_{ob}$ into $\tilde{\mathcal{A}}_{ob}$ is syntactic, adding equality/equivalence axioms between linked symbols.
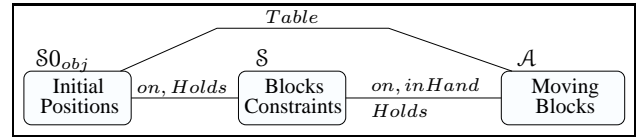

Figure 5: Diagrammatic view of $\mathcal{A}_{ob}$

Proposition 2 (Semantics) *Let $\varphi \in \mathcal{L}(A_{bw})$ and let φ' be its translation to $\mathcal{L}(\mathcal{A}_{ob})$. $A_{bw} \models \varphi$ iff $\mathcal{A}_{ob} \models \varphi'$.*

The translation of φ to $\mathcal{L}(\mathcal{A}_{ob})$ is straightforward. Symbols of φ that appear in more than one object have the same semantics in all of them, so we can arbitrarily pick one. For example, $A_{bw} \models \neg inHand(A, S0)$ iff $\mathcal{A}_{ob} \models \neg S.inHand(S0_{obj}.A, S0_{obj}.S0)$.

Reasoning With Objects

(Amir & McIlraith 2000) provides message-passing algorithms in the style of (Pearl 1988) for reasoning with theories that are decomposed in this way. Roughly speaking, if object S is connected to object $\mathcal{A}$, sentence φ_1 from $\mathcal{L}(S)$ can be logically combined (after translation) with a sentence φ_2 from $\mathcal{L}(\mathcal{A})$ only if $\mathcal{L}(\varphi_1)$ is included in the link.

More generally, given a graph of links and objects (as in Figure 5) and a query, we first decide on an object that can express the query in its vocabulary (if there is no such object, we can create one and link it to the others). Then, we transform the graph into a tree by running an algorithm called BREAK-CYCLES (we iteratively select a minimal cycle in the graph, remove a link and add its language to that of the other links in the cycle) and then perform MESSAGE-PASSING to prove the query, which roughly performs the following: Continuously perform consequence generation in each object. Send a proved formula (message) to another object only if the formula is in the vocabulary of the link to that object, and this other object is on the graph path to the object containing the query.

(Amir & McIlraith 2000) shows that this kind of algorithm is complete and sound for the semantics of Definition 1, and that it improves running time compared to standard deduction methods. For the propositional case, if T is a theory with n objects, each connected to d other objects in a tree structure, with each link having l symbols and $\mathcal{L}(T)$ has m symbols, then the running time for an analogous algorithm for SAT is $O(n * 2^{d*l} * f_{SAT}(m/n))$ (f_{SAT} is the time to compute SAT). This is an exponential time-improvement compared to standard reasoning techniques if the links' vocabularies are small.

We can now explain our modeling decisions from a computational perspective. To increase efficiency, we try to minimize the vocabularies of the links without giving up correctness of queries. This is why we omit the symbols $A, B, C, S0, res$ and several fluents from the links. Each symbol shows only in one object ($clear$ is defined by the fluent on in both $S, \mathcal{A}$). This is also the reason why we use the transitivity of equality to omit symbols from the links.

For a given domain theory (e.g., our blocks-world theory), $Holds$ and fluents that appear in the action object typically

appear on the link between the situation and action objects. This seems to limit the computational improvement due to such algorithms, and it seems to be an inherent property in theory structures of the form presented in Figure 5 (one object per class, connecting on $Holds$ and fluents). There are other structures that take advantage of an assumed tree of situations to give better performance (e.g., multiple objects, connected in a tree structure). We do not pursue these further here. Instead, our computational effort focuses on larger theories made of multiple domains, as described below.

The Frame Problem

The *frame problem* concerns the conclusion of non-effects of actions from the known effects in a concise, correct, expressive and elaboration-tolerant manner (see (Shanahan 1997)).

One of the simplest solutions generates *explanation closure* axioms (e.g., (Haas 1987), (Pednault 1989), (Schubert 1990)) from the effect axioms and the domain constraints. When it is possible, we mechanically (outside the logic) add axioms saying that if something has changed, then one of the enumerated actions occurred. For example, one axiom that is generated for explanation closure for $inHand$ in A_{bw} is $\neg inHand(x,s) \land inHand(x,res(a,s)) \Rightarrow (clear(x,s) \land handEmpty(s) \land a = pickUp(x))$.

(Reiter 1991) summarized the effort and showed how to generate such axioms automatically if there are no state constraints, (Lin & Reiter 1994) extended this process for the presence of state constraints, using deduction, and (McIlraith 2000) gave a closed-form solution in the presence of some restricted state constraints. These solutions also add other axiom sets, including unique names axioms (UNA) for sort "actions", preconditions for executing actions (summarized by the predicate $Poss(a,s)$) and foundational axioms for situations (here we call them *Peano axioms for situations*, although the time structure represented by situations is a tree rather than a line).

The diagram in Figure 6 presents the theory resulting from appending this solution to $\mathcal{A}_{ob}$, together with additional axioms for domain closure (DCA) and UNA for objects (these are not needed for the solution, but are sometimes assumed in domain theories). In this figure, $\mathcal{U}na_a, \mathcal{U}na_o \& Dca_o, \mathcal{P}oss, \mathcal{P}eano$ contain UNA for actions, UNA and DCA for objects, preconditions for execution and the Peano axioms, respectively. Our Peano axioms for situations include the first-order induction axiom $(\forall f)(Holds(f,S0) \land (\forall as)[Holds(f,s) \Rightarrow Holds(f,res(a,s))] \Rightarrow (\forall s)Holds(f,s))$ instead of the second-order axiom of (Lin & Reiter 1994).

The object $\mathcal{A}_p$ contains effect and explanation closure axioms with the following provision. To allow the introduction of the predicate $Poss(a,s)$, we need to make the effect axioms of **BWAct** dependent on $Poss(a,s)$. To do that, we create a subclass **BWPossAct** (shown in Figure 7) and replace the object $\mathcal{A}$ of the superclass with an object $\mathcal{A}_p$ of the subclass. The declaration *inherit:* $[Holds/Holds_{old}]$ is a directive to our compiler to replace the symbol $Holds$ with $Holds_{old}$ in all the axioms the class inherits from its superclass (as discussed above). This and the additional axiom adjusts our effect axioms to depend on $Poss$ (see Figure 7).
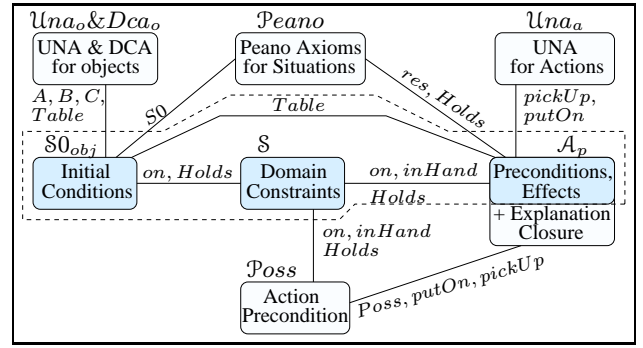

Figure 6: Frame-problem solution using objects.

Figure 7: Adding $Poss$ and explanation closure to **BWAct**.

Combining Action Theories

Now, we examine the way domain theories can be joined. Consider a domain theory regarding buying and selling blocks. In this theory, buying a block decreases the amount of money a robot has, but also places the purchased block in the robot's hand. Selling a block has the opposite effect.

We use two new fluents together with $inHand$: (1) $hasItem$ accounts for named items the robot possesses; and (2) $money$ is a function (fluent) that returns the amount of money that the robot has. Figure 8 presents some of the new classes. Figure 9 displays the diagram of the complete theory with the frame solution.

Figure 8: The domain of buying and selling blocks.

If the solution to the frame problem is not applied, we can combine the two domain theories by letting objects of corresponding positions in the two theories communicate. The two domains share only $S0, res, Holds$ and $inHand$. To see the situation diagrammatically, look at Figure 9 and consider only objects $\mathcal{S}, \mathcal{S}', \mathcal{A}, \mathcal{A}', \mathcal{S}0_{obj}, \mathcal{S}0'_{obj}$.

When we add the solution to the frame problem, there are two complications. First, the graph becomes more connected: UNAs potentially make all the objects dependent
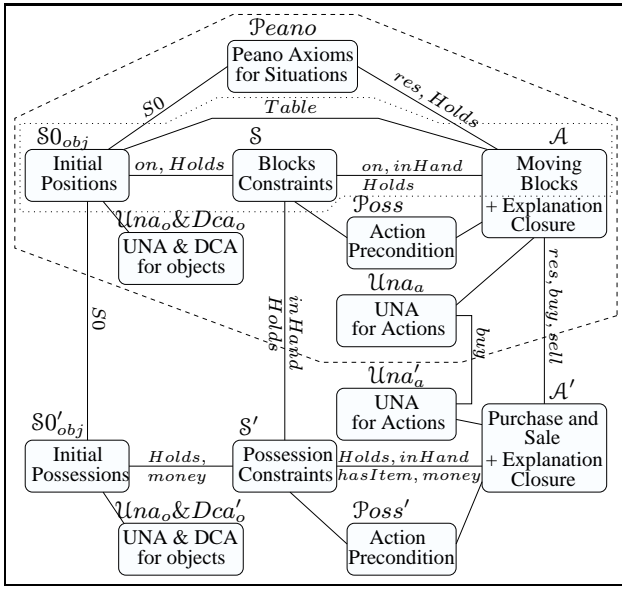
Figure 9: Connecting two domain theories.

on one another and explanation closure axioms add bandwidth to the links. Second, if we already have the solution for the separate domain theories, there are components that will have to be recomputed. These concerns are particularly important if we have a large number of connected domains.

Proposition 3 *For a theory made of many domain theories, if no qualification constraints are involved in computing $Poss$, then there is a formulation of the frame solution above (with or without UNA and DCA for objects) such that,*

1. *Any two domain theories that share fluents $\{f_i\}_{i \in I}$ are linked on exactly $\{f_i\}_{i \in I}, S0, res, Holds$ and actions $\{a_j\}_{j \in J}$ from the first domain that appear in the explanation closure for $\{f_i\}_{i \in I}$.*

2. *If we already have the frame solution for the separate domain theories, only the explanation closure axioms for $\{f_i\}_{i \in I}$ need to be recomputed (we ignore computations involved in solving the qualification problem (i.e., computing axioms for $Poss$)).*

3. *Domain theories are (directly) linked iff they share fluents or a removal of their links disconnects the graph.*

PROOF SKETCH   We use the structure depicted in Figure 9 to explain the case for two domains. Call the two domains 1 and 2. We need to formulate $Poss, Una_a, Una_o \& Dca_o$ such that they do not use any symbol that is in domain 2 and not in $\{f_i\}_{i \in I}, S0, res, Holds, \{a_j\}_{j \in J}$. This is enough, because So_{obj}, S, A are in the original domain axiomatization, for which this is trivially true, and $Peano$ uses only symbols $res, Holds, S0$ and the partial order $\prec$ on situations.

The axioms involving $Poss$ are always in the vocabulary of domain 1, following from our assumption that there are no qualification constraints. Also, we do not need to include DCA for actions, as there are no qualification constraints.

In each of Una_a, Una'_a we use the formulation of UNAs given in (McCarthy 1986). We define a total order, $<$, on action prototypes (this can be extended to a total order on actions, but we are not concerned with that here). Set an action prototype, A_2 from $\{a_j\}_{j \in J}$, to be the $<$-smallest in domain 2. Set another action, A_1 from domain 1, to be the $<$-largest in domain 1. Adding the axiom $A_1 < A_2$ to Una_a completes the needed specification for UNA.

Finally, UNA and DCA for objects can be formulated to be domain-dependent and also situation dependent (e.g., adding $block(x)$ as a precondition for axioms in domain 1). This concludes part 1, as the arguments above can be easily generalized to more than two domain theories.

For part 2, notice that DCA and UNA for objects and UNA for actions do not need to be recomputed when we link domain 2 to domain 1. When there are more than two domains involved, an increasing order on added domains can be used to guide axiomatizing $<$ on actions so that there are no inconsistencies (inconsistency may arise if we sanction $a < b < c < a$). Notice that if two domains share object symbols, then we must use a scheme that is similar to the one used for actions (sharing objects but not object symbols does not give rise to this problem).

For part 3, notice that all domain theories have the same semantics to $Holds, res, S0$ (the graph of domain theories is connected by our assumption). If two domain theories do not share a fluent, then they do not share an action (we assume that domains originally do not share actions, and they eventually need to share an action only as a result of explanation closure). This means that symbols that should have matching semantics in both domains already have matching semantics by virtue of the graph being connected. Thus, such domain pairs do not need to be directly connected.   ∎

Thus, adding the frame solution on top of the simple connection between the two domains adds to the links only actions participating in explanation closure of shared fluents. Figure 9 displays the complete theory of the blocks-world and buy-sell domains ($inHand$ is the only shared fluent). This result enables building composite theories like the one sketched in Figure 10 for a mobile robot that can perform electronic transactions, make phone calls, etc. Applying the reasoning algorithms described above to this theory results in an improved inference time, as the connectivity of this graph is kept low.

Combining Different Agents

We can represent different agents in the situation calculus by giving them different objects in the theory, and by adding an object to represent interactions. In the following, we assume that the agents' worlds do not interact, namely, that they deal with different fluents altogether (including the fluent $time$). We distinguish between actions that they perform separately (e.g., moving a block in their respective domains) and actions that influence the domains of both agents (e.g., money transfers between them). Those actions that influence both agents, are dealt with in a new interaction subtheory. In some cases we may choose to use the treatments of (Reiter 1996), (Pinto 1998) for this interaction subtheory, but in our
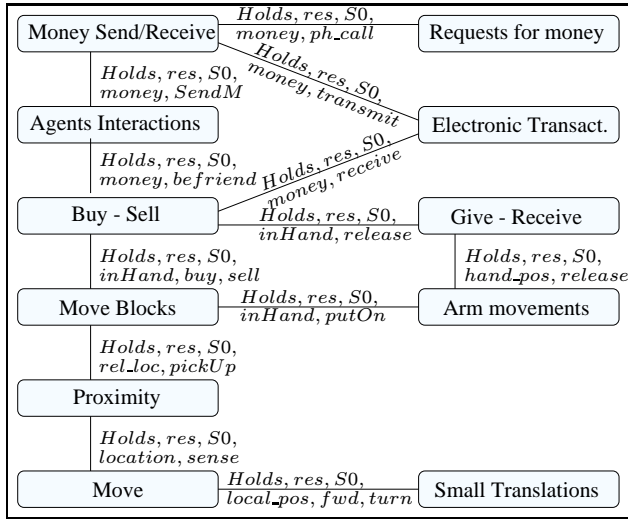
Figure 10: High-level diagrammatic view of a theory combining multiple domains

example we can do without them.

The approach we describe gives each agent a different perspective/knowledge of the same situation. It uses a single situations tree, but allows different perspectives on the *time* of a situation. Figure 11 is a diagrammatic view of a modification of *Daddy and Junior* (an example taken from (McCarthy & Costello 1998)). Daddy is stacking blocks of gold in New York, while Junior is stacking blocks of silver in London. If Junior needs more money, he may ask Daddy to send him some (requiring Daddy to sell a block). We omit the treatment for the frame problem here for clarity, but the approach used above works here too.
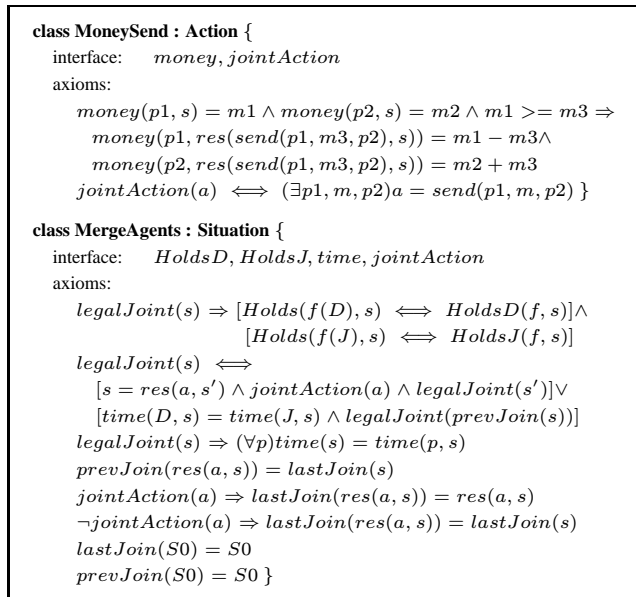

Figure 12: Merging perspectives on situations.

Figure 12 presents the necessary classes. **MoneySend** is an action class similar to those we have already seen. **MergeAgents** and the links to the object Agt of that class define our approach for merging the agents' theories (Figure 11), and we explain them below.

We care about time synchronization between the agents, so we assume some account of time in the different domain theories. We write $f(p)$ for the fluent that results from f after adding a dependency on p. The axioms of **MergeAgents** are explained as follows. First, if s is a legal joint situation ($legalJoint(s)$), then what holds in it ($Holds$) is exactly what holds in the different perspectives of the different agents ($HoldsD, HoldsJ$). If it is not a legal joint situation, we know nothing about it. Thus, the assumption that our agents' worlds do not interact is realized by giving each agent a different $Holds$ predicate.

The rest of the axioms in **MergeAgents** define legal joint situations. These axioms say that s is a legal joint situation iff the agents' perspectives of its history are consistent and their $time$ fluents are synchronized. The two perspectives on history are consistent only if all joint actions (actions for which $jointAction$ holds) in history were done in a legally joint situation. We do not care about actions done separately, as they cannot influence an agent other than the one who executed them. In particular, notice that time does not progress in one agent's perspective of the world as a result of the other agent executing an action.

For example, in Daddy's world ($\mathcal{S}_D$) the following is true after adding the explanation closure axioms for Daddy: $time(res(\mathcal{A}_J.pickUp(\mathcal{A}_J.A), S0)) = 0$. This is not a legal joint situation, because the agents' times are not synchronized. If we now add a *wait* action for Daddy and apply $\mathcal{A}_D.wait(t)$ with the proper time, a legal joint situation would result (in which they can perform $send(D, m, J)$).

Conditional Independence

Complete knowledge of the world of the links of $\mathcal{A}_J$ renders $\mathcal{A}_J$ independent of all other objects. Proof-theoretically, proving something in the language of $\mathcal{A}_J$ depends only on theorems proved in the language of its links to $\mathcal{S}_J$ and $\mathcal{A}'_J$ (theorems that may depend on further theorems proved elsewhere). This follows from Craig's interpolation theorem.

Theorem 4 ((Craig 1957)) *If $\alpha \vdash \beta$, then there is a formula γ involving only symbols common to both α and β, such that $\alpha \vdash \gamma$ and $\gamma \vdash \beta$.*

From a modeling point of view, this kind of conditional independence allows us to build an object considering only those objects that will interact with it. $\mathcal{A}_m$ of class **MoneySend** depends only on Agt of **MergeAgents** and $\mathcal{S}_m$ of **MultiHas** (a class that was not detailed here), which in turn care only about money and time. Daddy's world is independent of Junior's, given Agt. This allows the design of theories by first considering their interfaces and only then writing their details. The construction of every domain theory can be done separately, caring only about those predicted inputs from connected theories, which must be in the vocabulary of the interfaces. In particular, we never care about
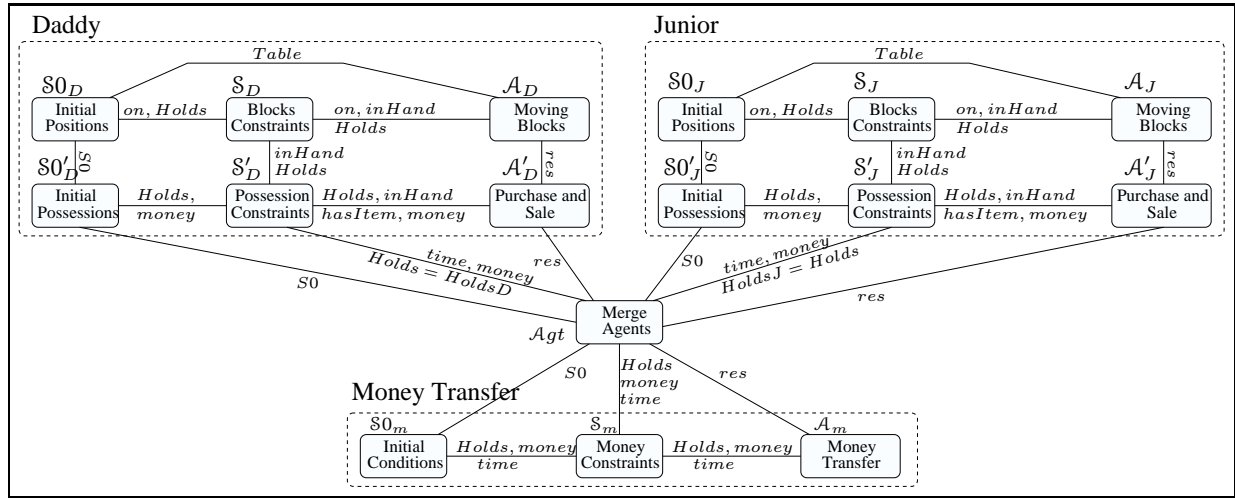
Figure 11: Diagrammatic view of Daddy and Junior in object-oriented situation calculus

theories that are independent from us, given our neighboring objects/domains.

This conditional independence is also utilized in the solutions to the frame problem. Explanation closure axioms help us make sure that actions in one domain do not influence domains that are not directly connected to it. Money transfers do not change the state of blocks on the table. This analysis can computationally enhance methods for computing solutions to the frame problem.

Finally, rewriting of theories is somewhat easier with this modeling technique. If a portion of the theory needs to be revised, the ramifications of this revision are easier to observe and predict. If the object structure is kept, then the influence of intra-object modifications can be predicted by checking the impact on the interface of the changed object/s.

Related Work

We focus our attention on object-oriented knowledge representation, as this material is spread out on many disciplines and the previous sections already included some references to related situation calculus literature. Many object-oriented knowledge representation systems are relevant to this work. However, systems available up until OOFOL (Amir 1999) either lack expressivity, lack object-oriented tools, extend the expressivity beyond the simple FOL semantics, or are not concerned with knowledge engineering at all.

The work on object-oriented Prolog (O-O Prolog) (e.g., (Shapiro & Takeuchi 1983)) provides limited expressivity and raises different challenges for object-oriented approaches than modeling in FOL. In contrast to FOL, O-O Prolog structures are built using a programming perspective, and the program flow in O-O Prolog is identical to the program flow in ordinary Prolog. Object-oriented structures in FOL supply heuristic information for theorem proving.

Another close object-oriented approach is Frame systems. Currently, there is no adequate object-oriented approach for full FOL in Frame systems. For example, Ontolingua (e.g.,

(Fikes, Farquhar, & Rice 1997)), one of the most expressive Frame systems to date, allows different sentences to be associated with each frame. However, it does not differentiate between a sentence put in one frame and the same sentence put in a different frame. The situation is similar in other and earlier frame systems such as KRYPTON/KL-ONE (Brachman, Fikes, & Levesque 1983),(for a recent survey see (Fridman-Noy & Hafner 1997)). Logical systems examined in the related field of *description logics* (e.g., (Borgida 1996)) are concerned with tasks related to classification in languages that is typically restricted (some systems can express full FOL theories, but in an unnatural way after translation).

Object-Oriented Databases are conceptually different from our approach. Logics for Object-Oriented Databases (e.g., (Kifer, Lausen, & Wu 1995)) try to give specific semantics to object-oriented databases. They focus on specific languages that allow us to reason about object membership in a class, class inheritance and properties of objects. In our work we do not try to reason about objects, but rather use the object-oriented technology to design large logical theories.

Approaches to formalizing and using context (e.g., (McCarthy & Buvač 1998; Guha & Lenat 1990)) typically have a significantly more expressive language, therefore requiring a more elaborate semantics. Furthermore, no object-oriented tools are supplied in any application of context known to the author (but some of it is hinted in (Ghidini & Serafini 1998)).

Finally, Bayes Nets (Pearl 1988) and Object-Oriented Bayes Nets (Koller & Pfeffer 1997) are closely related in spirit to our development here. Some of the ideas presented here came up while observing the work done in these fields.

Conclusions

In this paper we presented three results. First, we presented object-oriented techniques and tools that make designing large situation calculus theories simple. Conditional independence, low-bandwidth links, information hiding and encapsulation (looking at one subject at a time) are powerful

tools in designing large theories in general and situation calculus theories in particular.

Second, we showed that it is possible to build situation calculus theories separately and then associate them with loosely interacting agents. The builder of a simple theory does not need to predict its use in a multiple-agents setup. Thus, agents can be treated in the situation calculus without abandoning the *result* formalism. We made an implicit assumption that the agents' separate actions are independent and had their joint actions explicitly stated as such.

Third, our theories can be designed with low-bandwidth links between interacting domains. This leads to good performance expectation for reasoning with these theories.

For simple theories, object-oriented structures are sometimes excessive. This situation is familiar from other fields that use object-oriented techniques. However, it seems that large theories are difficult to build and comprehend without such structures. In this paper we adjusted this methodology to the modeling of situation calculus theories. We hope that the results we presented here will support the scaling up of logical theory engineering.

Acknowledgments

References

Amir, E., and Maynard-Reid, P. 1999. Logic-based subsumption architecture. In *Intl' Joint Conf. on Artificial Intelligence (IJCAI'99)*.

Amir, E., and McIlraith, S. 2000. Paritition-based logical reasoning. In *Proc. KR'2000*. Available at http://www-formal.stanford.edu/eyal/papers/oo-reason.ps.

Amir, E. 1999. Object-Oriented First-Order Logic. Technical report, NRAC'99. Submitted to Elect. Trans. on AI (http://www.ida.liu.se/ext/etai)/RAC area.

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

Brachman, R. J.; Fikes, R. E.; and Levesque, H. J. 1983. KRYPTON: A functional approach to knowledge representation. *Computer* 16(10):67–73.

Craig, W. 1957. Linear reasoning. a new form of the herbrand-gentzen theorem. *Journal of Symbolic Logic* 22:250–268.

Fikes, R.; Farquhar, A.; and Rice, J. 1997. Tools for assembling modular ontologies in ontolingua. In *Natl' Conf. on Artificial Intelligence (AAAI '97)*, 436–441. AAAI Press.

Fridman-Noy, N., and Hafner, C. D. 1997. The state of the art in ontology design. *AI Magazine* 18(3):53–74.

Ghidini, C., and Serafini, L. 1998. Distributed first order logic. In *Proceedings of Frontiers of Combining Systems*.

Guha, R. V., and Lenat, D. B. 1990. Cyc: A midterm report. *AI Magazine* 11(3):33–59.

Haas, A. R. 1987. The case for domain-specific frame axioms. In *proc. workshop on the frame problem*, 343–348.

Kifer, M.; Lausen, G.; and Wu, J. 1995. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM* 42(4):741–843.

Koller, D., and Pfeffer, A. 1997. Object-oriented Bayesian networks. In *Proc. of the 13th Conf. on Uncertainty in Artificial Intelligence (UAI-97)*, 302–313. Morgan Kaufmann.

Lin, F., and Reiter, R. 1994. State constraints revisited. *Journal of Logic and Computation, Special Issue on Actions and Processes*.

McCarthy, J., and Buvač, S. 1998. Formalizing Context (Expanded Notes). In *Computing Natural Language*, volume 81 of *CSLI Lecture Notes*. CSLI, Stanford University, Stanford, California. 13–50.

McCarthy, J., and Costello, T. 1998. Combining Narratives. In *Proc. KR'98*.

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

McCarthy, J. 1986. Applications of Circumscription to Formalizing Common Sense Knowledge. *Artificial Intelligence* 28:89–116.

McIlraith, S. 2000. Integrating actions and state constraints: a closed-form solution to the ramification problem (sometimes). *Artificial Intelligence* 116(1-2):87–121.

Morgenstern, L. 1998. Inheritance comes of age: Applying non-monotonic techniques to problems in industry. *Artificial Intelligence* 103(1–2):237–271.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann.

Pednault, E. P. D. 1989. ADL: exploring the middle ground between STRIPS and the situation calculus. In *Proc. KR'89*, 324–332.

Pinto, J. 1998. Concurrent actions and interacting effects. In *Proc. KR'98*.

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

Reiter, R. 1996. Natural actions, concurrency and continuous time in the situation calculus. In *Proc. KR'96*.

Schubert, L. K. 1990. Monotonic solution of the frame problem in the situation calculus. In *knowledge representation and defeasible reasoning*. Kluwer. 23–67.

Shanahan, M. 1997. *Solving the Frame Problem, a mathematical investigation of the common sense law of inertia*. Cambridge, MA: MIT press.

Shapiro, E., and Takeuchi, A. 1983. Object oriented programming in concurrent prolog. *New Generation Computing* 1:25–48.