H. Kirchner G. Levi (Eds.)

CC01-632

Algebraic and Logic Programming

Third International Conference
Volterra, Italy, September 2-4, 1992
Proceedings

BIBLIOTHEQUE DU CERIST

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Program Committee

Egidio Astesiano (Genova)
Hubert Comon (Orsay)
Alexander Herold (München)
Joxan Jaffar (Yorktown Heights)
Hélène Kirchner (Nancy), Co-chair
Jan Willem Klop (Amsterdam)
Giorgio Levi (Pisa), Co-chair
Horst Reichel (Dresden)
Mario Rodriguez-Artalejo (Madrid)
Vijay Saraswat (Palo Alto)
Gert Smolka (Saarbrücken)

Professor Wolfgang Wechler from Würzburg University, the initiator of this series, was initially a member of this committee. His premature and sudden death has left his colleagues and friends very shocked by this tragic event.

Local Organization

Roberto Bagnara, Piero Bonatti, Maurizio Gabbrielli, Roberto Giacobazzi, Anna Maria Manunta, Maria Chiara Meo, Danilo Montesi, Stefania Pellegrini, Chiara Tricomi.

List of Referees

M. Alpuente, R. Amadio, J.-M. Andreoli, J. Andrews, I. Bethke, A. Boudet, A. Brodsky, F. Bry, A. Corradini, G. Costa, E. Domenjoud, M. Falaschi, G. Ferrari, T. Frühwirth, D. de Frutos Escrig, M. Gabbrielli, A. Gavilanes-Franco, R. Gerth, R. Giacobazzi, A. Gil-Luezas, A. Giovini, E. Hamoen, N. Heintze, M. Henz, D. Hofbauer, M.T. Hortalá-González, C. Jutla, J.R. Kennaway, C. Kirchner, F. Klay,

G. Kucherov, T.M. Kuo, H. Leiss, P. Lim, D. Lugiez, M. Maher, P. Mancarella, L. Maranget, B. Marre, K. Marriott, M. Martelli, A. Masini, A. Middeldorp, M. Meier, E. Moggi, E. Monfroy, R. Nieuwenhuis, F. Orejas, Y. Ortega-Mallén, V. van Oostrom, C. Palamidessi, P. Panangaden, D. Pedreschi, M. Posegga, S.D. Prestwich, L. Puel, G. Reggio, P. Réty, F. Rossi, J. Rouyer, A. Rubio, M. Rusinowitch, A. Salibra, J. Schimpf, P.-Y. Schobbens, B. Thomsen, R. Treinen, L. Vigneron, P. Viry, M. Vittek, A. Voronkov, F.-J. de Vries, R.C. de Vrijer, J. Würtz, D. Yankelevich, R. Yap, W. Zadrozny, H. Zantema.

BIBLIOTHEQUE DU CERIST

Contents

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

Outline of an Object-Oriented Calculus of Higher Type

(Abstract of Lecture)

Hassan Aït-Kaci

hak@prl.dec.com

Digital Equipment Corporation, Paris Research Laboratory
85, avenue Victor Hugo, 92500 Rueil-Malmaison, France

BIBLIOTHEQUE DU CERIST

Object-oriented programming was introduced (essentially thanks to Smalltalk) as a bag of purely empirical, but nonetheless seductive, ideas. The most important of these were class hierarchies, object instance-variables, methods, message-passing, self, and object identity. Thus, this set of concepts is by no means a monolithic entity but consists of several orthogonal notions *de facto*, and somewhat incidentally, amalgamated under this same qualifier.

After adopting an initial attitude towards these ideas between scorn and derision, formalists have recently made a radical change of opinion. Several, among the best, have taken a sudden interest in formalizing object-oriented programming and proposed theoretical constructions of impressive mathematical complexity in order to explain rigorously the foregoing empirical concepts. This is in evident contrast with their intuitive understanding which is nevertheless of a disarmingly elementary simplicity.

Be that as it may, I strongly believe, as a scientist, that a sound scientific approach requires that a formal rendition should be given which characterizes precisely and exactly the essence of an empirical phenomenon. On the other hand, I am no less convinced that the tools and efforts employed in achieving this must be of intrinsic complexity at most comparable to that of the object of study. Even further, I think that formalization has a justification only if it pays in return the object of study by simplifying it, generalizing it, or better yet, improving it. From this standpoint, then, it is clear that the formal models of object-oriented programming currently offered are blatantly inadequate.

Where did these formalists go wrong? First of all, they simply omitted to ask themselves the question, necessary *a priori*, whether everything in object-oriented programming's bag of ideas were of a common essence. Secondly, and more importantly, they did not question the actual need for some of these features nor their use. For example, is it really necessary to distinguish between instance-variables and method names? Or, is it clean to introduce the concept of "self" as it is done to allow an object to refer to itself? Or, is message-passing anything other than calling a curried function as it has been partially evaluated on its first-argument? And if so, why this asymmetry limiting this capability only to first arguments? Finally, should the very idea of inheritance not rather rest on a notion of approximation, quite more advantageous for static analysis,

instead of a mere arbitrary data organization scheme for supporting code sharing and overloading?

Taking these questions as a guideline, as well as a few others reassessing taken-for-granted mottos related to object of higher-types (*e.g.*, left contravariance), I will outline *SCOOP* – a Simple Calculus for Object-Oriented Programming. The aim of *SCOOP* is to embody a formal calculus particularly simple and uniform based on approximation structures of first and higher types. It is meant to account rigorously for an essential part of object-orientation (namely, inheritance, instance-variables, methods, self-reference, and message-passing) while staying faithful to their empirical intuition. It simplifies and demystifies the concept of self-reference that causes awful complications in other formal accounts. In addition, *SCOOP* brings about substantial improvements over the empirical concepts as it eliminates frustrating discrepancies such as the asymmetry of message-passing towards first argument only, or the useless distinction between instance-variables and methods. Finally, it offers to construe of inheritance as approximation (realized through a structural endomorphism) that fully commutes with evaluation. This is quite a novel insight as it provides the calculus with an automatic power of abstract interpretation based on object refinement. In other words, it allows to compile a program with the same interpreter used at run-time.

Although this research is still at its infancy, I believe that *SCOOP* and the methodology on which it rests offer an exciting potential. I shall describe the operational model of *SCOOP* in some detail, and illustrate its behavior on concrete examples.

BIBLIOTHEQUE DU CERIST

This article was processed using the LaTeX macro package with LLNCS style

BIBLIOTHEQUE DU CERIST

High-Level-Replacement Systems for Equational Algebraic Specifications

H. Ehrig

Fachbereich Informatik, Technische Universität Berlin
D-1000 Berlin 10 Germany

F. Parisi-Presicce

Dipartimento di Matematica Pura ed Applicata, Università de L'Aquila
I-67010 Coppito (AQ) Italy

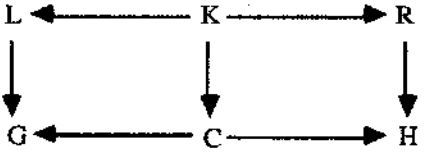
Abstract Equational algebraic specifications and the corresponding specification morphisms have been defined in the literature in several ways. Although apparently equivalent, they are significantly different with respect to standard categorical constructions, leading to categories of algebraic spacifications which are not equivalent. The nonequivalence of these categories of algebraic specifications is also significant in the context of high-level-replacement (HLR) systems, a generalization at the categorical level of the well known algebraic approach to graph grammars based on double pushout. Unexpectedely, only for some of the categories the properties needed to prove the Church-Rosser, Parallelism and Concurrency Theorems for High-Level-Replacement systems are valid.

1 Introduction

The use of algebraic specifications for the definition of abstract data types dates back to at least 1974 and has influenced, since then, the theoretical foundations of some formal methods for the development of reliable software as well as some applications. The 'simplest' framework adopted for the algebraic specifications is the equational one, in which properties of the functions being defined are expressed as universally quantified equations between pairs of terms. Several authors have used and analyzed equational specifications and the differences among the formalizations have often been considered negligible. These differences have then been extended to distinct formalizations of the notion of specification morphism f: SPEC1 $\rightarrow$ SPEC2, all based on the accepted definition of signature morphism f_Σ : SIG1 $\rightarrow$ SIG2 , but differing in the way the equations are related. Some definitions of specification morphisms require that the translation f#(E1) of the equations of SPEC1 induced by the signature morphism f_Σ be among the equations of SPEC2, other morphisms

require only that they be derivable (in the usual equational logic) from those of SPEC2.

Different morphisms give rise to different categories in which the standard categorical constructions are carried out. These constructions are typically limits or colimits [9] which model, for example, the modular definition of specifications via the mechanism of parametrization and parameter passing. We have shown that two of the four categories considered are equivalent and that the three non equivalent categories are consistent with respect to pushout constructions but not with respect to pullbacks, that is, the functors relating the different categories do not preserve pushouts and pullbacks, but while the different pushout objects have the same category of models, the different pullbacks do not. The differences among these categories of algebraic specifications are even more significant for the corresponding High Level Replacement (HLR) Systems [6,7]. These systems generalize at the categorical level the algebraic approach to graph grammars based on a double pushout construction [4], in which a production p consists of two morphisms $L \leftarrow K \rightarrow R$ (selected from a subset M of morphisms) and a direct derivation p: $G \Rightarrow H$ from G to H consists of two pushouts, in the category under consideration, as in the following diagram



The object C represents the context which is not affected by the transformation and which is "glued" to the new part R via the interface K.

The HLR-systems can be seen as a categorical rewriting system in which the original rewriting of graphs has been replaced by the rewriting of other structures, such as algebraic specifications, hypergraphs, graphics, place-transition nets, jungles, etc.. Applications of these replacement systems range from modular system design [15, 16] in which derivations can be automatically translated to interconnections of the modules whose interfaces have been used as productions; to logic programming [2, 3] in which derivations simulate the refutation procedure; to rule-based systems [10, 8] with the analysis of conflicting rules and the consequent reduction of the search space.

Among the important properties of these rewriting systems, we can list the Church-Rosser Theorem and the Parallelism and Concurrency Theorems. The CR Theorem indicates when the application of two rules (or productions) is independent of the order in which they are applied; the Parallelism Theorem states that the effect of applying sequentially two independent productions is equivalent to the simultaneous application of (the coproduct of) the productions, while the Concurrency Theorem handles the case of the sequential application of two rules which are not independent. In this paper we have refined the axiomatic proofs of these Theorems in the context of HLR-systems and shown that only one of the nonequivalent categories of algebraic specifications satisfies the HLR properties which guarantee all three results, while for the other ones only subsets of these properties are satisfied depending on the appropriate choice of the distinguished class M of morphisms used in the productions. This indicates that only one of the formalizations of equational specifications is the most appropriate to fully exploit, for example, the correspondence between derivations and interconnections of

BIBLIOTHEQUE DU CERIST

modular systems [15].

In the next section, we define the different categories and characterize, for each of them, the mono- , epi- , and iso-morphisms, the initial and final objects and the categorical constructions of pushouts and pullbacks. In section 3, we present the different results, without proofs, concerning the weak and (lack of) strong equivalence between pair of categories. In section 4, after reviewing briefly the Church-Rosser- , Parallelism- , and Concurrency Theorems, we show which of the HLR properties that imply them are satisfied by the different categories with different choices of the class M of morphisms. Some open problems are stated in the last section. All proofs are omitted and can be found (in painstakingly details) in [13].

2 Categories of Equational Algebraic Specifications

We first define five different alternatives for categories of standard equational algebraic specifications that have been introduced in the literature, where in each case an equational algebraic specification SPEC = (S, OP, E) consists [9] of a set S of sorts, a set OP of operation symbols N: s1 ... sn → s and a set E of equations of the form (X, t1, t2) with t1, t2 terms of the same sort with variables X. .

2.1 Definition

Type 0: The specification morphisms f : SPEC → SPEC' are signature morphisms $f_\Sigma = (f_S : S \to S',\ f_{OP} : OP \to OP')$ satisfying $f_\Sigma (N: s1 \ldots sn \to s) =$ $f_{OP}(N) : f_S(s1) \ldots f_S(sn) \to f_S(s)$ and such that the translated equations f#(E) are derivable from E' (see [9, 16]).

Type 1: Same as Type 0, but with the condition f#(E) ⊆ cl(E') where cl(E') is the closure of E' under derivability.

Type 2: Same as Type 0, but with the condition f#(E) ⊆ E' [6, 7, 15, 11].

Type 1': Type 2 restricted to specifications SPEC = (S, OP, E) where E is already closed under derivability (theories in the sense of [1]).

Type 3: Specifications SPEC = (S, OP, E) where

+ the set of equations is labelled, and different labels e may correspond to the same triple (X, t1, t2) representing the equation e:t1 = t2, and

+ specification morphisms are triples $f = (f_S:S \to S', f_{OP}:OP \to OP', f_E:E \to E')$ with the usual compatibility properties for f_S and f_{OP} and for $(e:t1 = t2) \in E$ we have $(f_E(e):f\#(t1) = f\#(t2)) \in E'$ (see [12]).

In fact type 0 is most often used in the literature and it is well-known that the type 0-condition "f#(E) derivable from E' " is equivalent to the type 1-condition "f#(E) ⊆ cl(E')" in the sense that both types define the same category. In the following we will only use type 1 in order to avoid explicit notions of derivability.

The category having algebraic specifications as objects and morphisms of type i will be denoted by **SPECi** for i = 1, 2, 3, 1'. We use the terminology type 1' because the

BIBLIOTHEQUE DU CERIST