A (Not Very Much) Annotated Bibliography on Integrating Object-Oriented and Logic Programming

Available from `menaik.cs.ualberta.ca:  pub/oolog`

Compiled by: Vladimir Alexiev
`vladimir@cs.ualberta.ca`

March 27, 1993

Contents

1 Introduction

Object-oriented computing emerged in the last decade as an effective means to deal with the "software crisis" by making the software engineering an easier process. Logic programming on the other hand can be viewed as an attempt to put (part of) mathematical logic in service to programming. LP is widely used for applications where it is not quite clear what direction the computation will take (what will be the inputs and what the outputs) but the relations among the concepts in the domain are clear This is the case for some AI tasks, prototypical programming, etc. LP, however, largely misses the needed software engineering devices (modules, information hiding, reuse etc.) to make it a viable choice for large programs.

It would be very useful to integrate OOP and LP in a seamless and natural way in order to exploit the synergism between the two. An important issue in such an integration would be to preserve the clarity of the two paradigms and merge them organically so that not to get an ad-hoc conglomerate of unrelated concepts ala PL/1. The two paradigms are equally basic and self-important so it would not be appropriate to make one of them the master and the other the slave of such a merger.

A preliminary literature survey of the area shows that right now OOP+LP is a hot topic. Without much effort it was possible to compile a bibliography of about 180 items; there are at least 50 different mergers and/or languages proposed. Some of them are approaches one would like to avoid: implementing OOP in LP or LP in OOP (often very inefficiently), binding the OOP+LP merger to Constraint LP, Concurrent LP or Deductive and OO Databases (whereas the merger has its own importance) etc. But there are also basic ideas which seem very attractive: represent message passing by unification, methods by the set of clauses comprising a predicate, inheritance by extending a predicate with clauses from another logic theory (world, module) etc.

The main obstacle to a merger seems to be that the LP paradigm does not support the notion of *mutable state*: a logic variable is either free or bound once and for all (logic possesses the property of *referential transparency*). When represented in LP, most often state is carried by predicate's parameters. The task of defining formally what an object is (devising logics to model OO features) seems most important at present because there is no widely accepted formalization yet and this hinders the smooth implantation of OOP into LP.

I have used the following sources to compile this bibliography: Computer and Control Abstracts (everything from 1989 to 1992), ACM Guide to Computing Literature (1991 and 1992; this is hardly usable as a reference), online databases COMPENDEX (1987-1992) and INSPEC (1988(?)-1992; this is most comprehensive but I had hardly enough access to it). I have also searched Index to Scientific Reviews and got some titles but haven't included them yet. And of course, I have scanned the reference lists of the articles I have read.

I have tried to outline the boundaries of subareas in the OOP+LP area and to figure out what is already done and what is still to be done. However the resulting division is imperfect for a number of reasons:

- it is highly subjective;
- sometimes a paper falls equally well under two or more divisions;
- papers which I have not read and which have undescriptive title inevitably are classified wrongly;
- I have tried not to split related work by the same author or team: all related articles are put in the section which best fits the most important of them.

As the title says, this is a not-very-much annotated biblography. It is not completed yet both in the sense that there are titles I would like to include and, more importantly, I have read only about 30 or 40 of the articles. My comments are rarely longer than five lines; the ten-line abstracts which you can see with some of the articles have come form COMPENDEX (and are copied without permission, I guess). Generally all annotations which do not end on "n Refs" are written by me, but about 10% of them (the shortest ones) are just "abstracted abstracts" from CCA and have little value. Now you may think that my annotations are very short and make little sense, but this is because the life is short; and I made it for myself, not for you :-). However corrections, additions and suggestions for reorganization *are most welcome* at the address above. I am not qualified to be and I won't serve as a referee of all these articles, only with a joint effort this bibliography could possibly have some value (not that I got a lot of comments yet :-(.

2 Logics to Model Object-Oriented Notions

This is the subtopic I deem most important at present because we still don't have a clear logical understanding of what an object is (at least I don't; maybe after reading all this I will gain one :-). Object-oriented computing enjoys most wide acceptance but no semantics for it does yet.

It is not necessarily the case that having a good logic of objects will buy us a good merger of logic *programming* with OOP: the particular logic may have little practical value. So having a clear logical semantics of OOP is probably not a sufficient condition for a good merger, but definitely it is a necessary one.

Logics which deal only with inheritance or modules for LP are in the next section. Here are logics which aim at an overall treatment of objects and/or which try to model mutable state. Some of the the logics included here should probably go to the section Concurrent LP.

A couple of ecclectic remarks about the subarea follow: An early work on OO logic is Maier's O-LOGIC [40, 41]. Later it was extended and refined in Kifer's F-LOGIC [31, 33] and Chen and Warren's C-LOGIC [5].

Goguen and Meseguer has done a considerable early work falling between this subarea and the one described in the next section [19, 20].

Another direction is the application of Linear Logic to the problem: Andreoli and Pareschi's LINEAR OBJECTS [2, 3, 1, 4, 5].

A recent approach by Meyer and Wieringa [45, 53] and by Jungclaus [4, 26] (influenced by Ehrich, Fiadeiro, Saake and Sernadas) is to use fully the developments in Abstract Data Types and only add object identities and Dynamic Logic (a form of Modal Logic with modalities formed by events: message sends to objects) to model changing state.

Uustalu [51] uses Modal Logic tro model inheritance and regards evolution of state the same as inheritance of behavior: state is inherited from the previous time instant. This approach is somewhat similar to Pimentel [46].

By the way, should we talk about an "Italian school" in this subarea? (Andreoli, Brogi, Lamma, Leonardi, Mello, Pareschi)

References

[1] J.-M. Andreoli and R. Pareschi. LO and behold! concurrent structured processes. In *ECOOP-OOPSLA'90*, Ottawa, Ontario, 1990. (*SIGPLAN Notices*, 25(10):44–56, Oct. 1990).

[2] J.-M. Andreoli and R. Pareschi. Linear objects: Logical processes with built-in inheritance. In D. H. D. Warren and P. Szeredi, editors, *Seventh International Conference on Logic Programming*, pages 495–510, Jerusalem, Israel, 1990. The MIT Press.

[3] J.-M. Andreoli and R. Pareschi. Logic programming with linear logic. In *Extensions of Logic Programming*, LNAI. Springer-Verlag, 1990.

[4] J.-M. Andreoli and R. Pareschi. Linear objects: Logical processes with built-in inheritance. *New Generation Computing*, 9(3-4):445–473, 1991.
Objects are regarded as proof processes (this is a borrowing from concurrent LP). Object state is modeled by the arguments occuring during the proof. "Linear" objects mean that there may be more than one literal in the clause head (non-Horn clauses).

[5] J.-M. Andreoli and R. Pareschi. Linear objects: A logic framework for open system programming. In A. Voronkov, editor, *International Conference on Logic Programming and Automated Reasoning LPAR'92*, pages 448–450, St. Petersburg, Russia, July 1992.

[6] A. Brogi, E. Lamma, and P. Mello. A general framework for structuring logic programs. Technical report, C.N.R. "Progetto Finalizzato Sistemi informatici e Calcolo paralello", 1990.

[7] A. Brogi, E. Lamma, and P. Mello. Objects in a logic programming framework. In A. Voronkov, editor, *First Russian Conference on Logic Programming*, number 592 in LNAI, pages 102–113. Springer-Verlag, 1991.
Objects are represented by logic theories, inheritance is expresses as metalevel axioms, messages are equated to requests to prove a goal. A clear semantic characterization is provided.

[8] A. Brogi and F. Turini. Metalogic for knowledge representation. In J. A. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Second International Conference*, pages 61–69, Cambridge, CA, 1991. Morgan Kaufmann.
Metalogic is shown adequate to represent a number of KR methods and inference modes: structuring of logic theories in terms of hypothetical reasoning and contextual LP; object-orientation of theories by means of heararchical reasoning/inheritance and encapsulation of theories as objects.

[9] Q. Chen. High-order logic programming framework for complex objects reasoning. In *Thirteenth Annual International Computer Software and Applications Conference - COMPSAC'89*, pages 711–718, Orlando, FL, 1989.
The theoretical foundations of a strongly typed high-order rule language, HILOG, are developed by introducing appropriate mathematical concepts to reformulate the logic programming (LP) notions. This work is significant for enhancing the LP capability to support object orientation, Abstract Data Types, and knowledge representation with type hierarchies and for applying LP and deductive database techniques to practical applications involving complex objects. 16 Refs.

[10] W. Chen and D. S. Warren. Objects as intensions (logic programming). In R. A. Kowalski and K. A. Bowen, editors, *Fifth International Conference and Symposium on Logic Programming*, pages 404–419, Seattle, WA, Aug. 1988.

[11] W. Chen and D. S. Warren. C-Logic of complex objects. In *Eight ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 369–378, Philadelphia, PA, Mar. 1989.
Our objective is to have a logical framework for natural representation and manipulation of complex objects. We start with an analysis of semantic modeling of complex objects, and attempt to understand what are the fundamental aspects which need to be captured. A logic, called C-logic, is then presented which provides direct support for what we believe to be basic features of complex objects, including object identity, multi-valued labels and a dynamic notion of types. C-logic has a simple first order semantics, but it also allows natural specification of complex objects and gives us a framework for exploring efficient logic deduction over complex objects. (Author abstract) 24 Refs.

[12] J. W. de Bakker and E. P. de Vink. CCS for OO and LP. In S. Abramsky and T. S. E. Maibaum, editors, *International Joint Conference on Theory and Practice of Software Development (TAPSOFT'91), Volume 2: Colloquium on Combining Paradigms for Software Development*, number 494 in LNCS, pages 1–28, Brighton, UK, Apr. 1991. Springer-Verlag.
The title decrypts to: "Comparative Continuation Semantics for Object-Oriented and Logic Programming".

[13] H.-D. Ehrich, M. Gogolla, and A. Sernadas. Objects and their specification. In M. Bidoit and C. Choppy, editors, *Eigth Workshop on Abstract Data Types*, number 655 in LNCS, pages 40–66. Springer-Verlag, 1992.

[14] H.-D. Ehrich, J. A. Goguen, and A. Sernadas. A categorial theory of objects as observed processes. In J. W. deBakker, W. P. deRoever, and G. Rozenberg, editors, *REX/FOOL Workshop*, number 489 in LNCS, pages 203–228, Noordwijkerhood, Netherlands, 1990. Springer-Verlag, 1991.

[15] H.-D. Ehrich, G. Saake, and A. Sernadas. Concepts of object-orientation. In R. Studer, editor, *Second Workshop of "Informationssysteme und Künstliche Intelligenz: Modellierung"*, number 322 in IFB, pages 1–19, Ulm, Germany, 1992. Springer-Verlag.

[16] J. Fiadeiro and T. Maibaum. Describing, structuring and implementing objects. In J. W. deBakker, W. P. deRoever, and G. Rozenberg, editors, *REX/FOOL Workshop*, number 489 in LNCS, pages 275–310, Noordwijkerhood, Netherlands, 1990. Springer-Verlag.

[17] J. C. Freytag, R. Manthey, and M. Wallace. Mapping object-oriented concepts into relational concepts by meta-compilation in a logic programming environment. In K. R. Ditrich, editor, *Secont International*

Workshop on Advances in Object-Oriented Database Systems, pages 204–208, Ebernburg, Germany, Sept. 1988.

[18] H. Gallaire. Merging objects and logic programming: Relational semantics. In *AAAI-86: Fifth National Conference on Artificial Intelligence*, pages 754–758, Philadelphia, PA, Aug. 1986.

[19] J. A. Goguen and J. Meseguer. Equality, types, modules and generics for logic programming. In *Logic Programming: Relations, Functions and Equations.* Prentice-Hall, 1986.

[20] J. A. Goguen and J. Meseguer. Unifying functional, object-oriented and relational programming with logical semantics. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 417–477, Cambridge, MA, 1987. MIT Press.

[21] J. A. Goguen and D. Wolfram. On types and FOOPs. In R. Meersman et al., editors, *Object-Oriented Databases: Analysis, Design and Construction*, pages 1–22. North-Holland, 1991.

[22] J. Grabowski. Metaprograms for change, assumptions, objects, and inheritance. In A. Pettorossi, editor, *Third International Workshop on Meta-Programming in Logic, META-92*, pages 336–351, Uppsala, Sweden, June 1992.

[23] C. A. Gunter and J. C. Mitchell. *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design.* The MIT Press, 1993. To appear.

[24] T. Hartmann, R. Jungclaus, and G. Saake. Aggregation in a behavior oriented object model. In O. L. Madsen, editor, *European Conference on Object-Oriented Programming (ECOOP'92)*, number 615 in LNCS, pages 57–77, Utrecht, Netherlands, 1992. Springer-Verlag.

[25] Y. J. Hiang. Epistemic logics and epistemic objects. In *UK IT 88 Conference Publication*, pages 120–123, Swansea, UK, July 1988.

[26] R. Jungclaus. *Logic-Based Modeling of Dynamic Object Systems.* PhD thesis, Technical University Braunschweig, Germany, 1993.

[27] R. Jungclaus and G. Saake. Formal specification of object-oriented systems. In S. Abramsky and T. S. E. Maibaum, editors, *International Joint Conference on Theory and Practice of Software Development (TAPSOFT'91), Volume 2: Colloquium on Combining Paradigms for Software Development*, number 494 in LNCS, pages 60–82, Brighton, UK, Apr. 1991. Springer-Verlag.

[28] D. Kato, T. Kikuchi, R. Nakajima, J. Saawada, and T. Tsuiki. Modal logic programming. In *VDM & Z: Formal Methods in Software Development*, pages 29–40, Kiel, Germany, Apr. 1990.
Presents several languages based on Modal Logic which is useful not only for temporal reasoning but also for structuring. Overview of the work done at Kyoto University, Japan. 13 Refs.

[29] N. Kesim and M. Sergot. On the evolution of objects in a logic programming framework. In *International Conference on Fifth Generation Computer Systems*, pages 1052–1060, ICOT, Japan, 1992.

[30] M. Kifer. A first-order formalization of object-oriented languages. *Data Engineering*, 14(2):13–17, June 1991.

[31] M. Kifer and G. Lansen. F-LOGIC: A higher order language for reasoning about objects, inheritance and scheme. In *1989 ACM SIGMOD International Conference on Management of Data*, portland, OR, May 1989. (*SIGMOD Record* 18(2):134–146, Feb. 1990).

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

[33] M. Kifer and J. Wu. A logic for object-oriented logic programming (Maier's O-LOGIC revisited). In *Eight ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 379–393, Philadelphia, PA, Mar. 1989.
We present a logic for reasoning about complex objects, which is a revised and significantly extended version of Maier's O-LOGIC. The logic naturally supports complex objects, object identity, deduction is tolerant to inconsistent data, and has many other interesting features. It elegantly combines the object-oriented and value-oriented paradigms and, in particular, contains all of the predicate calculus as a special case. Our treatment of sets is also noteworthy: it is more general than ELPS and COL, yet it avoids the semantic problems encountered in LDL. The proposed logic has a sound and complete resolution-based proof procedure. (Author abstract) 45 Refs.

[34] C. S. Kwok. A survey of structuring mechanisms for logic programs. In *International Computer Science Conference*, pages 179–188, Hong Kong, Dec. 1988.
Three mechanisms are surveyed: modularization, object-orientation and use of metalanguage. **82 Refs.**

[35] E. Laenens, D. Sacca, and D. Vermeir. Extending logic programming. In *1990 ACM SIGMOD International Conference on Management of Data*, Atlantic City, NJ, May 1990. (*SIGMOD Record* 19(2):184-193, Feb. 1990).

[36] E. Laenens and D. Vermeir. A logical basis for object-oriented programming. In J. van Eijck, editor, *European Workshop on Logics in Artificial Intelligence (JELIA'90)*, pages 317–332, Amsterdam, Netherlands, Sept. 1990.
Describes Ordered Logic (OL) which models object identity, multiple inheritance, defaults. It considers partially-ordered sets of logic theories.

[37] L. Leonardi and P. Mello. Combining logic- and object-oriented programming language paradigms. In B. D. Shriver, editor, *Twenty-First Annual Hawaii International Conference on System Sciences*, volume II: Software Track, pages 376–385, Kailua-Kona, HI, Jan. 1988.
The usefulness and synergetic advantages of combining logic- and object-oriented programming in a declarative framework are explored. Rather than present another specific combination of logic and object programming, the authors discuss different kinds of extensions. 33 refs. **(This is a review of the developments in the area until 1987).**

[38] R. Li and A. Sernadas. Reasoning about objects using a tableau method. *Journal of Logic Computing*, 1(5):575–611, Oct. 1991.

[39] P. Loucopoulos and R. Zicari, editors. *Describing and Structuring Objects for Conceptual Schema Development*, Chichester, UK, 1991. John Wiley & Sons.

[40] D. Maier. A logic for objects. Technical Report CS/E-86-012, Oregon Graduate Center, Nov. 1986.

[41] D. Maier. A logic for objects. In J. Minker, editor, *Workshop on Foundations of Deductive Databases and Logic Programming*, pages 6–26, 1986.

[42] P. Mello and A. Natali. Objects as communicating PROLOG units. In *ECOOP'87: European Conference on Object-Oriented Programming*, number 276 in LNCS, pages 181–192, Paris, June 1987. Springer-Verlag.

[43] J. Meseguer. A logical theory of concurrent objects. In *ECOOP/OOPSLA'90*, Ottawa, Ontario, 1990. (*SIGPLAN Notices*, 25(10):101–115, Oct. 1990).

[44] J. Meseguer. Multiparadigm logic programming. In G. Levi and H. Kirchner, editors, *Third International Conference on Algebraic and Logic Programming*, number 632 in LNCS, pages 158–200. Springer-Verlag, 1992.

[45] J.-J. C. Meyer and R. J. Wieringa. Actor-oriented system specification with dynamic logic. In S. Abramsky and T. S. E. Maibaum, editors, *International Joint Conference on Theory and Practice of Software Development (TAPSOFT'91), Volume 2: Colloquium on Combining Paradigms for Software Development*, number 494 in LNCS, pages 337–357, Brighton, UK, Apr. 1991. Springer-Verlag.
See also [53].

[46] S. G. Pimentel and J. L. Cuadrado. A Horn clause theory of inheritance and temporal reasoning. In J. P. Martinsand and E. M. Morgado, editors, *EPIA '89: Fourth Portuguese Conference on Artificial Intelligence*, number 390 in LNCS, pages 63–72, Lisbon, Portugal, Sept. 1989. Springer-Verlag.

[47] G. Razek. Combining objects and relations. *SIGPLAN Notices*, 27(12):66–70, Dec. 1992.

[48] U. S. Reddy. Objects as closures: Abstract semantics of object-oriented languages. In *1988 ACM Conference on* LISP *and Functional Programming*, pages 289–297, Snowbird, UT, July 1988.
A denotational semantics for SMALLTALK-80 using continuations to model side effects.

[49] A. Sernadas, J. Fiadeiro, C. Sernadas, and H.-D. Ehrich. The basic building blocks of information systems. In E. Falkenberg and P. Lindgreen, editors, *Information System Concepts: An In-Depth Analysis*, pages 225–246, Namur, Belgium, 1989. North-Holland.

[50] B. Stavtrup. A Proposal Regarding Invisible Logic For Object-Oriented Languages. *Journal of Object-Oriented Programming*, 5(1):63–65, 1992.
The author makes a proposal to the ANSI C++ Committee for an extension of the operator-overloading facilities of C++ so that whitespace can also be used for this purpose. I was halfway through the article when I asked myself: "Wait a minute, what month is this issue?". It turned out to be April (now read aloud the first letters of the title). Stavtrup is, of course, a screwed-up variant of Strostrup.

[51] T. Uustalu. Combining object-oriented and logic paradigms: A modal logic programming approach. In O. L. Madsen, editor, *European Conference on Object-Oriented Programming (ECOOP'92)*, pages 98–113, June 1992.
Based on the author's MS Thesis at Tallinn Technical University, Estonia. First a brief account of existing attempts to integrate OOP and LP is given and a categorization is provided. Then the author describes three modal logics (MU, MU' and MU") which provide for different inheritance modes: overriding/cumulative, syntactic/semantic. Objects can provide/receive iheritance selectively for each predicate (method). State change is treated exactly as inheritance: objects inherit their state from the previous time instant in the same way they inherit (part of) their behavior from their ancestors in the inheritance lattice. Thus a two-dimensional modal logic, 2MU, is proposed.

[52] M. Wand. Type inference for record concatenation and multiple inheritance. In *Fourth Annual Symposium on Logic in Computer Science*, pages 92–97, Asilomar Conference Center, Pacific Grove, CA, June 1989. IEEE Computer Society Press. (An extended version appeared in *Information and Computation*, 93(1):1–15, July 1991).
We show that the type inference problem for a lambda calculus with records, including a record concatenation operator, is decidable. We show that this calculus does not have principal types, but does have finite complete sets of types: that is, for any term M in the calculus, there exists an effectively generable finite set of type schemes such that every typing for M is an instance of one the schemes in the set.
We show how a simple model of object-oriented programming, including hidden instance variables and multiple inheritance, may be coded in this calculus. We conclude that type inference is decidable for object-oriented programs, even with multiple inheritance and classes as first-class values.

[53] R. J. Wieringa. A formalization of objects using equational dynamic logic. In C. Delobel, M. Kifer, and Y. Masunaga, editors, *Second International Congress on Deductive and Object-Oriented Databases (DOOD'91)*, number 566 in LNCS, pages 431–452, Munich, Germany, Dec. 1991. Springer-Verlag.
The author first argues that all requirements specified in the OODB Manifesto except object identity and ones

stemming from it (mutable state etc), are accounted for by work done in Abstract Data Types and Equational Order-sorted logic. After that he describes an extension to account for these notions: dynamic logic. Its central concept is the one of object identity; it models changing state by a modification of the possible worlds semantics where modalities are created by a special kind of entities: *events*. The whole system is called Conceptual Model Specification Language (CMSL). See also [45].

[54] M. I. Wolczko. Semantics of object-oriented languages. Technical Report (and PhD Thesis) UMCS–88–6–1, Department of Computer Science, University Manchester, May 1988.

3 Logics for Inheritance Systems; Modules (Worlds) for Logic Programming

Work on modules for LP (multiple logic theories in one program) and inheritance for logic theories is the earliest effort to buy structuring concepts to LP: there has been "worlds" even in the early Prolog II of Colmerauer. Another form of inheritance which is considered in LP is inheritance amongst the data elements (terms); it modifies the way unification works. There is a wealth of theories of inheritance available in the literature.

Although in OOP modules and inheritance are completely orthogonal notions, in LP they are kind of mixed (probably because LP is badly missing both). This is why they are mixed in this section.

The use of the word "worlds" for LP modules is probably just a coincidence with its use in the term "possible worlds", but nevertheless various forms of Modal Logics are widely used to formalize notions of modules and inheritancs.

Monteiro and Porto seem very knowledgeable in this area.

References

[1] K. Akama. Inheritance hierarchy mechanism in PROLOG. In *Fifth Conference on Logic Programming*, number 264 in LNCS, pages 12–21, Tokyo, Japan, 1986. Springer-Verlag.
Maintains a class/instance hierarchy. PROLOG variables can be typed by a class name (so-called Class-Bound Variables). Unification is extended to account for CBVs. Is-a and Part-of hierarchies are treated similarly.

[2] V. Breazu-Tannen, T. Coquand, C. A. Gunter, and A. Scedrov. Inheritance as explicit coercion (preliminary report). In *Fourth Annual Symposium on Logic in Computer Science*, pages 112–129, Asilomar Conference Center, Pacific Grove, CA, June 1989. IEEE Computer Society Press. (An extended version appeared in *Information and Computation*, 93(1):172–221, July 1991).
We present a method for providing semantic interpretations for languages which feature *inheritance* in the framework of statically checked, rich type disciplines. We illustrate our approach on an extension of the language FUN of Cardelli and Wegner, which we interpret via a translation into an extended polymorphic lambda calculus. Our approach interprets inheritances in FUN and *coercion functions* already *definable* in the target of the translation. Existing techniques in the theory of semantic domains can be then used to interpret the extended polymorphic lambda calculus, thus providing many models for the original language. Our method allows the simultaneous modeling of *parametric polymorphism*, *recursive types*, and *inheritance*, something that was regarded as problematic because of the seemingly contradictory characteristics of inheritance and type recursion on higher types. We identify the main difficulty in providing interpretations for explicit type disciplines featuring inheritance, namely that programs can type-check in more than one way. Since interpretations follow the type-checking derivations, *coherence* theorems are required, (that is, one must prove that the meaning of a program does not depend on the way it was type-checked), and we do prove them for our semantic method. Interestingly, proving coherence in the presence of recursive types, variants, and abstract types forced us to reexamine fundamental equational properties that arise in proof theory (in the form of commutative reductions) and domain theory (in the form of strict *vs.* non-strict functions). (Author abstract).

[3] A. Brogi, E. Lamma, and P. Mello. Inheritance and hypothetical reasoning in logic programming. In L. C. Aiello, editor, *9th European Conference on Artificial Intelligence (ECAI'90)*, pages 105–110, Stokholm, Sweden, Aug. 1990.

[4] M. Bugliesi. A declarative view of inheritance in logic programming. In K. Apt, editor, *Joint International Conference and Symposium on Logic Programming*, pages 113–127. The MIT Press, 1992.

[5] W. Cook and J. Palsberg. A denotational semantics of inheritance and its correctness. In *OOPSLA'89*, New Orleans, LA, 1989. (*SIGPLAN Notices*, 24(10):433-443, Oct. 1989).

[6] M. D. A logical analysis of modules in logic programming. *Journal of Logic Programming*, 1(2):79–108, 1989.

[7] C. Dichev. Logic programming with worlds. In B. du Boulay and V. Sgurev, editors, *Artificial Intelligence V: Methodology, Systems, Applications (AIMSA'92)*, pages 57–66. Elsevier Science Publishers, 1992.
Considers various ways to combine clauses for the same predicate residing in different "worlds" (modules). Clauses from the subworld (the specialization) can override completely the predicate, or replace only clauses which unify, or just add themselves to the predicate in the superworld (the generalization).

[8] T. Finin and J. McGuire. Inheritance in logic programming knowledge bases. *Journal of Computer Languages*, 16(3-4):290–310, 1991.

[9] K. Furukawa, R. Nakajima, and A. Yonezawa. Modularization and abstraction in logic programming. *New Generation Computing*, 1(2):169–178, Dec. 1983.

[10] Y. Goldberg, W. Silverman, and E. Shapiro. Logic programs with inheritance. In *International Conference on Fifth Generation Computer Systems*, pages 951–960, ICOT, Japan, 1992.

[11] E. Gregoire. Reducing inheritance theories to default logic and logic programs. In H. Jaakkolo and S. Linnainmaa, editors, *Scandinavian Conference on Artificial Intelligence (SCAI'89)*, pages 493–458, Tampere, Finland, June 1989.

[12] H. Kauffmann and A. Grumbach. MULTILOG: MULtiple worlds in LOGic programming. In *Seventh European Conference on Artificial Intelligence (ECAI'86)*, volume 1, pages 291–305, Brighton, UK, 1986.

[13] P. Mello. Inheritance as combination of Horn clause theories. In D. Lenzerini, D. Nardi, and M. Simi, editors, *Inheritance Hierarchies in Knowledge Representation and Programming Languages*. Wiley and Sons, 1991.

[14] L. Monteiro and A. Porto. Contextual logic programming. In G. Levi and M. Martelli, editors, *Sixth International Conference on Logic Programming*, pages 284–299. The MIT Press, 1989.

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

[16] L. Monteiro and A. Porto. A transformational view of inheritance in logic programming. In D. H. D. Warren and P. Szeredi, editors, *Seventh International Conference on Logic Programming*, pages 481–494. The MIT Press, 1990.

[17] E. Sandewall. Nonmonotonic inference rules for multiple inheritance with exceptions. *IEEE*, 74(10):1345–1353, Oct. 1986.
The semantics of inheritance 'hierarchies' with multiple inheritance and exceptions is discussed, and a partial semantics in terms of a number of structure types is defined. Previously proposed inference systems for inheritance with exceptions are discussed. A new and improved system is proposed, using a fixed number of

nonmonotonic inference rules. The hierarchy is viewed as a set of atomic propositions using the two relations `isa` (subsumption) and `nisa` (nonsubsumption). General results concerning systems of nonmonotonic inference rules can immediately be applied to the proposed inference system. 7 Refs.

[18] D. T. Sannella and L. A. Wallen. A calculus for the construction of modular PROLOG programs. *Journal of Logic Programming*, 12:147–178, 1992.

[19] C. Solnon and M. Rueher. Inference of inheritance relationships from PROLOG programs: A system developed with PROLOG III. In M. Bruynooghe and M. Wirsung, editors, *Programming Languages Implementation and Logic Programming (PLILP'92)*, pages 489–490, 1992.
The first step on the way to objects are abstract types (eventually polymorphic). In the theoretical part of the article types are equated with their extents but in practice the system infers them constructively as a Descartes product, union or intersection of known types. For logic variables the type of the variable is a subtype of the intersection of its types in all its occurences in a clause. A further objective is to specialize types using constraints.

[20] R. H. Thomason and J. F. Horty. Logics for inheritance theory. In M. Reinfrank, J. de Kleer, M. L. Ginsberg, and E. Sandewall, editors, *Second International Workshop on Non-Monotonic Reasoning*, pages 220–237, Grassau, Germany, June 1988.

[21] D. Touretzky. *The Mathematics of Inheritance Systems.* Pitman, London, 1986.

[22] Y.-K. Yang. Behind the inheritance relations in a semantic network. In *IEEE Southeastcom '90 — Technologies Today and Tomorrow*, pages 289–296, New Orleans, LA, Apr. 1990.
There are many confused meanings on the use of ISA (is-a), AKO (a-kind-of), and ISPART (is-part) relations. This problem can be solved only by defining the precise meaning of the ISA, AKO, and ISPART relations, and by recognizing either a class node or an object node for a given node in a semantic network. The author makes clear, precise, and consistent definitions for these inheritance relations based on the two fundamental types of information these relations are intended to represent: classes and objects. The features explored from these relations make clear the use of these three relations and what properties each has. By dividing the properties related by ISA, AKO, and ISPART relations into property values and property attributes, it is found that a property value relates to a specific class or object only and is not an inheritable property, while a property attribute corresponds to the universal quantification in predicate logic and is inheritable by the descendants of a class or object. 11 Refs.

4 Languages Integrating OOP and LP

There are quite a lot of languages (about 30) aiming to integrate OOP and LP. However none of them seems perfect for reasons discussed in the beginning of Section 2. Nevertheless the good ideas abound.

Languages which are heavily bound to Concurrent LP or Constraint LP are grouped in the corresponding sections.

Aït-Kaci [1] and Zaniolo [37] are almost "classics" on the subject. Tyugu [31] has done some really early work in Tallinn, Estonia. McCabe [22] is the only book on the OOP+LP subject known to me. Conery [5, 4, 6] has got a really nice idea about how to model mutable state by literals which pass from one side of a clause to the other side. Kahn [15, 16] seems "the guy with the bunch of ideas" (see also Section 7).

References

[1] H. Aït-Kaci and R. Nasr. LOGIN: A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3(3):185–215, Oct. 1986.
An elaboration of the PROLOG language is described in which the notion of first-order term is replaced by a more general one. This extended form of terms allows the integration of inheritance—an IS-A taxonomy—directly into the unification process rather than directly through the resolution-based inference mechanism of PROLOG.

This results in more efficient computations and enhanced language expressiveness. The language thus obtained, called LOGIN, subsumes Prolog, in the sense that conventional Prolog programs are equally well executed by LOGIN. (Author abstract) 10 refs.

[2] H. Aït-Kaci and A. Podelski. Towards a meaning of LIFE. In *Programming Languages Implementation and Logic Programming (PLILP'91)*, pages 255–274, Passau, Germany, Aug. 1991.
Logic, Inheritance, Functions, Equations (LIFE) is regarded as a composition of three separate instances of the Constraint Logic Programming (CLP) Scheme. Object unification is represented by constraint solving. Type-theoretic, logical and algebraic renditions of the system are provided.

[3] K. Benkerimi and P. M. Hill. Object-oriented programming in Gödel: An experiment. In A. Pettorossi, editor, *Third International Workshop on Meta-Programming in Logic (META'92)*, pages 177–191, Uppsala, Sweden, June 1992.

[4] J. S. Conery. HOOPS: an object-oriented Prolog. Technical report, University of Oregon, 1987.

[5] J. S. Conery. Object-oriented programming with First-Order Predicate Calculus. Technical Report CIS-TR-87-09, University of Oregon, Aug. 1987.

[6] J. S. Conery. Logical objects. In R. A. Kowalski and K. A. Bowen, editors, *Fifth International Conference and Symposium on Logic Programming*, pages 420–434, 1988.
Models mutable state by introducing a new kind of literals—object literals—whose arguments carry the state. A program clause may have object literals both in the body and in the head. The object literal in the body represents object state prior to executing the method, and the literal in the head represents the state after the execution, e.g.
```
push(X), stack(S) :- stack([X|S]).
pop(X), stack([X|S]) :- stack(S).
```
The head is deemed a conjunction of literals, therefore "object clauses" are not really clauses (which are disjunctions of literals). However this does not change the inference method drastically (it is very similar to normal binary resolution) because object literals are not pursued by themselves, but only together with the "real" goals. Thus the proof that some object exists is constructed in parallell with the proof that is has certain properties.

[7] M. Dalal and D. Gandopadhyay. OOLP: A translation approach to object-oriented logic programming. In W. Kim, J.-M. Nicolas, and S. Nishio, editors, *First International Conference on Deductive and Object-Oriented Databases (DOOD'89)*, pages 593–606, Kyoto, Japan, Dec. 1989.
The paper describes two languages: OOLP and OOLP+ which are translated to Prolog.

[8] B. Freeman-Benson. Kaleidoscope: Mixing objects, constraints, and imperative programming. In *ECOOP/OOPSLA'90*, Ottawa, Ontario, 1990. (*SIGPLAN Notices* 25(10):77–88, Oct. 1990).
Tries to integrate the imperative programming paradigm with the declarative-constraint one. The former provides sequencing, the latter provides object relations. Variables are regarded as streams. Multiple views are a natuaral consequence of this integration. 32 Refs.

[9] E. Gullichsen. BiggerTalk: An object-oriented Prolog. Technical Report STP-125-85, MCC-STP, Austin, TX, Nov. 1985.

[10] I. H. and K. H. Extending logic programming to object programming: the system LAP. In *IJCAI'87*, pages 34–39, Milan, Italy, 1987.

[11] J. S. Hodas and D. Miller. Representing objects in a logic programming language with scoping constructs. In D. H. D. Warren and P. Szeredi, editors, *Seventh International Conference on Logic Programming*, pages 511–526. The MIT Press, 1990.

[12] M. H. Ibrahim. KSL: A reflective object-oriented programming language. In *1988 International Conference on Computer Languages*, pages 186–193, Miami Beach, FL, Oct. 1988.

[13] M. H. Ibrahim and F. A. Cummins. KSL/Logic: Integration of logic with objects. In *1990 International Conference on Computer Languages*, pages 228–235, New Orleans, LA, Mar. 1990. IEEE Computer Society Press.
KSL/Logic is an integration of logic and object-oriented programming that adds the declarative framework and deductive reasoning of logic programming to the powerful modeling capabilities of the object-oriented paradigm. Predicates, logic expressions, and the generalized search protocol of KSL/Logic are implemented as an integral part of KSL, a reflective, object-oriented programming language. KSL/Logic provides capabilities that go beyond those of Prolog to permit domain-based reasoning, functional arguments, matching of complex object patterns, and object representation of facts. The syntax and semantics of KSL/Logic are described, and the object implementation of its predicate resolution is examined. 13 Refs.

[14] M. H. Ibrahim and F. A. Cummins. Objects with logic. In *Cooperation. ACM 18th Annual Computer Science Conference*, pages 128–133, Washington, DC, Feb. 1990.
This paper describes an approach to the integration of logic and object programming where predicates, logic expressions, and a generalized search protocol that support Prolog-like reasoning are implemented as an integral part of an object-oriented language. This logic programming facility provides 1) domain-based reasoning, 2) functional arguments, 3) support of the abstraction power of object-oriented languages, and 4) matching of complex object patterns, none of which are available in Prolog. The integration does not require logic facts to be local predicates in the environment; instead, facts are represented as objects in the application model. This permits recursive reasoning and backtracking on predicates that are defined on different domains. The design concepts and implementation of this approach are presented and its application is illustrated by an example. (Author abstract) 10 Refs.

[15] K. M. Kahn. Uniform: a language based upon unification which unifies (much of) Lisp, Prolog and Act 1. In *IJCAI'81*, pages 933–939, 1981.
This early paper claims that the same program may serve as: function, inverse function, predicate, pattern, generator. An extended form of unification may serve as pattern matching, evaluation, message passing, inheritance (an example of the latter is unifying the *description* of red-chairs with the description of big-chairs to get big-red-chairs). Unification is augmented beyond simple Prolog's syntactic unification (which is still the default) by asking the two entities to unify themselves the way they see fit. One can add unification *rules* (e.g. that two expressions unify; similarly to the defining rules in non-free word algebras) thus invoking simplification mechanisms.

[16] K. M. Kahn. Intermission—Actors in Prolog. In K. L. Clark and S. A. Tärnlund, editors, *Logic Programming*, pages 213–228. Academic Press, 1982.
First discusses deficiencies of Prolog: no types (only terms and lists), no lazy evaluation ("virtual data streams"). Claims that implementing Actors in Prolog will carry over the well-understood semantics of Prolog into Actors. The implementation is clean, general, flexible and *very* inefficient. Each new object/method is introduced by extending the predicate `sent(target, message, result)`. Therefore there is no compact (at one place) description of actors: all actor behaviors go through `sent`. The `target` is an actor generally of the type `list(type, acquaintances)`. Delegation: when an actor cannot handle a message, it passes it to its `proxy`. The equivalent of instance variables can be added dynamically.

[17] E. Laenens, D. Vermeir, and B. Verdonk. LOCO, a LOgic-based language for Complex Objects. In *ESPRIT'89: Sixth Annual Esprit Conference*, pages 604–616, Brussels, Nov. 1989.

[18] Y. Lou and Z. M. Ozsouoglu. LLO: An object-oriented deductive language with methods and method inheritance. In *1991 ACM SIGMOD International Conference on Management of Data*, Denver, CO, May 1991. (*SIGMOD Record*, 20(2):198–207, June 1991).

[19] J. Malenfant, G. Lapalme, and J. Vaucher. ObjVProlog: Metaclasses in logic. In S. Cook, editor, *European Conference on Object-Oriented Programming (ECOOP'89)*, pages 257–269, Nothingham, UK, July 1989.

[20] J. Malenfant, G. Lapalme, and J. Vaucher. Metaclasses for metaprogramming in logic. In *Second International Symposium on Meta-Programming in Logic*, pages 257–271, Leuven, Belgium, Apr. 1990.

[21] J. Malenfant, G. Lapalme, and J. Vaucher. ObjVProlog-D: A reflexive object-oriented logic language for distributed computing. *OOPS Messenger*, 2(2):78–81, Apr. 1991.
Implemented in Quintus Prolog. Runs on a network of workstations.

[22] F. G. McCabe. *Logic & Objects*. International Series in Computer Science. Prentice-Hall, 1992.
This is the only book in the area known to me. It is based on the author's PhD Thesis at Imperial College, University of London (1988). It describes the Logic & Objects extension of IC-Prolog. The language provides for multiple logic theories, encapsulated in blocks and marked by "labels" (object identifiers). The labels can be any terms (they may have arguments) thus parameterizing the object theory:

```
person(Age,Sex): {
sex(S) :- S=Sex. % return sex
likes(Person) :- Person:sex(OtherSex), % call to another object (theory)
OtherSex<>Sex. % hmm
}
```

A label may inherit another one by "class rules"; the inheritance is accumulating or overriding; multiple inheritance is allowed; there are no explicit classes (prototype-based language):

```
person<=animal. % class rule: establishes inheritance
person<=socialAgent % multiple inheritance allowed
jim<=person(30,male). % instanceOf the same as subclassOf
penguin<<bird. % overriding inheritance: do not inherit "fly"
penguin: {fly :- fail.} % because penguins don't
```

It also integrates (conditional) equalities and functions, and has destructive assignment:

```
merge([H1|T1],[H2|T2])=[H1|merge(T1,[H2|T2])] :- H1<=H2.
merge([H1|T1],[H2|T2])=[H2|merge([H1|T1],T2)] :- H1>H2.
```

Table of contents: review of a number of mergers; description of the language; discussion of the mixed programming methodology which emerges (divide & conquer *and* browse & modify); examples: *L&O* graphics, a general packer program and a traveling salesman algorithm with graphical interfaces; formal semantics of the proposed extension by translation of *L&O* clauses to Prolog clauses. Although the problem of mutable state is not addressed (assignment does not count; the semantics just disregards it), the language demonstrates very convincingly the synergism of OOP+LP.

[23] G. Mints and E. H. Tyugu. The programming system PRIZ. *Journal of Symbolic Computing*, 5:359–375, 1988. See [31].

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

[25] Z. Palaskas and P. Loucopoulos. Amore — object-oriented extensions to Prolog. In *Technology of Object-Oriented Languages and Systems (TOOLS'89)*, pages 379–393, Paris, France, 1989.
A language for specifying and maintaining information bases. The paper describes also the Rubric runtime system developed in Amore. 39 Refs.

[26] Z. Palaskas, P. Loucopoulos, and F. van Assche. Amore — object-oriented extensions to Prologfor the Rubric implementation environement. In *Sixth Annual ESPRIT Conference*, pages 475–489, Brussels, Belgium, Nov. 1989. Kluwer Academic Prublishers.

[27] J. Plaser. The multiparadigm language G. *Journal of Computer Languages*, 16(3-4):235–258, 1991.

[28] O. R.A. Towards an algebra for constructing logic programs. In *IEEE Symposium on Logic Programming*, pages 152–160, 1985.

[29] F. Staes, E. Laenens, and D. Vermeir. A seamless integration of graphics and dialogues within a logic based object-oriented language. *Journal of Visual Computing*, 1(4):313–332, Dec. 1990.
A User Interface subsystem for the KIWIS system implemented in the language LOCO [17]. 9 Refs.

[30] R. B. Terwilliger and P. A. Kirslis. PK/C++: An object-oriented, logic-based, executable specification language. In *Twenty-Second Annual Hawaii International Conference on System Sciences*, Kailua-Kona, HI, Jan. 1989.
ENCOMPASS is an environment that supports software development using formal techniques similar to the Vienna Development Method (VDM). In ENCOMPASS, software can be specified using the PLEASE family of executable specification languages. PK/C++, the latest member of the PLEASE family, differs from its predecessor by having C++ rather than ADA as its base language, by having an operational as well as declarative semantics, and by being based on flat rather than standard PROLOG. PK/C++ specifications can be used in proofs of correctness. They are also executable, so that initial specifications can be validated and refinements can be verified using testing-based techniques. The authors give an overview of ENCOMPASS, describe PK/C++ in reasonable detail, and give an example of development using the language. 24 Refs.

[31] E. Tyugu. Three new-generation software environments. *Communications of the ACM*, 34(6):46–59, June 1991.
The system PRIZ has been under development at the Institute of Cybernetics, Tallinn, Estonia since mid-seventies. This article describes three environmants of different sophistication based on this system: EXPERT-PRIZ, a simple expert system shell; C-PRIZ, a language integrating imperative programming (C); and NUT (New UTopist; UTOPIST was the initial programming language of PRIZ), an object-oriented environment.
Data elements in UTOPIST (attributes of objects) can be bound by both inter- and intra-object relations. These relations are used by a propositional theorem prover to generate a prooof that it is possible to compute a certain datum given some other data. This (constuctive) proof is used to synthesize a program which computes the datum. So PRIZ has a quite non-standard logic component: compile-time proof and program generation instead of run-time clause resolution. These technique are called Propositional Logic Programming and Structural Program Synthesis. NUT is a prototype-based (as opposed to class-based) language: any object can be used as a template (type) for a new object (and of course as a component of a new object). Limited polymorphism is supported through a generic type (`any`) . It is notable that some of the modern programming paradigms (constraint programming, object-orientation, employing logic in computation) have been considered so early.

[32] E. H. Tyugu. Propositional logic programming. *Computers and Artificial Intelligence (Czechoslovakia)*, 8(4):357–368, 1989.

[33] E. H. Tyugu et al. NUT—an object-oriented language. *Computers and Artificial Intelligence*, 5(6):521–542, 1986.

[34] K. Y. Amalgamating multiple programming paradigms in PROLOG. In *IJCAI'87*, pages 76–86, Milan, Italy, 1987.

[35] H. Yasukawa, H. Tsuda, and K. Yokota. Objects, properties, and modules in QUIXOTE. In *International Conference on Fifth Generation Computer Systems*, pages 257–268, ICOT, Japan, 1992.

[36] S. Yokoi. A PROLOG based object-oriented language SPOOL and its compiler. In *Fifth Conference on Logic Programming*, number 264 in LNCS, pages 116–125, Tokyo, Japan, 1986. Springer-Verlag.
Compiles SPOOL to PROLOG. Methods are represented by PROLOG program clauses and messages by PROLOG calls. SPOOL has multiple inheritance and metaclasses.

[37] C. Zaniolo. Object-oriented programming in PROLOG. In *International Symposium on Logic Programming*, pages 265–270, Atlantic City, Atlanta, Feb. 1984.
Objects are represented as predicates with instance variables modeled by predicate arguments (e.g. `regular-polygon(Sides, Length)`). Methods are attached to predicates by a binary constructor *Object* `with:` *List-of-methods*. Each method is a set of independent clauses which an use the instance variables and delegate

responsibilities through inheritance. Mutable state is modeled by `assert/retract` (modification of the logic program).

5 Implementation-Oriented Developments

I have gathered in this section papers dealing with implementations of OOP in LP or LP in OOP, treating specific implementation isues, discussing various environments, describing what seem to be not very complete and/or general languages, etc.; said shortly, papers I deemed not fitting very well in the previous section. The distinction however is neither very clear nor principled, so generally one should look in both sections.

References

[1] G. Castelli and F. Mariani. An EIFFEL class for the integration of object-oriented and declarative programming. In *Technology of Object-Oriented Languages and Systems (TOOLS'89)*, pages 395–400, Paris, France, 1989.
An EIFFEL class, `Inference`, forms the basis for an extensible PROLOG interpreter. 39 Refs.

[2] P. T. Cox. Using object-orientation to implement logic programming. In *1990 ACM SIGSMALL/PC Symposium on Small Systems*, pages 106–114, Arlington, VA, Mar. 1990.
Implemented in PROGRAPH, a picture-based language. 22 Refs.

[3] T. Koschmann and M. W. Evens. Bridging the gap between object-oriented and logic programming. *IEEE Software*, 5(4 or 5?):36–42, July 1988.
A description is given of an interface that was developed between LOOPS and Xerox QUINTUS PROLOG. LOOPS is an extension to the Xerox AI Environment to support object-oriented programming; Xerox QUINTUS PROLOG is a version of PROLOG that runs on Xerox Lisp machines. Such a bridge enables all the support tools of both environments to be accessed, and degradation of performance that occurs when one language is implemented on top of another is avoided. The interface has three layers. At the lowest level, a set of PROLOG predicates gives the PROLOG programmer access to LOOPS objects. This lowest level is the bridge from PROLOG to LOOPS. At the next level, programming tools in the LOOPS environment let object methods be defined in PROLOG. At the highest level, the PROLOG programmer can treat PROLOG clauses as LOOPS objects that can be manipulated outside the PROLOG database. Each layer can be used independently. 9 Refs.

[4] E. Lamma, M. P., and N. A. An extended Warren Abstract Machine for the execution of structured logic programs. *Journal of Logic Programming*, 14:187–222, 1992.

[5] D. Lanovaz and D. Szafron. An object-oriented inference engine for PROLOG. Technical Report TR 90-18, University of Alberta, 1990.
Implementation of PROLOG in SMALLTALK. Each clause is a (persistent) object which knows how to unify and execute itself as well as which other clauses to call. Part of the GÖDEL project.

[6] G. L. Lazarev. PROLOG/V: PROLOG in the SMALLTALK environment. *Dr. Dobb's Journal of Software Tools*, 13(11/145):68–80, 98–102, Nov. 1988.
Describes briefly (mainly by examples) a commercial PROLOG system developed and integrated in SMALLTALK/V.

[7] M. Levy and R. N. Horspol. Translation of PROLOG to C++. Internal report, University of Victoria, 1990.

[8] V. Loia and M. Quaggetto. Extending CLOS towards logic programming: A proposal. *OOPS Messenger*, 4(1):46–51, Jan. 1993.
Common Lisp Object System (CLOS) provides generic functions, multiple inheritance, meta-object protocol, and declarative method combination. The authors show how PROLOG can be run in it. Selection rule (which

clause to try next) is extended over PROLOG's one: each clause is an instance of a generic method and can check for the input/output mode of its head literal (and even finer details like whether an argument is a `cons`) before unification. Computiation rule (which subgoal to prove next) is the same as in PROLOG: from left to right. Backtracking is implemented by CLOS generators employing lazy evaluation (authors define two macros: `send-value` and `multi-let*`). It seems that there is no compiler from PROLOG to this system yet: CLOS methods are hand-crafted. Bad English.

[9] B. E. Mayfield and J. C. Na. PROLOG methods for COMMON LISP + FLAVORS. In *Knowledge-Based Systems and Neural Networks: Techniques and Applications*, pages 29–39, Stillwater, OK, Nov. 1990.
PLOS (PROLOG/LISP Object System) gives coexistence to PROLOG and COMMON LISP. It includes extended inheritance of PROLOG methods.

[10] E. Meirlaen, J. M. Trinon, and R. Venken. An object-based prototyping workbench in PROLOG. In *Fifth Annual ESPRIT Conference*, volume 1, pages 423–437, Brussels, Belgium, Nov. 1988.

[11] F. Mellender. An integration of logic and object-oriented programming. *SIGPLAN Notices*, 23(10):181–185, 1988.

[12] C. Michel and M. Rueher. Logic programming, object-oriented programming and rapid prototyping. In *Second International Workshop on Software Engineering and its Applications*, pages 417–431, Toulouse, France, Dec. 1989.

[13] T. Reix. SP-OBJECT. Object extensions in the SP-PROLOG V.2.1 system. In *Technology of Object-Oriented Languages and Systems (TOOLS'89)*, pages 395–400, Paris, France, 1989.
SP-PROLOG is the language of choice of Bull SA. The extension is integrated in the system (it is not implemented by a preprocessor). 33 Refs.

[14] S. Roggenbuck, R. Gebhardt, and W. Ameling. PROLOG as method language in an object-oriented programming environment (in German). *Angewandte Informatik*, 31(5):181–188, 1990.

[15] A. Schmidt and F. Belli. Extension of PROLOG for object-oriented programming in logic. In *3rd International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems - IEA/AIE'90*, pages 1153–1161, Charleston, SC, July 1991.
In this paper, we attempt extending Logic Programming 'smoothly' in order to allow object-orientation in a PROLOG-like environment. We call our extension PROLOOP (Yet another PROLOG-based Language for Object-Oriented Programming). PROLOOP is the essential component of a PROLOG-based environment (PROVIRO) to develop knowledge and rule-based expert systems. PROVIRO consists of a series of pragmatic components as to testing (PROTEST), knowledge version control (PROVERS), self actualization of the documentation (PROSELF), etc. The potential of PROLOOP stems from its simplicity. This simplicity makes PROLOOP easy to use and to extend, allows to achieve a high degree of reliability of PROLOOP programs, increases their maintainability, etc. Because of its artlessness, PROLOOP is also a good example for understanding and teaching object-oriented programming. Nevertheless, PROLOOP possesses sufficient expression power which we demonstrate by including non-trivial examples produced in a real project. (Author abstract).

[16] E. P. Stabler, Jr. Object-oriented programming in PROLOG. *AI Expert*, pages 46–57, Oct. 1986.

[17] S. Tyszberowicz and A. Yehudai. OBSERV - a prototyping language and environment combining object-oriented approach, state machines and logic programming. In *Twenty-Third Annual Hawaii International Conference on System Sciences*, volume II: Software Track, pages 247–256, Kailua-Kona, HI, Jan. 1990.
The OBSERV methodology for software development is based on rapid construction of an executable specification (or prototype) of a system, which may be examined and modified repeatedly to achieve the desired functionality. The objectives of OBSERV also include facilitating a smooth transition to a target system and providing the means for reusing specification, design, and code of (sub)systems. Of particular interest is the handling of

embedded systems, which are likely to have concurrency and real-time requirements. The OBSERV prototyping language combines several paradigms to express the behavior of a system. The object-oriented approach provides the basic mechanisms for building a system from a collection of objects, with well-defined interfaces between them. Finite-state machines are used to model the behavior of individual objects. At a lower level, activities that occur within objects are described with the logic-programming paradigm, thus allowing a nonprocedural description when possible. An attempt has been made to provide flexible tools for executing (simulating) the prototype being built, as well as for browsing and static checking. The current implementation of the tools is window-based but not graphical. 26 Refs.

[18] S.-i. Wu. Integrating logic and object-oriented programming. *OOPS Messenger*, 2(1):28–37, Jan. 1991. Describes a logic extension to C++ called LOGIC++. Member functions of C++ classes can be written in PROLOG. A preprocessor accepts methods written in PROLOG and produces C++. The implementation does not seem horribly efficient.

[19] S.-i. Wu. LOGIC++: An integrated logic and object-oriented language. In *USENIX C++ Conference*, pages 235–243, Washington, DC, Apr. 1991.

6 Formal Specification of Object-Oriented Systems

A number of OO Software Engineering methodologies has been proposed (e.g. Wirfs-Brock, Wilkerson and Wiener; Rumbaugh; Booch; Coad and Yordon), but most of them lack rigor. This section includes articles which are aimed to a formal specification of OO systems.

References

[1] J. Fiadeiro, C. Sernadas, T. Maibaum, and G. Saake. Proof-theoretic semantics of object-oriented specification constructs. In R. Meersman, W. Kent, and S. Khosla, editors, *Object-Oriented Databases: Analysis, Design and Construction. Fourth IFIP TC2 WG2.6 Working Conference on Database Semantics (DS-4)*, pages 243–284, Windermere, UK, 1990. North-Holland.

[2] H. B. M. Jonkers. Introduction to COLD-K. In M. Wirsing and J. A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications*, number 349 in LNCS, pages 139–206. Springer-Verlag, 1989.

[3] H. B. M. Jonkers. Inheritance in COLD. In J. A. Bergstra and L. M. G. Feijs, editors, *Algebraic Methods II: Theory, Tools and Applications*, number 490 in LNCS, pages 277–301. Springer-Verlag, 1991.

[4] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. Object-oriented specification of information systems: The TROLL language. Technical Report Informatik-Bericht 91-04, Technical University Braunschweig, Germany, 1991.

[5] G. Saake and R. Jungclaus. Specification of database applications in the TROLL language. In *International Workshop on Specification of Database Systems*, pages 228–245, Glasgow, UK, July 1991.

[6] G. Saake, R. Jungclaus, and H.-D. Ehrich. Object-oriented specification and stepwise refinement. In J. de Meer, V. Heymer, and R. Roth, editors, *International IFIP Workshop on Open Distributed Processing*, volume 1 of *IFIP Transactions C: Communication Systems*, pages 99–121, Berlin, Germany, Oct. 1991. North-Holland.

[7] A. Sernadas, C. Sernadas, and H.-D. Ehrich. Object-oriented specification of databases: An algebraic approach. In *Thirteenth International Conference on Very Large Databases (VLDB'87)*, pages 107–116, Brighton, UK, 1987.

[8] F. J. van der Linden. Object-oriented specification in COLD. Technical Report RWR-508-re-92007, Phillips Research Labs, Eindhoven, Netherlands, Sept. 1992. `flinden@prl.phillips.com`.

[9] F. J. van der Linden. Formal methods: From object-based to object-oriented. Technical Report RWR-508-re-93004, Phillips Research Labs, Eindhoven, Netherlands, Jan. 1993. `flinden@prl.phillips.com`.

7 Connections to Concurrent Logic Programming and Actors

Concurrent LP is intimately connected to OOP+LP, I believe, because of a reason more pragmatic then causal: when one seeks to write really large programs in LP, one has to consider both concurrency (in order to achieve speed) and object orientation (in order to cope with the complexity of the application). On the other hand, because objects are inherently distributed, it is natural to describe distributed systems in terms of objects/actors. However OOP+LP has importance in itself: it can be applied very profitably in sequential programming as well.

This section contains articles both theoretical and dealing with programming languages. There is a lot of articles on Concurrent LP in the most current literature which I have omitted.

A lot of work in this direction has been done in Japan under the auspices of the Fifth Generation Computer Systems Project [1, 2, 6, 8, 9, 10, 16, 17, 18, 21, 22]. Another author to be mentioned here is Kahn.

References

[1] T. Chikayama. ESP–Extended Self-contained PROLOG–as a preliminary kernel language of fifth generation computers. *New Generation Computing*, 1:11–24, 1983.

[2] T. Chikayama. Unique features of ESP. In *International Conference on Fifth Generation Computer Systems*, pages 292–298, Tokyo, Nov. 1984.

[3] A. Davison. POLKA: a PARLOG object-oriented language. Technical report, DOC, Imperial College, London, 1988.

[4] A. Davison. From PARLOG to POLKA in two easy steps. In J. Maluszyński and M. Wirsing, editors, *Third International Symposium on Programming Language Implementation and Logic Programming, PLILP'91*, number 528 in LNCS, pages 171–182. Springer-Verlag, 1991.
The parallel LP language PARLOG is enhanced by some basic OO concepts (encapsulation, data hiding, message passing) to form PARLOG++. Adding multiple inheritance and self-communication renders POLKA.

[5] A. Eliens. Extending PROLOG to a parallel object-oriented language. In *IFIP WG 10.3 Working Conference*, pages 159–170, Lyon, France, Dec. 1989.

[6] K. Fukunaga and S. Hirose. An experience with a PROLOG-based object-oriented language. In *OOPSLA'86*, Portland, OR, Sept. 1986.

[7] T. Hartmann and R. Jungclaus. Abstract description of distributed object systems. In M. Tokoro, O. Nierstrasz, , and P. Wegner, editors, *ECOOP'91 Workshop on Object-Based Concurrent Computing*, number 612 in LNCS, pages 227–244, Geneva, Switzerland, 1991. Springer-Verlag.

[8] Y. Ishikawa and M. Tokoro. Concurrent object-oriented knowledge representation language ORIENT84/K: Its features and implementation. In *OOPSLA'86*, Portland, OR, Sept. 1986.

[9] Y. Ishikawa and M. Tokoro. ORIENT84/K: A language with multiple paradigms in the object framework. In *Nineteenth Annual Hawaii International Conference on System Sciences*, volume II: Software Track, Honolulu, HI, Jan. 1986.

[10] R. Iwanaga and O. Nakazawa. Development of the object-oriented logic programming language CESP. *Oki Technical Review*, 58(142):39–44, Nov. 1991.
Common ESP brings OO to LP. Comes with an adaptible environment.

[11] K. M. Kahn. Vulcan: Logical concurrent objects. In E. S. Shapiro, editor, *Concurrent* Prolog*: Collected Papers*, volume 2, pages 274–303. MIT Press, 1986.

[12] K. M. Kahn. Objects–a fresh look. In S. Cook, editor, *European Conference on Object-Oriented Programming (ECOOP'89)*, pages 207–223, Nothingham, UK, July 1989.
Objects are considered as perpetual recursive predicates which consume mesage streams.

[13] K. M. Kahn, E. D. Tribble, M. S. Miller, and D. G. Bobrow. Objects in concurrent logic programming languages. In *OOPSLA'86*, Portland, OR, Sept. 1986.

[14] K. M. Kahn, E. D. Tribble, M. S. Miller, and D. G. Bobrow. Vulcan: Logical concurrent objects. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 75–112, Cambridge, MA, 1987. MIT Press.

[15] P. Mello. Concurrent objects in a logic programming framework. In *ACM SIGPLAN Workshop on Object-Based Concurrent Programming*, San Diego, CA, Sept. 1988. (SIGPLAN Notices 24(4):37–39, Apr. 1989)
Some papers discuss and point out the synergetic advantages of combining logic and object-based programming. This work is mainly focussed on outlining those advantages with particular reference to concurrency. At the state of the art the most interesting, complete proposals on this topic that deal with concurrency are built on the top of the Concurrent-Prolog logic programming language (CP), which is intrinsically parallel and not compatible with the sequential model of Prolog. An alternative approach could be to directly extend Prolog, the most widely used logic programming language, in order to introduce in it concepts that are typical of parallel and distributed object-oriented systems without losing the advantages of a declarative language. 14 Refs.

[16] H. Miyoshi and K. Furukawa. Object-oriented parser in the logic programming language ESP. In *Natural Language Understanding and Logic Programming, First International Workshop*, pages 107–119, Rennes, France, Sept. 1984. North-Holland.
In this paper we propose an object-oriented parsing mechanism for logic programming language ESP. In this object-oriented parser, each program component is abstracted as a class, and access between the two classes is performed by a message-passing mechanism. Since grammatical categories are also abstracted as classes, the intrinsic grammatical features which are implemented as predicate arguments in DCG, are described as instances of category classes. This helps to simplify Horn clause grammar rule description. Being implemented in the logic programming language ESP, the fundamental mechanism of DCG is also applicable to our parser. 16 refs.

[17] T. Mori and R. Iwanaga. The development of a programming environment for an object-oriented logic programmming language — ESP. *Oki Technical Review*, 57(137):53–58, May 1991.
Discusses an environment for the language ESP on machine PSI (Personal Sequential Inference under SIMPOS (Sequential Inference Machine Program and Operating System.

[18] M. Ohki, A. Takeuchi, and K. Furukawa. An object-oriented programming language based on the parallel logic programming language KL1. In J.-L. Lassez, editor, *Fourth International Conference on Logic Programming*, MIT Press Series in Logic Programming, pages 894–909, 1987.

[19] E. Shapiro and A. Takeuchi. Object-oriented programming in Concurrent Prolog. *New Generation Computing*, 1:25–48, 1983.

[20] J. Vaucher, G. Lapalme, and J. Malenfant. SCOOP: Structured Concurrent Object-Oriented Prolog. In *European Conference on Object-Oriented Programming (ECOOP'88)*, number 322 in LNCS, pages 191–211. Springer-Verlag, 1988.
Object state change is modeled by non-logical means: `assert/retract` predicates to modify the logic program.

[21] K. Yoshida and T. Chikayama. A'UM = stream + object + relation. In *OOPSLA'89*, New Orleans, LA, 1989. (*SIGPLAN Notices*, 24(10):55-58, Oct. 1989).

[22] K. Yoshida and T. Chikayama. A'UM—a stream-based concurrent object language. *New Generation Computing*, 7:127–157, 1990.

8 Connections to Constraint Logic Programming

Since the definition of the Constraint Logic Programming (CLP) Scheme by Jaffar and Lassez CLP has underwent a very fast development. It may be interesting to note that the common area between Concurrent LP and Constraint LP has already been "institutionalized": Concurrent Constraint Programming.

References

[1] W. Havens, S. Sidebottom, G. Sidebottom, J. Jones, and R. Ovans. ECHIDNA: a constraint logic programming shell. In *Twelfth Pacific Rim International Conference on Artificial Intelligence*, Seoul, Corea, 1992.
Integrates object-oriented schema Knowledge Representation, Constraint Logic Programming on reals, intelligent backtracking via justification-type reason maintenance. Objects are persistent predicate schemata. Message passing is done through unification.

[2] M. van Biema, G. Q. Maguire, and S. Stolfo. The constraint-based paradigm: Integrating object-oriented and rule-based programming. In *Twenty-Third Annual Hawaii International Conference on System Sciences*, volume II: Software Track, pages 358–366, Kailua-Kona, HI, Jan. 1990.

[3] S. Watari, Y. Honda, and M. Tokoro. MORPHE: A constraint-based object-oriented language supporting situated knowledge. In *International Conference on Fifth Generation Computer Systems*, pages 1044–1051, ICOT, Japan, 1992.

9 Deductive Object-Oriented Databases

This area is relatively young: the first conference on it was held in 1988 [14]. It is highly correlated to OOP+LP because these databases have to have some database programming language, right? But once again, OOP+LP is a self-important area which may have quite general application.

References

[1] S. Abiteboul. Towards a deductive onbject-oriented database language. *Data and Knowledge Engineering*, 5(4):263–289, 1990.
The author describes a logic-based language for databases with sets, tuples, lists, object identity and structural inheritance. Methods can be overloaded/supplied externally.

[2] A. M. Alashqur, S. Y. W. Su, and H. Lam. Rule-based language for deductive object-oriented databases. In *Sixth International Conference on Data Engineering*, pages 58–67, Los Angeles, CA, Feb. 1990.
A deductive rule-based language for object-oriented databases is presented. A deductive rule in this language derives new patterns of associations among objects of some selected classes if these objects fall in certain 'base' or other derived patterns. The patterns of object associations derived by a rule are held in a subdatabase whose intension consists of some selected classes and their associations. In other words, the structure of a derived subdatabase is represented using the structural constructs provided by the object-oriented data model and hence can be uniformly operated on by other rules to further derive new subdatabases. Therefore, the world of subdatabases is closed under this rule-based language. 23 Refs.

[3] F. Cacace, S. Ceri, S. Crespi-Reghizzi, L. Tanca, and R. Zicari. Integrating object-oriented data modeling with a rule-based programming paradigm. In *1990 ACM SIGMOD International Conference on Management of Data*, Atlantic City, NJ, May 1990. (*SIGMOD Record* 19(2):225–236, Feb. 1990).

[4] Y. Caseau. A deductive object-oriented language. In *Annals of Mathematics and Artificial Intelligence*, Feb. 1991. Special issue on Deductive Databases.

[5] W. Chen, M. Kifer, and D. S. Warren. HiLog as a platform for database languages (or why predicate calculus is not enough). In R. Hull, R. Morrison, and D. Stemple, editors, *Second International Workshop on Database Programming Languages*, pages 315–329 or 121–135?, Glenden Beach, OR, June 1989. Morgan-Kaufmann.

[6] A. El Habbash, J. Grimson, and C. Horn. Towards an efficient management of objects in a distributed environment. In *Second International Symposium on Databases in Parallel and Distributed Systems*, pages 181–190, Dublin, Ireland, 1990.
A prototype of an object-oriented system implemented in C-Prolog is described. Its main objective is to demonstrate system features that would support efficient management of objects and object-oriented databases in a persistent and distributed environment. Mechanisms at the low level of the system were considered to support object distribution, mobility control, and configuration management in a simple and uniform way. Objects exist in clusters, which are transparent to the applications. The prototype is a framework for a self-organizing object-oriented distributed system. 23 Refs.

[7] M. Fornarino, A.-M. Pinna, and B. Trousse. An original object-oriented approach for relation management. In J. P. Martinsand and E. M. Morgado, editors, *EPIA '89: Fourth Portuguese Conference on Artificial Intelligence*, number 390 in LNCS, pages 13–26, Lisbon, Portugal, Sept. 1989. Springer-Verlag.

[8] G. Gardarin. New approaches to advanced database applications. In *First European Conference on Information Technology for Organizational Systems*, page 843, Athens, Greece, May 1988.

[9] J. Grant and T. K. Sellis. Extended database logic. complex objects and deduction. *Information Sciences*, 52(1):85–110, Oct. 1990.
Database logic was proposed in the late 1970s as a generalization of first-order logic in order to deal in a uniform manner with relational, hierarchic, and network databases. At about the same time, the study of deductive (relational) databases has become important, primarily as a vehicle for the development of expert database systems. Also, Prolog, the main logic-programming language, has become prominent for many applications in artificial intelligence, and its connections with deductive databases have been investigated. Although the relational model provides a suitable framework for traditional, essential data-processing applications, several researchers have found the need for complex objects in newer applications, such as engineering databases. In this paper we show how database logic can be extended in two directions: (1) to include complex objects, and (2) to provide deductive capabilities for hierarchic and network databases. Thus, extended database logic provides a logical formalism as the foundation for the study of complex objects. (Author abstract) 18 Refs.

[10] P. M. D. Gray and G. J. L. Kemp. OODB with entity-based persistence. In *Colloquium on Very Large Knowledge-Based Systems*, volume 96, page 4, Stevenage, England, June 1990. IEE, Michael Faraday House.
At Aberdeen we have built an object-oriented database (P/FDM) as a natural extension of Shipman's Functional Data Model (FDM) which is itself founded on entity-relationship concepts. We are using this to store 50 Mb of protein structure data, including the coordinates of every atom in over 80 proteins. The database is mainly used to search for fragments of protein backbone that are of interest, either because of their shape, or because of their relationship to other substructures (helices, sheets, loops). It also has a completely general query language (Daplex), which is founded on set- abstraction, list comprehensions and functions. It can call out to functions which may do arbitrary computations, combined with database search and updates. Most of the database system is written in compiled Prolog; this calls to C routines which access UNIX file structures. The Daplex language is compiled into Prolog, and one can also write complex searches directly in Prolog. 7 Refs.

[11] S. Greco and P. Rullo. Complex-Prolog: A logic database language for handling complex objects. *Information Systems*, 14(1):79–87, 1989.
The logic language paradigm represents a natural extension of relational databases. However, one of its limitations is the lack of suitable data abstraction mechanisms for modeling complex objects. This paper describes Complex-Prolog, a logic database language that provides facilities for data abstraction, notably, the notions of object identity, class and inheritance. This language was designed as an attempt to integrate concepts from logic programming and semantic data models. A formal definition of Complex-Prolog, interspersed with a number of examples, is given. The paper concludes with a description of its implementation. (Author abstract) 24 Refs.

[12] S. Greco and P. Rulo. Complex: An object-oriented logic programming system. *IEEE Transactions on Knowledge and Data Engineering*, (4):344–359, Aug. 1992.
The programming language of the system is Complex-Datalog.

[13] P. Kanellakis and S. Abiteboul. A logical database query language with object identity and strong typing. In G. Levi and M. Martelli, editors, *Sixth International Conference on Logic Programming*, pages 675–692. The MIT Press, 1989.

[14] W. Kim, J.-M. Nickolas, and S. Nishio, editors. *First International Conference on Deductive and Object-Oriented Databases (DOOD'89)*. North-Holland, 1990. Also in *Data and Knowledge Engineering* 5(4), Oct. 1990.

[15] M. Kramer, G. Lausen, and G. Saake. Updates in a rule-based language for objects. In L.-Y. Yuan, editor, *18th International Conference on Very Large Databases (VLDB'92)*, pages 251–262, Vancouver, Canada, 1992.

[16] S. Nurcan and J. Kouloumajian. Structured data in structured logic programming environemnts. In *International Conference on Databases, Parallel Architectures and their Applications (PARBASE'90)*, page 547, Miami Beach, FL, Mar. 1990.

[17] C. Quiming. A deductive database approach for complex objects. *Journal of Computing Science Technology (China)*, 5(3):225–235, July 1990.

[18] C. V. Ramamoorthy and P. C. Sheu. Logic-oriented object bases. In *Third International Conference on Data Engineering*, pages 218–225, Los Angeles, CA, Feb. 1987.
The authors propose the framework of logic-oriented object bases, i.e., databases that are constructed based on object model and augmented by mathematical logic. Adopting logic as a formal means for knowledge representation, they have developed both algorithmic and knowledge-based approaches to relate objects, to evaluate declarative queries that involve high-level concepts, and to schedule declarative update requests such that changes to objects can be made consistently. 25 refs.

[19] G. Saake. Descriptive specification of database object behaviour. *Data & Knowledge Engineering*, 6(1):47–73, Jan. 1991.
Traditional database design methodologies are not appropriate for the specific requirements of object-oriented database systems and new database application areas. Apart from semantic complications arising from object-oriented database structures with complex objects, arbitrary data types as attribute domains, or generalization hierarchies, specification and semantics of dynamic database behaviour has to be of main interest for typical object-oriented applications, too. We propose the use of a temporal logic as a specification language for dynamic object behaviour and point out the formal semantics of such database dynamics specifications. A layered conceptual database design methodology is presented together with a discussion on design support techniques for behaviour specifications. Finally, implementation aspects are treated. (Author abstract) 48 Refs.

[20] P. C.-Y. Sheu and R. L. Kashyap. Query optimization in distributed logic-oriented object bases. *Journal of Parallel and Distributed Computing*, 8(1):60–71, Jan. 1990.

We define a logic-oriented object base to be a deductive database based on an object data model. Like conventional database, a logic-oriented object base system can be constructed on top of a computer network such that distribution of logical and physical components of the system is kept hidden from the users. A distributed logic-oriented object base differs from a distributed relational database in many aspects. For instance, objects are organized hierarchically and objects are retrieved through customized methods. In this paper we investigate the problem of query optimization in distributed logic-oriented object bases. (Author abstract) 25 Refs.

10 Applications to Knowledge Representation

This section introduces some of the application areas which would benefit from a good OOP+LP merger (actually most of the works mentioned here are not mere uses of the technology, but develop quite good and original ideas for a merger). Here are only applications dealing with representing complicately interconnected complex entities (typically for knowledge-based systems). Other applications are enlisted in the next section. Note that it was not possible to divide articles very well between this and the previous section. Some of the presented works may be applied for more general software engineering.

References

[1] G. Antoniou. Logical approaches to structured knowledge bases. In B. du Boulay and V. Sgurev, editors, *Artificial Intelligence V: Methodology, Systems, Applications (AIMSA'92)*, pages 47–56. Elsevier Science Publishers, 1992.

[2] M. Balaban. The Generalised-Concept formalism – an object-oriented, logical framework for knowledge representation. In *Second International Symposium on Methodologies for Intelligent Systems, Colloquia*, pages 179–189, 1987.

[3] M. Balaban and S. Strack. Logster – a relational object-oriented system for knowledge representation. Technical Report TR 88-7, SUNY at Albany, 1987.
A logic kernel wrapped in an object-oriented interface. Uses the Generalised-Concept formalism [2].

[4] R. Brachman. Knowledge representation theory meets reality: some brief lessons from the Classic. In *International Conference on Fifth Generation Computer Systems*, pages 1063–1065, ICOT, Japan, 1992.

[5] R. Brachman, A. Borgida, D. McGuinness, P. Patel-Schneider, and L. Resnick. The Classic knowledge representation system of KL-ONE: The next generation. In *International Conference on Fifth Generation Computer Systems*, pages 1036–1043, ICOT, Japan, 1992.

[6] F.-a. Chen and Y.-f. Zhu. POKRS: A Prolog-based object-oriented knowledge representation system. In *1988 IEEE International Conference on Systems, Man, and Cybernetics*, pages 285–288, Beijing/Shenyang, China, Aug. 1988.
The authors introduce the design and implementation of a Prolog-based object-oriented knowledge representation system (POKRS), which is a developing environment of knowledge systems. The system includes a logic-based object-oriented knowledge representation language based on the combination of object-oriented programming and logic programming. The language has both the capability of knowledge-base organization from object-oriented programming and expressive power from logic programming. The authors describe the language implementing method, an algorithm of message receiving and method searching in the inference mechanism, and the functions of knowledge-base inquiry and maintenance. The system also provides a structural editor as knowledge input tool. POKRS has been implemented in Prolog-KABA language on an IBM-PC microcomputer. 16 Refs.

[7] I. Dimitrov. Systemic programming: a new paradigm for knowledge representation. In Trappl, editor, *Cybernetics and Systems*. World Scientific, 1990.

[8] I. Dimitrov. A systems-based fraamework for knowledge representation. In *Artificial Intelligence IV: Methodology, Systems, Applications (AIMSA'90)*. Elsevier Science publishers, 1990.
An approach borrowed from systems science which is orthogonal to OOP. The KB is represented by means of a set of interconnected subsystems and specification how to combine the methods of the subsystems in order to get the system method. Combinations include ones from CLOS: prologues, main part, epilogues, "around" methods and means to notify one's parents/children in the part-of hierarchy. Particularly good for modeling mechanical systems.

[9] I. Dimitrov. Systems-based knowledge representation: Relations and methods. In B. du Boulay and V. Sgurev, editors, *Artificial Intelligence V: Methodology, Systems, Applications (AIMSA'92)*, pages 203–212. Elsevier Science Publishers, 1992.

[10] A. Doman. OBJECT-PROLOG: Dynamic object-oriented representation of knowledge. In T. Henson, editor, *SCS Multiconference on Artificial Intelligence and Simulation: The Diversity of Applications*, pages 83–88, San Diego, CA, Feb. 1988.

[11] G. Q. Huang and J. A. Brandon. AGENTS: Object-oriented PROLOG system for cooperating knowledge-based systems. *Knowledge-Based Systems*, 5(2):125–136, June 1992.
AGENTS is a multiparadigm language to express the collaborations among cooperating expert systems in an OO language and the deductions inside each agent in a LP language.

[12] M. S. Ibrahim and S. W. Woyak. An object-oriented environment for multiple artificial intelligence paradigms. In *Second International IEEE Conference on Tools for AI*, pages 77–83, Herndon, VA, Nov. 1990.
EDS/OWL integrates uniformly access-oriented, rule-based and LP paradigms in an extensible OO environment.

[13] H. Ito and H. Ueno. ZERO: Frame + PROLOG. In *Fourth Conference on Logic Programming*, number 221 in LNCS, pages 78–82, Tokyo, Japan, 1985. Springer-Verlag.
Logic programs stuffed in the slots of frames (though there is also an external general KB of clauses).

[14] V. Karakostas and P. Loucopoulos. Verification of conceptual schemata based on hybrid object-oriented and logic paradigm. *Information and Software Technology*, 30(10):587–594, Dec. 1988.
Contemporary conceptual modeling languages are concerned with the represenational adequacy of knowledge about a universe of discourse and with the efficient organization of this knowledge in structures that help overcome the problems of size and complexity in the modeled reality. In the paper it is argued that a conceptual modeling language should also facilitate the verification of captured requirements by exercising the conceptual schemata derived from the use of such a language. A conceptual modeling language is presented that is based on a hybrid representation scheme that makes use of object-oriented and logic approaches, and it is shown how this language can be used to verify requirements during the development of information systems. (Author abstract) 27 Refs.

[15] K. Lee and S. Lee. Object-oriented approach to data/knowledge modeling based on logic. In *Sixth International Conference on Data Engineering*, pages 289–294, Los Angeles, CA, Feb. 1990.
The object-oriented data model has gained popularity in developing database systems for new applications which include AI, CAD, and OIS. In modeling such applications, it is necessary to capture not only data semantics but also knowledge semantics, such as constraints and deductive rules. The authors describe an approach to data/knowledge modeling which combines various modeling features of the object-oriented data model with deductive capabilities of the deductive database system. 22 Refs.

[16] F. Mizoguchi, H. Ohwada, and Y. Katayama. LOOKS: Knowledge representation system for designing expert systems in a logic programming framework. In *International Conference on Fifth Generation Computer Systems*, ICOT, Japan, Nov. 1984.

[17] M. Tokoro and Y. Ishikawa. An object-oriented approach to knowledge systems. In *International Conference on Fifth Generation Computer Systems*, pages 623–631, ICOT, Japan, 1984.
The architecture has three parts: behavioral (implemented by logic), Knowledge Base, and monitoring.

[18] A. Turnheim, D. Raveh, and I. Bogomolni. SOLOG–system object oriented logic development. In *1990 IEEE International Conference on Computer Systems and Software Engineering - COMPEURO '90*, pages 556–557, Tel-Aviv, Israel, May 1990.
A brief overview is presented of an intelligent tool, SOLOG, which addresses the problem of system logic development. This tool is part of an environment which is used to support rapid prototyping and development of complex systems. A case study is presented of logic development with SOLOG. 3 Refs.

[19] C. Welsch and G. Barth. Reasoning objects with dynamic knowledge bases. In J. P. Martinsand and E. M. Morgado, editors, *EPIA '89: Fourth Portuguese Conference on Artificial Intelligence*, number 390 in LNCS, pages 257–268, Lisbon, Portugal, Sept. 1989. Springer-Verlag.
Object-oriented programming has proven its appropriateness for simulating real worlds, in particular for imitating human societies and their ability to solve problems. Object-oriented software is easy to modify and extend, a property of great importance for AI applications. Logic programming on the other hand stands out for its declarative specification language, built-in inference capabilities and clear theory. A well known feature of logic programming is the separation of knowledge representation and inference method.
We present a framework which amalgamates object-oriented and logic programming. It combines the object-oriented view with the logic formalism. Objects are considered as reasoning entities whose knowledge bases may change over time. They communicate via messages in order to ask for or to provide information. In response to new information, an object may have to update its knowledge. Operationally, reactions to messages are inference processes based on PROLOG's inference by resolution mechanism. Great importance is devoted to simple and intelligible semantics of knowledge base alterations being the only way to change states. To this end, an object's knowledge base is divided into three parts: assumptions, reflections and reactions, each consisting of Horn clauses. Only assumptions are allowed to be altered. Knowledge can not be modified while an inference process is going on, resulting in easy-to-understand and easy-to-formalize semantics.

11 Other Applications

The variety of applications listed in this section suggests that OOP+LP will be useful in numerous domains.

References

[1] J.-P. A. Barthes and Y. Le Noan. Command and control system based on a multi-media object-oriented data base and a logic programming language. In *Annual AI Systems in Government Conference*, pages 126–132, Washington, DC, Mar. 1989.
The authors present two systems, VORAS and G-BASE, for storing a large number of objects in a Lisp environment, saving them permanently in secondary storage, and providing shared access. Objects use a simple, flexible and powerful model of recursive frames called the property driven model. A number of mechanisms have been implemented such as browsing, complex queries, object-oriented programming, and deduction, leading to a desired result. In utilizing G-BASE, the French Navy has discovered two unique advantages of the product: 1) G-BASE manages all information available and presents it in an intelligent way; and 2) the expert system developer can use the data base objects and relationships without having to build and manage a data base. 35 refs.

[2] E. Corsetti, A. Montanari, and E. Ratto. A methodology for an incremental, logical specification of real-time systems. In *EUROMICRO '90 Workshop on Real Time*, pages 87–94, Horsholm, Denmark, June 1990.
A methodology for an incremental, logical specification of real-time systems which is based on an object-oriented extension of a logical specification formalism is presented. Such an object-oriented framework makes available

primitives for identifying, partitioning, and structuring the elements of a specification. In such a way, it supports a twofold modality of dealing with abstraction, i.e. specialization and decomposition, that provides a guideline for specifications development. In particular, it provides the specifier with the ability to deal with different time granularities within a single specification. That is, it allows the specifier to describe the behavior and the properties of a system and its environment with respect to different time scales, and to switch among them in a suitable way. It also allows an extension of temporal verification and validation of specifications taking into account the incremental development and the resulting layered structure of specifications. 25 Refs.

[3] G. Fleischanderl, G. Friedrich, W. Neijdl, and J. Retti. Integrating logic, object-oriented and procedural paradigms in a fault diagnosis and monitoring system. In *Second International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIES'89)*, June 1989.

[4] P. Loucopoulos and V. Karakostas. Modelling and validating office information systems: an object and logic oriented approach. *Software Engineering Journal*, 4(2):87–94, Mar. 1989.
Developing information systems for the office environment of today requires powerful representation formalisms and techniques capable of modeling all office elements. Furthermore, these formalisms should provide appropriate facilities for the validation of a conceptual schema. In the paper, it is argued that an office modeling approach should provide semantic account for the various aspects of the schema, as well as facilities for simulating its behavior. A conceptual modeling language is presented that combines the object oriented and logic programming paradigms, and it is demonstrated how this language can be used to validate the conceptual design of an office information system. (Author abstract) 24 Refs.

[5] B. Muller. Enhancing software engineering capabilities of PROLOGby object-oriented concepts. In F. Belli and F. J. Radermacher, editors, *5th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems - IEA/AIE'92*, pages 127–138, Paderborn, Germany, June 1992.

[6] L. F. Pau. Context knowledge and search control issues in object-oriented prolog-based image understanding. *Pattern Recognition Letters*, 13(4):279–290, 1992.

[7] A. S. Watson and S. H. Chan. PROLOG-based object oriented engineering DBMS. *Computers and Structures*, 40(1):11–21, 1991.
In this paper we present the primary concepts of PBASE, a prototype object oriented database system. PBASE is intended to support the needs of engineering applications with specific reference to structural engineering. To address the engineering requirements the object oriented data model used in PBASE incorporates several enhancements, including Schema Evolution, Composite Objects, Declarative Methods and Version Management. Schema evolution allows dynamic changes to the class definitions and the class lattice. Composite objects support the is-part-of relationship between assemblies and components. Declarative methods introduce semantics into objects while version management supports the tracking of objects' versions and alternatives as they evolve during the design process. (Author abstract) 44 Refs (First International Conference on the Application of Artificial Intelligence Techniques to Civil and Structural Engineering - CIVIL-COMP'89, London, England).

[8] K. Wiederanders. CSO-PROLOG: A language for knowledge-based object-oriented programming, distributed execution and simulation. In *European Simulation Multiconference*, pages 747–752, Nuremberg, Germany, June 1990.
This language is developed at Multilogic Computing Ltd. , Budapest, Hungary. It integrates combined knowledge (functions, clauses), object-orientation (objects, classes), parallel execution (processes) and simulation (model time).

12 Acknowledgements and Notes

Ralf Jungclaus and Christo Dichev have kindly sent me some references from their own bibliographic databases.

A more thorough survey on Logics for LP will be available from `menaik.cs.ualberta.ca: pub/oolog` in August 1993.

Vote for the creation of the Usenet newsgroup `comp.object.logic`!