DOMAIN EQUATIONS AND VALID ISOMORPHISMS IN ALL MODELS OF (HIGHER ORDER) LANGUAGES. A short discussion

KIM BRUCE[1], GIUSEPPE LONGO[2]
[1]Williams College (Massachusetts)
[2]Dipartimento di informatica Università di Pisa

One of the main problems in the last 10-15 years research in Mathematical Computer Science has been suggested by the attempt to specify the semantics of functional languages. By this we mean the solution of recursive domain equations, by using methods and tools from Mathematical Logic.

As a matter of fact, Scott's motivation for finding a non-trivial solution to the equation

(0) $D = A + [D \to D]$

was based on mathematical investigations of the semantics of paradigmatic functional languages, λ-calculus and the theory of Combinators (Combinatory Logic, C.L.).

By a "non-trivial" solution to the above equation we first mean that A is non-empty or that D is larger than a singleton set. Then, of course, the set $[D \to D]$ cannot include all functions over D, by Cantor's theorem. For the purpose of Computer Science, though, $[D \to D]$ has to be at least rich enough so that, for some suitable interpretation of the natural numbers by a countably

Research partially supported by National Science Foundation Grants MCS-8202851 and INT-8300436 and Consiglio Nazionale delle Ricerche contract n.8300031.01, under the auspices of the U.S.-Italy Cooperative Science Program.

infinite subset of D, it contains all computable functions over it. This is also required in order to turn D into a model of λ-calculus or C.L., for both languages have this strong computational power.

For the reader with no direct experience in denotational semantics of programming languages, the interest and meaning of the equation in (0) may still be rather obscure. Let's be more specific.

Let Ide be the set of identifiers of a programming language; Num the set of all integers; State the set of all possible states of a (possibly imaginary) computer; Proc the set of all parameter-less procedures, and Val the set of all storable values. In the language we have in mind, identifiers can denote integer and procedures. The following equations represent the relationship between these sets:

(1) $\text{State} = \text{Ide} \to \text{Val}$

(2) $\text{Proc} = \text{State} \to \text{State}$

(3) $\text{Val} = \text{Num} + \text{Proc}.$

Note that in (1), (2) and (3), $A \to B$ represents a set of functions from A to B, while $A+B$ represents the disjoint union of A and B. By substituting for Proc in (3) and then the new right hand side of (3) in (1) we obtain:

(4) $\text{State} = \text{Ide} \to [\,\text{Num} + (\text{State} \to \text{State})\,].$

In other words, the intended domains are recursively specified. Such a "circular" definition of data types is commonly used in actual programming. As mentioned above, the existencxe of a solution to (4), i.e., the consistency of (1) + (2) + (3), is not a trivial fact in classical mathematics, mainly if one wants to retain the expressiveness of high level programming languages. Several methods have been published. Scott [1972] showed how to solve (0), for $A = \varnothing$, and Scott [1976] gave a general treatment using retracts of $P\omega$. Later papers introduced neighborhood systems (Scott [1980]) and then information systems (Scott [1982], Larsen and Winskel [1984], and Coppo, Dezani, and Longo [1983]). A categorical approach to solving such systems is given in Smyth and Plotkin [1982] and Lehman and Smyth [1981] (see

also Plotkin [1978] or Wand [1979]). Applications of recursive data specifications may be found in Gordon [1979], Milne and Strachey [1976], Stoy [1978], and Tennent [1981].

In Bruce and Longo [1983] we present an elementary algebraic approach to constructing domains satisfying these equations. While many of the results are old, we hope that this concrete approach is more accessible to computer scientists. In particular the only use of category theory in this paper is the notion of functor which we define herein. Briefly, a natural partial order is given in the category of complete partial order such that the usual domain constructors turn out to be "continuous" functions. This is applied both for the semantics of ordinary functional languages and of language which allow the use of types as parameters.

In Bruce and Longo [1984] the equations which hold in all models of typed λ-calculus are characterized. This is done by a simple theory whose axioms for type equality are just

$$\alpha = \alpha \quad \text{and} \quad \alpha \to (\beta \to \gamma) = \beta \to (\alpha \to \gamma)$$

plus the expected inference rules for equality. Since Cartesian Closed Categories are models of typed β-calculus, our result characterize the isomorphisms which hold in all CCC's.

Moreover, by a suitable extension of that theory, we characterize the valid equations in higher order models, i.e. in models of the second order calculus.

An analogy

The solution of domain equations may be probably regarded as one of the main issues in present and forthcoming research in Mathematical Computer Science and as a major connection between the later and Mathematical Logic. It should be considered that "equations" have been a basic way for applying mathematical tools to other scientific areas. Think, for example, to the equations for "motion" in Physics or to the role of partial differential equations in describing a huge amount of physical and engeneering problems. Proofs of existence (and unicity) of such equations motivated large developements in Mathematics and provided methods for several concrete applications.

What has semantics to do with this? Consider say the formal equation

(5) $x^2 = -1$

One may get to (5) by pure algebraic-syntactic manipulations of symbols and formal operations. In order to find a solution of (5) one has to interpret the symbols "=", "-" and the constant symbol "1" over a suitable space where an "interpretation" of x may be found realizing or satisfying that equation. The complex numbers or the cartesian plane would do the job.

Similarly, Scott's equation (0) may be written, for $A=\varnothing$, as

(0) $X = X^X$

The problem of finding a non-trivial solution to (0) is clearly a hard problem. As in the case of (5) with complex numbers (or their interpretation in the cartesian plane), one has to invent a suitable universe for interpreting the operations and symbols in (0). Then an interpretation or value-assignement has to be found which realizes the given equation.

In this case the mathematical tools are bounded from category Theory: the universe is provided by suitable Cartesian Closed Categories, where formal exponentiation is interpreted by the "function space" functor and "=" by an isomorphism. The objects obtained as inverse limits over suitable chains give a solution to (0), by providing an interpretation for X.

BIBLIOGRAPHY

Bruce, K. and G. Longo (1983), "An elementary approach to the solution of recursive domain equation," preprint.

Bruce, K. and G. Longo (1984), "Provable isomorphisms and domain equations in models of typed languages," preprint, Williams College.

Bruce, K. and A. Meyer (1984), "The semantics of second order polymorphic lambda calculus," preprint.

Coppo, M., M. Dezani, and G. Longo (1983), "Applicative Information Systems," Information and Control, to appear.

Gordon, M. (1979), *The Denotational Description of Programming Languages*, Springer-Verlag.

Larsen, K., and G.Winskel (1984), "How to solve recursive domain equations effectively using information systems, incomplete, preliminary version," preprint.

Lehmann, J. J., and M. B. Smyth (1981), "Algebraic specification of data types: a synthetic approach," Mathematical Systems Theory, 14, 97-139.

McCracken, N. J. (1979), "An investigation of a programming language with a polymorphic type structure," Ph.D dissertation, Syracuse University.

McCracken, N. J. (1984), "A finitary retract model for the polymorphic lambda calculus," to appear, Information and Control.

Milne, R., and C. Strachey (1976), *A theory of Programming Language Semantics*, Chapman and Hall.

Plotkin, G. D. (1978), "The category of complete partial orders: a tool for making meanings," lecture notes, Pisa Summer School.

Scott, D. S. (1972), "Continuous lattices," *Toposes, Algebraic Geometry and Logic*, LNM 274, Lawvere, ed., Springer-Verlag.

Scott, D. S. (1976), "Data types as lattices," SIAM J. on Computing, 5, 552-587.

Scott, D. S. (1980), "Lectures on mathematical theory of computation," manuscript, Univ. of Oxford, 135 pp.

Scott, D. S. (1982), "Domains for denotational semantics," ICALP proceedings.

Smyth, M. D., and G. D. Plotkin (1982), "The category-theoretic solution of recursive domain equations," SIAM J. on Computing, 11, 761-783.

Stoy, J. (1978), *Denotational Semantics: The Scott-Strachey approach to Programming Languages*, Cambridge, M.I.T. Press.

Tennent, R. (1981), *Principles of Programming Languages*, Prentice-Hall.

Wand, M. (1979), "Fixed point construction in order enriched categories," T.C.S. 8, 13-30.