Object-Oriented First-Order Logic

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

Abstract

First-Order Logic (FOL) provides richness of representation, but is unwieldy in modeling large, complex tasks. On the other hand, the Object-Oriented paradigm has been successful in treating large, complex tasks in both software and knowledge engineering. This paper describes a novel logic that extends FOL to create counterparts of the Object-Oriented paradigm. We call this logic Object-Oriented First-Order Logic (OOFOL). In this logic, objects are theories that are connected via interface vocabularies to other objects, classes are used to provide a reusable logical template, and inheritance is used to adapt classes to specialized tasks. We show that such a logic can be given simple semantics using FOL. A variant of the logic that allows some links between the objects to be unidirectional is also examined. This variant is called Directed Object-Oriented First-Order Logic (Directed OOFOL). It is shown that such a system can be given semantics using the nonmonotonic method of Circumscription.

Inference algorithms for the logics are proposed. It is shown that decomposing theories into objects enables construction of large theories while maintaining high-performance inference. Comparing the two logics we observe that inference in Directional OOFOL is sometimes faster than inference in OOFOL. The utility of the new logics is demonstrated through a few examples.

1 Introduction

First-Order Logic (FOL) provides a richness of representation, but does not necessarily make change and reuse easy. The difficulty of building and reusing first-order theories is visible in the obstacles encountered in different domains, such as Commonsense Reasoning, Knowledge Reuse and Formal Verification. On the other hand, the Object-Oriented paradigm has proved successful in facilitating software engineering (Meyer 1997), database engineering (Lausen & Vossen 1998) and knowledge engineering (e.g., (Brachman, Fikes, & Levesque 1983), (Lenat & Guha 1990)).

In this paper, we propose a novel logic that has expressivity that includes that of FOL and uses object-oriented structures. We show that this logic can be given simple semantics using FOL. A second logic that also allows for directionality of inference is shown to be semantically characterizable using Circumscription. In these logics, classes are theories equipped with interfaces, objects are "copies" of their classes with links to other objects, and inheritance is used to define classes that specialize their parents. We show the existence of efficient inference algorithms for both logics.

Many approaches to the combination of Object-Oriented ideas and logical Knowledge Representation (KR) exist: Frame systems (e.g., KRYPTON (Brachman, Fikes, & Levesque 1983)), Context Systems (e.g., CycL (Lenat & Guha 1990)) and Object-Oriented Prolog dialects (e.g., Prolog++ (Moss 1994)). We compare these approaches with ours at the end of this paper. Before that, Section 2 describes the motivation and structures embodied in our Object-Oriented logics, Section 3 describes our syntax and semantics, Section 4 describes several applications of our logics and Section 5 describes entailment methods.

Throughout this paper, we treat logic as an engineering tool and view it through the eyes of an engineer. This approach influences our design decisions, as will be evident. All proofs are omitted for lack of space. They are available in the full version (Amir 1999).

2 An Object-Oriented Approach to Logic

The intuition behind our approach to Object-Oriented logic is that a theory can be presented as a collection of theories communicating with each other via *interface vocabularies*. From this point of view, *a theory is a graph of smaller theories* (the graph is possibly directed). If the interface vocabularies are small, there is a significant performance improvement due to the algorithms that can be applied, compared to "flat" theories. Also, directionality of such interfaces allows us to restrict inconsistencies from influencing other parts of the theory. Modifications in the internals of these theories have only local impact, an impact which is circumscribed by the interface vocabularies.

To make our discussion more concrete, let us look at a simple example. The *Cup* object, depicted in Figure 1 below, describes some of the characteristics of a typical cup. As with other objects, it has a language/vocabulary, a set of axioms, an interface vocab-

ulary and a set of links to other objects. In their simple form (non-directional), the links specify equalities between symbols in the object's interface vocabulary and other objects' interface vocabulary symbols.

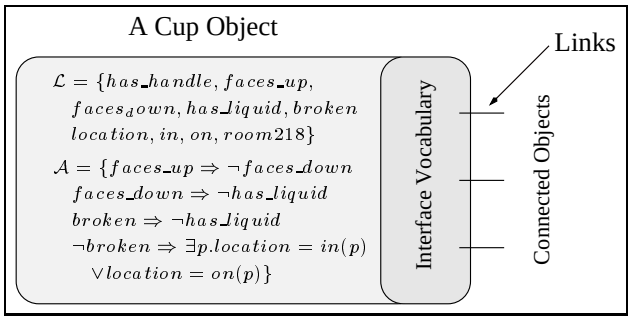

Figure 1: A Cup object.

The workings of the object are best explained in terms of the intended inference procedure. In the Cup object we may prove that the cup is not facing down, given that it has liquid inside. The interface vocabulary and the links are then used to communicate this fact (the validity of our conclusion) to the connected objects. If the propositional symbol $faces_down$ is part of the interface vocabulary, then those objects that are *linked* to this symbol will get the axiom $\neg faces_down$. Other objects that do not have a link to this symbol will not get this axiom sent to them. There is nothing communicated if $faces_down$ is not in the interface vocabulary.

More generally, all objects prove consequences in parallel, sending results to those other objects interested in them. These results are then incorporated into the theorem-proving process of the receiving objects. Axioms sent from object to object are restricted to the communication vocabulary, but the language is full FOL. For example, if an object *Liquid1* has links to $has_liquid, location, in$ in the *Cup* object, then sentences like $has_liquid \Rightarrow \exists p\ location = in(p)$ will be sent to *Liquid1*. This situation is shown in Figure 2.

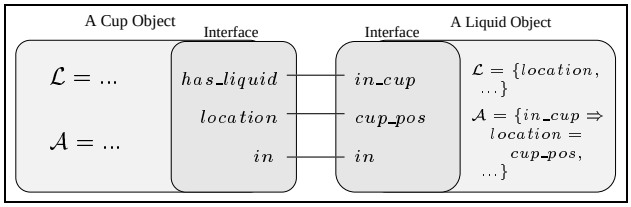

Figure 2: A Cup object connects to a Liquid object.

The use of fixed interfaces allows encapsulation of the internals of objects. This way, we can add or replace internal axioms with higher-grain axioms, describing the type of liquid, the heat, the degrees of tilt of the cup or the material of which the cup is made. Since we can expect the interface to remain the same, we can verify the change by examining changes to results in the interface, paying less attention to other internal results.

To allow reuse of theories and following Object-Oriented Design tradition, we define a *class* to be a "template" of objects. A class can be thought of as an object without specified links. Alternatively, an object is simply a copy of its class with links to other objects. We allow different uses of two objects of the same class in the same theory (linked to possibly different objects).

Object-Oriented approaches provide modularity through *inheritance*. Inheritance is a tool that is heavily used in many KR systems and is sometimes reasoned about using nonmonotonic reasoning systems. Since we are interested in OOFOL as an engineering tool, we prefer to implement inheritance of one class from its parent class outside the logic. Figure 3 describes Covered_Cup, a subclass of Cup, underlining the inherited part.

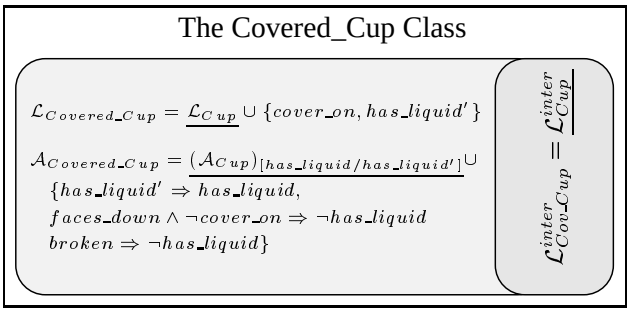

Figure 3: Covered_Cup is a subclass of the Cup class.

A subclass inherits all the axioms and vocabulary of its superclass, but may add symbols to the vocabulary and the interface vocabulary and may also add axioms. To enable subsumption of axioms we allow replacing symbols used in the superclass with new symbols (i.e., subsumption of axioms is done by subsuming symbols[1]). In our example, the symbol has_liquid must be substituted by has_liquid' in all of $\mathcal{A}_{Cup}$ to allow for a new definition for has_liquid.

This kind of inheritance is similar to inheritance in Object-Oriented programming languages, where the interface of the subclass must include the interface of the superclass, but may "implement" some of the internals differently. This definition of an is-a hierarchy over classes allows us to use a class as an abstracted version of its subclasses, adding more detail in the subclass if needed.

3 Object-Oriented FOL

We now define Object-Oriented Logics more formally. We show that the object-oriented structures we formally define can have intuitive FOL semantics, if the links are bidirectional. We also show that if the links

[1] This enables reusing superclass results in the subclass.

are directional, we can give an intuitive semantics to the system using Circumscription.

In what follows, we use some notations that are common in object-oriented programming languages, namely, we use $Obj.c$ (or $N.c$) to refer to the symbol c in the object Obj (with the name N). For simplicity of exposition, we shall not define classes and objects recursively (i.e., classes whose theories are themselves OOFOL theories). A recursive definition may sometimes be useful, but it requires only a simple technical step and makes our notations significantly more dense. Thus, this definition is omitted, but we note that all the results that follow hold for this more general case.

3.1 Language

First, we define our language: classes and objects, interfaces, links and theories. Unless otherwise mentioned, all languages mentioned are full first-order.

A *Class* C is a tuple $\langle \mathcal{L}_C, \mathcal{A}_C, \mathcal{I}_C \rangle$ corresponding to an FOL vocabulary ($\mathcal{L}_C$), a set of FOL axioms ($\mathcal{A}_C$) in this language, and $\mathcal{I}_C \subset \mathcal{L}_C$ the interface vocabulary of C.

An *OOFOL Theory* T is a set of statements

$$Object(C, N, L),$$

(defining an object) where C is a class reference (a class description as above, or a reference to such a description), N is a unique name for the object (typically a string of symbols) and L is a linking relation $L = \langle (P_i, N_i, Q_i) \rangle_{i \le n})$ which specifies the links between this object and other objects in the theory. Here P_i is a symbol in the interface vocabulary of C, N_i is a name of another object in T and Q_i is an interface symbol of this other object. This way an object is simply "declared" to be in the theory. We assume that no such statement in T refers to a name N_i (in the linking relation) for which no corresponding object exists in T.

3.2 FOL Semantics

Unlike object-oriented approaches to logic that use context (see section 6), we take advantage of the limited expressivity of our language (we cannot quantify over objects) and the limited interaction between our objects to give a simple FOL semantics to OOFOL theories.

We define the semantics by making objects have mutually exclusive vocabularies and viewing the links as equality/equivalence assertions. For an OOFOL theory T, define $\mathcal{L}(T) = \{N.P \mid Object(C, N, L) \in T,\ P \in \mathcal{L}_C\}$. This language gives a unique vocabulary for each object in T. We apply a simple translation into this vocabulary (replacing P by $N.P$ in the axioms of the object N for each symbol P in its vocabulary) and add the equality/equivalence assertions:

$$T_\flat = \{(\mathcal{A}_C)_{[P/N.P \mid P \in \mathcal{L}_C]} \mid Object(C, N, L) \in T\} \cup \{\forall \vec{x} (N.P(\vec{x}) \equiv N'.P'(\vec{x})) \mid Object(C, N, L) \in T,\ L(P, N', P')\} \quad (1)$$

where "$\equiv$" means "$=$" if P, P' are function symbols, and $\vec{x}$ may be of arity 0 (if P, P' are propositional or constant symbols).

An *OOFOL structure* in the language $\mathcal{L}(T)$ is structure of $\mathcal{L}(T)$ as an FOL language.

Definition 3.1 (FOL Semantics for OOFOL) *Let T be an OOFOL theory and let $\mathcal{M}$ be an OOFOL structure of $\mathcal{L}(T)$. A formula $\varphi \in \mathcal{L}$ is satisfied in $\mathcal{M}$, written $\mathcal{M} \models \varphi$, iff $\mathcal{M}$ satisfies φ in the FOL sense.*

$\mathcal{M}$ is a model of *T, written $\mathcal{M} \models T$ iff $\mathcal{M} \models T_\flat$ (in the FOL sense).*

In other words, the semantics of an OOFOL theory as an undirected graph of connected theories has the links state equality/equivalence between symbols and otherwise have the theory behave like an FOL theory.

EXAMPLE If T is the OOFOL theory containing the two connected objects from Figure 2, then

$$\begin{aligned} T = \{&Object(Cup, \text{“}Cup1\text{”}, \langle \\ &(has_liquid, \text{“}Liquid1\text{”}, in_cup), \\ &(location, \text{“}Liquid1\text{”}, cup_pos), \\ &(in, \text{“}Liquid1\text{”}, in)\rangle), \\ &Object(Liquid, \text{“}Liquid1\text{”}, \langle\rangle)\} \end{aligned}$$

The equality/equivalence sentences that we add to define $T_\flat$ and the semantics for T are:

$$\begin{aligned} &Cup1.has_liquid \equiv Liquid1.in_cup \\ &Cup1.location \equiv Liquid1.cup_pos \\ &\forall x (Cup1.in(x) = Liquid1.in(x)) \end{aligned}$$

Notice that the definition of $Liquid1$ above did not include any links, because the links were already specified for $Cup1$. ■

3.3 Directional Semantics

The previous section gave semantics that is sufficient to represent an object-oriented system that has no direction associated with the links in the objects graph. From a procedural perspective, this approach corresponds to allowing influence to flow in both directions of a link. In Figure 2, if we proved $Cup1.has_liquid$, then we can infer $Liquid1.in_cup$ and vice versa.

Sometimes, we would like to have the influence go in only one direction. For example, if we want to allow different accuracies of reasoning in two connected objects O_1, O_2. *Accuracy* here means that although something of interest to O_1 may come up in O_2, we prefer to have O_1 ignore that. Abstractions and approximations are examples of such differences in accuracy. This ability to to ignore influence is of particular importance in time-dependent tasks.

To give directional semantics to OOFOL theories, we convert the equivalence/equality axioms that we added in section 3.2 into defaults. This is done simply by adding *abnormality predicates* in front of each link equality/equivalence. We later *Circumscribe* these abnormalities. The circumscription will be done for each

pair of linked-objects separately. Thus, we write the corresponding formula to (1) for each pair separately:

$$\mathcal{A}_\flat(O_1,O_2) = \{(\mathcal{A}_{C_i})_{[P/N_i.P|P\in\mathcal{L}_{C_i}]} \mid i=1,2\}\cup \{\forall\overrightarrow{x}(Ab_{N_1.P_1,N_2.P_2}(\overrightarrow{x}) \Rightarrow N_1.P_1(\overrightarrow{x}) \equiv N_2.P_2(\overrightarrow{x})) \mid L_2(P_2,N_1,P_1)\} \quad (2)$$

where $O_i = Object(C_i, N_i, L_i)$.

We use Circumscription (McCarthy 1986) for our semantics. Circumscription is a method for nonmonotonic reasoning where, given a set of axioms T, the extensions of some predicate symbols $\overrightarrow{P}$ are minimized while allowing some other symbols $\overrightarrow{Q}$ to vary their semantics. This is achieved by replacing T with the second-order formula (or FOL schema) $Circ[T;P;Q]$:

$$T \wedge \forall p,q[T(p,q) \wedge p \leq P \Rightarrow p = P].$$

In the following semantics, to have influence flow only in one direction, we circumscribe the abnormality predicates Ab_{O_1,O_2} (the set of all abnormality predicates in $\mathcal{A}_\flat(O_1,O_2)$) in the axioms of the object-pair, $\mathcal{A}_\flat(O_1,O_2)$. We allow only the symbols of O_2 to vary, thus allowing influence to flow from O_1 to O_2, but not vice-versa.

Definition 3.2 (Directional Semantics) *We say that $\mathcal{M}$ is a Directed OOFOL model of an OOFOL theory T, written $M \models_D T$, iff it is a model of*

$$\bigwedge_{O_1,O_2\in T} Circ[\mathcal{A}_\flat(O_1,O_2); Ab_{O_1,O_2}; \mathcal{L}(O_2)]$$

We define $M \models_D \varphi$, for a formula $\varphi \in \mathcal{L}(T)$, and $T \models_D \varphi$, for an OOFOL theory, as usual.

Roughly speaking, in this definition, the links of O_2 serve as *inputs* and the links of O_1 serve as *outputs*. This distinction is made precise by allowing the circumscription to vary only $\mathcal{L}(O_2)$, holding $\mathcal{L}(O_1)$.

EXAMPLE We are going to show that influence between the objects depicted in figure 4 flows only in the arrow's direction. As before, the link $L_2(B,"O",A)$

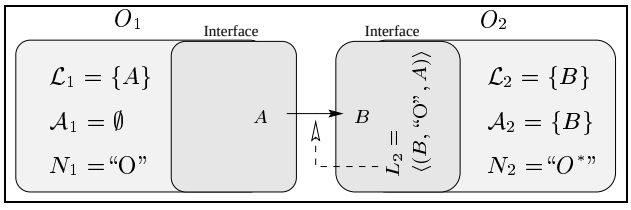

Figure 4: Two directionally linked objects. Influence flows from O_1 to O_2.

roughly says "link my B to O's A". Proving $O_2.B$ does not allow us to prove $O_1.A$.

$$Circ[B \wedge (\neg ab \Rightarrow (B \iff A)); ab; B] \not\models A$$

This is because the only models (in the propositional language $\{A,B\}$) that satisfy $B \wedge (\neg ab \Rightarrow (B \iff A))$ are $\{M_1 = \{\neg A, B, ab\}, M_2 = \{A, B, \neg ab\}, M_3 = \{A, B, ab\}$. M_1, M_2 are not comparable by the preference relation of our circumscription, because they differ on A (which is held constant in our circumscription). M_2 is preferred to M_3 and thus we have two minimal models of our circumscription, one of which does not satisfy A. ■

3.4 Inheritance

For OOFOL, there are four differences that we wish to allow between the child and the parent classes: (1) the child's language is an expansion of the parent's; (2) the child class inherits all the axioms of the parent, subject to replacement of non-logical symbols (in all of the axioms in the same way) to some of the new symbols in the child's language; (3) the child class may have additional axioms; and (4) the interface language may be expanded.

This way, we inherit all the class's symbols, but some of the symbols may be subsumed/redefined by new symbols (intending the new ones to build on the old ones). The *body* of each class (the axioms of that class) will still be valid in the new subclass, but its influence on the interface symbols will be different, depending on the changes we introduce to the language.

Formally, for a set of axioms A and non-logical symbols c, c', $A_{[c/c']}$ is the set of axioms A with every occurrence of the symbol c replaced by an occurrence of the symbol c'. $A_{[c_1/c'_1,\ldots,c_n/c'_n]}$ is similarly defined.

A class $C = \langle\mathcal{L},\mathcal{A},\mathcal{I}\rangle$ is a *subclass* of $C' = \langle\mathcal{L}',\mathcal{A}',\mathcal{I}'\rangle$ if $\mathcal{L}' \subseteq \mathcal{L}, \mathcal{A}'_{[c_1/c'_1,\ldots,c_n/c'_n]} \subseteq \mathcal{A}$ $(n \geq 0)$ and $\mathcal{I}' \subseteq \mathcal{I}$.

A detailed example is provided in Section 4.1.

4 Applications

In this section we briefly describe a few examples demonstrating the suggested uses of our modeling language. Some of the examples were abstracted for lack of space.

4.1 Model-Based Reasoning

Below, we give a fragment of a theory describing the functionality of a car. We use qualitative proportionalities (see (Forbus 1996)), to represent the relationships among different variables of the car (in our case these are monotonically increasing and monotonically decreasing functions). Figure 5 describes some of the classes for this domain. The notation *class C1 : C2* serves to point that $C1$ is a subclass of $C2$.

Our theory fragment is built from these classes as follows:

$Object(Brakes, Brakes1, \langle\rangle)$.
$Object(Tires, Tires1,$
$\quad \langle(brakes_force, Brakes1, apply_force)\rangle)$.
$Object(Car, Car1,$
$\quad \langle(tire_force, Tires1, apply_force),$
$\quad\ (bad_traction, Tires1, bad_traction)\rangle)$.

class Qualitative_Proportionalities {
axioms: $x > y \Rightarrow rel_pos(x) > rel_pos(y)$
$x > y \Rightarrow rel_neg(x) < rel_neg(y)$ **}**

class Brakes : Qualitative_Proportionalities {
interface: $apply_force(x)$

axioms: $pedal_pressure(x) \wedge oil_pressure(y) \Rightarrow$
$apply_force(rel_pos(x) * rel_pos(y))$
$pedal_pressure(0) \Rightarrow apply_force(0)$
$oil_pressure(70)$ **}**

class Tires : Qualitative_Proportionalities {
interface: $bad_traction(wheel), apply_force(x),$
$engine_force(x), brake_force(x)$

axioms: $brake_force(x) \wedge engine_force(y) \Rightarrow$
$apply_force(rel_pos(y) - rel_pos(x))$
$\forall wheel.(\neg balanced(wheel) \vee old(wheel) \Rightarrow$
$bad_traction(wheel))$
$\forall wheel.(balanced(wheel) \wedge new(wheel))$
$\forall x.new(x) \iff \neg old(x)$ **}**

class Car : Qualitative_Proportionalities {
interface:
$tire_force(x), bad_traction(wheel)$
axioms:
$\exists w.bad_traction(w) \wedge tire_traction(t) \Rightarrow t < Good$
$\forall w.\neg bad_traction(w) \wedge tire_traction(t) \Rightarrow$
$t = Good$
$tire_force(f) \wedge tire_traction(t) \wedge speed(s) \Rightarrow$
$stop_time(rel_neg(s) * rel_pos(f) * rel_neg(t)).$
}

Figure 5: Some of the classes in the CAR theory.

The situation is semantically/diagrammatically depicted in Figure 6. The reason for using directional

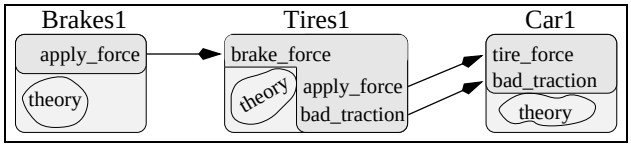

Figure 6: Diagrammatic structure of the CAR theory fragment.

semantics here instead of the simpler undirected semantics (FOL semantics) is that we wish to enforce the direction of influence. Nothing the tires do can change the force applied to the brakes. Nevertheless, this choice of semantics is a matter of specific application at hand, and one can easily find an application where this assertion is FALSE (e.g., if a tire explodes, that may influence the force applied on the brakes). In that case, we should take the FOL semantics instead.

Using this theory, we can prove that increasing the pressure on the brakes will make the car stop faster, if we keep the rest constant (e.g., not increase the pressure on the accelerator).

If we wish to specialize our knowledge of the Brakes system, we add the class **ABS** as a subclass to **Brakes**.

class ABS : Brakes {
interface: $apply_force(x), abs(x)$
axioms: $old_apply_force(x) \wedge x \geq ABS_step \Rightarrow$
$apply_force(abs(x))$
inherit: $[apply_force/old_apply_force]$ **}**

In the subclass, we inherit all the axioms of **Brakes**, replacing the symbol $apply_force$ with old_apply_force, add one more axiom and add an interface symbol abs.

4.2 Reasoning About Action: PMON

Let us examine one way of representing the temporal logic PMON (Sandewall 1994), (Gustafsson & Doherty 1996). Roughly speaking, the *features and fluents* language $\mathcal{L}(\mathcal{FL})$, that is used for PMON, has a theory divided into several subtheories: Γ_{OBS} (observations), Γ_{SCD} (schedule of actions/events), Γ_{UNA} (unique names axioms), Γ_T (axiomatization of time) and Γ_{NCG} (no-change/frame axiom). Different axioms are allowed in each category and we refer the reader to (Gustafsson & Doherty 1996) for a relatively recent work on the subject.

Since the semantics of PMON can be defined by Circumscription using the policy

$$\Gamma_{NCG} \wedge \Gamma_{OBS} \wedge \Gamma_{SCD} \wedge \Gamma_{UNA} \wedge \Gamma_T \wedge Circ[\Gamma_{SCD}(Occlude); Occlude;]$$

(see (Gustafsson & Doherty 1996)), we can represent PMON using the Directed OOFOL theory schematically displayed in Figure 7. In the figure, OCC stands

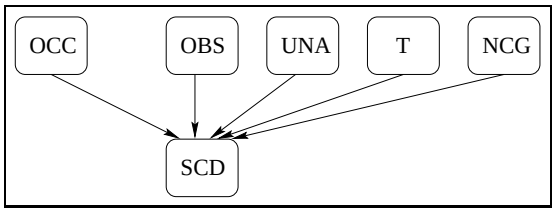

Figure 7: PMON in Directed OOFOL.

for the theory including the single axiom $\forall x\, Occlude(x)$. The rest of the objects represent their respective theory. The links and interfaces in the theory may include all of the symbols of each of the objects. This seems to render some of the computational savings due to the algorithms presented in Section 5 insignificant. The modeling techniques we present in the rest of this section help rectify this situation.

4.3 Hierarchical Reasoning

Suppose that a robot is given the assignment of navigating from one office to another. Since the robot moves are miniscule and are supposed to be run and monitored

at real-time, reasoning about the entire plan can be extremely expensive. If one adds to that the fact that the robot should reason about the behavior of other objects/agents (e.g., the elevator, doors, etc), brute-force reasoning becomes intractable and knowledge modeling becomes difficult.

With Directed-OOFOL one can represent and reason about several layers of abstraction concurrently, leading to the desired behavior at the low-level sensors/actuators. Figure 8 describes the theory. A more

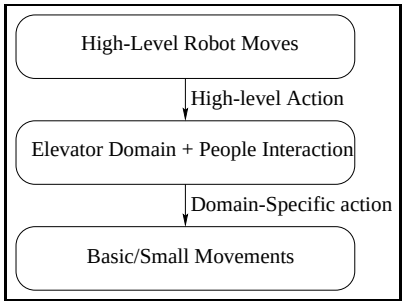

Figure 8: A hierarchical decomposition.

detailed description of a use of a similar theory is given in (Amir & Maynard-Reid 1999).

4.4 Multiple Agents

When we want to reason about several agents, decomposing the domain according to the different agents and their interactions can yield significant gain in modeling effort and reasoning efficiency. For example, if one has three robots that should each perform some different task, yet they have to interact every hour to consider battery-recharging, we can create a class *Robot* that includes the action theory for that robot and a class *Recharge* that includes the action theory for the multiple robots involved. Each object *Rob1, Rob2, Rob3, Rob1', Rob2', Rob3'* is of class *Robot*. An action schedule can be given to each of the robots, and the results of executing that schedule are fed into the *Recharge* object (of class *Recharge*). The results of the recharging process are then fed into the separate schedules, and the robots may continue in their separate tasks. The complete picture is depicted in Figure 9.

5 Algorithms

Algorithms for inference in OOFOL and Directional OOFOL may reduce the theories to the appropriate semantics and perform inference on the result. We are more interested in algorithms that can take advantage of the special structure of our theories. We present such an algorithm (each semantics with its own variant) for the special case of totally-ordered theories.

5.1 Totally Ordered Theories

We say that an OOFOL theory T is *totally ordered* if the objects in T can be sorted into a chain $\langle Obj_1, ..., Obj_n \rangle$

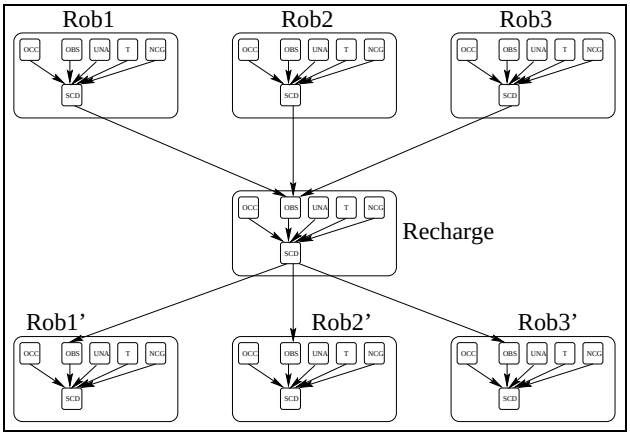

Figure 9: Multiple Agents.

such that every object Obj_i in the chain is directionally linked (or simply *linked* in the FOL-semantics case) to its predecessor Obj_{i-1} and there are no other links (i.e., no "loops").

For totally ordered theories, we can give equivalent semantics to our Directional Semantics using Prioritized Circumscription. The proofs for this and the following theorems are omitted for lack of space and are found in (Amir 1999).

Theorem 5.1 (Directional Semantics II) *Let $T = \{Obj_i\}_{i \leq n}$ be a totally ordered OOFOL theory. $\mathcal{M}$ is a model of T iff it is a model of*

$$\bigwedge_{i \leq n} Circ[T; Ab_i; Ab_{i+1}, ..., Ab_n, \mathcal{L}(Obj_{i+1}), ..., \mathcal{L}(Obj_n)]$$

where $Ab_i = Ab_{O_i, O_{i+1}}$.

Note that the formula above can be written as a single Prioritized Circumscription formula after the symbols in $\mathcal{L}(Obj_{i+1}), ..., \mathcal{L}(Obj_n)$ are eliminated from the varied part (Cadoli, Eiter, & Gottlob 1992).

5.2 Message-Passing

Applying our original intuition from Section 2, the following algorithm performs inference in one object, passes results to the appropriate objects and repeats the process in subsequent objects. Various inference methods can be used to prove a given query in a given object, but for simplicity we assume forward reasoning and a *totally ordered* theory T. We describe the different variances needed for each of the semantics after describing the algorithm.

For two objects $Obj = Object(C, N, L), Obj' = Object(C', N', L')$, let $\mathcal{L}_{Obj,Obj'}$ be the language that includes the set of "linking symbols" of Obj, Obj', $\{P \mid L'(Q, N, P)$ for some $Q\}$. For every sentence φ in $\mathcal{L}_{Obj,Obj'}$, we say that the *translation of φ from Obj to Obj'* is the sentence resulting from following L' in replacing every symbol $N.P$ in φ by $N'.Q$ such that $L'(Q, N, P)$. $\mathcal{L}_=$ is the first-order language with the sole

relation "=". Figure 10 describes our message-passing algorithm. Theorems 5.2 and 5.3 validate our approach for the FOL semantics.

PROCEDURE MESSAGE-PASSING
Given a totally ordered OOFOL theory $T = \{Obj_i\}_{i \leq m}$ and a query G in $\mathcal{L}(Obj_n)$ (for some $n \leq m$),

1. Concurrently perform inference for each of the objects $Obj_i \in T$ with $i \leq m$.
2. For each $i \leq n$, when we prove a formula $\varphi \in \mathcal{L}_{Obj_i, Obj_{i+1}}$, add the translation of φ from Obj_i to Obj_{i+1} to the set of axioms of Obj_{i+1}.
3. For each $n < i \leq m$, when we prove $Obj_i \models \varphi$, for $\varphi \in \mathcal{L}_=$, add φ to the set of axioms of Obj_n.
4. If we proved G in Obj_n, return YES.

Figure 10: An algorithm for proving G from a Directed OOFOL theory T by message-passing.

Theorem 5.2 (Completeness for OOFOL) *Let $T = \{Obj_i\}_{i \leq n}$ be a totally ordered OOFOL theory with $Obj_i = \langle C_i, N_i, L_i \rangle$. Let $\varphi \in \mathcal{L}(Obj_n)$ be a sentence. If $T \models \varphi$, then there is $a \leq n$ and a sequence of sentences $\varphi_a, \ldots, \varphi_n$ such that $\varphi = \varphi_n$, $Obj_a \models \varphi_a$ and every $a \leq \forall i < n$ $\varphi_i \in \mathcal{L}_{Obj_i, Obj_{i+1}}$ such that $Obj_{i+1} \models \varphi_i' \Rightarrow \varphi_{i+1}$, where φ_i' is the translation of φ_i from Obj_i to Obj_{i+1}.*

Theorem 5.3 (Soundness for OOFOL) *Assume that there are $\varphi_a, ..., \varphi_n$ such that $Obj_a \models \varphi_a$ and for every $a \leq \forall i < n$ $\varphi_i \in \mathcal{L}_{Obj_i, Obj_{i+1}}$ such that $Obj_{i+1} \models \varphi_i' \Rightarrow \varphi_{i+1}$, where φ_i' is the translation of φ_i from Obj_i to Obj_{i+1}. Then, $T \models \varphi$.*

Theorems 5.4 and 5.5 validate our approach to the Directional Semantics.

Theorem 5.4 (Completeness for Directional Sem.) *Let $T = \{Obj_i\}_{i \leq m}$ be a totally ordered OOFOL theory with $Obj_i = \langle C_i, N_i, L_i \rangle$. Let $n \leq m$, $\varphi \in \mathcal{L}(Obj_n)$ be a sentence and $E = \{\psi \in \mathcal{L}_= \mid \exists i > n\ Obj_i \models \psi\}$. If $T \models \varphi$, then there is a $\leq n$ and a sequence of sentences $\varphi_a, \ldots, \varphi_n$ such that $E_n \cup \{\varphi_n\} \models \varphi$, $Obj_a \models \varphi_a$ and every $a \leq \forall i < n$ $\varphi_i \in \mathcal{L}_{Obj_i, Obj_{i+1}}$ such that $Obj_{i+1} \models \varphi_i' \Rightarrow \varphi_{i+1}$, where φ_i' is the translation of φ_i from Obj_i to Obj_{i+1}.*

For soundness, we need to assume that influence does not "try" to flow back (i.e., if $O_{i+1} \models \varphi'$ for some $\varphi \in \mathcal{L}_{O_i, O_{i+1}}$, then $T \setminus \{O_j \mid j \leq i\} \models \varphi$).

Theorem 5.5 (Soundness for Directional Sem.) *Assume that there are $\varphi_k, ..., \varphi_l$ such that $k \leq l$ and $O_k \models \varphi_k$ and $k \leq \forall i < l$, $\varphi_i \in L_{O_i, O_{i+1}}$ and $O_i \models \varphi_i' \Rightarrow \varphi_{i+1}$, where φ_i' is the translation of φ_i from O_i to O_{i+1}. Then, $T \models \varphi_l$.*

There are two main differences between the uses of MESSAGE-PASS for the two semantics: (1) for FOL Semantics we can ask queries only from the last object in the sequence, Obj_m; and (2) MESSAGE-PASS is not sound for Directional Semantics if influence "tries" to flow back, in the sense described above.

5.3 Branches and Cycles

To generalize over the totally ordered case, our approach is to rewrite every structure into a reduced totally ordered OOFOL theory by aggregating groups of objects into single objects.

Assume that we have an arbitrary OOFOL theory $T = \langle Obj_i \rangle_{i \in \mathcal{I}}$ and we wish to prove $T \models \varphi$, where $\varphi \in \mathcal{L}(Obj_i)$ for some $i \in I$. All we need to do is to unify objects such that we are left with a new theory T' that is totally ordered. This unification will most likely result in an increased "bandwidth" for the links among the new objects, and this is the price we must pay for such an algorithm.

To see how this happens, see Figure 11. Here we see that, to use the MESSAGE-PASSING method to prove $O4.p \vee O4.q$, we cannot simply pass messages from $O1$ to $O2, O3$ to $O4$; rather, we have to unify O_2, O_3 into O_2' (on the right side of the figure).

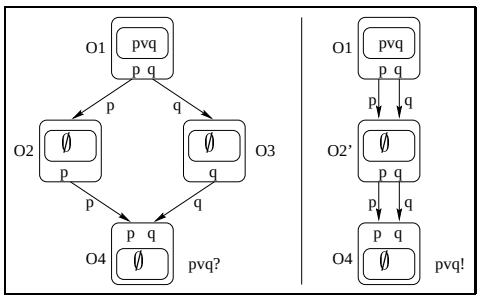

Figure 11: An example of joining nodes.

One may think that we can generalize MESSAGE-PASSING to work for a circular Directed OOFOL theory by simply continuing to transfer formulas from one object to the next until we prove our goal. Unfortunately, there are instances in which MESSAGE-PASSING is not complete for such theories.

6 Other Object-Oriented KR Systems

The seemingly closest approach to ours is the work on object-oriented Prolog (e.g., (Shapiro & Takeuchi 1983) and (Moss 1994)). Besides the obvious difference of full FOL expressivity vs horn clauses with the Closed World Assumption, there are several key differences. The program flow in Object-oriented Prolog is identical to the program flow in ordinary Prolog, while FOL has no *theorem proving flow* that one can generalize to an object-oriented version. Prolog and its object-oriented versions are based on queries, whereas OOFOL institutes the FOL-equivalent to queries in order to make give FOL the object-oriented *proof flow*. The object structure in both O-O prolog and OOFOL allows only some queries to be asked. In the OOFOL case, the object structure also gives heuristic information for the theorem prover on how to aggregate computation and what information should flow from object to object. Such information is not used in the unit-resolution systems of O-O Prologs.

Another close approach is Frame Systems. Currently, there is no adequate object-oriented approach for full FOL in Frame systems. KRYPTON (Brachman, Fikes, & Levesque 1983), which was one of the early Frame systems, had an A-Box part which allowed arbitrary FOL sentences to be stated. This structure was completely flat and had nothing of the object structure that we gave to our OOFOL. Ontolingua (Gruber 1993), which is one of the most expressive Frame systems to date, allows different sentences to be associated with each frame. Unlike OOFOL, there is no difference between a sentence put in one frame and the same sentence put in a different frame. In particular, there is no "scope" in which the axiom is active/inactive. Frame systems also traditionally use only a restricted form of the axiom inheritance used in this paper (for the A-Box part of these systems). *Overriding* is achieved (if at all) using nonmonotonic techniques.

Compared to the approaches for formalizing context (e.g., CycL (Lenat & Guha 1990) and QLC (Buvač 1996)), our logic allows only prescribed connections between the different objects (contexts usually have no restrictions), and instead of appealing to modalities for representing the different theories, we use an FOL semantics (and Circumscriptive/Preferential semantics for the directional case). Our semantics also has one universe of elements, whereas contexts are usually considered to have distinct domains of elements. Finally, for OOFOL we devised much more efficient general-purpose algorithms than those available for context systems (see (Lenat & Guha 1990)).

7 Conclusions

We presented an object-oriented approach to First-Order Logic. We showed that our approach has a simple FOL semantics in the undirected case and a simple circumscriptive semantics in the directional case. These semantics are in contrast to previous approaches to decomposing logical theories into small fragments. We demonstrated several applications of our logics and showed that in some cases a significant performance improvement can be gained by using OOFOL structures, due to the specialized algorithms we can apply.

Compared to FOL, the links/objects mechanisms that OOFOL provides eliminate some spurious deductions that the designer has decided are not useful. The OOFOL structures also provide natural ways to specify heuristic information for reasoning in the logics. Finally, the object-oriented approach encourages the knowledge engineer to aggregate knowledge for a particular use in one place, hiding most of the details of the "implementation" from the outside world by the provision of circumscribed interfaces. This facilitates knowledge reuse and information integration.

From the perspective of FOL and nonmonotonic reasoning (NMR), we presented a fragment of FOL (respectively NMR) that has some promising computational properties. In this work we did not delve into the decidability and computational-complexity issues involved in using such a structure. Nevertheless, we can already see some natural theorems emerging. This, together with further development of applications of the logics we presented here, is our current work.

8 Acknowledgments

I wish to thank Pedrito Maynard-Reid II for discussing related ideas and projects with me. Also, discussions with Erik Sandewall, Sheila McIlraith and Mike Genesereth were very helpful. Finally, I wish to thank Erika Henik, Patrick Doyle and the anonymous reviewers for commenting on drafts of the paper. This research was supported by an Air Force grant F49620-97-1-0207.

References

Amir, E., and Maynard-Reid, P. 1999. Logic-based subsumption architecture. In *Proc. 6th Intl Joint Conf. on Artificial Intelligence (IJCAI-99).*

Amir, E. 1999. Object-Oriented First-Order Logic – full version. http://www-formal.stanford.edu/eyal/-nmr/oofol-full.ps.

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

Buvač, S. 1996. Quantificational logic of context. In *Proc. Natl Conf. on Artificial Intelligence (AAAI '96).*

Cadoli, M.; Eiter, T.; and Gottlob, G. 1992. An efficient method for eliminating varying predicates from a circumscription. *Artificial Intelligence* 54(3):397–410.

Forbus, K. 1996. Qualitative reasoning. In *CRC Handbook of Computer Science and Engineering.* CRC.

Gruber, T. R. 1993. A translation approach to portable ontology specifications. *Knowledge Aquisition* 5(2):199–220.

Gustafsson, J., and Doherty, P. 1996. Embracing occlusion in specifying the indirect effects of actions. In *Proc. 5th Intl Conf. on Knowledge Representation and Reasoning (KR '96)*, 87–98.

Lausen, G., and Vossen, G. 1998. *Models and Languages of Object-Oriented Databases.* Addison-Wesley.

Lenat, D. B., and Guha, R. V. 1990. *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project.* Addison-Wesley.

McCarthy, J. 1986. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence* 28:89–118.

Meyer, B. 1997. *Object-Oriented Software Construction.* Prentice-Hall, 2nd edition.

Moss, C. 1994. *Prolog++, The Power of Object-Oriented and Logic Programming.* Addison-Wesley.

Sandewall, E. 1994. *Features and Fluents.* Oxford University Press.

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