Several Types of Types in Programming Languages

Simone Martini
University of Bologna, Italy & INRIA Sophia-Antipolis, France

We seldom realise that the concept of "type" in programming languages we understand nowadays is not the same it was perceived in the sixties, and that it was largely absent in the programming languages of the fifties. Moreover, we now superpose the concept of "type" in programming languages with the concept of the same name in mathematical logic – an identification which may be good for today, but which is the result of a (slow) convergence of two different paths. Tracing this story with the accuracy it merits, it is well beyond the limits of this paper. We will simply make several remarks (some historical, some of more conceptual character) that we hope will be useful as a basis for a further investigation. The thesis we will argue is that there are three different characters at play in programming languages, all of them now called *types*: the *technical concept* used in language design to guide implementation; the *general abstraction mechanism* used as a modelling tool; the *classifying tool* inherited from mathematical logic. We will suggest three possible dates *ad quem* for their presence in the programming language literature, suggesting that the emergence of the concept of type in computer science is relatively independent from the logical tradition, until the Curry-Howard isomorphism will make an explicit bridge between them.

From types to "types"

The first question is when the word "*type*" entered the technical jargon. Contrary to folklore, early documentation on FORTRAN does not use the word, at least in the technical sense. It uses "type" in a generic sense, where "kind" or "class" could be used (e.g., "32 types of statement"). For a precise occurrence of our technical term we must wait for the report on Algol 58 [10]. There, "type" is used as a collective representative for "special types, e.g., *integral*, or *Boolean*" (page 12). Algol 58 is the result of a meeting held in May 1958, between an ACM group and a European group. Both preparatory papers *do not* use "type". It is during meeting that the committee realised that different concepts could be grouped together, and given a name as a collective – types were born. It is also remarkable that, at least from these references, there is no clue that in this process the term "type" from mathematical logic had any role. The process will come to maturity in Algol 60 [1]: "The various "types" (integer, real, Boolean) basically denote properties of values". Observe the word "types" under quotes, as to stress that it is no longer the ordinary word, but the technical one.

Data types and abstractions

Algol 58 treats arrays separately from types, and Algol 60 makes no change in this – types denote properties of just "simple" values. To enlarge Algol's primitive data, [7] advocates the definition of new data spaces (Cartesian product, disjoint union, power set) in terms of given base spaces. McCarthy's paper is at a general, meta-level, but it sets a roadmap on how to introduce new types in programming languages – instead of inventing a new palette of primitive types, provide general, abstract mechanisms for the construction of new types from the base ones. The challenge to amend the "weakness of Algol" was taken up in more concrete forms, and in similar ways, by Hoare [4], and Dahl and Nygaard [3], around 1965. Of these two papers it will be Hoare's one to have the major, immediate impact – with an explicit reference to McCarthy's project it introduces at the same time the concepts of (dynamically allocated) *record* and *typed reference*. The paper is fundamental because types change their ontology – from an implementation issue, they programmatically become a general abstraction mechanism. This happens at three levels. First, it implements McCarthy's project into a specific programming language, extending the concept of type from simple to structured values, thus opening the way to the modern view of *datatypes*. Second, types are explicitly proposed as a linguistic modelling tool – a record type naturally represents a class of complex and articulated values. Finally, the combination of record types and typed references provides a robust abstraction over the memory layout used to represent them, because the type checker may statically verify that the field of a record obtained by dereferencing is of the correct type required by the context – primitive types are true abstractions over their representation. Hoare's proposal will find its context into [13], and finally will be implemented in Algol W [12], which will have a significant impact and is an important precursor of Pascal. It will be [9] to bring to full development the types as an abstraction mechanism, which will be further elaborated and formulated in modern terminology in [11].

Classifying values

Types in mathematical logic are a discipline for separating formulas denoting values from formulas that "do not denote". The opposition "denoting" vs. "non denoting" becomes, in programming languages, "non producing errors" vs. "producing errors". Types as a classifying discipline for programs are found in the programming language literature as early as in the PhD thesis of Morris [8]. But the connection to types of logic is implicit, even unacknowledged. A lack of acknowledgement which is going to persist – none of [9] or [11] cites any work using types in logic. Certainly the *Zeitgeist* was ripe for the convergence of the two concepts; the Curry-Howard isomorphism [5] will be the catalyst for the actual recognition[1], which comes only in [6], written and circulated in 1979, which presents a complete correspondence between proof-theory and functional languages. This slow mutual recognition of the two fields tells a lot on their essential differences. For most of the "types-as-a-foundation-of-mathematics" authors, types where never supposed to be actually

[1] For a lucid account of the interplay between types, constructive mathematics, and lambda-calculus in the Seventies, see [2], Section 8.1.

used by the working mathematician. It was sufficient that *in principle* most of the mathematics could be done in typed languages. Types in programming languages, on the contrary, while being restrictive in the same sense, are used everyday by the working computer programmer. And hence, from the early days, computer science had to face the problem to make types more "expressive", and "flexible". The crucial point, here and in most computer science applications of mathematical logic concepts and techniques, is that computer science never used ideological glasses (types per se; constructive mathematics per se; etc.), but exploited what it found useful for the design of more elegant, economical, usable artefacts. But this is the subject of an entirely different paper.

References

[1] J.W. Backus, F.L. Bauer, J. Green, C. Katz, J. McCarthy, A.J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J.H. Wegstein, A. van Wijngaarden, M. Woodger, "Report on the algorithmic language ALGOL 60", *Communications of ACM* 3(5), 1960, pp. 299-314.

[2] F. Cardone, J.R. Hindley, "Lambda-calculus and combinators in the 20th century", in D.M. Gabbay, J. Woods (eds.), *Logic from Russell to Church*, volume 5 of *Handbook of the History of Logic*, North-Holland, 2009, pp. 723 – 817.

[3] O.-J. Dahl, K. Nygaard, "Simula: An ALGOL-based simulation language, *Communications of ACM* 9(9), 1966, pp. 671-678.

[4] C.A. R. Hoare, "Record handling", *ALGOL Bulletin* 21, 1965, pp. 39-69.

[5] W.A. Howard, "The formulae-as-types notion of construction", in J.P. Seldin and J.R. Hindley (eds.), *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, 1980, pp. 479-490.

[6] P. Martin-Löf, "Constructive mathematics and computer programming", in L.J. Cohen and al. (eds.) *Logic, Methodology and Philosophy of Science VI, 1979*, North- Holland, 1982, pp. 153-175.

[7] J. McCarthy, "A basis for a mathematical theory of computation, preliminary report", in *Papers Presented at the May 9-11, 1961, Western Joint IRE-AIEE-ACM Computer Conference*, IRE-AIEE-ACM '61 (Western), 1961, pp. 225-238.

[8] J.H. Morris, *Lambda-calculus models of programming languages*. PhD thesis, MIT, December 1968.

[9] J.H. Morris, "Types are not sets", in *Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, ACM, 1973, pp. 120-124.

[10] A.J. Perlis, K. Samelson, "Preliminary report: International algebraic language", *Communications of ACM* 1(12), 1958, pp. 8-22.

[11] J.C. Reynolds, "Towards a theory of type structure", in *Programming Symposium Proceedings. Colloque sur la programmation*, Springer, 1974, pp. 408-423.

[12] R.L. Sites, "Algol W reference manual", Technical Report STAN-CS-71-230, Computer Science Department, Stanford University, 1972.

[13] N. Wirth, C.A.R. Hoare, "A contribution to the development of ALGOL", *Communications of ACM* 9(6), 1966, pp. 413-432.