Estratto da

R. Ferro e A. Zanardo (a cura di), *Atti degli incontri di logica matematica*, Volume 3, Siena 8-11 gennaio 1985, Padova 24-27 ottobre 1985, Siena 2-5 aprile 1986.

Disponibile in rete su http://www.ailalogica.it

ILLOGICAL PROGRAMMING

GABRIELE RICCI

Summary. We motivate and report some work on automated programming that yielded an experimentally tested program generator. Its theory focuses the objects to be computed more than the programming machinery involved. Thus, the tools developed for it differ from the logical ones in their use and in some technicalities.

0 A bottom up approach. A common feature of present applications of Logic to programming is their high level globality. E.g. in practical applications, Logic is used for modeling or inspiring a whole programming language and, in theoretical ones, for formalizing its semantic [1]. On the contrary, this communication concerns some work motivated by an approach complementary to the usual one. The outcome of this work, in fact, is a (small) program generator, viz. something closer to a single ready to use program than to a wide programming language.

(Program generators, even as trivial as a printing procedure with parametrized formats, immediately satisfy user's needs, though within a restricted field. Also, they often provided the high level programming languages with their building blocks.)

Two questions rise from a bottom up approach to high level programming as ours. Does it work even for non trivial tasks ? Does it have something to do with Logic ?

1 What got. An affirmative hint for the former question is provided by a sample of output programs from our generator. Its input parameter is an algebra, to be provided by user procedures at now or, within a possible high level translator, by declarations only. Here is what the generator returns you after such an input.

Input algebra	Generated program serves to
Finite vector space	Integrating systems of linear difference equations on a Galois field. (NEW)
Complete semilattice [2] of union	Finding weakly connected components in a graph. (WELL KNOWN)
Boolean set algebra	Decomposing an encoded clock (with delays) into "behavioral" components. (NEW)
Complete max and successors on partial tables of natural numbers	Scheduling of PERT projects. (WELL KNOWN)

The ability of performing such heterogeneous tasks comes from a simple idea: get a program performing a single non trivial task (in our case, solving eigenvalue equations) within usual vector spaces and "parametrize" it by allowing algebras other than vector spaces.

2 Relevance to Logic. As far as the latter question in O is concerned, our bottom up approach leads us to two opposite findings. In fact, tools similar to the usual Logic ones do be needed, whereas their use and some technical details conflict against the usual ones.

A need for "combinators" rises while writing a program generator as ours. In fact, the algebra parametrization requires a lot of new theory oriented to computational purposes. This is difficult, since present Universal Algebra comes from a merely extensional view of functions, contrary to traditional geometry, where analytic geometry provides an intensional view also (hence,

contrary also to Whitehead's proposal [5]).

The only analytic tool (feebly) accepted by universal algebraists (Menger's superassociative systems) is an imported item, yet they did not pick up the accompanying [3] proposal, about some kind of combinators, at all. (Combinators, at least as a notational aid, do be useful for expressing categorical constructs intensionally and specifically.) Thus, our parametrization demands some rebuilding of the very foundations of Universal Algebra. Yet, there was no need for a wide rebuilding. E.g. Set-Theory was not relinquished.

In such circumstances, the useful "combinators" differ from the logical ones. To begin with, we cannot throw pairs away by Schönfinkel repeated applications (nor by diads) easily. Then, we cannot consider **C** being a trivial combinator, e.g. as a generalization of matrix transposition.

(We can easily get it even in the case of classical geometry. In fact, consider the function χ mapping a vector v into the linear form χv identified by its coordinates. Thus, in the projective case, χ is Plücker isomorphism between a space and its dual. Then, $\mathbf{C}\chi$ does not look a "transposed" χ at all. It is the isomorphism from "Cayley" monoid of matrix product onto "Klein" endomorphism monoid, see 2.6 of manuscript A in 3.)

A further difference, related with retaining pairs, is that **B** comes to have **two** principal types. Probably, we would not find difficulties of this kind, if we had an embedding available of Algebra and Categories into Combinators, viz. if somebody had solved a converse of the problem tackled by Lambek (the equivalence between Combinators and some Categories).

Programming our generator, i.e. purposefully choosing its input algebra, pushes the user to approach other Logic tools. Here, the problem arises from the input algebra looking unrelated with the generated programs. It

looks like as the generator "invent" the algorithms it generates. Contrary to conventional programming, programmer 's intuition looses control over the logic of a generated program. Likely, this problem will occur in any form of automated programming.

Here is where Category Theory plays a role. In fact, it offers conditions stating when output relevant objects can be made to form an input algebra and it hints how the algebra has to be structured. However, these categorical results are the low level ones, viz. the ones from the applied categories of [2]. Moreover, a still lower level and some intensional machinery would likely benefit the programmer.

3 Available material and work in progress. The program generator, its description and test runs are in C. The relevant theory is in A and B. A also contains some algebraic application and B some link with other Computer Science problems. (All papers are manuscripts available from the Author.) Some further theory, relevant to certain extensions done in C, is in [4].

A Universal eigenvalue equations, 1982.
B You don't need numbers to do integrations, 1983.
C A Whitehead generator: a new high level programming springs from a an old conjecture, 1985.

At now, a faster algorithm (exploiting an extension of superassociative systems) is being implemented for our generator.(Indeed, the programs generated by the present algorithm are too slow, when they can compare with possible conventionally programmed ones. The feasibility of "illogical" automated programming would be shown even on a cheap videogame machine.

References

[1] H.P. Barendregt, The λ-calculus:its syntax & semantics, North-Holland, 1984.
[2] E.G. Manes, Algebraic theories, Springer-Verlag, 1976.
[3] K.Menger, Superassociative systems and logical functors, Math. Annalen **157** (1964).
[4] G.Ricci, P-algebras & combinatory notation,Riv.Mat.Univ.Parma,**5**(1979).
[5] A.N.Whitehead, Universal Algebra, **1**, Cambridge U.ty Press, 1898.