F-logic Programming and Terminological Constraints

Christophe RIGOTTI, Mohand-Saïd HACID and Jean-François BOULICAUT

Laboratoire d'Ingénierie des Systèmes d'Information
INSA Lyon Bâtiment 501
F-69621 Villeurbanne Cedex
Tél. : (33) 72.43.85.88 Fax : (33) 72.43.87.13
{crig, mohand, jef}@lisiecrin.insa-lyon.fr

1. Introduction

The CLP scheme of Höhfeld and Smolka [12] has been developed to set up a semantic foundation for the logic programming language LOGIN [1]. Since then, a few instances of this scheme have been built e.g. LIFE [2] (leaving apart its functional component) and TaxLog [4] [10]. The constraints used in these languages have a logical formulation that is rooted in an attributive description formalism (i.e. feature logics and terminological logics) [22]. The ψ-term formalism of LOGIN and LIFE is generalized by a feature logic [28]. TaxLog uses both a terminological logic [21] to define classes (conceptual knowledge of a domain) and an assertional formalism to describe individual objects. The second common property of these languages is that they concentrate the object-oriented flavor at the term level. The first-order Prolog terms are replaced by more general structures that enable a kind of object-oriented programming.

A closely related integration is studied in the ψ-TOOL[1] project. A logic programming language is considered whose clauses are extended by means of constraints based on a terminological logic, the so-called terminological constraints. The main difference with related works is that we do not use Prolog clauses but a logic programming language based on Frame Logic (F-logic) [16]. An overview and a basic operational semantics of such a programming language have been given in [24]. The essential feature of F-logic is its ability to use atoms to describe objects. It can be seen as a first order logic in which some predicate symbols are interpreted as the commonly used relations in object-oriented languages (e.g. inheritance link, class membership, attribute value). The allowed semantic structures are those in which these

1 Programming with Simple Inferences in Terminological and Object-Oriented Logics

predicates are interpreted by relations satisfying the basic properties of the object-oriented paradigm (e.g. the predicate that denotes inheritance must be interpreted by a partial ordering relation).

Designing such an integration, we have at hand a logic programming language that supports two kinds of object-oriented programming: one at the term level (following the TaxLog view) and the other at the atom level (due to the F-logic key notions). The former provides facilities to describe the domain structure of an application by means of classes and instances. The second allows to encode the problem solving knowledge of an application in an object-oriented way (e.g. a path in a state-space will no longer be interpreted by a relation that holds between some states, but it will rather be a class with instances that are particular paths).

In addition to this double layer of object-oriented programming facilities, an extra feature will be gained if F-logic atoms and terminological constraints are used together. If it is straightforward that constraints restrict the kind of terms that can be constructed by the program, we will see that these constraints can also apply to F-logic atoms (e.g. some class membership can not hold w.r.t. the current set of constraints). Hence, we will be able to constrain the objects when describing the problem solving knowledge (e.g. to restrict the kind of transitions allowed in a path).

2. Maturity of the paradigms

The O-logic [19] has been one of the first attempts to design a logic that accounts for the structural aspects of object-oriented languages. Since then, other works (e.g. COL [5], extension of LDL [30]) have been made, among which the F-logic [14] [16] appears to be the most general one. F-logic has a model-theoretic semantics and a sound and complete resolution-based proof procedure. Although no built-in mechanism is provided to handle the dynamic part of the object-oriented paradigm (i.e., state transitions), it has already been used to define the formal semantics of several languages in the field of databases (ESQL2 [11], XSQL [15], DOODLE [9]) and logic programming [17].

Regarding the constraint language, its underlying formalism, the so-called terminological logics, is better known. Terminological logics [21] have been designed as a formal base for the knowledge representation system of the KL-ONE [7] family (e.g. Classic [6], Loom [18] or Back [13]). The reasoning services available in these systems have been elegantly formalized as a satisfiability check of a constraint system [27]. This work has been recently extended [8] to also

handle cycles in the descriptions and assertions about individual objects within a sound and complete calculus for deciding set of constraints satisfiability.

3. Integration overview

We give an informal overview of the two underlying paradigms and their integration. Formal aspects of F-logic and terminological logic can be found respectively in [16] and [21]. The basic operational semantics of the resulting language has been defined by means of a non deterministic abstract machine. This description is introduced in section 3.3 and can be found in [25]. It appears to be the merge of two kinds of operational semantics as described in [26].

A program is split in two parts: a set of constraints $\mathcal{CT}$ (global constraints) and a set of extended clauses $\mathcal{P}$.

An extended clause is of the form:

Bo :- B1, ..., Bn {κ1, ..., κs}.

Where the head Bo and the body B1, ..., Bn are F-logic atoms, and κi (i=1..s) are constraints. The declarative meaning of a set of constraints is the conjunction of the declarative meanings of its members. The declarative semantics of an extended clause of $\mathcal{P}$ is the universal closure of $(B1 \wedge ... \wedge Bn \wedge \{\kappa 1, ..., \kappa s\} \wedge \mathcal{CT}) \Rightarrow Bo$.

To recall the origin of atoms and constraints, we use the syntax commonly devoted to them in F-logic and terminological logics. However, a more homogeneous syntax may be used.

3.1. Description of objects at the term level

At first sight we consider only terms being variables or constants. These terms denote names for the classes and the individual objects.

We use constraints that allow two kinds of descriptions. One stands for the so-called Terminological Box (TBox) and the other for the Assertional Box (ABox) used in Terminological Knowledge Representation Systems [21].

In these systems, the TBox describes the classes while the ABox describes the individual objects. Here, the same dichotomy is used and we consider terminological and assertional constraints.

Examples deal with the problem of computing plans in a state-space.

Let C1, C2, R, X, Y be terms and N a natural number.

The assertional constraints are of the following forms:

- X : C1 — X is an instance of class C1.
- X R Y — one of the values of the role R for object X is Y (in the object-oriented parlance a role R is called a multi-valued attribute).

For the sake of briefty, only restricted (but essential) forms of terminological constraints are given:

- C1 < C2 — C1 is a subclass of C2.
- C1 < not(C2) — C1 and C2 are disjoint classes.
- C1 < all R C2 — instances of the class C1 are objects such that values of role R are instances of the class C2.
- C1 < at-most N R — for a given instance of C1 it may not exist more than N different values for the role R.

Let us define a set of global constraints $\mathcal{CT}$.

$\mathcal{CT}$ = {transition < at-most 1 startFrom, transition < at-most 1 endAt,
move < transition, winningPath < all winningMove move }

This set describes the following three classes: ‘transition’, ‘move’ and ‘winningPath’:

- instances of class ‘transition’ have two single-valued attributes: ‘startFrom’ and ‘endAt’;
- ‘move’ is simply a subclass of ‘transition’;
- instances of class ‘winningPath’ have a multi-valued attribute ‘winningMove’ whose values must be instances of class ‘move’.

We do not give details about the satisfiability test for this constraint language. Formalizations of the operational semantics of the Terminological Knowledge Representation Systems have been given [27,8]. They inspired the satisfiability test defined in [25] that could be used here.

Both kinds of constraints (terminological and assertional) can be used in the constraint part of the extended clauses:

```
p(X,Y) :- q(X), r(Y,Z)
        { X : lastMove, X endAt Z, lastMove < move, lastMove < all endAt winningState }.
/* p, q and r are some predicate symbols */
```

Despite of our restricted form of terms (constants or variables) *à la* Datalog [29], it appears that a definite tree can be described with a set of terminological constraints. So, a syntactic sugar, inspired by those of LIFE [3] (representation of rational trees with ψ-terms), may be used to allow direct mention of conventional Prolog terms.

Such a syntactic sugar has been proposed in [26]. It differs from the solution used in LIFE in that functors are not represented by sorts. Therefore two terms beginning with different functors will never unify.

3.2. Description of objects at the atom level

Let us consider the F-logic atoms used to describe subclass inclusion, class membership, attribute value and conventional Prolog atoms.

Let C1, C2, A, X, Y, T1, ..., Tn be terms and let p be a predicate symbol.

The atoms are of the following forms:

•	C1 :: C2	C1 is a subclass of C2.
•	X : C1	X is an instance of C1.
•	X [A ->> Y]	one of the values of the multi-valued attribute A of the object X is Y.
•	X [A -> Y]	the value of the single-valued attribute A of object X is Y.
•	p (T1, ...,Tn)	a conventional (Prolog-like) atom.

Notice that this syntax should be slightly modified for the sake of clearness. Original notations from F-logic and terminological constraints have been preserved in this paper and it appears sometime confusing (e.g., C1 < C2 and C1 :: C2 both denote that C1 is a subclass of C2).

As a short illustration, let us consider the following (partial) problem solving knowledge:

- X is a winning path if it contains a winning move starting from the same state as X;
- An X contains a winning move M if M is a valid move that leads to the final state of X (the winning position);
- An X contains a winning move M1 if M1 is a valid move and can be followed by a move M2 which is a winning move of X.

This knowledge can be directly encoded in the program composed of the global constraints set $\mathcal{CT}$ of section 3.1 and of the following extended clauses.

```
X : winningPath :- X[winningMove ->> M] { M startFrom Y, X startFrom Y }.
X[winningMove ->> M] :- M : validMove { M endAt Y, X endAt Y }.
X[winningMove ->> M1] :- M1 : validMove, X[winningMove ->> M2]
                                { M2 startFrom Y, M1 endAt Y }.
```

Notice that terminological constraints given in $\mathcal{CT}$ should not be considered as attribute declarations. Any attribute which satisfies current constraints may occur in the clauses. In particular, any unconstrained attribute can occur (e.g., the 'startFrom' attribute).

As valid moves have been represented as instances of a class, this structure can be refined.

```
lowCostMove :: validMove.
M : lowCostMove :- p(M).          /* where p is a predicate symbol */
```

Finally, let us outline the ability to constrain directly the objects used to describe the problem solving knowledge. In the following example, interesting paths are restricted to the ones that contain only low cost moves with no more than four moves:

```
r(X) :- ..., X : winningPath, ...
                { ..., winningPath < all winningMove lowCostMove,
                       winningPath < at-most 4 winningMove, ... }.
```

3.3. Operational semantics

In [25], an operational semantics for such a multi-paradigm language is given by means of a non deterministic abstract machine. Three kinds of non-deterministic choices are to be considered: two choices cope with the resolution inference rule (L-choice and C-choice i.e. which atom has to be proved and which clause has to be selected) while the third one is related to specific F-logic inference rules. Here, only the L-choice is fixed (the leftmost atom is chosen).

A machine state is a pair < G ; S > where G is the current goal to solve (the remaining conjunction of atoms to be solved) and S is the current set of constraints to be satisfied. G is denoted by a sequence of atoms [Go, ..., Gm] where each Gi is an atom. F-logic atoms are considered to be in a classical prefixed way (e.g., C1 :: C2 is supposed to be :: (C1,C2)).

Hence, each atom has the following form:

p (X1, ..., Xh) where p is a predicate symbol and Xi (i=1..h) are terms.

In this abstract machine, we use also equality constraints over terms (terms are restricted to be variables or constants). {A=B} means that A and B denote the same object.

The first transition rule reflects the well-known resolution principle.

a) Resolution

From the current state:

state = < p(X1, ..., Xh), G1, ..., Gm ; S > (h≥0)

Given the selected clause (chosen non-deterministically and after a proper variable renaming):

p (Y1, ..., Yh) :- B1, ..., Bn {κ1, ..., κs}. (s≥0)

The state:

< [B1, ..., Bn, G1, ..., Gm] ;

S ∪ τ (p (X1, ..., Xh)) ∪ {κ1, ..., κs} ∪ {X1=Y1} ∪ ... ∪ {Xh=Yh} >

is reached if and only if the constraint system S ∪ τ (p (X1, ..., Xh)) ∪ {κ1, ..., κs} ∪ {X1=Y1} ∪ ... ∪ {Xh=Yh} is satisfiable.

The function τ translates the current atom to be proved into a set of terminological constraints as follows:

atom L	τ (L)
:: (C1,C2)	{C1 < C2}
: (X1,C1)	{X1 : C1}
-> (X,R,Y)	{X R Y, X : at-most 1 R}
->> (X,R,Y)	{X R Y}
other atoms	∅

So each time an atom is deleted by applying a transition rule, the current constraint system is augmented by a new constraint set that describes this atom.

The initial state of the machine consists in the goal to be solved and the $\mathcal{CT}$ set of global constraints. A final state is a state with an empty goal and a satisfiable set of constraints. This set after simplification is called an answer. The set of all final states that can be reached from the initial state by means of the transition rules, gives the set of all answers.

Let us give now the three transition rules that reflect the F-logic inference rules of [16] for the chosen types of F-logic atoms.

b) Inheritance transitivity

state = < [:: (C1,C2), G1, ..., Gm] ; S >.
selected clause = :: (C3,C4) :- B1, ..., Bn {κ1, ..., κs}.
If S ∪ τ (:: (C1,C2)) ∪ τ (:: (C3,C4)) ∪ {κ1, ..., κs} ∪ {C1=C3} is satisfiable
then next state = < [B1, ..., Bn, C4 :: C2, G1, ..., Gm] ;
S ∪ τ (:: (C1,C2)) ∪ τ (:: (C3,C4)) ∪ {κ1, ..., κs} ∪ {C1=C3} >

c) Inheritance reflexivity

state = < [:: (C1,C2), G1, ..., Gm] ; S >
selected clause = no one
If S ∪ {C1 = C2} is satisfiable
then next state = < G1, ..., Gm ; S ∪ {C1 = C2} >

d) Subclass inclusion

state = < [: (X,C1), G1, ..., Gm] ; S >
selected clause = :: (C2,C3) :- B1, ..., Bn {κ1, ..., κs}.
If S ∪ τ (: (X,C1)) ∪ τ (:: (C2,C3)) ∪ {κ1, ..., κs} ∪ {C1 = C3} is satisfiable
then next state = < B1, ..., Bn, X : C2, G1, ..., Gm ; S ∪ {C1 = C3} >

4. Related works and perspectives

We propose the integration of a logic programming language based on the F-logic and a knowledge representation formalism based on a given terminological logic. The resulting language enables object-oriented programming at two levels while extensions of logic programming commonly focuses on one level at a time. LOGIN, LIFE and TaxLog concentrate the object orientation at the term level. Other languages like Echidna [23] and L&O [20] propose object-

oriented features at the clause level (classes and individual objects are described with sets of clauses). Both approaches enable object-oriented knowledge representations, but at different levels. The first level is concerned with the domain structure of an application while the second level deals with the encoding of problem solving knowledge. Our integration proposal is a step towards languages that offer both kinds of object-oriented representation facilities.

We are currently working on the design of an incremental algorithm to check the satisfiability of our constraint sets. Such an algorithm is needed to achieve a first implementation of an interpreter.

The further work will be twofold:

- to find application fields that make a full use of the double object-oriented level representation ability. We have already proposed an application to object-oriented database application debugging in [25]. Our approach handles the queries involved in an application program without running them against the database content. Queries are not evaluated, and we use only the database schema (class definitions and integrity constraints) to check whether the queries, and the operations performed upon their results, form a consistent sequence. The terminological constraints are used to represent both the database schema and the query representations.

- to study language extension to integrate constraints over some concrete domains, like those of Echidna and TaxLog.

References

[1] H. Aït-Kaci and R. Nasr, "LOGIN: a Logic Programming Language with Built-in Inheritance," *Jounal of Logic Programming*, vol. 3, no. 3, 185--215, October 1986.

[2] H. Aït-Kaci and A. Podelski, "Towards a Meaning of LIFE," *Journal of Logic Programming*, vol. 16, no. 3&4, 195--234, July, August 1993.

[3] H. Aït-Kaci, R. Meyer, P.V. Roy and B. Dumant, *Wild LIFE 1.0 User Manual (Draft) Proteus Project*, DEC-PRL, Paris, October, 1993.

[4] A. Abecker and P. Hanschke, "TaxLog: A Flexible Architecture for Logic Programming with Structured Types and Constraints," in *Proc. of the Workshop on Constraint Processing held in conjunction with CSAM'93,* Petersburg, July 1993, pp. 77--91.

[5] S. Abiteboul and S. Grumbach, "COL: a Logic-based Language for Complex Objects," in *Proc. Int. Conf. on Extending Data Base Technology,* Venise, Italy, March 1988, pp. 271--293.

[6] A. Borgida, R. Brachman, D. McGuiness and L. Resnick, "CLASSIC: A Structured Data Model for Objects," in *Proc. of the ACM SIGMOD Int. Conf. on Management of Data,* Portland, Oregon, June 1989, pp. 59--67.

[7] R. Brachman and J. Schmolze, "An Overview of the KL-ONE Knowledge Representation System," *Cognitive Science*, vol. 9, no. 2, 171--216, April 1985.

[8] M. Buchheit, F. Donini and A. Schaerf, "Decidable Reasoning in Terminological Knowledge Representation Systems," in *Proc. of the Int. Joint Conf. on Artificial Intelligence,* Chambery, France, August 1993.

[9] I. Cruz, "DOODLE: A Visual Language for Object-Oriented Databases," in *Proc.of the ACM SIGMOD Int. Conf. on Management of Data,* M. Stonebraker, San Diego, California, June 1992, pp. 71--80.

[10] T. Frühwirth and P. Hanschke, "Terminological Reasoning with Constraint Handling Rules," Tech. Rep., ECRC, München, Germany, 94-06, February 1994.

[11] G. Gardarin and P. Valduriez, "ESQL2: an Object-Oriented SQL with F-logic Semantics," in *Proc. of the Int. Conf. on Data Engineering,* IEEE, Phoenix, Arizona, February 1992.

[12] M. Höhfeld and G. Smolka, "Definite Relations over Constraint Languages," Tech. Rep., IBM Deutschland, Stuttgart, Germany, LILOG Report 53, October 1988, To appear in Journal of Logic Programming.

[13] T. Hoppe, C. Kindermann, J. Quantz, A. Schmiedel and M. Fisher, *BACK V5: Tutorial and Manual*, Technische Universitat Berlin, March, 1993.

[14] M. Kifer and G. Lausen, "F-logic: A higher-order language for reasoning about objects, inheritance and schema," in *Proc.of the ACM SIGMOD Int. Conf. on Management of Data,* 1989, pp. 134--146.

[15] M. Kifer, W. Kim and Y. Sagiv, "Querying Object-Oriented Databases," in *Proc.of the ACM SIGMOD Int. Conf. on Management of Data,* M. Stonebracker, San Diego, California, June 1992, pp. 393--402.

[16] M. Kifer, G. Lausen and J. Wu, "Logical Foundations of Object-Oriented and Frame-Based Languages," Tech. Rep., Dep. of Computer Science, SUNY at Stony Brook, NY 11794, 93/06, April 1993, revised June 1993, to appear in Journal of ACM.

[17] M. Lawley, "A Prolog Interpreter for F-logic," Tech. Rep., Griffith University, Queensland, Australia, April 1993, p. 14.

[18] R. MacGregor, "Inside the LOOM Description Classifier," *SIGART Bulletin*, vol. 2, no. 3, 88--92, June 1991.

[19] D. Maier, "A Logic for Objects," in *Proc. of the Workshop on Foundations of Deductive Databases and Logic Programming,* Washington, August 1986, pp. 6--26.

[20] F. McCabe, *Logic and Objects*. Prentice Hall, 1992.

[21] B. Nebel, *Reasoning and Revision in Hybrid Representation Systems*, vol. 422, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1990.

[22] B. Nebel and G. Smolka, "Attributive Description Formalisms ... and the Rest of the World," in *Text Understanding in LILOG: Integrating Computational Linguistics and Artificial Intelligence*, O. Herzog and C. Rollinger, Eds. Springer-Verlag, Berlin, Germany, 1991.

[23] R. Ovans and S. Sidebottom, "Solving Logic Problems with Object-Oriented Constraint Logic Programming," in *Proc. of the IEEE Pacif Rim Conf. on Communications, Computers and Signal Processing,* 1993.

[24] C. Rigotti, M.S. Hacid and J.F. Boulicaut, "Does a Deductive Object-Oriented System produce Object-Oriented Answers? A Hybrid Reasoning Approach," in : Proc. of the 10th conference INFORSID'94, Aix en Provence (F), 17-19 mai 1994, 15 p.

[25] C. Rigotti, M.S. Hacid and J.F. Boulicaut, "Une Approche Multi-Paradigmes pour tester la cohérence des applications BDOO", To appear in : Proc. of "Bases de Données Avancées" BDA'94, Clermont-Ferrand (F), 30-31 Aout- 1-2 Septembre 1994, 15 p.

[26] C. Rigotti, J.F. Boulicaut and M.S. Hacid, "Vers une Typologie des Sémantiques Opérationnelles pour les Extensions Objets de Prolog", To appear in : Proc. of JFPL'94, 3° Journées Francophones de Programmation en Logique (M-M. Corsini Ed.), Bordeaux (F), 25-27 mai 1994, 15 p.

[27] M. Schmidt-Schauss and G. Smolka, "Attributive Concept Description with Complements," *Artificial Intelligence*, vol. 48, 1--26, 1991.

[28] G. Smolka, "A Feature Logic with Subsorts," in *Unification in Grammar*, J. Wedeking and C. Rohrer, Eds. MIT Press, 1992.

[29] J.D. Ullman, *Principles of Database and Knowledge-Base Systems*, Computer Science Press, 1988.

[30] C. Zaniolo, "Object Identity and Inheritance in Deductive Databases--an Evolutionary Approach," in *Proc. of the First Int. Conf. on Deductive and Object-Oriented Databases (DOOD'89),* Elsevier Science North-Holland, Kyoto, Japan, December 1989, pp. 7--24.