Noname manuscript No.
(will be inserted by the editor)

The Porphyrian Tree and Multiple Inheritance

A Rejoinder to Tylman on Computer Science and Philosophy

Lorenz Demey

Received: date / Accepted: date

Abstract Tylman (2017) has recently pointed out some striking conceptual and methodological analogies between philosophy and computer science. In this paper, I focus on one of Tylman's most convincing cases, viz. the similarity between Plato's theory of Ideas and the Object-Oriented Programming (OOP) paradigm, and analyze it in some more detail. In particular, I argue that the (Neo)platonic doctrine of the Porphyrian tree corresponds to the fact that most object-oriented programming languages do not support multiple inheritance. This analysis further reinforces Tylman's point regarding the conceptual continuity between classical metaphysical theorizing and contemporary computer science.

Keywords Porphyrian tree · object-oriented programming · multiple inheritance · diamond problem · computer science · Neoplatonic metaphysics.

1 Introduction

Tylman (2017) has recently pointed out some striking analogies between certain philosophical theories on the one hand, and certain constructs and techniques from contemporary computer science on the other. He argues that these analogies are due to the fact that philosophy ultimately has the same aim as computer science, viz. to explain the nature of (significant parts of) reality by constructing models of it. Since human cognition has not substantially changed (in the biological sense) in the past 25 centuries, it should come as no surprise that the same types of basic intuitions about the nature of the world keep on resurfacing throughout the intellectual history of mankind, either in classical metaphysical theorizing or in contemporary computer science. This is in line with van Benthem's (2007, p. 80) suggestion that "computer science and its more pregnant form of Artificial Intelligence are just – in Clausewitz' happy phrase – 'the continuation of philosophy by other means'" and Martini's (2016a, p. 225) remark that "computer science never used ideological glasses [...] but exploited what it found useful for the design of more elegant, economical, usable artefacts".

Lorenz Demey
Center for Logic and Analytic Philosophy
KU Leuven
Leuven, Belgium
E-mail: lorenz.demey@kuleuven.be

In this paper I will focus on one of Tylman's most convincing examples: the analogy between Plato's theory of Ideas and the Object-Oriented Programming (OOP) paradigm. In particular, the paper's goal is to show that a later development of Plato's theory, viz. the Neoplatonic doctrine of the *Porphyrian tree* of Categories, is closely related to one of the main controversial issues in the OOP paradigm, viz. *multiple inheritance*. The historical and systematic details of this case study further reinforce and refine Tylman's original point.

The paper is organized as follows. First, Sect. 2 briefly rehearses Tylman's analogy between Plato's theory of Ideas and the OOP paradigm, and discusses some potential criticisms of this analogy. Next, Sect. 3 introduces the doctrine of the Porphyrian tree as a further development of Plato's theory, and Sect. 4 shows how this doctrine is related to the issue of multiple inheritance in OOP. Finally, Sect. 5 discusses two subtle limitations of the extended analogy, and Sect. 6 offers some concluding thoughts.

2 Plato's Theory of Ideas and Object-Oriented Programming

The exact interpretation of Plato's theory of Ideas is a matter of extensive scholarly debate, but for our current purposes it suffices to focus on just a few of its core features (Tylman 2017, Table 1). The central claim of this theory concerns the sharp distinction between unchangeable, eternal Ideas and destructible observable entities. Ideas serve as models or blueprints for observable entities, and a single Idea can serve as a blueprint for multiple observable entities (thereby purportedly explaining the commonalities between those entities).

Object-oriented programming (OOP) is one of the most successful programming paradigms in the history of computing; see Joque (2016) for a historical-philosophical perspective. It was first introduced in the 1960s, and is nowadays present in a wide variety of programming languages, such as C++, Java, Python, Perl, C#, Object Pascal (Delphi), Ruby, Objective-C, PHP and Smalltalk. Again, we focus on just of a few of its core features (Tylman 2017, Table 2). The most common implementations of the OOP paradigm draw a sharp distinction between classes and objects. Classes are usually compile-time constructs (and thus 'eternal' from the program's perspective), whereas objects may be freely created and destroyed during runtime. Classes serve as blueprints for the objects instantiating them, and a single class can have multipe objects as instances. The definition of a class typically involves a number of variables (sometimes called 'fields') and a number of methods, which capture the class's properties and behavior, respectively.

By comparing these two sets of core features, Tylman (2017) convincingly argues that there is a clear analogy between Plato's theory of Ideas and the OOP paradigm. However, this argument is not entirely unproblematic (also cf. Sect. 5). In particular, since the analogy does not take into account some of the other core features of OOP (such as encapsulation, i.e. information hiding), it seems to lose some of its specificity, and might equally well be applicable to other, more generic ways of organizing our knowledge into data types. For example, some of the historical precursors of OOP languages (such as Algol W and Algol 68) already provided general abstraction and modelling mechanisms (Martini 2016a,b), and thus it seems that Tylman's analogy should equally apply to those earlier languages.

Despite this potential criticism, the analogy is still essentially on the right track, and I will now further reinforce it by focusing on one particular aspect. Tylman (2017, pp. 6–7) explicitly notes that both Plato's Ideas and the classes in an object-oriented program constitute a *hierarchy*. Additionally, he remarks that in object-oriented programming, this class hierarchy is based on the notion of *inheritance*: if we have a superclass and a subclass in the hierarchy, then the subclass inherits all the fields and methods from the superclass

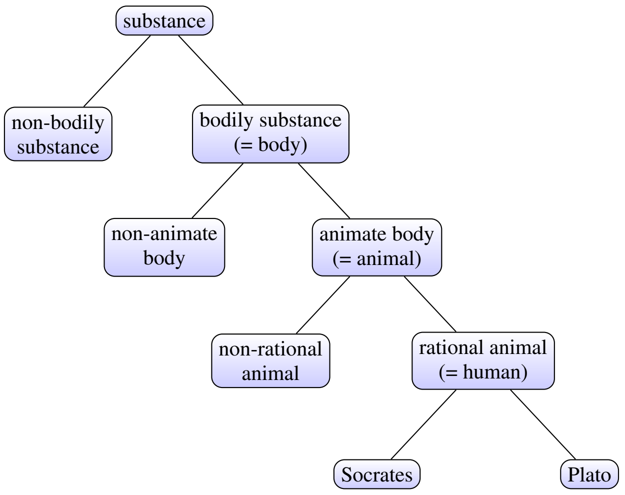

Fig. 1 Example of a Porphyrian tree.

(and will typically have some additional fields and methods). In the next two sections, I will focus on the precise nature of these hierarchies of Ideas and classes.

3 The Porphyrian Tree

Plato's theory of Ideas was further extended and refined by his student, Aristotle, leading to the latter's theory of the Categories (Studtmann 2013). In the 3rd century CE, the Neoplatonic philosopher Porphyry defended the view that Plato's and Aristotle's theories are essentially in agreement with each other (Emilsson 2015). His most influential work, the *Isagoge* (Greek for 'introduction') (Barnes 2003), explained Aristotle's theory of the Categories, making use of the doctrine that would become known as the 'Porphyrian tree' (*arbor Porphyriana*). A modern example of such a tree is shown in Fig. 1, and will be discussed below. The actual tree diagram is not present in the Isagoge, but was only developed in one of Boethius' commentaries on that work (5th–6th century CE), and later, by 13th-century logicians such as William of Sherwood and Peter of Spain (Verboon 2008, 2010, 2014). The Porphyrian tree was discussed and taught to logic students well into the 19th century (Blum 2012; Moktefi and Shin 2012). Similar diagrams are still used today in philosophical logic and metaphysics, but they have also found their way into disciplines such as formal ontology, linguistics and cognitive science (Chisholm 1989; Thomasson 2004; Hacking 2007; Kaczmarek 2002; Seuren and Jaspers 2014). Finally, as discussed in Franklin (1986) and Archibald (2014), Linnaeus's tree diagrams in biology are a concrete version of the Porphyrian tree, replacing the abstract Categories with the names of actual biological species.[1]

The Porphyrian tree, as shown in Fig. 1, is relatively straightforward to understand. It starts with the most fundamental Category, viz. substance. This is the highest genus, and it

[1] The historical development of the Porphyrian tree is surprisingly similar to that of another logical diagram, viz. the *square of opposition*. The theory embodied by this diagram was first expounded by Aristotle, but the actual diagram was only constructed by a later commentator (Apuleius, 2nd century CE). It, too, was highly popular among medieval logicians, and is nowadays still used in logic and philosophy, but also in various other disciplines, including linguistics and computer science (Parsons 2012; Smessaert and Demey 2014, 2015; Demey and Smessaert 2016, 2017).

can be divided into two species, viz. non-bodily substance and bodily substance (body). The latter is a species of substance, but it can also be regarded as a genus itself, and be divided into further species, viz. non-animate and animate body (animal). The latter species can again be regarded as a genus itself, and be divided into futher species, viz. non-rational animal and rational animal (human). The latter can only be regarded as a species (not as a genus to be divided into further species), and encompasses concrete human beings (i.e. concrete observable entities), such as Socrates and Plato.

When a medieval author such as Peter of Spain talked about "Porphyry's tree (*arbor Porphirii*)" (Copenhaver et al 2014, pp. 134–137), he was without a doubt using the word 'tree' as a visual metaphor, comparing the hierarchical organization of species and genera with the organic structure of a tree in nature, consisting of a trunk, branches and leaves (Verboon 2014). Nevertheless, I would like to propose that one can also look at the Porphyrian tree in the contemporary, mathematical sense of the word 'tree', viz. as a *connected acyclic graph* (Diestel 2006). In particular, from the acyclic nature of a tree, it follows that once two branches have diverged from each other, they can never come back together again (since that would constitute a cycle in the graph). In Porphyrian terminology: a genus can have multiple species under it, but a species cannot belong to multiple genera. Looking at Fig. 1, we immediately see that the Porphyrian tree is indeed a tree in this mathematical sense. The reason for this is that the different species falling under a given genus do not overlap with each other; for example, an entitity belonging to the genus *body* cannot simultaneously belong to the species *non-animate body* and *animate body*.

4 The Issue of Multiple Inheritance

We have just seen that in (Neo)platonic philosophy, the hierarchy of Ideas/Categories has the form of a tree, in the mathematical sense of the word. Given Tylman's (2017) original analogy between Plato and the OOP paradigm, it therefore seems reasonable to expect that the hierarchy of classes in an object-oriented program also has the form of a tree. In particular, this would mean that although a given class can have multiple subclasses inheriting from it, it is impossible for a given class to inherit from multiple superclasses.

In computer science, this is better known as the issue of *multiple inheritance*, which is highly contentious: "Multiple inheritance is a controversial part of object-oriented programming. [. . .] Unfortunately, multiple inheritance brings with it a set of problems that does not have simple, elegant solutions" (Mitchell 2004, p. 359). Consequently, most of the object-oriented programming languages mentioned in Sect. 2 do *not* support full-fledged multiple inheritance; for example: Java, C#, Object Pascal (Delphi), Ruby, Objective-C, PHP and Smalltalk.[2] For programs written in these languages, the hierarchy of classes is thus a tree (in the mathematical sense of the word), just like the Porphyrian tree of Ideas/Categories.

A specific version of the issue of multiple inheritance is a problem that has been called 'the fork-join problem' (Sakkinen 1989), 'the diamond problem' (Malayeri and Aldrich 2009), and even 'the diamond of doom', 'the diamond of dread' and 'the diamond of death' (by various users of the online programming platform Stack Overflow). This problem is illustrated by the graph in Fig. 2: class D simultaneously inherits from two superclasses,

[2] For reasons of space, I will not go into the details of these various programming languages. However, it should be mentioned that some of these languages do support some 'lightweight version' of multiple inheritance. For example, in Java, a class is *not* allowed to inherit from multiple superclasses, but it *is* allowed to implement multiple interfaces (Gosling et al 2015). Finally, there also exist languages that fully support multiple inheritance, such as C++, Python and Perl.

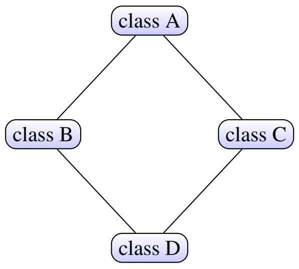

Fig. 2 The diamond problem in object-oriented programming.

viz. B and C, which in turn inherit from a common superclass, A. (This graph is thus not a tree, since it contains a cycle.) If the class A contains a method, its subclasses B and C will inherit it and can override it in potentially conflicting ways; however, since D is a subclass of both B and C, it is not at all clear which version of the method it should inherit (that from B or that from C), i.e. the *behavior* of (instances of) D is left fundamentally underdetermined. Similarly, if the class A contains a variable, its subclasses B and C will inherit it and can specify potentially conflicting default values for it; however, since D is a subclass of both B and C, it is not at all clear which default value of the variable it should inherit (that from B or that from C), i.e. the *state* of (instances of) D is left fundamentally underdetermined.[3]

This can be clarified by means of the following toy example. At a university, there are two types of people: students and employees. Furthermore, in the Belgian university system, a PhD student has a dual status: she is simultaneously a student and an employee. This can be modeled as in Fig. 2, with classes Person (A), Student (B), Employee (C) and PhDStudent (D). The Person class has a method celebrateBirthday(), which is inherited by the Student and Employee classes: the former implements this method by having the student drink beer with her friends at the dorm, whereas the latter implements it by having the employee bring cake for her colleagues at work. However, since PhDStudent is a subclass of both Student and Employee, it is not clear which implementation of celebrateBirthday() it should inherit. Similarly, the Person class has a variable numberOfEarlierDegrees, which is inherited by the Student and Employee classes: the former assigns value 0 to it by default, whereas the latter assigns 1 to it by default. However, since PhDStudent is a subclass of both Student and Employee, it is not clear which default value of numberOfEarlierDegrees it should inherit.

5 Limitatons of the Extended Analogy

We have just extended Tylman's original analogy between Plato's theory of Ideas and OOP, by showing that the (Neo)platonic doctrine of the Porphyrian tree corresponds to the fact that most object-oriented programming languages do not support multiple inheritance. Just like Tylman's original analogy, however, the extended analogy has some (quite subtle) limitations, which I will now discuss.

First of all, in the case of the Porphyrian tree of Ideas/Categories, a given concrete (observable, destructible) entity cannot belong to a genus without also belonging to some

[3] A similar issue arises in non-monotonic logic, where it is sometimes called the 'Nixon diamond' (Strasser and Antonelli 2016): consider the defeasible statements that Quakers are usually pacifists (cf. class B with one method) and that Republicans are usually not pacifists (cf. class C with a conflicting method). Given that Richard Nixon happens to be both a Quaker and a Republican (cf. class D inheriting from both B and C), it is not at all clear whether we should conclude that he is or that he is not a pacifist.

species of that genus. For example, one cannot belong to the genus animal without also belonging to either the species non-rational animal or the species rational animal. By contrast, in an object-oriented program, it is perfectly possible for an object to be an instance of a given class, without being an instance of any of its given subclasses. This shows that the notion of an object in OOP is still less 'concrete' than the notion of a destructible, observable entity in (Neo)platonic philosophy. (It should be noted that this also constitutes an objection to Tylman's original analogy.)

Secondly, there seems to be a *modal* difference between the acyclic nature of both hierarchies. In the case of the Porphyrian tree of Ideas/Categories, the occurrence of a 'diamond'-like cycle is *logically impossible*. After all, since a genus splits into two species that are incompatible with each other (e.g. bodily substance and non-bodily substance as species of the genus substance; cf. Fig. 1), these two species simply cannot have a joint subspecies (continuing the example: that subspecies would have to be simultaneously bodily and non-bodily in nature, which is logically impossible). By contrast, in the class hierarchy of an object-oriented program, two subclasses of a common superclass are not necessarily incompatible with each other (e.g. Student and Employee as subclasses of Person; cf. our toy example), and hence it is not *logically impossible* for those two classes to have a common subclass (although this will typically still cause other problems, as discussed in Sect. 4).

6 Conclusion

Tylman (2017) has argued that Plato's theory of Ideas is strikingly similar to the object-oriented programming paradigm. In this paper, I have explored this analogy in more detail, and shown that the tree-like (acyclic) nature of the Porphyrian tree from (Neo)platonic philosophy corresponds to the fact that most object-oriented programming languages do not support multiple inheritance. Although this extended analogy has some (subtle) limitations, it further reinforces Tylman's point regarding the conceptual continuity between classical metaphysical theorizing and contemporary computer science. As a computer scientist, Tylman (2017, p. 13) views this continuity as a reason "to seek inspiration for [. . .] computer science ideas in some less-than-obvious [i.e. philosophical] sources". As a philosopher, I would like to add that, *vice versa*, contemporary computer science can itself also be a fruitful source of new philosophical insights (Demey 2014). An increasing awareness of this mutually beneficial relation has led to the recent establishment of conference series such as History and Philosophy of Computing (HaPoC) and Computability in Europe (CiE).

Acknowledgements I would like to thank Hans Smessaert, Margaux Smets, Wouter Termont, Leander Vignero and two anonymous referees for their helpful feedback on an earlier version of this paper. The two issues discussed in Sect. 5 were first raised by Vignero and Termont, respectively. The ideas presented in this paper were first proposed in one of the additional theses of my PhD dissertation (Demey 2014). My research is financially supported by a Postdoctoral Fellowship of the Research Foundation–Flanders (FWO).

References

Archibald JD (2014) Aristotle's Ladder, Darwin's Tree. The Evolution of Visual Metaphors for Biological Order. Columbia University Press, New York, NY
Barnes J (ed) (2003) Porphyry's Introduction. Translation and Commentary. Oxford University Press, Oxford
Blum PR (2012) Studies on Early Modern Aristotelianism. Brill, Leiden
Chisholm RM (1989) On Metaphysics. University of Minnesota Press, Minneapolis, MN

Copenhaver BP, Normore CG, Parsons T (eds) (2014) Peter of Spain, Summaries of Logic. Text, Translation, Introduction and Notes. Oxford University Press, Oxford

Demey L (2014) Believing in logic and philosophy. PhD thesis, KU Leuven

Demey L, Smessaert H (2016) Metalogical decorations of logical diagrams. Logica Universalis 10:233–292

Demey L, Smessaert H (2017) Combinatorial bitstring semantics for arbitrary logical fragments. Journal of Philosophical Logic

Diestel R (2006) Graph Theory. Springer, Heidelberg

Emilsson E (2015) Porphyry. In: Zalta E (ed) Stanford Encyclopedia of Philosophy, CSLI, Stanford, CA

Franklin J (1986) Aristotle on species variation. Philosophy 61:245–252

Gosling J, Joy B, Steele G, Bracha G, Buckley A (2015) The Java® Language Specification. Java SE 8 Edition. Oracle America, Redwood City, CA

Hacking I (2007) Trees of logic, trees of Porphyry. In: Heilbron JL (ed) Advancements of Learning: Essays in Honour of Paolo Rossi, Olschki, Florence, pp 221–263

Joque J (2016) The invention of the object: Object orientation and the philosophical development of programming languages. Philosophy & Technology 29:335–356

Kaczmarek J (2002) On the Porphyrian tree structure and an operation of determination. Bulletin of the Section of Logic 31:37–46

Malayeri D, Aldrich J (2009) CZ: Multiple inheritance without diamonds. In: OOPSLA '09, ACM, New York, NY, pp 21–40

Martini S (2016a) Several types of types in programming languages. In: Gadducci F, Tavosanis M (eds) History and Philosophy of Computing (HaPoC 2015), Springer, Berlin, pp 216–227

Martini S (2016b) Types in programming languages, between modelling, abstraction, and correctness. In: Beckmann A, Bienvenu L, Jonoska N (eds) Pursuit of the Universal (Computing in Europe 2016), Springer, Berlin, pp 164–169

Mitchell JC (2004) Concepts in Programming Languages. Cambridge University Press, Cambridge

Moktefi A, Shin SJ (2012) A history of logic diagrams. In: Gabbay DM, Pelletier FJ, Woods J (eds) Handbook of the History of Logic. Volume 11. Logic: A History of its Central Concepts, North-Holland, Amsterdam, pp 611–682

Parsons T (2012) The traditional square of opposition. In: Zalta E (ed) Stanford Encyclopedia of Philosophy, CSLI, Stanford, CA

Sakkinen M (1989) Disciplined inheritance. In: Cook S (ed) ECOOP '89, Cambridge University Press, Cambridge, pp 39–56

Seuren P, Jaspers D (2014) Logico-cognitive structure in the lexicon. Language 90:607–643

Smessaert H, Demey L (2014) Logical geometries and information in the square of opposition. Journal of Logic, Language and Information 23:527–565

Smessaert H, Demey L (2015) Béziau's contributions to the logical geometry of modalities and quantifiers. In: Koslow A, Buchsbaum A (eds) The Road to Universal Logic, Springer, Basel, pp 475–494

Strasser C, Antonelli GA (2016) Non-monotonic logic. In: Zalta E (ed) Stanford Encyclopedia of Philosophy, CSLI, Stanford, CA

Studtmann P (2013) Aristotle's Categories. In: Zalta E (ed) Stanford Encyclopedia of Philosophy, CSLI, Stanford, CA

Thomasson AL (2004) Methods of categorization. In: Varzi AC, Vieu L (eds) Formal Ontology in Information Systems, IOS Press, Amsterdam, pp 3–16

Tylman W (2017) Computer science and philosophy: Did Plato foresee object-oriented programming? Foundations of Science (DOI 10.1007/s10699-016-9506-7)

van Benthem J (2007) Logic in philosophy. In: Jacquette D (ed) Philosophy of Logic, North-Holland, Amsterdam, pp 65–99

Verboon AR (2008) Einen alten Baum verpflanzt man nicht. Die Metapher des Porphyrianischen Baums im Mittelalter. In: Reichle I, Siegel S, Spelten A (eds) Visuelle Modelle, Fink, Munich, pp 251–268

Verboon AR (2010) Lines of thought: Diagrammatic representation and the scientific texts of the Arts faculty, 1200–1500. PhD thesis, Leiden University

Verboon AR (2014) The medieval tree of Porphyry: An organic structure of logic. In: Salonius P, Worm A (eds) The Tree: Symbol, Allegory, and Mnemonic Device in Medieval Art and Thought, Brepols, Turnhout, pp 95–116