Combining Object-Oriented and Logic Paradigms: A Modal Logic Programming Approach

Tarmo Uustalu[1]

Dept. of Mathematics, Institute of Cybernetics, Estonian Acad. Sci.
Akadeemia tee 21, EE-0108 Tallinn, Estonia

Abstract. In this paper, a number of existing solutions and suggestions towards combining the object-oriented (OO) and logic programming paradigms have been briefly studied and categorized, and a sketch of a new solution has been proposed which tries to capture the essence of OO in terms of modal logic, and which could be implemented as a modal logic programming system.
The proposal is based on the belief that two orthogonal dimensions - object hierarchy and time - are involved in OO that in many respects are similar. For the one-dimensional (static) case, three logics - MU, MU′, and MU″ - have been designed, each allowing a different variety of inheritance modes. The two-dimensional (dynamic) case has been treated in a logic 2MU and its corresponding variants. Under certain OO-motivated definitions of Horn clause, the resolution calculi of these logics turn out to be complete.

1 Introduction

Since 1983-4 [ST83, Zan84], more than 20 mergers of the OO and logic paradigms (see Section 2) have been proposed by different authors. Most of these aim at achieving some synergic effect, since the advantages that the two paradigms exhibit, are different. OO offers modularity, encapsulation, and sharing, whereas logic programming offers reasoning and deductive retrieval via unification and backtracking. A deeper difference between the two paradigms, however, lies not in their different advantages but rather in what they are about and for. Roughly, this deeper difference could be put as follows.

OO is a technique meant to create top-level orderliness and manageability. It is not based on any particular formalism, being rather a heuristic (in the best sense of the word) that works on many formalisms and makes things convenient. The logic paradigm, on the contrary, cares more about bottom-level properliness and purity. It assumes a concrete formalism, elaborated in much detail, and idolizes uniformity. Sadly enough, the logic paradigm lacks in strong heuristics for applying this formalism, sometimes turning out to be inept when larger applications are to be dealt with.

In this paper, yet another merger of OO and logic is proposed. In so doing, logic is viewed as a formalism, and OO is regarded as a technique that can be applied to different formalisms. An attempt is made to find equivalents to several OO concepts in logic,

[1]The research reported in this paper was carried out at The Norwegian Institute of Technology (NTH) in Trondheim while holding a scholarship of the International Department of NTH.

and to build up a logic-based formal model of the OO schema. The solution is more of a theoretic than pragmatic value, and its objective is twofold: to obtain a semantics to OO in terms of some logic, and to see what organizing effect logical "simulation" of OO could have on logic programming.

The logical tool used throughout this paper is modal logic (more exactly, logic of belief). The concrete logics that are attained appear to be elegant (they are normal, rely on simple and quite classical axioms etc.). Moreover, resolution calculi can be constructed for these logics which for classes of formulae corresponding to the actual OO practice prove to be complete.

There are similarities in our approach to that of McCabe's [McC88], where the main tool, however, is classical logic.

The organization of the paper is as follows. Section 2 is a brief summary of the ongoing work on merging the OO and logic paradigms and on the "OO-independent" structuring trends within the logic paradigm. Section 3 presents the choices that our approach is based on, and provides some justification of these. Section 4 describes our base logic MU ("Modal Units"), and contains a thorough motivation derived from a simple example. Sections 5-7 are brief descriptions of the advanced variants of MU. Section 8 discusses practical issues in using the mentioned logics for programming. Section 9 points out conclusions and possible directions for future work.

Proofs, as well as other technicalities, have been omitted in this paper. They can be found in [Uus91].

2 Related Work

2.1 Mergers

A good classification and study of mergers that had been suggested until 1987, has been presented in [LM88]. In the majority of mergers, logic means Horn clause logic, and we propose to distinguish between three main groups of such mergers:

- OO base language, extended with logical constructs,
- two base languages (OO and logic), interfaced,
- logic base language, extended with OO constructs.

The examples of *mergers with an OO base language* are Orient84/K [IT86], KSL/Logic [IT90], and CBL [BMS90]. In Orient84/K and KSL/Logic, deductive retrieval is facilitated through built-in methods. Besides that, in KSL/Logic, all logical constructs are viewed as objects. In CBL, an OO base language has been extended with means for expressing complex conditionals (rules).

The example of a *language interface* is Prolog/Loops [KE88], where facilities have been provided for making calls to Loops objects from Prolog, and for setting Prolog goals from Loops (in the form of messages).

Mergers with a logic base language are the most numerous. The historically earliest proposals by Shapiro and Takeuchi [ST83] and Zaniolo [Zan84] have had strong influ-

ence on the development of this group, and may thus be also viewed as belonging here, even though they add no new syntax to their logic base languages, and only show how to program in the OO spirit, using the existing means.

In nearly all mergers of this group, methods are represented by predicates, defined in one or more clauses. Differences appear in solving the following questions:

- what are objects and classes (and attributes, consequently)?,
- how can attribute values be changed?,
- what does inheritance mean?

Objects, classes, attributes. In the majority of mergers, objects and classes essentially are sets of clauses (theories), referred to by some atomic name. Attributes, thereby, are either non-logical (imperative) variables, or predicates that are given values in the clause sets of objects. In Zaniolo's proposal, classes are predicates, with argument places corresponding to attributes. Objects, therefore, are literals, with arguments instantiated with concrete attribute values. In Mandala [FTK..84], objects are predicates, with one argument meant for indicating the revision (in the form of a *stream*), and others for attribute values.

In Conery's merger [Con88], where predicate interpretation changes during the program execution, classes are predicates, with one argument meant for an atomic object name, and others corresponding to attributes. In "Objects as Intensions" [CW88], an object is an individual variable, standing for the infinite list of its states during history.

The concept of attribute is often understood more generally in mergers than in conventional OO (see Section 3 for details).

Attribute value changes. Zaniolo's approach enables no dynamics. ESP [Chi84], SPOOL [FH86], and Prolog++ [Mos90] make use of *imperative assignments*. Predominantly, states are revised exploiting the *assert and retract* built-in predicates. In ObjVProlog [MLV89], semantically cleaner predicates -- *assume and forget* -- are made use of. Shapiro and Takeuchi, Mandala, and Vulcan [KTMB87] (on its hidden underlying level) exploit *streams*. These lists of as yet unprocessed messages serve there as object revision identifiers.

Conery's proposal and "Objects as Intensions" change states at resolution and through stepwise instantiation of the history lists, respectively.

Inheritance. Conventional OO presumes *overriding inheritance* which in the logic context means that if a given class owns a definition for a called method, and the invocation fails, then no definitions from superclasses will be searched for or tried. In mergers with logic, it seems equally reasonable to allow further pursuing until either success will be achieved, or the space of appropriate definitions will be used up. Such inheritance is called *cumulative*. [Gal86] terms these modes of inheritance as *call/return* and *success/failure*, respectively.

Most mergers use overriding inheritance. In Prolog/KR [Nak84], ESP, and POL [Gal86], cumulation is the default. Overriding is facilitated through special language constructs (the simplest of them -- *cut* -- is readily available in Prolog). In CPU [MN87], the user has to metaprogram the inheritance semantics.

Our proposal, MU, falls under the third group of the above classification.

There are solutions that do not restrict logic to mean just Horn clause logic, like FOOPlog [GM87] and Maude [Mes90], which put much emphasis on algebra (algebraic

specifications, rewriting), or PRIZ [MT90], where deduction is used for synthesizing algorithms from specifications and not in execution.

The list of mergers, mentioned in this section, does not pretend to be complete, but rather to illustrate the variety of approaches taken.

2.2 Trends of Structuring Within the Logic Paradigm

Within the logic paradigm, tendencies towards thinking along the lines of software engineering are a relatively recent development. Two orthogonal directions of structuring can be observed in logic programming: *modularization* (structuring of programs) and *sorting* (structuring of data, i.e. grouping of individuals; sort ≈ type).

As examples of research in the direction of *modularization*, we mention [Mil86] and [Che87]. In a number of studies, modules are viewed as *worlds*, with more or less explicit parallels drawn to the possible-worlds' semantics of *modal logics*. From among the merger systems, mentioned in Section 2.1, Prolog/KR, Mandala and CPU take this approach. From proper modal extensions of Prolog, we mention MULTILOG [KG86] and MOLOG [Far86]. In OO applications of the modal logic paradigm, the *accessibility relation* between worlds correlates with the *isa* relation.

Sorting relies on *many-sorted logics*. Plain many-sorted logics offer no relation between sorts. To provide a counterpart for subtype/supertype (i.e. isa) relation, *order-sorted logics* have been developed (for overview, see [Obe89]). In those, a partial ordering on sorts is assumed which basically is a *subsort/supersort relation*. A good example of an order-sorted logic programming language is EPOS [HV87]. The LOGIN [AN86] language (which served as a starting point for the ALF [Mel88] merger of OO and logic) can also be regarded as relying on a variant of order-sorted logic (*feature types*), its peculiarity being that there are no sorts there, and individuals are ordered instead. The ordering itself, thereby, is not programmer-declared, but induced by the structure of terms that denote individuals. What "normally" would be sorts, are simulated by means of individuals in LOGIN. In a sense, the ordering in LOGIN is a hybrid of individual/sort and subsort/supersort relations. In OO terms, this pretty exactly corresponds to a classless (delegation) system with objects ordered by an object/prototype relation.

Objects in OO are hybrids of programs and data (data-drivenness), and this implies that neither modularization nor sorting is ideally satisfactory for modelling OO in logic. There is a reason, however, to prefer modularization. Objects as modules have identity in the form of module (world) names. In the case of choosing the sorting approach, objects would have to be represented by terms encoding their current states, and possess no identity. There would be no means for deciding whether some objects just occasionally have equal states or are identical indeed. A change in an object's state would mean that the object was exchanged for a new object.

In FOOPlog, objects are facilitated by means of sorting, but this has been achieved by powerfully generalizing the notion of sort (thus forcing sorting to subsume modularization). There, attribute values (the material manifestations of objects) are grouped into (*visible*) sorts, whereas objects themselves are grouped into so-called *hidden sorts*. Hidden sorts are defined as sets of algebrae, which are nothing less than

semantic counterparts of theories or modules. A thorough justification of this approach can be found in [GW91].

3 Principles and Terminology

The principles of OO have two aspects, *facilitating* and *restrictive*. An OO language must possess constructs for working with objects (*modularity*). There must be constructs for expressing explicit calls to objects (e.g., message passing), but certain calls may be forbidden (e.g., "remote calls to attributes" -- *encapsulation*). Similarly, there must also be constructs for expressing implicit calls, *sharing* (e.g., inheritance or delegation), but there may be limitations on what should be sharable and how.

We feel that logical models (i.e. semantics in the form of logics) of OO should concentrate on the facilitating role, and thus we allow a broad interpretation of the OO concepts. Restrictions can be introduced syntactically (by narrowing the set of well-formed formulae, logically speaking). Therefore, both objects and classes will be termed *objects* below; object/class, subclass/superclass, and object/prototype relations will be termed *isa*; and inheritance, delegation, and even the relation of an object's method being stored in that object's class will be termed *inheritance*.

Next we will argue that there is no deep reason to distinguish between attributes and methods either. In our opinion, the evolution of answers to the question what attributes and methods are, in conventional OO and in mergers thereafter, has been the following:

- **Attributes are hidden, methods are public.** Assuming that we neglect the restrictive aspect of the OO principles, this yields no distinction at all.
- **Attributes values are individuals, methods are operations on them.** Though true about conventional OO languages, this view has been abandoned in most mergers with logic. As it implies that attributes are dynamic, and as the most dynamic thing in generally static Prolog is predicates, there is ground for further generalizations.
- **Attributes are dynamic, methods are static** (and both are predicates). This is so in many mergers, and it is specifically emphasized in SCOOP [VLM88] and ObjVProlog [MLV89]. Now not only may an attribute be "multivalued" (the predicate be non-functional -- only LOCO [LVV89] has a construct for requiring univaluedness), but it can also be "parametric" (the predicate has input argument places) and "conditional" (the definition clauses do have bodies -- in SCOOP and ObjVProlog, however, dynamic predicate definition clauses are restricted to facts only).

From this position we can make one reformulation.

- **Attribute values are inherited over time, methods are inherited over the object hierarchy.** Just as objects own method definitions, we can say that time instants (revisions) own value assignments. Then the search for a definition of an object's method from superobjects becomes analogous to looking up an attribute's value backwards in the history of assignments (usually the latest assignment is interesting, i.e. overriding is applied).

The succession relation of time instants is analogous to the isa relation in the object hierarchy. Treating the time dimension in full analogy with the object universe dimension, we obtain a facility of simple *versioning*. It becomes possible to make calls ("send messages") to the past, so to retrieve old values, if necessary. Note that this approach also solves the *frame* (or *persistence*) *problem*: at a transition to a new revision, the

values that are not being changed require no updating. The analogy between objects and revisions was pointed out in [KG86].

In our approach, we will not distinguish between objects and classes (classless system). Both attributes and methods will be called *properties*, and both objects and time instants will be called *units* (with the latter we follow the tradition of [MN87], [MP90]). A *unit hierarchy* is formally a pair $\langle U, \text{isa}\rangle$, where U is a non-empty finite set, and isa is a partial ordering on U. isa* will further denote the reflexive-transitive closure of isa.

4 MU -- "Modal Units"

4.1 Motivation

L. Monteiro and A. Porto [MP90] point out that in addition to the overriding/cumulation dimension, one more dimension in modes of inheritance can be discerned -- that of *syntactic/semantic*. These two dimensions yield four different modes. In this section, we will study cumulative modes, postponing the discussion of overriding until Section 8.

Suppose u, $v \in U$, u isa v, and that everything that v owns is subject to inheritance. *Semantic* (or *relational* or *predicate*) *inheritance* means that the literals that hold in v, are available as facts to u, i.e. that u is allowed to use the model (semantics) of v for resolutions. *Syntactic* (or *definitional* or *clause*) *inheritance* means that the clauses that v owns, are available to u, i.e. that u is allowed to use the syntax of v for resolutions.

In other words, if inheritance is semantic, then in order to find the predicate interpretations in u, first the predicate interpretations must be computed separately for the clause sets of u and v, and then the results (models) must be composed. In case of syntactic inheritance, on the contrary, the clause sets of u and v (syntaxes) have first to be composed, and then only the predicate interpretations for the composed clause set can be computed. As can be noticed, the key question is whether u cannot or can use its own clauses when demonstrating the goals of clauses inherited from v.

Modal logic (logic of belief) enables us to mark the two different usages of inherited clauses more explicitly. We now move to motivating our first logic MU ("Modal Units"). Assume that for every unit $u \in U$, we have a modal operator $[u]$, where $[u]A$ is read as "A inheritably in u".

The two steps in interpreting under inheritance suggest two axioms:

$$\mathbf{K}^v_u. \quad [v]A \rightarrow [u]A, \text{ if } u \text{ isa } v,$$

and

$$\mathbf{K}_u. \quad [u](A \rightarrow B) \rightarrow ([u]A \rightarrow [u]B).$$

The first axiom corresponds to *composition* (it "copies" clauses), the second to *model computation*.

Assume that v owns a clause $p \leftarrow q$. We show that it is natural to write this as $[v]p \leftarrow [v]q$ in case we want it to be inheritable semantically, and as $[v](p \leftarrow q)$ in case we aim at syntactic inheritance. Indeed, both $[v]p \leftarrow [v]q$ and $[v](p \leftarrow q)$ yield $[u]p \leftarrow [v]q$, while only $[v](p \leftarrow q)$ enables to derive $[u]p \leftarrow [u]q$, just as it ought to be.

The clause $[v]p \leftarrow [v]q$ can be given an alternative equivalent form $[v](p \leftarrow [v]q)$, if we admit an axiom

$$\mathbf{4}_{u,z}. \quad [z]A \rightarrow [u][z]A.$$

This does no harm, as it claims all units to have access to see what holds in other units (*positive introspection*), which is in concert with the OO world-view. In fact, we will let units also to see what does not hold in other units (*negative introspection*, see axiom $\mathbf{5}_{u,z}$ in Section 4.2).

The purpose of the new writing for semantically inheritable clauses is that we wanted attain a uniform form for all acceptable clauses (we call them *Horn clauses* of MU): in each clause there will be one modal operator over the whole clause (*clause modality*), and optional modal operators over goals in the body (*goal modalities*). Clause modalities indicate the owner of the clause, goal modalities determine who has to demonstrate the goals (a missing modality means that the demonstration task is passed back to the caller -- syntactic inheritance).

As an example, consider a clause $[v](p \leftarrow [v]q,r,[z]s)$, where u isa v, and z is a unit, incomparable with u and v in the sense of isa*. The following is a proof that together with $\leftarrow[u]p$ it yields $\leftarrow[v]q,[u]r,[z]s$:

$$\dfrac{\dfrac{\dfrac{\dfrac{[v](p \leftarrow [v]q,r,[z]s)}{[u](p \leftarrow [v]q,r,[z]s)}\,\mathbf{K}^{v}_{u}}{[u]p \leftarrow [u][v]q,[u]r,[u][z]s}\,\mathbf{K}_{u}}{\leftarrow [u]p \qquad [u]p \leftarrow [v]q,[u]r,[z]s}\,\mathbf{4}_{u,v},\,\mathbf{4}_{z,v}}{\leftarrow [v]q,[u]r,[z]s}$$

This clause involves a mixture of syntactic and semantic inheritance, but also *message passing*, showing MU to be even more powerful than the formalism of [MP90]. In the conventional OO notation, this clause could be written as:

```
object v is
   ...
   method p is q; self:r, z:s
   ...
```

4.2 Formal

In the formal presentation, we adhere to the terminology and denotations from [Che80] and [Cat88]. Proofs and technicalities can be found in [Uus91]. To simplify the presentation, we shall consider propositional fragments everywhere. Section 4.3 gives hints for generalizations into the full predicate logic.

For any modal operator $[a]$, $\langle a \rangle$ will stand for $\neg[a]\neg$.

Language

For any $u \in U$, the language of MU has a modal operator $[u]$.

Axiomatics

The *Hilbert-style calculus* of MU consists of the following rules and axioms:

- *The Hilbert-style rules and axioms of the classical propositional logic.*
- *Modal rules and axioms.* For any $u, v, z \in U$, we have:

$\mathbf{N}_u$. $A \,/\, [u]A$,

$\mathbf{K}_u$. $[u](A \to B) \to ([u]A \to [u]B)$,

$\mathbf{D}_u$. $[u]A \to \langle u \rangle A$,

$\mathbf{K}^v{}_u$. $[v]A \to [u]A$ if u isa v,

$\mathbf{4}_{u,z}$. $[z]A \to [u][z]A$,

$\mathbf{5}_{u,z}$. $\langle z \rangle A \to [u]\langle z \rangle A$.

MU is normal in the sense of [Cat88].

Semantics

A complete *Kripke semantics* of MU is defined by the following restrictions on the accessibility relations:

$\mathbf{d}_u$. seriality: for any $w \in W$ there exist $w' \in W$ such that wR_uw',

$\mathbf{k}^v{}_u$. inclusion: if u isa v then for any $w, w' \in W$ if wR_uw' then wR_vw',

$\mathbf{iv}_{u,z}$. quasi-transitivity: for any $w, w', w'' \in W$ if wR_uw' and $w'R_zw''$ then wR_zw'',

$\mathbf{v}_{u,z}$. quasi-euclidity: for any $w, w', w'' \in W$ if wR_uw' and wR_zw'' then $w'R_zw''$.

The completeness of this semantics follows from the Catach completeness theorem for normal multimodal logics [Cat88].

The *standard Kripke frames* for MU are determined by the following conditions:

- $W = \{0\} \cup \bigcup_{u \in U} W_u$, where W_u's are pairwise non-intersecting non-empty finite sets;
- for any $w, w' \in W$ and any $u \in U$,

wR_uw' iff there exists $u' \in U$ such that $w' \in W_{u'}$ and u' isa* u.

Resolution calculus

There are two kinds of Horn clauses. *Input clauses* are formulae $[v](p \leftarrow \Gamma, n\Lambda)$, where p is an atomic formula, v is a unit, Γ and Λ are vectors of atomic formulae, and n is a vector of modal operators. *Goal statements* are formulae $\leftarrow m\Pi$, where Π is a vector of atomic formulae, and m is a vector of modal operators.

The *resolution calculus* of MU consists of one rule:

$$\frac{\leftarrow [u]p,\ m\Pi \qquad [v](p \leftarrow \Gamma, n\Lambda)}{\leftarrow [u]\Gamma, n\Lambda, m\Pi} \quad \text{if } u \text{ isa}^* v$$

For Horn clauses, the given resolution calculus is complete.

4.3 Predicate Case

The most important modifications at the transition from the propositional fragment of MU to full predicate MU are the following:

• To the axiomatics, the *Barcan formula* $\forall x[u]A \rightarrow [u]\forall xA$ must be added for all $u \in U$.

• In the Kripke semantics, it has to be required that individual and function symbols be rigid, and the domain be constant.

• The resolution rule has to be supplemented with the unification step.

5 MU′ -- Inheritability Not Inevitable

5.1 Motivation

The assumption of everything being inheritable is nearly always too strong. It is often necessary to permit a unit to own non-inheritable beliefs. To facilitate this, a logic MU′ will be described below, where for each unit we will have two modal operators $[u]$ and $[u\downarrow]$. $[u]A$ will read as "A in u", whereas $[u\downarrow]A$ will read as "A inheritably in u". Note that $[u]$ of MU has been renamed into $[u\downarrow]$ in MU′.

Like in MU, Horn clauses in MU′ will have clause and goal modalities. Since in MU′, we will allow clauses either to be inheritable or to hold for sure only just in their owner units, arbitrary modal operators will be acceptable as clause modalities. Goal modalities, on the contrary, will be limited to non-arrow modal operators, since it seems unnatural to require some unit to demonstrate a goal, if such a task was only passed to this unit's ancestor.

All theorems that we mentioned when presenting MU, are valid about MU′ (as well as about MU″ and two-dimensional logics, see Sections 6-7), too.

5.2 Formal

Language

For any $u \in U$, the language of MU′ has two modal operators: $[u]$ and $[u\downarrow]$.

Axiomatics

The *Hilbert-style calculus* of MU′ consists of the following rules and axioms:

• *The Hilbert-style rules and axioms of the classical propositional logic.*

• *Modal rules and axioms.* For any u, v, $z \in U$, we have: ($\circ$ and $\bullet$ can be either $\downarrow$ or the empty word)

$\mathbf{N}_{u\circ}$. $A \;/\; [u\circ]A$,

$\mathbf{K}_{u\circ}$. $[u\circ](A \rightarrow B) \rightarrow ([u\circ]A \rightarrow [u\circ]B)$,

$\mathbf{D}_{u}$. $[u]A \rightarrow \langle u\rangle A$,

$\mathbf{U}_{u}$. $\langle u\rangle A \rightarrow [u]A$,

$\mathbf{K}^{u\downarrow}{}_{u}$. $[u\downarrow]A \rightarrow [u]A$,

$\mathbf{K}^{v\downarrow}{}_{u\downarrow}$. $[v\downarrow]A\rightarrow[u\downarrow]A$ if u isa v,

$\mathbf{4}_{u\bullet,v\circ}$. $[z\circ]A\rightarrow[u\bullet][z\circ]A$,

$\mathbf{5}_{u\bullet,z\circ}$. $\langle z\circ\rangle A\rightarrow[u\bullet]\langle z\circ\rangle A$.

Semantics

A complete *Kripke semantics* of MU′ is defined by the following restrictions on the accessibility relations:

$\mathbf{d}_u$. seriality: for any $w\in W$ there exist $w'\in W$ such that wR_uw',

$\mathbf{u}_u$. uniqueness: for any w, w', $w''\in W$ if wR_uw' and wR_uw'' then $w'=w''$,

$\mathbf{k}^{u\downarrow}{}_{u}$. inclusion: for any w, $w'\in W$ if $wR_{u\downarrow}w'$ then wR_uw',

$\mathbf{k}^{v\downarrow}{}_{u\downarrow}$. inclusion: if u isa v then for any $w,w'\in W$ if $wR_{v\downarrow}w'$ then $wR_{u\downarrow}w'$,

$\mathbf{iv}_{u\bullet,z\circ}$. quasi-transitivity: for any w, w', $w''\in W$ if $wR_{u\bullet}w'$ and $w'R_{z\circ}w''$ then $wR_{z\circ}w''$,

$\mathbf{v}_{u\bullet,z\circ}$. quasi-euclidity: for any w, w', $w''\in W$ if $wR_{u\bullet}w'$ and $wR_{z\circ}w''$ then $w'R_{z\circ}w''$

The *standard Kripke frames* for MU′ are determined by the following conditions:

• $W = \{0\} \cup \bigcup_{u\in U} W_u$, where W_u's are pairwise non-intersecting non-empty finite sets, each having one distinguished element w_u;

• for any w, $w'\in W$ and any $u\in U$,

wR_uw' iff $w'=w_u$,

$wR_{u\downarrow}w'$ iff there exists $u'\in U$ such that $w'\in W_{u'}$ and u' isa* u.

Resolution calculus

Input clauses are formulae $[v\bullet](p\leftarrow\Gamma,\mathrm{n}\Lambda)$, where p is an atomic formula, v is a unit, • is either empty word or ↓, Γ and Λ are vectors of atomic formulae, and n is a vector of non-arrow modal operators. *Goal statements* are formulae $\leftarrow\mathrm{m}\Pi$, where Π is a vector of atomic formulae, and m is a vector of non-arrow modal operators.

The *resolution calculus* of MU′ consists of two rules:

$$\frac{\leftarrow[u]p,\ \mathrm{m}\Pi \qquad [u](p\leftarrow\Gamma,\mathrm{n}\Lambda)}{\leftarrow[u]\Gamma,\mathrm{n}\Lambda,\mathrm{m}\Pi}$$

$$\frac{\leftarrow[u]p,\ \mathrm{m}\Pi \qquad [v\downarrow](p\leftarrow\Gamma,\mathrm{n}\Lambda)}{\leftarrow[u]\Gamma,\mathrm{n}\Lambda,\mathrm{m}\Pi} \quad \text{if } u \text{ isa}^* v$$

6 MU″ -- Inheriting Not Inevitable

6.1 Motivation

The logic MU′ made it possible to block inheritance, i.e. to say of a formula which holds in a unit, whether it holds inheritably or just in this particular unit. The logic MU″ that we describe in this section, enables less practical but still conceivable blocking of inheriting, allowing us to say of a formula whether its holding in a particular unit can be derived without referring upwards, or whether it just holds in this unit.

For each unit $u \in U$, MU″ provides four modal operators. Modal formulae are read as follows:

$[u]A$	reads as	"A, without using inheritance, in u",
$[u\downarrow]A$	reads as	"A, without using inheritance, inheritably in u",
$[u\uparrow]A$	reads as	"A in u",
$[u\downarrow\uparrow]A$	reads as	"A inheritably in u".

Note that $[u\uparrow]$ of MU″ corresponds to $[u]$ of MU′, and that $[u\downarrow\uparrow]$ of MU″ corresponds to $[u\downarrow]$ of MU′.

In Horn clauses of MU′, we allowed clause modalities either to claim inheritability or not, whereas goal modalities could not require inheritability. In MU″, we act in a dual manner. Clause modalities have always to claim holding without using inheritance (since input clauses are a kind of "initial knowledge"), while goal modalities may require demonstration either without using inheritance or "just somehow". The roles of modalities are therefore as follows:

$[u]$	clause and goal modality,
$[u\downarrow]$	clause modality,
$[u\uparrow]$	goal modality,
$[u\downarrow\uparrow]$	internal "carrier" of inheritance ("hidden" in the resolution calculus).

6.2 Formal

Language

For any $u \in U$, the language of MU″ has four modal operators: $[u]$, $[u\downarrow]$, $[u\uparrow]$, $[u\downarrow\uparrow]$.

Axiomatics

The *Hilbert-style calculus* of MU″ consists of the following rules and axioms:

- *The Hilbert-style rules and axioms of the classical propositional logic.*
- *Modal rules and axioms.* For any u, v, $z \in U$, we have: ($\circ$ and $\bullet$ can be either $\downarrow\uparrow$, $\downarrow$, $\uparrow$, or the empty word)

$$\mathbf{N}_{u\circ}. \quad A \;/\; [u\circ]A,$$

$$\mathbf{K}_{u\circ}. \quad [u\circ](A\rightarrow B)\rightarrow([u\circ]A\rightarrow[u\circ]B),$$

$$\mathbf{D}_{u\uparrow}. \quad [u\uparrow]A\rightarrow\langle u\uparrow\rangle A,$$

$$\mathbf{U}_{u\uparrow}. \quad \langle u\uparrow\rangle A\rightarrow[u\uparrow]A,$$

$\mathbf{K}^{u\downarrow}{}_{u}$. $[u\downarrow]A\rightarrow[u]A$,

$\mathbf{K}^{u\downarrow\uparrow}{}_{u\downarrow}$. $[u\downarrow\uparrow]A\rightarrow[u\downarrow]A$,

$\mathbf{K}^{u}{}_{u\uparrow}$. $[u]A\rightarrow[u\uparrow]A$,

$\mathbf{K}^{u\downarrow}{}_{u\downarrow\uparrow}$. $[u\downarrow]A\rightarrow[u\downarrow\uparrow]A$,

$\mathbf{K}^{v\uparrow\downarrow}{}_{u\uparrow\downarrow}$. $[v\uparrow\downarrow]A\rightarrow[u\uparrow\downarrow]A$ if u isa v,

$\mathbf{4}_{u\bullet,v\circ}$. $[z\circ]A\rightarrow[u\bullet][z\circ]A$,

$\mathbf{5}_{u\bullet,z\circ}$. $\langle z\circ\rangle A\rightarrow[u\bullet]\langle z\circ\rangle A$.

Resolution calculus

Input clauses are formulae $[v\bullet](p\leftarrow\Gamma,n\Lambda)$, where p is an atomic formula, v is a unit, $\bullet$ is either empty word or $\downarrow$, Γ and Λ are vectors of atomic formulae, and n is a vector of non-arrow and $\uparrow$ modal operators. *Goal statements* are formulae $\leftarrow m\Pi$, where Π is a vector of atomic formulae, and m is a vector of non-arrow and $\uparrow$ modal operators.

The *resolution calculus* of MU″ consists of four rules:

$$\frac{\leftarrow[u]p,\ m\Pi \qquad [u](p\leftarrow\Gamma,n\Lambda)}{\leftarrow[u]\Gamma,n\Lambda,m\Pi} \qquad \frac{\leftarrow[u]p,\ m\Pi \qquad [u\downarrow](p\leftarrow\Gamma,n\Lambda)}{\leftarrow[u]\Gamma,n\Lambda,m\Pi}$$

$$\frac{\leftarrow[u\uparrow]p,\ m\Pi \qquad [u](p\leftarrow\Gamma,n\Lambda)}{\leftarrow[u\uparrow]\Gamma,n\Lambda,m\Pi} \qquad \frac{\leftarrow[u\uparrow]p,\ m\Pi \qquad [v\downarrow](p\leftarrow\Gamma,n\Lambda)}{\leftarrow[u\uparrow]\Gamma,n\Lambda,m\Pi}\ \text{if } u \text{ isa}^{*} v$$

7 Logics for Two Dimensions

7.1 Motivation

Each of the logics MU, MU′, and MU″ can be easily generalized for two dimensions. We will demonstrate how this can be done in case of MU. Assume we are given an object hierachy $\langle O, \text{isa}_O\rangle$ and a time instant hierarchy $\langle T, \text{isa}_T\rangle$. Let their corresponding logics be denoted MU_O and MU_T. The two-dimensional logic 2MU has then modalities for the elements of both O and T. It is natural to assume the formulations "A in object o at time t" and "A at time t in object o" to be equivalent, and this suggests that modalities from different one-dimensional logics should commute.

7.2 Formal

Language

For any $u\in O\cup T$, the language of 2MU has a modal operator $[u]$.

Axiomatics

The *Hilbert-style calculus* of 2MU consists of the following rules and axioms:

- *The Hilbert-style rules and axioms of the classical propositional logic.*

- *Modal rules and axioms of* MU_O.
- *Modal rules and axioms of* MU_T.
- *Commutation axioms.* For any $o \in O$ and $t \in T$, we have:

$$[o][t]A \equiv [t][o]A,$$
$$[o]\langle t\rangle A \equiv \langle t\rangle [o]A.$$

Resolution calculus

Input clauses are formulae $[\omega][\tau]\ (p \leftarrow \Gamma,\ \mathrm{n}\Delta,\ \vartheta\Sigma,\ \mathrm{n}'\vartheta'\Lambda)$, where p is an atomic formula, ω is an object, τ is a time instant, Γ, Δ, Σ, and Λ are vectors of atomic formulae, n and n′ are vectors of object modal operators, and ϑ and ϑ' are vectors of time modal operators. *Goal statements* are formulae $\leftarrow \mathrm{m}\theta\Pi$, where Π is a vector of atomic formulae, m is a vector of object modal operators, and θ is a vector of time modal operators.

The *resolution calculus* of 2MU consists of one rule:

$$\frac{\leftarrow [o][t]p,\ \mathrm{m}\theta\Pi \qquad [\omega][\tau]\ (p \leftarrow \Gamma,\ \mathrm{n}\Delta,\ \vartheta\Sigma,\ \mathrm{n}'\vartheta'\Lambda)}{\leftarrow [o][t]\Gamma,\ \mathrm{n}[t]\Delta,\ [o]\vartheta\Sigma,\ \mathrm{n}'\vartheta'\Lambda,\ \mathrm{m}\theta\Pi} \quad \text{if } o\ \mathrm{isa}_O^*\ \omega \text{ and } t\ \mathrm{isa}_T^*\ \tau$$

8 Pragmatics

In order to use the above-presented resolution calculi for programming, we must assume predicate definitions in units, and goals in clauses, to be ordered. We must also provide some *facility for declaring object and time instant hierarchies*. With objects, the situation is easy -- we can supplement the language with some non-clausal constructs that allow us to encode the hierarchy graph. Time is more problematic, since the analogy between the object universe and time dimensions is not universal:

- Among the time instants, one is always distinguished -- the current instant, or "now". "Now" never has descendants, though new instants can be created only from "now".
- When specifying in a program some action that would cause a new revision, we normally don't know at which instant this action will be performed at run-time. Therefore, there must be a possibility to express not fully instantiated time instant names.

To cope with this, we can let instant names themselves induce the time instant hierarchy. Instant names can be lists (of some atomic labels), for example, and isa can then be defined as the list/tail relation. This is essentially the same idea as that of stream programming. The only difference is that the frame problem has been solved: due to inheritance, there is no need any more to write clauses stating merely that some literal preserved its validity at some transition.

Overriding inheritance can be facilitated by means of cuts. In MU′, if searches for successful clauses for a goal $p(C1,\ldots,Cn)$ must end at a unit v, then the clause

```
[v↓] (p(X1,...,Xn) ← !, fail)
```

has to be added to the end of the definition of p in v.

9 Conclusions and Future Work

This paper used modal logic to give logical semantics to some aspects of OO. Units and different inheritance modes proved to allow clear and simple formalization in terms of logic of belief. It also appeared to be reasonable to regard the essential difference between attributes and methods to lie in which hierarchy they are inherited over, since this provides some solution to the frame problem, and strengthens uniformity.

The following problems should deserve future study:

- Possible connection of overriding inheritance with linear and non-monotonic logics.
- Elaboration of the pragmatics of working with the time dimension.
- Dynamic object hierarchies. At the present stage, the designed logics enable only objects be dynamic, whereas the isa links between them have to be static.

Only when the research was in the stage of completion, the author got to know about the work by M. Kifer et al. [KLW90]. The connection between their formalism and ours needs to be studied yet.

Acknowledgements

I am grateful to my scientific advisors Reidar Conradi and Jan Komorowski. I also thank Tore Amble and Antonio Porto for the inspiration they provoked directly and indirectly, and the anonymous referees for their helpful remarks.

References

[AN86] H. Aït-Kaci and R. Nasr. LOGIN: A logic programming language with built-in inheritance. *J. Logic Programming*, 3(3):185-215, 1986.

[BMS90] M. v. Biema, G. Q. Maquire, and S. Stolfo. The constraint-based paradigm: Integrating object-oriented and rule-based programming. In *Proc. 23rd Annual Hawaï Int'l Conf. on Syst. Sci.*, Kailua-Kona, Jan 1990, Vol 2, pp 358-66. Los Alamitos, CA: IEEE Comp. Soc. Press, 1990.

[Cat88] L. Catach. Normal multimodal logics. In *AAAI-88: Proc. 7th Nat'l Conf. on AI*, St Paul, Aug 1988, Vol 2, pp 491-5. 1988.

[Che80] B.F.Chellas. *Modal Logic: An Introduction*. Cambridge, UK: Univ. Press, 1980.

[Che87] W. Chen. A theory of modules based on second-order logic. In *Proc. 1987 Symp. on Logic Programming*, San Fransisco, Aug/Sept 1987, pp 24-33. Washington, DC: IEEE Comp. Soc. Press, 1987.

[Chi84] T. Chikayama. Unique features of ESP. In *Proc. 1984 Int'l Conf. on 5th Gener. Comp. Syst.*, Tokyo, Nov 1984, pp 292-8. Amsterdam: North-Holland, 1984.

[Con88] J. S. Conery. Logical objects. In *Logical Programming: Proc. 5th Int'l Conf. and Symp.*, Seattle, Aug 1988, pp 420-34. Cambridge, MA: The MIT Press, 1988.

[CW88] W. Chen and D. S. Warren. Objects as intensions. In *Logical Programming: Proc. 5th Int'l Conf. and Symp.*, Seattle, Aug 1988, pp 404-19. Cambridge, MA: The MIT Press, 1988.

[Far86] L. Fariñas del Cerro. MOLOG: A system that extends Prolog with modal logic. *New Gener. Computing*, 4(1):35-50, 1986.

[FH86] K. Fukunaga and S. Hirose. An experience with a Prolog-based object-oriented language. In *OOPSLA'86: OOP Syst., Lang. and Appl.: Conf. Proc.*, Portland, Sept/Oct 1986, pp 224-31. 1986. (*SIGPLAN Notices*, 21(11)).

[FTK..84] K. Furukawa et al. Mandala: A logic-based knowledge programming system. In *Proc 1984 Int'l Conf. on 5th Gener. Comp. Syst.*, Tokyo, Nov 1984, pp 613-22. Amsterdam: North-Holland, 1984.

[Gal86] H. Gallaire. Merging objects and logic programming: Relational semantics. In *AAAI-86: Proc. 5th Nat'l Conf. on AI, Philadelfia*, Aug 1986, Vol 2, pp 754-8. Los Altos, CA: Morgan Kaufmann, 1986.

[GM87] J. A. Goguen and J. Meseguer. Unifying functional, object-oriented and relational programming with logical semantics. In B. Shriver and P. Wegner, eds, *Research Directions in Object-Oriented Programming*, pp 417-77. Cambrdige, MA: The MIT Press, 1987.

[GW91] J. A. Goguen and D. Wolfram. On types and FOOPS. In R. Meersman et al., eds., *Object Oriented Databases: Analysis, Design and Construction*, pp 1-22. Amsterdam: North-Holland, 1991.

[HV87] M. Huber and I. Varsek. Extended Prolog for order-sorted resolution. In *Proc. 1987 Symp. on Logic Programming*, San Fransisco, Aug/Sept 1987, pp 34-43. Washington, DC: IEEE Comp. Soc. Press, 1987.

[IC90] M. H. Ibrahim and F. A. Cummins. KSL/Logic: Integration of logic with objects. In *Proc. 1990 Int'l Conf. on Comp. Lang.*, New Orleans, March 1990, pp 228-35. Los Alamitos, CA: IEEE Comp. Soc Press, 1990.

[IT86] Y. Ishikawa and M. Tokoro. Concurrent object-oriented knowledge representation language Orient84/K: Its features and implementation. In *OOPSLA'86: OOP Syst., Lang. and Appl.: Conf. Proc.*, Portland, Sept/Oct 1986, pp 232-41. 1986. (*SIGPLAN Notices*,21(11)).

[KE88] T. Koschmann and M. W. Evens. Bridging the gap between object-oriented and logic programming. *IEEE Software*, 5(5):36-42, 1988.

[KG86] H. Kauffmann and A. Grumbach. MULTILOG: MULtiple worlds in LOGic programming. In *ECAI'86: Proc. 7th Europ. Conf. on AI*, Brighton, July 1986, Vol 1, pp 291-305. 1986.

[KLW90] M. Kifer, G. Lausen and J. Wu. Logical foundations of object-oriented and frame-based languages. Technical Report 90/14, SUNY at Stony Brook, Dept. of Comput. Sci., August 1990.

[KTMB87] K. Kahn et al. Vulcan: Logical concurrent objects. In B. Shriver and P. Wegner, eds, *Research Directions in Object-Oriented Programming*, pp 75-112. Cambridge, MA: The MIT Press, 1987.

[LM88] L. Leonardi and P. Mello. Combining logic- and object-oriented programming language paradigms. In *Proc. 21st Annual Hawaï Int'l Conf. on Syst. Sci.*, Vol 2, pp 376-85. Washington, DC: IEEE Comp. Soc. Press, 1988.

[LVV89] E. Laenens, D. Vermeir, and B. Verdonk. LOCO, a LOgic-based language for Complex Objects. In *ESPRIT'89: Proc. 6th Annual Esprit Conf.*, Brussels, Nov/Dec 1989, pp 604-16. Dordrecht: Kluwer Acad. Publ., 1989.

[McC88] F. McCabe. *Logic and Objects*. London: Imperial College, 1988.

[Mel88] F. Mellender. An integration of logic and object-oriented programming. *SIGPLAN Notices*, 23(10):181-5, 1988.

[Mes90] J. Meseguer. A logical theory of concurrent objects. In *OOPSLA ECOOP '90 Proc.: Conf. on OOP Syst., Lang. and Appl., Europ. Conf. on OOP*, Ottawa, Oct 1990, pp 101-15. 1990. (*SIGPLAN Notices*, 20(10)).

[Mil86] D. Miller. A theory of modules for logic programming. In *Proc. 1986 Symp. on Logic Programming*, Salt Lake City, Sept 1986, pp 106-14. Washington, DC: IEEE Comp. Soc. Press, 1986.

[MLV89] J. Malenfant, G. Lapalme, and J. Vaucher. ObjVProlog: Metaclasses in logic. In *ECOOP'89: Proc. Europ. Conf. on OOP*, Nottingham, July 1989, pp 257-69. Cambridge, UK: Univ. Press, 1989.

[MN87] P. Mello and A. Natali. Objects as communicating Prolog units. In *ECOOP'87: Proc. Europ. Conf. on OOP*, Paris, June 1987, pp 181-92. Berlin: Springer-Verlag, 1987. (*LNCS*, 276).

[Mos90] C. Moss. An introduction to Prolog++. Research Report DOC 90/10, Imperial College, London, June 1990.

[MP90] L. Monteiro and A. Porto. Semantic and syntactic inheritance in logic programming. Draft report, Departamento di Informatica, Universidade Nova de Lisboa, Dec 1990.

[MT90] G. E. Mints and E. H. Tyugu. Propositional logic programing and the PRIZ system. *J. Logic Programming*, 9(3):179-93, 1990.

[Nak84] H. Nakashima. Knowledge representation in Prolog/KR. In *Proc. 1984 Int'l Symp. on Logic Programming*, Atlantic City, Feb 1984, pp 126-30. Silver Spring, MD: IEEE Comp. Soc. Press, 1984.

[Obe89] A. Oberschelp. Order-sorted predicate logic. In *Sorts and Types in AI: Workshop Proc.*, Eringerfeld, April 1989, pp 8-17. Berlin: Springer-Verlag, 1989. (*LNAI*, 418).

[ST83] E. Shapiro and A. Takeuchi. Object-oriented programming in Concurrent Prolog. *New Gener. Computing*, 1(1):25-48, 1983.

[Uus91] T. Uustalu. Combination of object-oriented and logic paradigms. MSc Thesis, Tallinn Technical University, 1991.

[VLM88] J. Vaucher, G. Lapalme, and J. Malenfant. SCOOP: Structured Concurrent Object-Oriented Prolog. In *ECOOP'88: Proc. Europ. Conf. on OOP*, Oslo, Aug 1988, pp 191-211. Berlin: Springer-Verlag, 1988. (*LNCS*, 322).

[Zan84] C. Zaniolo. Object-oriented programming in Prolog. In *Proc. 1984 Int'l Symp. on Logic Programming*, Atlantic City, Feb 1984, pp 265-70. Silver Spring, MD: IEEE Comp. Soc. Press, 1984.