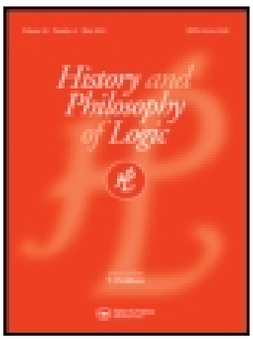

History and Philosophy of Logic

ISSN: 0144-5340 (Print) 1464-5149 (Online) Journal homepage: http://www.tandfonline.com/loi/thpl20

Proceeding in Abstraction. From Concepts to Types and the Recent Perspective on Information

Giuseppe Primiero

To cite this article: Giuseppe Primiero (2009) Proceeding in Abstraction. From Concepts to Types and the Recent Perspective on Information, History and Philosophy of Logic, 30:3, 257-282, DOI: 10.1080/01445340902872630

To link to this article: https://doi.org/10.1080/01445340902872630

HISTORY AND PHILOSOPHY OF LOGIC, **30** (August 2009), 257–282

Taylor & Francis
Taylor & Francis Group

Proceeding in Abstraction. From Concepts to Types and the Recent Perspective on Information

GIUSEPPE PRIMIERO[†‡§]

[†]Centre for Logic and Philosophy of Science, University of Ghent, Belgium; [‡]IEG, Faculty of Philosophy and Computing Laboratory, University of Oxford, Great Britain; [§]GPI, Department of Philosophy, University of Hertfordshire, Great Britain

Received 22 November 2007 Revised 30 January 2009 Accepted 8 February 2009

This article presents an historical and conceptual overview on different approaches to logical abstraction. Two main trends concerning abstraction in the history of logic are highlighted, starting from the logical notions of concept and function. This analysis strictly relates to the philosophical discussion on the nature of abstract objects. I develop this issue further with respect to the procedure of abstraction involved by (typed) λ-systems, focusing on the crucial change about meaning and predicability. In particular, the analysis of the nature of logical types in the context of Constructive Type Theory allows elucidation of the role of the previously introduced notions. Finally, the connection to the analysis of abstraction in computer science is drawn, and the methodological contribution provided by the notion of information is considered, showing its conceptual and technical relevance. Future research shall focus on the notion of information in distributed systems, analysing the paradigm of information hiding in dependent type theories.

1. Introduction

Abstraction is one of the core notions in both ancient and modern formal logic. As a central subject for any philosophical approach to logic, its historical development is fascinating from various perspectives.

A fair amount of scholarly work, especially in linguistics and metaphysics, has been produced concerning the nature of abstract entities and abstract objects, both with respect to theories of Universals in Antiquity and the Middle Ages, and to more recent systematic and formal approaches.[1] Abstraction, intended as a logical procedure, is the other side of the coin. This issue arises from the development of modern logic and today it plays a relevant role in mathematics and computer science. The basic distinction between 'abstraction' and 'abstract' spreads across the history of logic, starting with the ancient Greeks, and since then has presented a number of overlapping and continuosly changing readings. The aim of the present article is to provide an overview of the links between two main readings of logical abstraction: I shall refer, on the one hand, to the 'bottom–up' process of abstraction based on predication; on the other, to its counterpart, the 'top–down' model based on the elucidation of meanings.

This task is accomplished by introducing this basic distinction at the origin of logic with reference to Plato and Aristotle, and reconsidering the change induced by medieval commentators (section 2). The next step is represented by the analysis of the first paradigmatic interpretation of abstraction due to modern logic, namely via the introduction of the notion of function and its translation into the notion of logical type (section 3). The aim is to reach the analysis of abstraction provided

[1] See for example *Hart 1979*, *Bealer 1982*, *Künne 1982*, *Künne 1983*, *Zalta 1983*, *Hale 1987*, *Campbell 1990*, *Bacon 1995*, *Lowe 1995*.

History and Philosophy of Logic ISSN 0144-5340 print/ISSN 1464-5149 online © 2009 Taylor & Francis
http://www.informaworld.com DOI: 10.1080/01445340902872630

by the various versions of λ-systems (sections 4 and 5), in order to present the appropriate interpretation of this notion in procedural semantics (section 6 and 7) and to explain a new methodological view, based on epistemic information (section 8).

2. At the origins of abstraction

A philosophical outlook on the formal procedures of abstraction can start nowhere else than from the first steps in logic.

The first formulation of a theory of abstract terms in the history of philosophy is probably Plato's Theory of Ideas, on the basis of which the first (coherently presented) epistemology of the history of philosophy is built. The missing formal presentation of such a theory can be extracted from different passages of the Platonic dialogues. Notoriously, the nature of the most general Forms, the related problem of how does one get to know them, and their connection to existing (concrete) objects, are the essential features of Plato's theory of knowledge and of his metaphysics.[2] The element of Plato's philosophy that is crucial to the issue of the present analysis is the conceptual priority of Ideas over the material objects realizing them: Forms exist explicitly separated from all the particulars. The latter are thus determined in their essence by their partaking of the former, which implies in turn the inherence of Forms to existents. For this, a passage from the *Phaedo* (100a–101e) remains illuminating:

> I think that if anything is beautiful besides absolute beauty it is beautiful for no other reasons than because it partakes of absolute beauty; and this applies to everything. [...] I hold simply and plainly and perhaps foolishly to this that nothing else makes it beautiful but the presence or communion (call it which you please) of absolute beauty, however it may have been gained; about the way in which it happens, I make no positive statement as yet, but I do insist that beautiful things are made beautiful by beauty.

This relation of partaking (i.e. the combined relation given as $\pi\alpha\rho\sigma\upsilon\sigma\acute{\iota}\alpha$ and $\kappa\sigma\iota\nu\omega\nu\acute{\iota}\alpha$), the intertwining of particular and general Forms, is notoriously difficult to explicate, and the present article will not try to formulate a new thesis on this topic. The reference to Plato's Theory of Ideas is intended as a shortcut to present the connection between abstraction and knowledge, instantiated as a relation of predication. This view in the context of Platonic metaphysics and ontology is emphasized by the fact that the general Forms are considered in their relation to concrete objects with respect to the central topic of judgment-formation. Platonic Ideas are standard paradigms, to which individuals conform, and this appears manifestly in the fomulation of correct judgments.[3] Despite the obvious connection that Platonic Forms bear to judgments and predicative Forms, the disentanglement of their formal relation is a difficult one, and the theory of judgments was never explicitly considered by Plato as the logical *tertium* between abstracts and their terms.

Such clarification comes only with the Copernican turn due to the Aristotelian explanation of the relation between objects and abstract terms: it consisted of

[2] Probably the most complete reference for Plato's Theory of Ideas is still *Ross 1971*.

[3] See e.g. *Theaetetus* 201d–202c and *Republic*, 476a.

abandoning the ontological priority of the general Forms over their participating individuals, and assuming explicitly the relation of predication as the basis for defining abstraction. Aristotle maintains that the logical relation of predication is the starting point for any account of the Categories, intended as the Forms of both what there is and of what can be said.[4] Under such reading, the notion of general form is translated into that of a *category of predication*; as a specification of the latter, a general predication will make explicit the relation of an individual to a substance (*οὐσία*), the first category to which the others refer. This implies that the relation between terms in an act of predication and categories proceeds in the opposite direction than in Plato, and the analysis of predicative expressions amounts to a proper abstraction procedure, rather than to the relation between abstract terms and their instances. The Aristotelian understanding of abstraction can be formulated in terms of the main thesis that abstract methods (or abstraction procedures) are methodologically distinct from abstract terms, considered as entities with some ontological status. These distinct modes of abstraction can be recollected as follows in Aristotelian terms:

- The notion of *universal* (*καθόλου*);
- The notion of object produced by (a method of) abstraction (*τὰ ἐξ ἀφαίρεσεος λεγόμενα*, *ἓν ἀφαίρεσει*, *δι΄ ἀφαίρεσεος*, *ἀφαίρειν*).

The first mode of abstraction exemplifies universal predication. Its definition refers to the classical form of judgment 'P belongs (*ὑπάρχειν*) to S', for which one says that P *belongs universally* to S if:

- The predicate P belongs to every instance of the subject S;
- That P belonging to S is due to S itself in that it is an S (in virtue of S and *qua* S), i.e. not by accident.

A universal is identified with a predicable satisfying the two previous conditions.[5]

On the other hand, the idea of (a method of) abstraction (by which a certain abstract term is produced) corresponds to a *removal operation*.[6] This idea is formulated in the general logical context of the *Analytics* in terms of the operation of removing particular predicates, namely those not falling under the previously given description of the universal. The removal operation preserves only the basic or definitional predicates for the subject at hand (its defining categories): for example, it considers perceptible magnitudes *qua* lengths, and this corresponds to proceeding from the particular to the more general of the categories.[7] This procedure leads to concept-formation by means of classification of the properties belonging to objects or entities, and thus provides their hierarchy of universality. Moreover, this procedure of determining the universals starting from particulars is crucial because it defines the demonstrative process:

[4] Aristotle, *Categories*, par. 2.

[5] For this explanation see e.g. *Metaphysics B*, 4, 1000a1; Γ, 9, 1017b 35; *Z*, 13, 1038b11–13; *Posterior Analytics* I, 4, 73b 26–74a 3.

[6] The standard example is given in *Physics*, in terms of perceptibles defined as physical magnitudes, something which by nature can be added or removed. Cf. Aristotle, *Physics*, Book 3.

[7] See e.g. *Posterior Analytics*, I, 18, 81b 3–7 and *Metaphysics*, M 2, 1077b 10.

> [...] demonstration does not necessarily imply the being of Forms nor a One beside a Many, but it does necessarily imply the possibility of truly predicating one of many; since without this possibility we cannot save the universal and if the universal goes, the middle term goes with it, and so demonstration becomes impossible.[8]

As it appears by this short analysis, the distinction between formal procedures of abstraction and abstract entities (concepts) is fully developed in the Aristotelian framework of predication. In such a paradigm, the notion of concept is the abstract entity obtained by progressive universalization of predications, and this surely conflicts with the Platonic approach according to which abstract terms are ontologically anterior to their use in predication.

Predication is, therefore, the pivotal notion to look at for an understanding of abstraction. This strategy is at hand in the *ante rem*/*in re* distinction for universals with respect to particulars introduced in Middle Ages. The relation between general names and particular existing entities, first considered by Plato and examined at length for example in his *Parmenides*,[9] is one of the legacies bequeathed to the Middle Ages. Porphyry in his *Isagoge*, or introduction to Aristotle's *Categories*, considered the problem of Universals explicitly.[10]

Boethius discussed it in his *Commentaries on Porphyry's Introduction*, very much in the Aristotelian vein of mental abstraction, whereas in the *De consolatione philosophiae* he reflects a more Platonic approach of innate ideas as memories of previous existence. Abelard, on the other hand, maintained that Ideas pre-exist the creation as patterns which determine divine Providence in creating the best of the possible worlds (a thesis known as exemplarism).[11]

But the most relevant influence on the theory of universals, and a turning point in the history of abstraction, is the formulation of the semantic theory of *suppositio*. Ockham's interpretation is surely among the most relevant ones, based on a notion of *generality* from which a theory of abstract entities was derived. Notoriously, the theory of supposition provides a semantic treatment for the property of terms in a sentence, which is the context of semantic validity of a categorematic term in a proposition.[12] Ockham made use of this powerful tool in connection with the theory of universals, in order to consider the relation between categorematic and syncategorematic

[8] Aristotle, *Posterior Analytics*, I, 11, 77a 5–9.

[9] The dialogue between Zeno, Parmenides and Socrates having as a subject the Zenonian doctrine that 'things are many' finds an obvious and direct connection to the relation of particulars with the Forms; see e.g. *Parmenides*, 127 d6–e4.

[10] From the Introduction of the *Isagoge*:

> I shall not say anything about whether genera and species exist as substances, or are confined to mere conceptions; and if they are substances, whether they are material or immaterial; and whether they exist separately from sensible objects, or in them immanently. This sort of problem is very deep, and requires a more extensive investigation. But I shall now try to show you that the ancients, and especially the Aristotelians, had very reasonable theories about these [genus and species] and the others I mentioned [specific difference, exclusive property, accident].

A complete and satisfactory overview of the different doctrines about universal starting with Porphyry's explanation is given in *Aaron 1953*.

[11] See for example *Brower & Guilfoy 2004*, especially the chapters *Logic* by C.J. Martin and *Metaphysics* by P. King.

[12] Usually it has been simply identified with the modern theory of reference. Nowadays this identification has been limited or even rejected. For a modern logical interpretation of the theory of supposition see e.g. *Dutilh Novaes 2007*.

terms[13]: Do universal terms have signification proper to themselves? Ockham maintains that a term which supposits generally in a proposition (i.e. under the specification of the syncategorematic 'all') supposits for every term contained in the appellative domain determined by the general noun. In paragraphs 6–8 of his *Summa Logicae*,[14] he states that each of two names which are respectively the abstract and the concrete of the same concept (e.g. humanity-man, animality-animal, hotness-heat and so on) are not synonymous, i.e. they do not supposit for the same thing and what is predicated of one of them cannot also be predicated of the other one.

This amounts, in turn, to formulating the problem of predication for general abstract and concrete terms in relation to meaning: abstract and concrete names can be equivalent in signification to several expressions, and generality of reference must ultimately be secured by the categorematic terms embedded in general propositions, rather than by expressions like 'some' and 'every' that introduce them. Thus, identity of meaning is established in terms of their definitions, and therefore appealing to their *supposita simplex*:

> [. . .] every universal is one particular thing and [. . .] is not a universal except in its signification, in its signifying many things.[15]

According to Ockham's nominalism, the referents of terms render the distinction between universal and particulars, whereas there exists a common *suppositum* or *meaning* determined as an affection of the soul. This is also confirmed by the thesis that a concrete term, being a predicate in a proposition, supposits for a form, as 'white' for 'whiteness' in 'Socrates is white'.[16] The interesting thesis held by Ockham is therefore that the term in the universal form introduces the context of semantic validity in any case of predication in which the same name is used in the concrete form: this also means that universals are applicable to concrete things insofar as these resemble each other and the concept resembles each of them.

For Aristotle the nature of abstract terms (concepts) is obtained by a procedure of removal from the concrete predications, a position that finds variants in Averroes, Aquinas and Scotus. By introducing explicitly the semantic relation of supposition in this context, Ockham describes an abstract term as allowing those concrete predications to be formulated by displaying the context of semantic validity in which they can be performed, thus working as their logical presupposition. The relation of conceptual priority and hierarchy of predication is thus clearly at the basis of a full clarification of the nature of abstraction.

A double path emerged in this evolution of the notion of abstraction: on the one hand, the role of predication; on the other, the notion of meaning. In the following, I shall refer to abstraction in this double acception: either as process of removal, based on predication; or as abstract term, meaning-determiner for possible predications. This double acception of the term 'abstraction' shows the conceptual shift from the metaphysical interpretation of abstract terms to the procedural semantics, which are at the core of the logical procedures known today from mathematics and computer science. The present analysis does not aim to show a conceptual priority of the one

[13] The counterparts of Aristotelian categories and of modern logical constants.

[14] In *Ockham 1349* these paragraphs are titled respectively: 'On concrete and abstract names that are synonyms', 'The correct account of abstract and concrete names' and 'On the third mode of concrete and abstract names'.

[15] *Ockham 1349*, par. 14, p. 78.

[16] *Ockham 1349*, par. 63, p. 189.

explanation over the other, nor to establish any metaphysical claim of validity for one of them. For this reason, I do not claim that one of the two approaches deserves the denomination 'abstraction' better than the other one, and I shall use the term 'abstraction' in both cases, specifying – when needed – which of the two acceptions is referred to. I will show how, in the history of logic, the interpretation of abstraction converges with respect to the problems of predication and meaning highlighted up to now, preserving the dichotomy between a result-based and a process-based approach, and where the procedural approach has progressively taken over, providing a complete and unified insight to the general problem of abstraction.

3. From functions to types

One of the most important conceptual changes for which modern logic has been responsible is the Fregean representation of *beurteilbare Inhälte* (judgeable contents) distinct from the related acts of assertion in a formal language, obtained by the introduction of the turnstile sign in the *Begriffsschrift*. The resulting new framework stresses the essential role of predication in the definition of concepts; moreover, it provides a completely new (logical) form for the notion of abstract terms. This latter step is the most interesting element for the evolution of the notion of abstraction: it leads to the notion of *function* as representing logical predication. Its most important consequence is obviously the connection to the problem of impredicativity and the development of the hierarchy of predications: this is the next step in our analysis, namely the comparison between abstract terms and the invention of *logical types*. In what follows, I shall investigate how these well-known events from the history of logic are intertwined with the notion of abstraction referred to in the analysis of the previous section.

The Fregean approach represents a direct critique of the Aristotelian understanding of abstraction as the determination of a unity among many separated enitites. The connection between abstraction and function in the *Begriffsschrift* is given in terms of a generic form of the *Abstraction Principle*. This principle implies in Frege's successive works two forms of conceptual abstraction. The first is expressed by the Fregean definition of number:

Definition 1 (Hume's Principle) *For any two concepts F* and *G*, *if F and G stay in a one-to-one correspondence, then they are to be associated with the same number:*

$$\sharp_x Fx = \sharp_x Gx \leftrightarrow Fx \equiv_{x,y} Gy. \tag{1}$$

The second form is the extensional abstraction represented by Law *V* of the *Grundgesteze der Arithmetik*, i.e. the schema:

Definition 2 (Basic Law V) *Given a function* $f(x)$, *let its extension be* $\{x \mid f(x)\}$, *and given a function* $g(x)$ *let its extension be* $\{x \mid g(x)\}$; *then it holds* $f = g$ *iff* $\forall x(f(x) \leftrightarrow g(x))$,

notoriously leading to contradiction by Russell's Paradox.

Conceptual abstraction becomes a failing procedure in that it allows concepts and their properties to be 'given independently of the objects that are abstracted from them'.[17] Abstracts and concepts determine each other in a way that makes it difficult

[17] *Fine 2008*, p. 2.

to separate the abstraction on concepts and the conceptualization of objects. The two principles of conceptual abstraction in Frege's work exemplify this problem:

1. Looking at *abstracts* as equivalence classes replaced the quest for a purely logical definition of the sort of abstract objects that such principles define;
2. using *extensions* to define functions reduces abstracts to concretes, but it necessitates disposing entirely of such extensions, forgetting about the required substituition procedures on the place-holders of the functional expression.

This formulation of the abstraction principle(s) makes abstract objects essentially rely on purely logical grounds, namely properties (size, and thus cardinality) of the corresponding models, and it allows one to identify the crucial requirements for truth on abstraction: equivalence of concepts and a non-circular identity criterion.[18] The abstraction principles give coherent and logically admissible grounds for defining (ontologically justified) abstract terms (in second-order logic for abstraction on concepts). Nonetheless, the holistic view on predicables shows it is a high price to pay.

The structure of Hume's Principle essentially allows one to refer to the extension of a concept only for a countable number of objects. On the other hand, it can be claimed of both principles that the nature of their conditions for truth on abstraction is essentially the defining principle for functions, as this notion was introduced by Frege: if in an expression a simple or compound sign has one or more occurrences and that sign is recognized as replaceable in all of its occurrences by some other sign, the invariant part is then a function and the replaceable part is its argument.[19] With respect to the Fregean interpretation of abstraction, the priority relation between the criteria defining the predication relation and the (very same) properties as attributes of the resulting abstracts is crucial. In other words, Fregean abstraction requires, once again, explaining abstraction from either one of two viewpoints: as removal procedure or as meaning–determiner, now turned into – respectively – functional semantics versus ontological properties. The Aristotelian procedural ontology has turned into the functional determination of *Sinn* and *Bedeutung*, whereas the medieval ontological semantics became the Fregean third realm of abstracts: as a result, the Fregean abstracts are neither only the result of a removal operation in the Aristotelian way, nor just the term which settles the semantic range of validity for predications, in the Ockhamian style. The idea of definition by means of either form of abstraction, and the essential reducibility of Hume's Law to extensional abstraction, are the crucial elements for determining the conceptual priority between predication and abstract terms.

The Fregean functional structure as predicative model, obtained by the removal of the non-definitionally relevant parts of the predication, is meant to preserve semantic validity via the development of the theory of sense and reference. The concept

[18] The two corresponding formal criteria are clarified in *Fine 2008*, pp. 9–11, in the following terms: *tenability*, to take the universe of objects from which abstraction is formulated as a set with the related model and truth for that model; *stability*, to take a statement as true when it is such in all models of a sufficiently large cardinality. In the following I will extend the methodological analysis, further pointing at the logical difference of this realistic view on abstract terms with respect to a procedural notion of abstraction.

[19] *Frege 1879*, §9.

resulting by abstracting from an instance of a function[20] is the *Bedeutung* of a predicative expression.[21] In the substitution of the classical judgmental form 'S is P/P belongs to S' with the new functional structure $F(x)$, the role of the evaluation on x (the variable representing a place-holder for it) is therefore central. To build up a judgment means essentially to evaluate a concept for an argument by verifying the course of values of the related functional expression. According to Frege, therefore, a concept cannot play the role of the reference of the grammatical subject: it has to be converted into (or represented by) an object, which allows for its evaluation.

The structural remarks on the role of functions and objects do not lead to the very next step in the history of abstraction. The Fregean theory interpreting concepts as predicates with sense and reference under satisfaction of the appropriate evaluation procedure does not imply that predicates have to be logically possible: the existence of an object instantiating a self-contradictory predicate is obviously a completely different matter. Under the explanation of abstraction based on the notion of function, the Fregean non-deterministic semantics still overcomes a purely procedural view, claiming priority for the ontological status of abstracts.[22] Fregean abstraction in the form of functions mantains conceptual priority of abstract terms over their instances, but it is restricted by requiring fulfillment on the argument-place in order to give rise to proper evaluation for certain objects relativised to the domain at hand. The formal definition by abstraction of the old-fashioned notion of concept is therefore obtained by determining a class of given objects defined by the function, satisfying an equivalence relation R such that reflexivity, identity and transitivity hold on the objects that are members of the class. Such a definition for functional expressions has to specify their ranges of values only by means of terms for which they can be evaluated (the mentioned relativisation of the domain): this produces the notion of a function as corresponding to its extension, namely the correlation of arguments and values.[23] For a concept intended as an abstract term (represented by the function with the empty place-holder), its range corresponds to the usual logical extension, i.e. the set of objects that can positively evaluate it.[24]

[20] The well-known structure according to which a predicate in the form of a function is an unsaturated object, whereas the formulation of an argument represents the instantiation of a concept making the former saturated. See *Frege 1891*, p. 6, 16.

[21] *Frege 1892a*. Notoriously, Frege establishes the referents of singular terms to be the objects they stand for, and the referents of indicative sentences to be their ranges of values. See also his *1892b*, p. 193. *Fine 2008*, p. 17, ff. points to the sense/reference distinction for definitions with referential import vs. definitions with semantical import respectively. For a study of the constructive notion of reference based on the Fregean analysis, see *Primiero 2004*.

[22] In this line, *Fine 2008* claims the nature of Hume's Principle (and thus of the abstraction principle) to be more than just a principle defining the number operator. He claims that one can consider the principle as determining the domains of discourse on which abstraction is applied: 'It is somehow meant to succeed both in telling us what objects there are and in assigning them to the concepts. [. . .] the principle should determine [the domain] M on the basis of the underlying subdomain I of individuals (or non-abstracts). Thus the principle must not only determine the operator F as a function of M, it must also determine M as a function of I' (*Fine 2008*, p. 20).

[23] The latter is then an ordered pair, corresponding to the notion of a function as a graph; conceptually different is the simple dependent object considered before. In the next section, the notion of function involved in the analysis of type system will be a different one. The clarification of the distinction among these three notions is due to B.G. Sundholm in personal conversation.

[24] *Fine 2008* proposes a Fregean theory of abstraction that aims at solving a number of typical problems, such as the identity problem, considering an object the abstract of a concept with respect to a relation on concepts: 'Thus the notion relates three items: an object, which is the result of abstraction; a first-order concept, which is what is abstracted; and a second-order relation on concepts, which is the means of abstraction. In the case of numbers, for

The problematic applications of the Comprehension Principle rest on this conceptual priority of the meaning-function over the satisfaction value. Cases of paradoxical applications are due to the fact that extensions of concepts are themselves among the abstracts and therefore each extension should be identified with the class of concepts that have that extension. Notoriously, by treating ranges of values as ordinary objects, Frege allows a function to be applied to its very same course-of-values (corresponding to a function applied to its own graph). In 'Funktion und Begriff', Frege avoids this paradox of the *Begriffsschrift* by considering a first-level and a second-level form of abstraction, producing different kinds of functions.[25] Thus, essentially the same result of the later Russellian Ramified Type Theory (RTT)[26] was already at hand in the later developments of his theory of functions, but treating *Wertverläufe* as objects undermined the entire project of the *Grundgesetze*.

The resolution of the Fregean hypostatization of ranges of values of propositional functions in RTT is hidden in the derivation of the latter from a hierarchy of types. In the construction of propositional functions, Russell makes use of the substitution of variables by suitable arguments and this obviously requires that (some form of) the Fregean Abstraction Principle has been previously applied.[27]

The interesting case in the structure of Ramified Type Theory is represented by the construction of propositional functions from propositions with the use of abstraction. The basic language is built up by the basic sets of individuals, $I = \{a_1, a_2, \ldots\}$; variables, $V = \{x, y, z, \ldots\}$, with a strict relation $<$ defined over V; and functions R, together with an arity-defining map $\alpha : R \rightarrow \mathbb{N}$. When one considers the type of variables bound by quantification, the structure of ramified types is introduced: propositions are of order 0, and all quantificational expressions over basic propositions have to be of higher order. Thus, types and orders (a natural number) are introduced: O^0 is a ramified type; if $(t_1(a_1) \ldots t_n(a_n))$ are ramified types, with $a \in \mathbb{N}, a > \max(a_1, \ldots, a_n)$ a ramified type is $(t_1(a_1) \ldots t_n(a_n))^a$ where $a = 1 + \max(a_1, \ldots, a_n)$. All minimal ramified types are predicative types.

The result of abstraction seems to be again the functional empty structure to be fulfilled by proper arguments in place of the variables, producing appropriate objects. But the famous restrictive principle of impredicative definitions is then applied: whatever involves all of a collection cannot be taken as a part of it (*Vicious Circle Principle*). In the development of the relevant notion of function for this system of predication, an important conceptual change has occurred: the connection between predication and concepts as abstract entities is forgotten, being replaced by the notion of predicate, which represents the stable part of the abstraction procedure in terms of functions. Correspondingly, the abstraction procedure for the formation of propositional functions is the basis of RTT, which maintains the double hierarchy of simple types and of orders, where the difference is expressed by the different evaluation objects (arguments) they are satisfied by.

example, we may say that 0 is the abstract of the empty concept with respect to the relation of one-one correspondence' (*Fine 2008*, p. 28).

[25] *Frege 1891*, pp. 26–27.

[26] *Russell 1908* and *Whitehead & Russell 1910–12*.

[27] For a complete historical overview on type theory, starting from RTT, see *Kamareddine et al. 2004*.

The thesis that abstraction has now a rather different nature and plays a different role is confirmed by the understanding of generality for functions as interpreted by Russell. In RTT, two forms of abstraction are thus introduced:

1. *Abstraction from parameters*: For any propositional function f on predicative types $(t_1(a_1)\dots t_n(a_n))^a$ within domain of variables Γ, a new propositional function $f'((t_1(a_1)\dots t_{n+1}(a_{n+1}))^{\max(a,a_{n+1}+1)}$ – where $t_{n+1}(a_{n+1})$ is a new predicative type – is obtained by replacing all propositional and individual variables the function can range over with equal variables within a new domain Γ' that contains all previous variable plus the needed new ones;
2. *Abstraction from propositional functions*: For any propositional function f on predicative types $(t_1(a_1)\dots t_n(a_n))^a$ with free variable $y_1 < \dots < y_n$, a new propositional function $f'(y_1 < \dots < y_n)((t_1(a_1)\dots t_n(a_n))^a)^{a+1}$ within domain Γ' is obtained by constructing a higher-order predication on the possible values of f.

The definition of equality for abstraction from parameters is dependent on the strict order $<$ on the variables, so that the list of variables is ordered and difference in order defines different functions: in turn, appropriate substitution procedures preserving strict order define the set of equivalent functions. The second form of abstraction is the one needed to obtain higher-order propositional functions, and it uses higher-order variables. In the first case, one refers to apparent variables (the function is always true); in the second, to real variables (any value of the function is asserted). The notion of abstraction which leads to functions as independent objects (and in turn to contradiction whenever the appropriate hierarchy of predications is not considered) acts on the values (real variables), what Russell called a proper 'propositional function'.[28] Thus the type of a function depends not only on the type of arguments, but rather also on the type of apparent variables (place-holders).[29]

Typing procedures introduced by Russell are not just the solution to the problem of impredicativity: they present a new interpretation of the notion of function and a different approach to abstraction. In particular, the latter requires an explanation of the role of types as objects of a higher level. This can already be seen in the restriction on the formulation of propositional functions provided by the definition of appropriate contexts Γ, Γ' for the allowed types.

The traditional interpretation of the notion of function due to Leibniz, Bernoulli and Euler, is that of an analytical expression in one or more variables. Frege and Russell formulated the logical notion based on the relation of predication, which refers to substitution and evaluation as its defining operations. In the Fregean interpretation a function also stands by itself as an independent object of individual type, defined by the correlation to its course of values. By the Russellian Theory of Types, functions as formal structures of predication are admissible on the basis of the order of their objects, and thus strictly depend on their evaluations. This evolution

[28] *Russell 1908*, p. 157.

[29] A condition which Russell notoriously restricted by formulating the Axiom of Reducibility (AR): for each formula f there is a formula g with a predicative type such that f and g are logically equivalent, where a type is predicative if none of its objects are of a higher order than the order of the elements of the class to which that object should belong. The meaning of this principle for the determination of the range of a variable is crucial to the later understanding of abstraction procedures, and it shall be reconsidered in relation to type systems and the methodological clarification of the last section.

has led to a different model of function, which deeply influenced the notion of type: it restores the old-fashioned notion of function as rule rather than as graph, i.e. it consists of an operation from an argument to a value. I will now consider this model of function, in particular by comparing the type-free and the typed versions of λ-calculi: this makes it possible to analyse the related notion of abstraction, and to introduce the connection between functions and the modern notion of logical types.

4. Abstraction in type-free λ-systems

Abstraction has been progressively formalized by means of functional expressions: propositional functions à la Russell replaced Fregean concepts, and real variables took the place of the abstraction procedure. This change is further exemplified in λ-calculi, first developed by Church in his *1940* combining the Russellian calculus and the operation of deramification which removes all the orders on types.

In a type-free structure the objects of study are both functions and arguments; the alphabet of such a calculus is formed by λ-terms, which are formal expressions for functions and for applications of functions. A type-free λ-system uses variables as the basic terms of the calculus. The set of so-called pre-terms of a type-free λ-system is composed of variables, abstraction on variables and application on variables.[30] Abstraction consists in moving from the expression M to $\lambda x.M$, expressing the function $x \mapsto M$, from a set of variables to a formula. The formula $\lambda x(\lambda yM)$ expresses the nested abstraction over x over y with respect to expression M: its meaning is that of a function that maps any argument to a function (e.g. taking the function from any argument to itself, one defines the identity function). On the other hand, application consists in moving from two expressions M and N to the term MN, expressing the result of $M(N)$ (which provides an instance of a predication). The operation of β-reduction allows for going from the abstracted term $\lambda x.M$ applied to argument N to $M[x/N]$, which expresses the substitution of N in place of free occurrences of x in M. The definition of equivalence classes on the evaluation of pre-terms defines the closure operation on λ-terms of the type-free λ-system.

This standard analysis of the operations of abstraction and instantiation within a type-free λ-system shows immediately that the related notion of abstraction provides no connection to abstract terms, either semantically or ontologically defined. Moreover, it is precisely the type-free structure that led to interpreting the functional model in terms of graphs: in the language no theoretical distinction between function and argument is formulated, one considers both an expression FA which denotes the data F considered as algorithm applied to the input A, and the expression FF that denotes F applied to itself. In this kind of language, everything actually is or is meant to represent a function, based on a composed process:[31]

- *Abstraction*: consists in the replacement of a part of an expression by means of a variable element, the process called *functionalization*;
- *Application*: consists in the substitution of a certain value in place of the variable, to form a new expression, and accounts for the process of computation of its output.

[30] For a full introduction to type-free and typed λ-systems, the standard reference is *Barendregt 1984*.

[31] See e.g. *Laan 1997*, pp. 4–5.

The combination of the two parts is essential to the formulation of a function, and application is in fact the main operation, whereas abstraction is complementary. The other relevant operation is, notoriously, reduction, consisting of the process of computing from a λ-abstracted term to its value. The usual procedure of evaluation is not, here, the one of application (that we have seen to induce only a procedure as argument of another procedure): the process of calculating values is obtained by β-reduction $M \rightarrow_\beta N$.

The model of abstraction at hand in these languages is different when compared with the one holding for the Fregean notion of function, especially because of the type-free nature of the terms. The result of abstraction is performed by an operator, and it produces a function rather than being a function formalizing a predicate or a concept. Consider the numerical expression $2+3$, and its transformation into a function, by which one takes into account first the λ-term $(\lambda x.x+3)2$ which is the β-expansion of the given numerical expression: in this transformation we have an argument (2) replaced by the argument-variable (x) via the λ-abstractor. What is peculiar in this operation is that one already has the abstracted term (which in turn performs the role of an abstractor operator on values) without necessarily having the starting term which is abstracted: the operation of removing an argument as a function construction process is reinterpreted as the application of β-expansion to a λ-term. Instantiation corresponds to application plus β-reduction of the function.[32] The peculiar property in this form of abstraction is the conceptual identity between the predicative part and the argument, so that everything is an operation. In this sense, there is no term representing the result of an abstraction procedure, nor a term determining the (semantic) context of predication.

The abstraction operators are such that they can be applied to functions without considering the order of progressively higher types: the abstraction is a pure operation, not a complete process. And functions are first-class citizens. The resulting notion of function for these calculi is therefore considered in terms of evaluation (function values), rather than in terms of objects (abstract functions). In other words, no (standardly interpreted) function-object distinction prevails and abstraction is a primary operation only along with application. Moreover, the definition of abstraction requires as its basis the notion of β-normal form of a λ-term which corresponds to a propositional function (by the calculus such expression in normal form is unique, if it exists); then the set of abstractions of a propositional formula corresponds to the set of β-expansions of the corresponding λ-term.

In the following section, I shall consider the effect of reintroducing types in such a λ-system. One way to look at this step is to consider the reduction of the abstraction process to procedural semantics. If such a task has to be accomplished, one has to explain how a 'procedural' meaning determines the abstract terms corresponding to the objects defined.

5. Abstraction for typed λ-systems

The formulation of the typed version of λ-calculi affects the simple version in an essential way: the range or domain of λ-terms reverts to being fixed (i.e. typed), as in standard mathematical functions. That is, a typed λ-term is again a function, for example, from natural numbers to natural numbers $n \rightarrow n^2$. In this way the

[32] *Laan 1997*, p. 43.

operation of λ-abstraction in the typed version strengthens the role of the function/object structure: abstraction is contentual, within a definite meaning-determiner. From this perspective, there is a historical event that effected the relation between abstraction and meaning in the evolution of typed λ-calculi, namely the different versions introduced by Curry (*1934*) and Church (*1940*)[33].

I shall in the following use a standard vocabulary for the language of typed λ-calculi: a set of type variables $\{\alpha, \beta, \dots\}$ with corresponding arbitrary types $T = \{\tau_1, \tau_2, \dots\}$; a set of contexts (also called bases) $C = \{x_1 : \tau_1, \dots, x_n : \tau_n\}$, with domain of a context $\Gamma = \{x_1, \dots, x_n\}$ given by members of the set of variables in the λ-terms $\Lambda : \{M, N, \dots\}$, and its range by the set of simple types.

In typed systems à la Curry, a typability relation is defined among the sets of contexts, λ-terms and simple types by rules for instantiation of a member of a context (start-rule), λ-abstracted term (introduction) and application (elimination); the system is completed by a standard β-contraction rule. A context is a set of assumptions stating that certain elements have certain types, and this is used to build type expressions of the abstraction and application kind.

In the typed λ-calculus à la Church, on the other hand, one starts from a (denumerable) set of λ-terms, variables and the set of simple types. This formulation of the language has an explicit declaration of type for each variable, whereas in the system à la Curry the type of a function is inferred from the related environment. In the explicitly typed form of expressions, every term of the calculus now has a normal form, i.e. all possible β- and η-reductions terminate, which makes the set of typable λ-terms entirely recursive. Church-style languages are thus based on the correspondence between terms and types, i.e. the definition of the latter in terms of the former. The rules for the formation of abstracted and applied terms reflect the typability relation:

$$\frac{\begin{array}{c}\Gamma, x : \tau_1 \\ \vdots \\ M : \tau_n\end{array}}{\Gamma \vdash \lambda x : \tau_1 . M : \tau_n : \tau_1 \rightarrow \tau_n} \qquad \frac{\begin{array}{c}\Gamma \vdash M : \tau_1 \rightarrow \tau_n \\ \vdots \\ \Gamma \vdash N : \tau_1\end{array}}{\Gamma \vdash MN : \tau_n.} \tag{2}$$

The first rule (Abstraction) says that from a context Γ and assumption of type τ_1, if one derives a term M of type τ_n, then one obtains the term of functions from τ_1 to τ_n by λ-abstraction: for each term $M : \tau_n$ and each variable $x : \tau_1$, $(\lambda x : \tau_1 . M : \tau_n) : \tau_1 \rightarrow \tau_n$ is a term. The second rule (Application) brings back this abstracted term to a function of instantiated terms: for each couple of terms in context, $M : \tau_1 \rightarrow \tau_n$ and $N : \tau_1$, the application $(MN) : \tau_n$ is a term.

The formulation of the typed version of λ-calculi follows intuitively the Brouwer-Heyting-Kolmogorov (BHK) interpretation: a proof of an implication is a construction, and a construction of an implication is a function. Thus, the end formula of a derivation Π is the type of Π; application corresponds to modus ponens, i.e. by the application MN one will intend M to be a function type of the form $\tau_1 \rightarrow \tau_n$, N to be of type τ_1 and the result to be of type τ_n; and abstraction can be seen as the discharging of assumptions in natural deduction. It is in view of the role of assumptions in abstraction processes that a new understanding of this procedure is developed.

When considering this formulation of the abstraction rule, a relevant distinguishing feature among the two formulations of typed λ-systems appears: in

[33] See *Sorensen and Urzyczyn 2006*, especially sections 3.1–3.3.

the version formulated by Church, abstractions have declared domains, so that they assume the form $\lambda x:\tau M$, for x in the set of variables of type τ, abstracted with respect to expression M; on the other hand, in Curry-style systems, abstractions have no domain, that is they are of the form $\lambda x.M$. In these systems an extremely important conceptual shift is therefore related to the notion of abstraction: the bound variables formulated in environments convey *abstract information* related to the term syntax. λ-systems à la Church provide fixed types for all variables (in terms of their typing contexts) and terms, and expressions contain full type information (whereas a so-called type-assignment system Curry-style would not have such full information in the basic syntax). The procedure that allows transforming a fully built term into a core one, i.e. one providing only the necessary type information where every pseudo-term induces a type-free λ-term, can be seen as an operation of erasing the domain of information: this is the newly formulated idea of an abstraction process. Conversely, the typability of terms (based on the Curry–Howard isomorphism) consists of filling in a proof-trace with the missing formulas. Hence, contextual information describes the relevant data type under which a given reduction terminates.

I shall in the following define this abstract informational data contained in contexts as the process of forgetting data-constructions. According to this relation between the informational content of terms and the procedures of abstraction, one needs now to distinguish between two different uses of 'abstraction':

- *λ–abstraction terms* consist in proof-steps by generalization, formulating function types;
- *Abstract informational data* is the set of terms obtained by procedures of abstraction from an existential type.

This distinction is clearly based on the logical nature of types as logical objects of higher degree and the formulation of abstraction in terms of dependent term-constructions.[34] In the following, I will analyse further this connection, aiming to establish further links between the logical notion of type, the procedure of abstraction and the epistemic notion of information.

6. Dependent type systems and applications

The formulation of typed versions of λ-systems suggests a new logical form of abstraction procedures: from a process based on predication and determining the notion of function, abstraction has become the representation of a functional term (via predication) which refers to external assumptive data. The crucial step in this new

[34] It is important to notice here – especially with respect to the analysis provided in the next sections – that the formation of classes of type constructors depending on terms has no computational difference with respect to the class of non-dependent types in constructive systems. As shown in *Berardi 1990*, this dependence corresponds to the coding of high-order proofs in first-order arithmetic: this result is proved by showing that typable terms and representable functions are the same with and without type dependence; that the strong normalization property is equivalent with and without type dependence; and that the set of logical theorems of a given λ-calculus is characterized by proving that a logic with type dependence (even if it can prove some new theorems) is equivalent to a logic without type dependence. It will be my aim to show in the following section in which sense the dependency relation is conceptually relevant for the notion of predication and abstraction when interpreted in terms of informational abstract data.

interpretation of the notion of abstraction is represented by the use of contexts for type declarations, and the related formulation of the languages for dependent types.

The mutual dependency between types and terms in such languages is given in the form of terms depending on terms, and terms depending on types; but also, types depending on terms and types depending on types. The formulation of dependent types is therefore interesting because it shows two different forms of abstract terms for dependency: on the one hand, in a reduced functional expression $F:\tau_1 \to \tau_2$, such that $\mathrm{M}:\tau_1$, the term $FM:\tau_2$ depends on the term M; on the other hand, a generalization on variable types $M:\forall x.\alpha \to \alpha$ implies that the term $M\alpha:\alpha \to \alpha$ depends on the variable type α. This dependency applies also to higher degrees, provided functional abstraction holds both on $FM:\tau_2$ depending on the term M, and on $M\alpha:\alpha \to \alpha$ depending on the variable type α.[35]

The dependency of types on types tells another story: the definition of a functional λ-term $f=\lambda\alpha.\alpha \to \alpha$ *requires* the variable type α as its dependency relation. Formally, this amounts to saying that one should be able to find a constant for any instance of variable types α,β such that a functional relation $f=\alpha \to \beta$ is in that constant. One way of seeing this is precisely to assume objects of higher degree with respect to both terms and types, so called *kinds*. For any term M in type τ, then M is called the constructor of kind k, provided $\tau:k$.[36] The introduction of kinds recalls explicitly the formulation of higher-order meaning determining objects, which claim a conceptual priority over the instances.

A different form of dependency is that of types on terms. Provided τ is a type and k its kind, then the function $\tau \to k$ will be a kind. Given the functional abstraction $f:\tau \to (k' \in k)$, where k' is a constant in the kind k, and one predication of term $M:\tau$, a function $fM:k'$ is a term-dependent type (namely dependent on M).

It will be by means of the constructive strengthening of this dependency of types on types and terms (namely: accepting the definition of types by terms along with the conceptual presupposition of types by terms) that a complete procedural view on abstraction is reached. There is a common trait in the dependency of types on terms and types that relates to the notion of abstraction (in its double interpretation). Kind formation, type formation and term formation (predication), are all expressible within contexts of abstract data. This opens a completely new perspective, and it can be entirely expressed in the procedural formulation: the role of the meaning-determining abstract terms is entirely reduced to the procedure of reconstructing appropriate abstract information. The notion of abstract data in context corresponds – informally – to the conditions that allow predications to be performed. However, these formulas in context cannot be intended as arbitrary assumptions: rather, one needs to know that the related applications are *legal*. This reduces to the very simple observation that application justifies both functional abstraction and the abstraction on data types in the structure of context assumptions. This is what I shall call the *procedural view on abstraction*.

The polymorphic (second-order) typed λ-calculus, introduced independently by Girard (*1972*) and Reynolds (*1974*), is standardly obtained by adding to the language variables ranging over predicates, sets of functions and allowing quantification over

[35] See e.g. *Barendregt 1993*, p. 82.

[36] This is for example the structure of the language $\lambda\underline{\omega}$ in *Barendregt 1993*, where the constant $\Box$ (not part of the language) is such that $k:\Box$ means that k is a kind, is introduced as an axiom, and the introduction of types is given as members of kinds.

such variables. In formulating the second-order level of typed λ-calculi therefore, once again the usual operation of abstraction by generalized quantification over type variables is applied: type variables are types; the set of functions $\tau_1 \rightarrow \tau_2$ is a type, provided τ_1, τ_2 are types; and given type τ and type variable α, then $\forall\alpha\tau(\alpha)$ is a type. The role of the standard application process can again be expressed in terms of the BHK interpretation: a construction of $\forall\alpha\tau(\alpha)$ is a function that leads from every construction of α to a construction of $\tau_1(\tau_2)$. The obvious remark here is that the class of formulas the quantifiers can be taken to range over consists of the full set of second-order propositional formulas (that is including all types), so that the meaning of a formula $\forall\alpha\tau(\alpha)$ can be taken to be determined by all formulas satisfied by the substitution operation of the type variable α with a type constant τ, including the cases where this constant might be of the same or higher complexity than $\forall\alpha\tau(\alpha)$ itself: this obviously leads to impredicativity of second-order logic. But it is the reference to constructions that becomes crucial here: quantifiers range over definable types, namely those for which a construction method is provided. Well-typed λ-terms are thus obtained by inference rule formulated under context of assumptions, ordinary abstraction, polymorphic abstraction or type application.

Polymorphic abstraction (with its counterpart type application) is now the crucial element to look at. Polymorphic abstraction $\Lambda\alpha.M$ consists in taking a term M as a polymorphic procedure with respect to type parameter α. The introduction of the mechanism of parameters for defining access procedures (for example to the induction axiom, usually done in order to settle a difference between developers and users, a distinction which has been essentially produced by the AUTOMATH system) requires that abstraction be allowed only if always followed immediately by application;[37] thus, once again, the abstraction procedure cannot be considered by itself as eventually producing any kind of abstract object, it consists rather in the forgetting procedure on data construction. For the other standard inference rules (introduction, application, elimination), the very same property is reflected by the requirement that such a rule induces a proper term if free variables in the context are typed. Notice that this is an undecidable problem: it cannot be decided if the problem 'Given a type τ, is there a closed term for τ?' has in general a solution for the polymorphic λ-calculus.

These formal considerations lead to the formulation of two main *levels of abstraction*: the first is represented by taking variable types and allowing quantification over them; the second is clearly represented by allowing derivations to be performed on the assumption of abstract data types being reduced to term expressions.

In the direction of the first notion of abstraction went the idea of prototype proof, first formulated by Herbrand.[38] A prototype proof is, briefly, the proof of a universally quantified statement, whose verification is applicable to each specific instance of the quantified variable. It is executed by assuming a certain generic element of the set the quantification ranges over, in this way making the proof independent from that specific element, but rather dependent only on the assumption that that certain element for which the proof is done belongs to the same set the quantifier is ranging over.[39]

[37] See *Laan 1997*, p. 224.

[38] See *Herbrand 1971*, pp. 288–89 and the related analysis in *Longo 2000*.

[39] *Longo 2000* provides a reading of the notion of prototypic proof in type systems under the propositions-as-types interpretation, where one can consider this kind of proofs as λ-terms: this is done by considering a simple type called *generic*, i.e. such that it can be assumed as a variable, and whose proof is provided by a specific instance called 'parametric', i.e. which can be uniformly substituted.

On the other hand, we can interpret *abstract data types* (ADT) as models. ADT are defined as a set of data values (*abstract data structure*) and associated operations (*interface*) considered independently by any specific implementation. These data represent the result of an operation of abstraction intended as the process of deleting unnecessary details from necessary characteristics in order to solve a problem (i.e. to provide the correct operations on a certain set of data). This operation aims at defining data relevant to the problem plus the operations which are to be executed. Clearly, the process of obtaining the relevant data is exemplified by the *abstract predicates* entering into the solution of the problem; as in the case of the application function, this process of abstraction is never taken separately from the determination of those operations which are to be performed on the empty schema. A syntax obtained by induction on the structure of datatypes is known as *polytipic abstraction*: by this term one understands formalizations and verifications abstracted with respect to a large class of datatypes, which is especially relevant in functional programming. A simple example is that of the function *map* in the Hindler–Miller type system (*map*: $\forall A,\ B.(A \rightarrow B) \rightarrow (list(A) \rightarrow list(B))$) which provides a structure of transformation of data lists into other kinds of data, an untouched schema, irrelevant to the kind of data instantiated as object of that function.[40] Polytipic constructions can be used not only to define functions in a generic way, but rather also to formally state polytypic theorems and to synthesize polytypic proof-objects in a formal way. In this case one is dealing with a procedure of abstraction by which an empty model is obtained, to implement all the different data of a certain range of (differently) equivalent types, and to be used in terms of application: this model is intended to abstract from (in the sense of assuming) all the information needed to implement the construction. This obviously requires that the basic distinction between monomorphic and polymorphic languages holds, so that even a notion of universal polymorphism can be formulated as a function that works on a range of variables having a common structure.[41]

The analysis developed about abstraction for λ-systems, and in particular for their typed versions, has shown some relevant intuitions about this notion. These can be summarized as follows:

- Functional abstraction and application on types of ordered degrees present the structure of an empty model and its fulfillment, originated with the Fregean interpretation of predication by function;
- This compound structure provides a definitional procedure for terms rather than building an abstract term or object, like in the case of the function–concept structure;
- This stresses the procedural approach and avoids the ontological/metaphysical implications on abstraction.

In this analysis, different notions of abstraction are determined by the distinction between abstraction for types of higher order and abstraction as the empty

[40] See *Pfeifer & Ruess 1998* and *Pfeifer & Ruess 1999*.

[41] An example of application is given by the so-called *parametric polymorphism*, according to which the same object or function can be used uniformly in different type contexts without changes (provided all data are represented). See *Cardelli & Wegner 1985*, p. 477.

formulation of constructions with respect to a context of expressions in dependent types. In this way, one is led to approach the notions of abstraction and instantiation at distinct levels, where a full explanation for the epistemic meaning of this second form of abstraction is still apparently missing. It is in the reformulation of the problem of meaning in terms of information that the circle closes again.

7. Constructive types and abstraction

Among the formulations of dependent type theories, the intuitionistic version given by Martin-Löf is probably the best model to explore the connection between abstraction and procedural semantics, and to consider the role that contexts for dependent predications play with respect to this topic. The aim of the present section is to analyse the relation between (dependent) predication and meaning.[42]

The constructive version of type theory (CTT) is equipped with a conceptual and semantic frame that provides an interpretation for the procedural approach to abstraction, restoring the link between types and abstract semantic terms. Abstraction for such a theory provides a more general methodological approach: the epistemic notion of *information* shall be used to formulate it. The core thesis is that the syntactical (rule-based) procedures do not fully explain the notion of abstraction involved by types. On the one hand, the removal operation by which the notion of empty (polymorphic) model is obtained, and on the other hand, the notion of abstract (meaning-determining and predicative-component) object involved by the definition of types themselves are the two distinct aspects which we shall consider. The procedural semantics on dependent types is the bridge between them.

At the syntactic predicative level, the notion of functional abstraction is satisfied in terms of rules. Those regarding abstraction and application concern the informative content of expressions in terms of the related constructions, strictly following the BHK-interpretation seen for the typed versions of λ-calculi. Abstraction and application refer to the standard syntax of categorical and dependent judgments. The foundational perspective in CTT is epistemic and the starting point for introducing types is represented by predications. Types are given by the propositions-as-types principle and the sets-as-props identity. In the following I shall use Greek capital letters Γ, Δ, ... as metavariables for contexts of assumptions; Latin capital letters A, B, ... for metavariables for types; small Latin letters a, b, ... for proof-objects and x_1, x_2, ... as proof-variables. Both propositions and sets are justified by appropriate judgments, rightfully asserted if and only if one knows respectively an element of the set or a proof for the proposition A; moreover, one needs to know how to establish equality among such objects:

$$A : set$$
$$A : prop$$
$$a : A$$
$$a = b : A.$$

[42] The full formulation of CTT by Martin-Löf is spread over a number of papers and talks given at various conferences. For the purpose of this paper, *Martin-Löf 1984* and *Martin-Löf 1993* represent the basic references. A full introduction to the syntax and semantics of CTT is contained in the first chapter of *Primiero 2008*.

Two such types will be equal ($A = B : set/prop$) when they have same objects for which the same equality relation holds. To know an element in this setting is equivalent with formulating a construction. This extends to dependent judgments:

$$\begin{array}{c}(x_1 : A_1, \ldots, x_n : A_n)\\ a : A\end{array}$$

which states that a is a construction for the set/proposition A, provided the set of assumptions is verified, i.e. that appropriate constructions have been substituted for variables: $([x_1/a_1] : A_1, \ldots, [x_n/a_n] : A_n)$.[43]

The set J of judgments is therefore defined by the formula formation rules establishing that any expression of the form $a{:}A$ (categorical judgment) is a formula whenever a is a proof-constant of A in the set of propositional contents; and any expression of the form $(x_1 : A_1, \ldots, x_n : A_n)a : A$ (hypothetical judgment) is a formula whenever a is a proof-constant of A in the set of propositional contents, provided that $x_1, \ldots, x_n$ are proof-variables for $A_1, \ldots, A_n$ and substitution with proper constructions are performed.

Π-introduction is the rule to construct an independent object of the lowest individual type by the following derivation:

$$\frac{\begin{array}{c}(x : A)\\ b(x) : B(x)\end{array}}{\lambda((x)b(x)) : (\Pi x : A)B(x)} \tag{3}$$

which says that if x is a variable of the type A and $b(x)$ is a term of the type $B(x)$ (i.e. dependent on the term x in A), then a canonical element in A can be abstracted belonging to any of the elements in B. In this rule, functional abstraction in the sense of λ-calculi is at hand. The related rule of Π-elimination (Application) will be respectively as follows:

$$\frac{b : (\Pi x : A)B(x) \quad a : A}{b(a) : B(a)} \tag{4}$$

Formation of functions of higher type is instead obtained by a different form of abstraction:

$$\frac{\begin{array}{c}(x : A)\\ b : B\end{array}}{(x)b : A \to B} \tag{5}$$

which says that if x is a variable of type A and b is a term of type B, then $(x)b$ is a term of type $A \to B$; this is explained by the ordinary β-rule, expressing what it means to apply an abstraction to an object in A:

$$\frac{a : A \quad b : B(x : A)}{((x)b)(a) = b[x/a] : B[x/a]}. \tag{6}$$

[43] Here and in the following, round brackets are used for contents in contexts, whereas square brackets are used for the substitution procedures within contexts.

The explanation of these rules provide the proper distinction between universalization and (informational) abstraction on contents. The procedures of abstraction and instantiation express their contents in terms of information for the relevant constructions. The object of abstraction is here the informative content of constructions for judgements:

Principle 1 (Forget–Restore Principle (*Sambin & Valentini 1998*)) *To build up an abstract concept from a raw flow of data, one must disregard some information, and an abstraction is constructive when the information forgotten can be restored at will.*

Under this interpretation, abstraction corresponds to an operation of forgetting from computational information, whereas instantiation means restoring such information. In particular, by a procedure of abstraction one obtains the transition from the monomorphic to the polymorphic versions of the theory. This explanation represents the constructive counterpart of the principles of functional abstraction seen for typed λ-calculi.

How this notion of abstraction relates to predication and meaning in CTT is explained by giving a closer look at the sets-as-props identity. The philosophical basis of CTT is entirely given by such formal identity $prop \subseteq set$: the formulation of any judgment of the form $A : set/prop$ is justified by the appropriate construction, but it relies – from the point of view of the concepts involved – on the introduction of the notions of propositions and sets as categories of predication. To this aim, their formal expression is given by the use of appropriate axioms

$$prop : cat$$
$$set : cat$$

where *cat* stands for the predicative category. It is in view of the relation between the introduction of types as categories and their justification as constructions that the link with meaning is hereafter analysed.

By insisting on the procedure of abstraction in terms of removing the informational content of the constructions, one provides the judgment declaring truth of the involved types:

$$\frac{a : A}{A \quad true}. \tag{7}$$

This formulation simply expresses the propositions-as-sets interpretation, assuming that an appropriate presupposition for the first judgment is the introduction of A as either *prop* or *set*[44] and a multi-level typed λ-calculus can be provided for rigorous treatment of judgments of the form 'A true', on the basis of canonical expressions of the form $a : A$. In this sense, abstraction procedures of this form allow for type-expressions of the form $A : cat$

$$\frac{A : set(/prop)}{A : cat} \tag{8}$$

where abstraction applies on the construction justifying the judgment that A is a set (a proposition). From the procedural point of view, this corresponds to an

[44] Cf. *Nordström, Petersson & Smith 1990*, p. 37.

abstraction from the specific data type; its justification is entirely based on the appropriate construction, it corresponds to a *specification*, a procedure of instantiation, with the same structure as in ADT or in polytipic abstraction.

As formulated by the *Forgetting–Restore Principle*, information is the content of such procedures: in other words, the procedural semantics at the basis of CTT does not only allow for the removal of irrelevant data (as in the case of the constructions for the assumptions); it also allows abstraction from instances of concepts. Abstraction, in this sense, formulates the 'routine' from the intensional to the extensional level, corresponding to the model of a construction; instantiation (application) goes from the extensional to the intensional level, corresponding to the filling of an empty structure. The explanation of this abstraction procedure is determined by the relation between the syntactic definition and the semantic introduction of types, and it has to be given accordingly to the syntactic-semantic method of CTT. It corresponds to the thesis according to which there is a level at which types can be considered conceptually anterior to their objects, even though it is at the level of constructions only that the meanings of the former are explained.

This procedure leads to the analysis of types as objects of possible predication. This informal definition corresponds, in the formal lanugage of CTT, to judgments intended as presuppositions for expressions. The interesting aspect for the theory of abstraction concerns the fact that the meaning (justification) of a judgment $A : set/prop$ relies on its construction $a : A$, whereas the possibility of using its information content (for example in a context of assumptions) is given by the (justified) presupposition $A : cat$. In other words: on the one hand the semantic explanation of a judgment is given by the related instantiation and non-trivial identity ($\ldots : A$); on the other hand, its use as a meaningful object of predication is possible in terms of its introduction as a meaning category ($\ldots : cat$). The category declaration is therefore justified by the syntactic definition of types in terms of constructions; known types can instead be used as predicables, that is as terms apt to be predicated. This structure preserves in the first instance the notions of knowledge as predication and predicability as possibility of knowledge.

In the *order of the real* the abstraction procedure allows erasing information from arbitrary elements of well-defined types, i.e. abstraction is justified by the well-foundedness of predicative expressions with substitution and non-trivial identities. According to the epistemic formulation, to know a certain object (to formulate a construction) is the way a set is known to be inhabited, or a proposition is known to be true. At the (programming-)language level, such a distinction is reflected by the one between Data Type as a certain type given in terms of introduction rules for inductively generating canonical values of that type, and Abstract Data Type as the general structure allowing one to build a specific data type. This means that one does not obtain types at the conceptual level by means of abstraction, rather they are existent on the basis of their constructions and from these the process of asserting a type to be true is obtained by means of abstraction:

$$\text{Abstraction} \quad \begin{matrix} \text{INTENSIONAL} \\ \downarrow \\ \text{EXTENSIONAL} \end{matrix} \quad \text{Knowledge}$$

Higher types, i.e. types of the monomorphic kind, are explained instead as 'abstract terms' insofar as they provide a connection between predication and semantic context.

In the *order of concepts*, the nature of types is prior to objects, they can be clarified as meaningful abstract terms, or universal predicables. Where categories make predication possible, types are to be intended as abstract and purely informational meaningful entities, in the same way as a function is satisfied by its object. Types intended in this latter sense, i.e. as meaning-objects in the form of abstract entities, allow one to explain why the ontological question 'What is a type?', is complementary (but reduced to) the epistemic formulation 'What does it mean to know a type?':

CONSTRUCTION
Predicability ↑ Application
MODEL

In the following, I will suggest that a suitable interpretation for the meaning entities of higher order is that of *purely informational entities*.[45] This means that the introduction of the type of all types – initially suggested by Martin-Löf in the impredicative version of the theory – is overcome by a different conceptual formulation: the identification of propositions and types (and in turn also of sets) is given in terms of the formulation of the former as a category of predication, which means that the related quantification ranges only over the instances and not over the category itself.

In a slogan: the definition of types in terms of constructions implements the 'meaning is use' approach; the notion of category as informational entity allows one to extend it to a 'meaning in use' approach.

8. Abstraction as hidden information

Abstraction in dependent languages – and in particular as analysed in CTT – is neither just the kind of mathematical abstraction that gives rise to models, nor the form of predicative abstraction obtained by removal of non-universal terms we have seen at hand in the older philosophical interpretations. In this new logical framework, meaning and information are linked to each other. And one crucial paradigm change is the result, namely interaction. This new paradigm of abstraction is to be found in computer science, a very new form of understanding concepts and meanings.

The constructive profile of the definition of abstract terms from the previous section shows the new interpretation of logical abstraction at hand in CTT. The essentiality of a concept (in the Aristotelian sense) is no longer given by abstraction '*that eliminate inessential details*' and that '*inasmuch as such details constitute information* [represents] *information neglect*'.[46] The strict interplay between types and their terms (constructions), such that meaning of the former is always given in terms of the latter, consists of a procedure of *information hiding*, exemplified by the requirement of the reconstructibility of contents, substitution procedure on empty place-holders. Abstraction as a procedure of information hiding shall in the

[45] These distinctions are methodologically (other than conceptually) needed, as is shown by the Girard's Paradox about the existence of a type of types being an element of itself, established by extending the theory by means of the axiom $U : U$. See *Girard 1972*, further analysed in *Coquand 1995*. For an elaborate presentation of the inconsistency of type theory extended with an axiom for higher-order objects, see also *Jacobs 1989*.

[46] *Colburn & Shute 2007*, p. 176. Emphasis in the original.

following be further analysed with reference to computer science.[47] In parallel with such a new paradigm, the value of types as abstract terms is here explained not in their presenting some general emptyness on which their value is based, but rather in presenting always a filled (constructed) structure, one that might hide its information, as in the use of assumptions in context. A context is intended as nothing else than the load of needed data that is hidden to the user, and that can be recalled at will (as by the Forgetting–Restore Principle requirement).

In this perspective on abstract terms and abstraction procedures a major role is therefore played by the explanation of the notion of information. In the (meta-)language of CTT, the introduction of types corresponds to the introduction of meaningful informational entities; it expresses the notion of semantic information. On the other hand, operators and rules apply to syntactic data, whose procedures of forgetting and restoring consist essentially in saving and deleting syntactic information. The former notion of information is based on (justified by) the latter, but its use can be independent, precisely in virtue of the *information hiding* kind of abstraction. Hence, the notion of information represents in CTT an all-invasive concept related to abstraction and instantiation; it has an epistemic nature, it provides an abstract term corresponding to a meaningful (predicable) object; it recovers the steps of predication by reinterpreting removal as forgetting–restoring constructions, it allows the use of 'abstract entities' in terms of meaning. This is the methodological approach at the basis of the so–called *methods of levels of abstraction*.[48]

Let us refer to our structure of predication as an epistemic system. The system uses the 'hidden information' idea of abstraction to interpret expressions of the form $A : type$ and $x : A$. These expressions convey the same meaning that dependency on types and dependency on terms express in the formulation of the typed version of λ-systems. The expression $x : A$ corresponds to the definition of a 'typed variable'[49] in a dependent judgment, i.e. a judgment whose construction is dependent on a term. The related operation of typing is a presupposition for any such assumption, expressed by a formula of the form $A : type$ and expressing the dependency on a type. The 'type value' is what turns the typed variable into an observable[50] (constructable, predicable): this results in the instantiation of a predication, or substitution of the variable by a proper construction $[x/a : A]$ (objects of types). The degree of adequacy of a type corresponds to the formulation of proper environment $\gamma : \Gamma$ from which the agent is allowed to extract values to instantiate variables in context $[x : \Gamma]$. In the standard interpretation of the Method of Abstraction, the input is represented by a set of data, and the output is a model comprising information. The definition of a level of abstraction as a finite but non-empty set of observables corresponds in our system to a knowledge state. Such a knowledge state is represented by a predicative structure in which abstract terms (types/category of predications) are justified in terms of constructions, following the procedural semantics. To specify the level of abstraction means to clarify the range of meaningful concepts and instances of some predication procedure. A level of abstraction of greater resolution contains more information than a more abstract

[47] See *Colburn & Shute 2007*.

[48] See *Floridi 2008*.

[49] See *Floridi 2008*, p. 305.

[50] See *Floridi 2008*, p. 306.

one: thus types as meaning objects are purely informational entities, i.e. abstract terms determining the syntactical information provided by the data, and produced as output. Data types are here represented either as type-dependency or as term-dependency: each formulates a different level of abstraction. This epistemic interpretation allows for a full description of epistemic actions in the constructive frame, in particular by analysing the agent's state with respect to contents of information (satisfied by the constructive definition of types) and a weaker epistemic model given by the abstraction on those truthful contents.[51] This allows one to locate abstraction in an epistemic description of agents' knowledge processes, by preserving the connection both to predication and meaning.

The explanation of abstraction as an operation of information hiding can be further explored: contexts for dependent judgments are ways to display available computing resources for the agent; their contents express the operations that are supposed to be achieved if the entire epistemic program has to acquire meaning (express knowledge), ignoring in practice how they are achieved. The next level of abstraction is represented by the formulation of distinct agent-based protocols on contexts, to manage epistemic computational resources in multi-agent networks. To this aim, the second crucial element of the system is represented by the dynamics that can be defined on these informational repositories. This shall be a topic for further research.

9. Conclusions

In the evolution of the history of logic, through the disciplines that share its use, abstraction has received significantly different interpretations. There is no understanding of this notion without a complete account of these differences: from the formulation of empty structures of predications by removal of unnecessary elements, through the reification of abstract terms, to their linguistic formulation as meaning-determiners. By contrast, the formulation of λ-systems and their evolution into type-theories leads to a new paradigm, whose crucial element is an epistemic interpretation of the notion of information. This new approach is entirely taken up by the understanding of abstraction in computer science. The idea of information-hiding procedure is the basic step to extend the epistemic model of knowledge as proof-existence in terms of information-communication and exchange. This is essential for the understanding of the epistemic dynamics modelled by type-theories. The study of distributed systems from computer science can be a new model to inspire further evolution in the philosophy of logic.

Acknowledgements

Partial versions of this paper have been presented at the Logica 2006 Symposium (Hjenice, Czech Republic) and at the Logic Colloquium 2006 (Nijmegen, the Netherlands). A first partial version was published in Conference Proceedings as *Primiero 2007*. The author is Post-Doctoral Fellow of the Special Research Fund of Ghent University, whose support made the present formulation possible. Discussions with B.G. Sundholm, M. van der Schaar, P. Martin-Löf and B. Jespersen have been inspiring sources for my work on type theory. Discussions with L. Floridi, S.

[51] See *Primiero 2009*.

Sequoiah-Grayson and P. Allo have been important for my approach to epistemic information and its applications. The author wishes to thank three anonymous referees for their valuable comments and remarks.

References

Aaron, R.I. 1953. *The Theory of Universals*, Oxford: Clarendon Press.

Aristotle. *Works*. The Loeb Classical Library, vols. I, II, IV, XVII, XVIII, Cambridge: Harvard University Press. 1980–1997.

Bacon, J. 1995. *Universals and Property Instances. The Alphabet of Being*, Cambridge MA: Blackwell.

Barendregt, H.P. 1984. *The Lambda Calculus. Its Syntax and Semantics. Studies in Logic and the Foundations of Mathematics*, Amsterdam: North Holland.

Barendregt, H.P. 1993. 'Lambda calculi with types', in S. Abramsky, D.M. Gabbay and T.S.E. Maibaum, eds., *Handbook of Logic in Computer Science*, Vol. II. Oxford: Oxford University Press. pp. 117–309.

Bealer, G. 1982. *Quality and Concept*, Oxford: Clarendon Press.

Berardi, S. 1990. 'Type Dependence and Constructive Mathematics', PhD Thesis, Department of Informatics, University of Torino.

Brower, J.E. and Guilfoy, K. eds. 2004. *Cambridge Companion on Abelard*, Cambridge: Cambridge University Press.

Campbell, K. 1990. *Abstract Particulars*, Oxford: Blackwell.

Cardelli, L. and Wegner, P. 1985. 'On understanding types, data abstraction and polymorphism', *Computing Surveys*, **17**, 471–522.

Colburn, T. and Shute, G. 2007. 'Abstraction in computer science', *Minds & Machines*, **17**, 169–184.

Coquand, T. 1995. 'A new paradox in type theory', in *Proceedings of the Ninth International Congress of Logic, Methodology, and Philosophy of Science*, Studies in Logic and the Foundations of Mathematics, Vol. 134, D. Prawitz, B. Skyrms, and D. Westerstål, eds., Amsterdam: North-Holland, pp. 555–570.

Church, A. 1940. 'A formulation of the simple theory of types', *The Journal of Symbolic Logic*, **5**, 56–68.

Curry, H.B. 1934. 'Functionality in combinatory logic', *Proceedings of the National Academy of Science USA*, **20**, 584–590.

Dutilh Novaes, C. 2007. *Formalizing Medieval Logical Theories*, Logic, Epistemology and the Unity of Sciences Series, Vol. 7, Berlin, New York: Springer.

Fine, K. 2008. *The Limits of Abstraction*, Oxford: Oxford University Press.

Floridi, L. 2008. 'The method of levels of abstraction', *Minds & Machines*, **18**, 303–329.

Frege, G. 1879. *Begriffsschrift, eine der arithmetischen nachgebildeten Formelsprache des reinen Denkens*, Halle: Louis Nebert. Translated as *Concept Script, a formal language of pure thought modelled upon that of arithmetic*, by S. Bauer-Mengelberg in J. van Heijenoort, ed., *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*, 1967, Cambridge, MA: Harvard University Press, pp.1–82.

Frege, G. 1891. 'Funktion und Begriff', lecture at Jenaischen Gesellschaft für Medizin und Naturwissenschaft, Jena: Hermann Pohle. Reprinted in M. Textor, ed., *Funktion - Begriff - Bedeutung*, Göttingen: Vandenhoeck & Ruprecht, pp. 2–22.

Frege, G. 1892a. 'Über Sinn und Bedeutung', *Zeitschfrift für Philosophie und philosophische Kritik*, **100**, 25–50. Reprinted in M. Textor, ed., *Funktion - Begriff - Bedeutung*, Göttingen: Vandenhoeck & Ruprecht, pp. 23–46.

Frege, G. 1892b. 'Über Begriff und Gegenstand', *Zeitschrift für wissenschaftliche Philosophie*, **16**, 192–205. Reprinted in M. Textor, ed., *Funktion - Begriff - Bedeutung*, Göttingen: Vandenhoeck & Ruprechtx, pp. 47–60.

Girard, J.Y. 1972. *Interpretation fonctionelle et eleimination des coupures dans l'arithmetique d'ordre superieure*, These d'Etat, Universit Paris 7.

Hale, B., 1987. *Abstract Objects*, Oxford, New York: Blackwell.

Hart, W.D. 1979. 'The epistemology of abstract objects: access and inference II', *Proceedings of the Aristotelian Society*, **53**, 153–165.

Herbrand, J. 1971. W.D. Goldfarb, ed., *Logical Writings*, Dordrecht: Reidel Publishing Company.

Jacobs, B. 1989. 'The inconsistency of higher order extensions of Martin-Löf's Type Theory', *Journal of Philosophical Logic*, **18**, 399–422.

Kamareddine, F., Laan, T., and Nederpelt, R. 2004. *A Modern Perspective on Type Theory – From its Origin Until Today,* Applied Logic Series, Vol. 29, Dordrecht: Kluwer Academic Publishers.

Künne, W. 1982. 'Criteria of abstractness', in Smith B. ed., *Parts and Moments – Studies in Logic and Formal Ontology*, München, Wien: Philosophia Verlag, pp. 401–437.

Künne, W. 1983. *Abstrakte Gegenstände – Semantik und Ontologie*, Frankfurt am Main: Suhrkamp Verlag.

Laan, T. 1997. 'The Evolution of Type Theory in Logic and Mathematics', PhD Dissertation Thesis, Technische Universiteit Eindhoven, Enschede: Print Partners Ipskamp.

Longo, G. 2000. 'Prototype proofs in type theory', *Mathematical Logic Quarterly*, **46**, 257–266.
Lowe, E.J. 1995. 'The metaphysics of abstract objects', *The Journal of Philosophy*, **92**, 509–524.
Martin-Löf, P. 1984. *Intuitionistic Type Theory*. Naples: Bibliopolis.
Martin-Löf, P. 1993. *Philosophical Aspects of Intuitionistic Type Theory*, unpublished notes of lectures given at the Faculteit Wijsbegeerte Leiden.
Nordström, B., Petersson, K., and Smith, J. 1990. *Programming in Martin-Löf Type Theory. An Introduction*, The international series of monograph in computer science, Vol. 7. Oxford, New York: Clarendon Press, Oxford University Press.
Ockham, W. 1349. *Summa Logicae – Part One: Logic of Terms*; translated and introduced by M.J. Loux, Notre Dame–London: University of Notre Dame Press.
Pfeifer, H. and Ruess, H. 1998. 'Polytipic abstraction in type theory', in R. Back-house, T. Sheard, eds, *Informal Proceedings of Workshop on Generic Programming (WGP98)*, Marstrand, Sweden: Dept. of Computing Science, Chalmers University of Technology, and Göteborg University.
Pfeifer, H. and Ruess, H. 1999. 'Polytipic proof construction', *Lecture Notes in Computer Science*, **1690**, 55–72.
Primiero, G. 2004. 'The determination of reference in a constructive setting', *Giornale di Metafisica*, **26**, 483–502.
Primiero, G. 2007. 'On building abstract terms in type systems', *The Logica 2006 Yearbook,* Czech Academy of Sciences, Prague: FILOSOFIA Publisher, pp. 191–201.
Primiero, G. 2008. *Information and Knowledge – A Constructive Type-Theoretical Approach*, Logic, Epistemology and the Unity of Science Series, Vol. 10, Berlin, New York: Springer.
Primiero, G. 2009. 'An epistemic logic for becoming informed', *Knowledge, Rationality, and Action*, 441–467; *Synthese*, **167**, 363–389.
Reynolds, J. 1974. 'Towards a theory of type structure', in B. Robint, ed., *Proceedings of the Programming Symposium, Lecture Notes in Computer Science*, **19**, 408–425.
Ross, D. 1971. *Plato's Theory of Ideas*, Oxford: Clarendon Press.
Russell, B. 1908. 'Mathematical logic as based on the theory of types', *American Journal of Mathematics*, **30**, 222–262. Reprinted in J. van Heijenoort, ed. *From Frege to Gödel – A Source Book in Mathematical Logic*, 1879–1938, 1967, Cambridge, MA: Harvard University Press. pp. 150–182.
Sambin, G. and Valentini, S. 1998. 'Building up a toolbox for Martin-Löf type theory: subset theory', in G. Sambin, J. Smith, eds., *Twenty–five years of Constructive Type Theory*, Oxford: Clarendon Press. pp. 221–244.
Sørensen M.H. and Urzyczyn P. 2006. *Lectures on the Curry–Howard Isomorphism,* Studies in Logic and the Foundations of Mathematics, Vol. 149. Amsterdam: Elsevier.
Whitehead, N. and Russell, B. 1910–12. *Principia Mathematica*, 3 Vols., Cambridge: Cambridge University Press.
Zalta, E.N. 1983. *Abstract Objects – An Introduction to Axiomatic Metaphysics*, Dordrecht, Boston, Lancaster: Reidel Publishing Company – Kluwer.