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

TAVOLA ROTONDA
«LOGICA E INFORMATICA: DA INCIDENTI DI CONFINE A UN'ALLEANZA DI INTERESSE?»

INTERVENTO DI
HENK BARENDREGT

§ 1. Two types of semantics.

A semantics for a certain language L gives a "meaning" to expressions of L. Not always all expressions for the language will have a meaning, therefore this semantics should be viewed either as a partial map or as a total map with a specific element $\perp$ ("undefined") in its range.

There are two ways in which a semantics can be given: as a relative semantics or as an absolute one.

In a relative semantics for L an expression E of L (now called the object language) is translated into an expression S(E) of the target language L'. A meaning of expressions of L' is then supposed to be known. A well-known example of a relative semantics in the translation of a new natural language into our mother tongue. The word "acqua" means "water". Similary the translation of a higher order programming language like ALGOL into some assemble language consists of a relative semantics.Landin (1965) gave a different relative semantics of ALGOL by translating it into the lambda calculus.

On the other hand, the absolute semantics of a language is obtained in the way we learn our mother tongue. We are taught that water is something that comes out of a tap, can be put in a glass, can be drunk and can be used to wash ourselves with. Thereby we also obtain (part of) the meanings of the other words in italics. Wittgenstein's dictum "meaning is use" applies to the absolute semantic. We understand the meaning of an expression if we understand its appropriate use.

It often happens that semantic functions are composed with each other. If we have a relative semantics S of L into L', then this may be completed by giving an absolute semantics S' for L'. Alternatively, a relative semantics S' for L' into L" may be given and only then an absolute semantics S" for L". In the second case we have the composition S" $\circ$ S' $\circ$ S which has become an absolute semantics. Of course a relative semantics makes sense only if it is completed eventually by an absolute one. The nature of this final absolute semantics will be discussed more in section 3.

Sometimes the distinction is made between denotational and operational semantics. Denotational semantics can be viewed as a relative semantics with as target language set theory. Expressions of the object language are translated into expressions denoting sets (or elements of some complete partial order). When the abolute semantics for set theory is taken for granted, the denotational semantics becomes (after composition) also an absolute semantics.

Operational semantics is an absolute semantics.

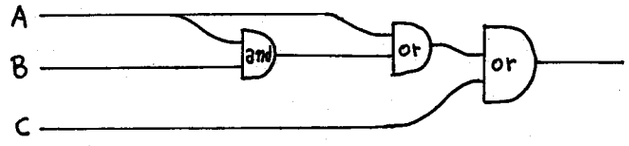
The meaning of an expression in the object language is explained by its operational behaviour.

Landin's relative semantics of ALGOL can be completed in two ways: 1. by giving an operational semantics for the lambda calculus; 2. by giving a denotational semantics for the lambda calculus. The first alternative was followed by Landin himself; the second one by Scott and Stratchay (1971).

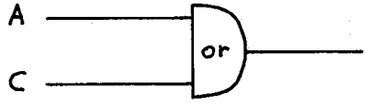
§ 2. Use of semantics.

There is a well established example of the use of semantics with an important industrial application: the Boolean interpretation of circuit of logical gates.

On the logic level of a computer circuits like the one in fig.1 are needed a zillion times. By using the Boolean semantics one obtains the expressions



(A & B) v A v C which is equivalent to A v C. Hence the circuit in fig. 1 can be simplified to



This kind of application is often used in the form of building up arbitrary logical funcions from only " nor " gates.

The above is a typical example of the use of semantics. An equivalence relation is induced on the objects of interest. Two objects (in the example the circuits) are equivalent if and only if the semantics of these are the same (in the example (A & B) v A v C and A v C are considered the same because they have the same truth table; in fact jet another semantics is involved). This induced equivalence relation gives an abstract view on the objects, words clarifying and program correctness proofs. Similarly the correctness of programs and program transformations may benefit from semantics. This was one of the motivations behinds the work of Landin and of Scott and Stratchay.

A more sophisticated application of these induced equivalence relations is in the proof of the characterization of invertibility in the $\lambda\beta\eta$ - calculus. Bergstra and Klop (19) have been using the denotational models P_∞ and P_ω for this purpose. The final result is purely in terms of the $\lambda\beta\eta$ - calculus, while the models have been used for obtaining the proper abstract view.

Summarising we state that althought semantics may seen to be for away in Platonic heaven its use is down to earth in the form of the induced equivalence relations (and other abstractions) on the objects of interest.

§ 3. Absolute semantics.

Although we have indicated the difference between absolute and relative semantics in section I and some use of absolute semantics in section 2, we did not say what

exactly is an absolute semantics. This will be discussed now.

Definition. An absolute semantics for a language L consists on an axiomatic theory T within the language L. The meaning of an expression E of L is the expression E itself modulo provable equivalence within T.

T is not required to be a complete theory: properties of the objects may be left undecided by the theory.

An axiomatic theory starts with primitive terms and axioms about them. From the primitive concepts we obtain defined concepts. From the axioms we derive the theorems. The axiomatic theory consists of primitive terms, axioms, defined terms and theorems. Euclid's view on the primitive terms and axioms was as follows. The primitive terms are concepts that are so clear that thay do not require any definition. The axioms are truths so obvious that they do not require any proof. Hilbert had another, more satisfactory, view: on to him the precise nature of the primitive concepts is irrilevant, as long as they satisfy the axioms: "the axioms form an implicite definition of the primitive terms". It is in this spirit that we interprete the absolute semantics.

It is clear that a relative semantics depends on an absolute semantics of the target language and has to be completed by taking the compositions of the various semantics involved. It can be asked whether the completed semantics can be given directly, without mentioning concepts from the intermediate languages. In a trivial way this is always so. However there are also interesting cases in which this happens. In Coppo et al. (1984) a lambda term

M is interpreted as the set of (generalized) types that M may have. In the appropriate setting this gives the same semantics as Scott's D_∞ models. By changing the structure of the types involved, Coppo et al. obtained also an alternative description of Park's (197) version of D_∞ in which the interpretation of the fixed point combinator is not the least fixed point operator. This all without mentioning complete lattices or partial order.

Summarising, an absolute semantics consists of our views that organize our data, our objects of interest. Similarly the semantics of our mother tongue depends on the image we have of the world.

References.

Bergstra, J.A and J.W. Klop
(1980) Invertible terms in the lambda claculus, Theor. Comput. Sci. II, 19-37.

Coppo M., M. Dezani-Ciancaglini, F. Honsell and G. Longo
(1984) Extended type structure and filter lambda models, in: Logic Colloquium '82, (ed. Longo Marcja and Lolli), North Holland.

Landin P. A correspondence between ALGOL 60 and Church's
(1965) lambda notation, Comm. Assoc. Comput. Mach. 8, 89-101 and 158-165.

Park, D. The Y-combinator in Scott's lambda calculus
(1976) models (revised version). Theory of computation report n. 13; Dept. of Comput. Sci. University of Warwick.

Scott. D.S. and C. Stratchay
(1971) Towards a mathematical semantics of computer languages, Proc. Symp. on computer and automata, Polytecnic Inst. of Brooklyn 21, 19-46.