WHY ARE SOFTWARE PATENTS SO ELUSIVE? A PLATONIC APPROACH

by

ODIN KROEGER*

Software patents are commonly criticised for being fuzzy, context-sensitive, and often granted for trivial inventions. More often than not, these shortcomings are said to be caused by the abstract nature of software—with little further analysis offered. Drawing on Plato's Parmenides, *this paper will argue (1) that the reason why software patents seem to be elusive is that patent law suggests to think about algorithms as paradigmatic examples and (2) that Plato's distinction between two modes of predication and the role of competence in his account of knowledge are helpful not only for conceptualising knowledge of algorithms, but also for understanding the limits of software patent regimes.*

KEYWORDS

Software patents, patent failure, computer-implemented inventions, algorithms, reification, Plato, Bessen, Meurer

1. INTRODUCTION

Judging from the software patent debate, algorithms seem to be an elusive matter. (1) James Bessen and Michael Meurer (2007), for instance, hold that 'software patents have especially severe boundary problems', because software patent claims employ 'functional language', which is abstract and therefore imprecise; what is more, they add (2008, p. 195) that software patent claims do not use 'abstract terms' by coincidence, but rather because 'many of the standard terms of art [in software engineering] are themselves abstract ideas'. (2) Sherly Abraham (2009, sec. 6), correspondingly, argues that software patents trouble patent offices and courts alike because al-

* odin.kroeger@univie.ac.at

gorithms, which is what software consists of, may perform different functions in different contexts.[1] (3) Jan Bergstra and Paul Klint (2007, sec. 8.1), by contrast, note that if the way in which algorithms are applied is not taken into account, what is left may be too insignificant to qualify for patent protection; and indeed, many—Hartmut Pilch (2004, p. 290) even claims 'most'—software patents appear to be trivial. More often than not, these difficulties are described as a consequence of the nature of algorithms: (1) algorithms are abstract, thus their boundaries are fuzzy; (2) algorithms are underdetermined, thus their function is context-dependent; (3) algorithms express logical truths, thus they tend to be trivial. Put in a nutshell, we are often told that algorithms are less than ideal candidates for patent protection because they are ideal objects.

Andrew Chin (2009, p. 200), however, points out two problems with this explanation: (1) formulating patent claims is 'an exercise in abstraction' at any rate, so that 'it is not immediately clear why the abstract nature of software should pose a special problem for the determination of patent scope'; (2) computer scientists and engineers seem to be able to communicate precisely about algorithms, regardless of their abstract nature. That is not to say that software patents work, the evidence gathered by Bessen and Meurer (2008, sec. 9.1.1) establishes beyond doubt that they do not, but we need a more complex explanation why algorithms seem to turn fuzzy, context-sensitive, and trivial as soon as they are the subject of a patent law suit. That being so, the purpose of this paper is to propose such an explanation; starting from the assumption that the characteristics displayed by ideal objects, including algorithms, in the context of patent litigation depend, among other factors, on the model of ideal objects that patent examiners and judges employ.[2] To be precise, in this paper 'model of ideal objects' refers to a set of—possibly implicit—assumptions about: (1) the functions of ideal objects, (2) the way in which they perform these functions, and (3) their ontological status in connection therewith.

So what model of ideal objects informs patent law? Rudolph Peritz (2008, p. 249) argues that the US patent regime 'seems to require a separation of inventions from ideas', Jeffrey Lefstin (2008, note 210) makes the

[1] Most of the literature on (1) and (2) focuses on the US, but should apply mutatis mutandis to other jurisdictions.

[2] This approach is inspired by Thomas Powers (2005), who argues that the 'meandering of case law' (p. 99) regarding copyright is a consequence of judges holding different views about the nature of ideal objects.

similar observation that the US Patent Office appears 'to have committed itself to a form of metaphysical realism', and Ben Klemens (2008, pp. 6–7) notes that algorithms in particular are often maintained to enjoy some kind of independent existence. Put differently, patents seem to come bundled with Platonism. More precisely, they seem to come bundled with Plato's theory of forms, which traditionally (see, e.g., Vlastos 1954) is taken to imply that: (P_1) ideal objects function as standards that are used to determine whether real objects possess certain predicates (e.g., there is an ideal object 'redness', to which we compare real objects in order to determine whether they are red); (P_2) they perform that function by being paradigmatic instances of the predicates the usage of which they regulate (e.g., 'redness' is the paradigmatic red); (P_3) in order for them to be able to perform that function, they need to be thought of as existing independently of real objects (e.g., 'redness' exists besides red things, rather than inhering in them). Following Peritz, Lefstin, and Klemens, it seems safe to presume that patent examiners and judges alike draw on a model of ideal objects that is congruent with the traditional reading of Plato's theory of forms.

Thus, the objective of this article can now be stated more precisely as follows: to assess whether the fact that algorithms tend to appear as fuzzy, context-sensitive, and trivial in the context of patent litigation can be explained in terms of the theory of forms. This will be done by drawing on Plato's *Parmenides* (1997), in which he not only discusses some weaknesses of that theory, but also how these weaknesses could be addressed and why we are inclined to think of ideal objects along the lines of the weak version of that theory. Having said that, there is no scholarly consensus on how the *Parmenides* should be interpreted; the reading employed here draws, among others, on those of Constance Meinwald (1991; 1992) and Wolfgang Wieland (1999). Since Plato's arguments need to be brought to bear on an example, the algorithm 'Ron's Code 4' (RC4) will be outlined in the next section (sec. 2). To this algorithm the so-called 'Third Man Argument', which—for the purposes of this paper—is the most interesting criticism Plato brings forward against his theory in the *Parmenides* (p. 132a–b), will then be applied (sec. 3). After that, the reasons that he identifies for which people tend to make assumptions about ideal objects that correspond to the weak version of the theory of forms (sec. 4) and the modifications that he proposes to rectify this theory will be related to the patent system (sec. 5). To conclude,

the implications of these findings for current software patent regimes will be discussed (sec. 6). With reference to James Bessen and Robert Hunt (2007, p. 167), the term 'software patents' is used in this paper to refer to all patents that encompass 'a logic algorithm for processing data that is implemented via stored instructions'; notably, this includes patents on computer-implemented inventions.

2. RC4

RC4 is a cipher that encrypts a given plaintext by combining it with pseudo-random numbers that are generated from a given key, it was developed by Ronald Rivest in 1987 for RSA Laboratories (see Mantin 2001, sec. 1.2). What makes RC4 such an apt example for the purposes of this article is that it can be understood without too much prior knowledge in computer science or mathematics—in contrast to more recent ciphers such as RC5 or RC6. What may seem to render RC4 a somewhat odd example, however, is that it has not been patented; but, in all likelihood, this is only because RC4 was leaked to the general public in 1994, that is, before algorithms were established as patentable subject matter in the US by the decision in *State Street Bank and Trust v. Signature Financial Group*, 149 F.3d 1368 (1998). RSA Laboratories has applied for and been granted patents for RC5 (US patent no. 5,724,428) as well as RC6 (US patent no. 5,835,600), the successors of RC4.

How does RC4 work? Similar to many other ciphers, RC4 comprises two components: a key scheduling algorithm (KSA), which initialises a pseudo-random generation algorithm (PRGA). First, the KSA creates an ordered list that ranges over all possible byte values, so that every value occurs once and only once (i.e., it counts from 0 to 255). Then, it swaps the elements of that list, where the elements that are to be swapped with each other are determined by the key provided. This list is called the 'substitution box'. Using that substitution box, the PRGA first swaps two elements of that box in order to ensure that its output remains pseudo-random even for large amounts of data and then selects another element of the box as pseudo-random byte. Different to the KSA, the PRGA uses only the current state of the substitution box to determine which elements are to be swapped. With the PRGA initialised by the KSA, the plaintext is encrypted by xoring each of its bytes with one pseudo-random byte generated by the PRGA (for more details see Thayer & Kaukonen 1999).

To illustrate, the following code, written in the programming language *Python*, implements RC4's KSA (called '__init__' in the example) and PRGA ('prga'):

```
class RC4:
        S = range(256)
        i = 0
        j = 0

        def __init__(self, key):
          j = 0

          for i in range(256):
             j = (j + self.S[i] + key[i % len(key)]) % 256
             self.S[i], self.S[j] = self.S[j], self.S[i]

        def prga(self):
          self.i = (self.i + 1) % 256
          self.j = (self.j + self.S[self.i]) % 256

          self.S[self.i], self.S[self.j] = (self.S[self.j],
            self.S[self.i])

          return self.S[(self.S[self.i] + self.S[self.j]) % 256]
```

Listing 1: RC4 in *Python*

With this description of RC4 in mind, we may now turn to the later Plato's analysis of the weaknesses of his middle-period theory of forms and see how this analysis applies to that algorithm.

3. ALGORITHMS AND THE THIRD MAN ARGUMENT

Plato's Third Man Argument (TMA) discusses precisely the model of ideal objects that has been argued to be employed by patent examiners and judges. What the TMA, with reference to Andreas Graeser (2003), is meant to show is that a model of ideal objects that includes the assumptions (P_1), (P_2), and (P_3) is inapt to formulate definite description of any ideal object (such a model will hereafter be referred to as 'TMA-model' for short). Gregory Vlastos (1954) and S. Marc Cohen (1971), among others, have shown that the TMA can be proven for more formalised versions of (P_1), (P_2), and (P_3), but for the purposes of this paper a simplified account of the TMA will do: (1) Suppose, there are two red flowers. (2) Since both of them are red, there must be some kind of form they have in common, namely red-

ness. (3) Since redness is not particular to any single thing, redness must exist independently of those things. (4) Since, according to the TMA-model, forms are thought of as paradigmatic instances, redness is the paradigmatic red. (5) But might we then not just as well ask what those two flowers and the ideal object 'redness' have got in common, that is, by virtue of what the red flowers and redness are related? How does redness, when looked at with 'the mind's eye' (*Parm.*, p. 132a), differ from the two red flowers? To sum up, according to the TMA-model, ideal objects on the one hand function as standards that inform our judgements about real objects, but on the other hand are understood to be so similar to real objects that one could rightly ask why ideal objects should be any better in performing that function than real ones. What the TMA shows is that this question cannot be answered on the basis of (P_1), (P_2), and (P_3).

To illustrate, let us go through the TMA using RC4 rather than red as our example. First, have a look at the following code, written in the programming language *Perl*, which also implements RC4's KSA (called 'new' in this example) and PRGA ('prga'):

```
package RC4;

sub new {
   my ($class, $self, $j) = (shift, {S => [0 .. 255]});

   for(my $i=0; $i<256; $i++) {
        $j = ($j + $self->{S}->[$i] + $_[$i % @_]) % 256;
        ($self->{S}->[$i], $self->{S}->[$j]) =
        ($self->{S}->[$j], $self->{S}->[$i]);
   }

   bless($self, ref($class) || $class); return $self;
}

sub prga {
   my $self = shift;
   $self->{i} = ($self->{i} + 1) % 256;
   $self->{j} = ($self->{j} + $self->{S}->[$self->{i}]) % 256;

   ($self->{S}->[$self->{i}], $self->{S}->[$self->{j}]) =
       ($self->{S}->[$self->{j}], $self->{S}->[$self->{i}]);

   return $self->{S}->[($self->{S}->[$self->{i}] +
       $self->{S}->[$self->{j}]) % 256];
}
```

Listing 2: RC4 in *Perl*

Although the earlier piece of code (listing 1) and this one (listing 2) are different, any person skilled in the art, in this case, a programmer who knows *Python* as well as *Perl*, will be able to tell that both of them implement the same algorithm. Put differently, they share the same form. So how can this form be spelled out? Usually, if the logical structure of an algorithm shall be illustrated, programmers use so-called 'pseudocode', that is, a description of the algorithm in natural language that mimics the structure of programming languages. Using such pseudocode, RC4's KSA and PRGA can be presented as follows:

```
class RC4
   S := { 0, ..., 255 }
   i := 0
   j := 0

   method ksa(key)
        j := 0

        for i := 0 to i = 255 do
           j := (j + self.S[i] + key[i mod length(key)]) mod 256
           swap i with j in self.S

        method prga
           self.i := (self.i + 1) mod 256
           self.j := (self.j + self.S[self.i]) mod 256

           swap self.i with self.j in self.S

           return self.S[(self.S[self.i] + self.S[self.j]) mod 256]
```

Listing 3: RC4 in pseudocode

We may be tempted to say that any code that implements the structure described above is an instance of RC4, but then we would have to explain the difference between the earlier examples (listing 1 and 2) and the pseudocode above (listing 3) that renders the latter but not the former able to function as a standard—and that turns out to be difficult. For if somebody asked 'what is the common logical structure of the pieces of code depicted in listings 1, 2, *and 3*?', then that question would appear to be no less justified than the same question asked only for listings 1 and 2. That is, listing 3 fails to answer that question in a satisfactory manner, but this is what we would expect from a standard. Moreover, if we tried to answer what the listings 1,

2, and 3 have in common, all we would end up with is yet another description of RC4—for which yet another question of the same kind could be posed. We could continue like this indefinitely, without ever arriving at a description that could be used as standard of what counts as instance of RC4.

How does this relate to software patent litigation? To answer this question we need to review two interpretations of the TMA: Wieland (1999, pp. 118–124) and Graeser (2003, sec. 4) agree that the regress just outlined obtains because the TMA-model reifies ideas, which means in this case that ideas are treated as objects, that is, as bearers of properties, that exist on their own. Wieland argues that such a treatment implies that ideas (1) tend to be removed from the contexts of their application, and that (2) knowledge of them is thought of as propositional, that is, as being structured similarly to beliefs, which often can be expressed as a relation between an object and a predicate (more on this in sec. 5).

This allows to account for the fact that algorithms tend to appear as (1) fuzzy, (2) context-sensitive, and (3) at times trivial to patent examiners and judges as a consequence of them relying on the TMA-model. (1) What the TMA shows first and foremost is that descriptions of ideas as paradigmatic instances need to be contextualised in order to be able to guide our judgements, that is, they do so to a larger extent than ordinary descriptions used in everyday life. Put simply, they seem to be fuzzy. For the same reason, (2) attempts to describe an idea as paradigmatic instance not only fail to include the idea's context of application, so that the resulting ideal objects will be applicable to a wide range of contexts, but also (3) tend to be unable to capture the inventive step needed to come up with a particular algorithm. Take, for instance, the description of RC4 given by listing 3. To be sure, there is nothing outstanding about any single instruction in that listing, similar instructions can be found in many other algorithms, yet that does not imply that RC4 is trivial. Quite to the contrary, the actual innovation of RC4 is that such a simple algorithm can encrypt data in a secure manner, but this is nothing that can be seen from listing 3. Metaphorically speaking, if we are looking for an innovation in such descriptions, then our 'mind's eye' is looking in the wrong direction.[3]

[3] That is not to say that there are no trivial software patents, but that the impression that a significant number of them is trivial might, at least in part, be caused by patent examiners, judges, and others drawing on the TMA-model of ideal objects.

Having said that, some may wonder why patent examiners and judges would rely on such a dysfunctional model of ideal objects in the first place. Plato does not address this explicitly, but some of his arguments provide helpful clues, to which we will turn now.

4. WHY PATENT EXAMINERS AND JUDGES MAY BE PLATONISTS

Plato, in spite of having shown that reifying ideas is fallacious, insists that the assumptions that lead to that reification perform important functions, so that they are difficult to avoid. To be precise, Plato advances two arguments in the *Parmenides* that may explain why patent examiners and judges tend to rely on the TMA-model: (1) We could not engage in any serious discussion, if we did not have some kind of idea of the subject matter at hand, that is, if we did not have a standard that allows us to assess each other's arguments as true or false, but in order to function as such a standard ideas must not be subject to our whim and be shared by all participants of the discussion—both of which suggests them to enjoy a certain degree of independence from us (cf. p. 135a–c). (2) Without forms, that is, if we did not assume them to somehow exist, statements about immaterial objects such as 'Pegasus has wings' would be meaningless, for there is no such thing as a real horse with wings that 'Pegasus' could refer to, yet such statements clearly are meaningful. If they were not, we would not even be able to deny that Pegasus exists, for in order to assert that nothing in the (material) world is Pegasus, we need to know what 'Pegasus' refers to (cf. pp. 160b–164b). Plato finds these two arguments to establish beyond doubt that we need to assume that there *are* forms, regardless of the difficulties that seem to follow from this assumption.

Wieland (1999, pp. 118–124) construes this result to imply that the way in which we use ideas differs depending on whether we draw on them to assess an argument or whether we make them the subject of an argument, to wit, that doing the latter may compel us to think about that idea as some kind of object, causing the regress described by the TMA; this is illustrated by the following two sentences:

(S_1) These roses are red.

(S_2) Redness is a colour.

Whereas 'red' in (S_1) functions as predicate, so that we will tend to think of red as a property that some real objects happen to possess, 'redness' in (S_2) functions as subject, to the effect that we seem to treat redness as if it were a proper object. What is more, expressing the content of (S_2) without reifying the property '… is red' to an ideal object 'redness' is tricky and the result would sound fairly contrived, at least in Indo-European languages. To put this differently, which assumptions about ideas (or ideal objects respectively) are functional depends on the role an idea plays in our judgements, that is, on whether that idea informs a judgement about something else or is itself the subject matter about which a judgement is made.

We can now see why patent examiners and judges should find the TMA-model appealing: the ideas embodied by patents are the subject matter not the standard of their judgements, at least, insofar as these judgements concern patents. For instance, how could one apply the doctrine of equivalents without assuming, at least implicitly, that there is some kind of ideal object that patent claims or inventions embody and that functions as a yardstick to which the software accused of infringing on a patent can be compared? What is more, for such a comparison to make sense, there must be some kind of correspondence between the standard that is used to assess an object and the object that is assessed, and intellectual property law in general seems to suggest that this correspondence can be thought of as one that holds between a pattern and its exemplifications. Put bluntly, being a Platonist is part of the job description of patent examiners and judges.

Graeser (2003, sec. 4), however, points out that the metaphysics of the TMA-model is at odds with some of Plato's other writings and regards this as evidence that Plato intends the TMA to show that any reading of his theory of forms that reifies ideas is misguided. So patent examiners and judges may be able to be Platonists without subscribing to the TMA-model after all. To assess this we will now review how Plato addresses the TMA.

5. RECTIFYING PLATO'S MODEL OF IDEAL OBJECTS

Plato, in the light of the TMA and other similar difficulties, (1) explores an amendment to his theory of forms and (2) clarifies the functions that forms can perform in different contexts, both of which are interesting avenues for a possible reform of software patents.

(1) Meinwald (1991; 1992) construes the second part of the *Parmenides* (pp. 135a–166c) to introduce a distinction between two modes of predica-

tion: predications *pros ta alla* (in relation to others) are ordinary predications (e.g., 'this flower is red' or 'RC4 was invented by Ronald Rivest'), they express a relation between an object (e.g., the flower or RC4) and something else (e.g., the colour red or Ronald Rivest); predications *pros heauto* (in relation to itself), by contrast, express the relation between natures of concepts (e.g., 'redness is a colour', 'RC4 is a cipher'), similar to a genus-species tree (see illustration 1). Thus, Meinwald holds that the TMA obtains because the theory of forms put forward by the middle-period Plato fails to distinguish between these two modes of predication; in other words, descriptions of ideal objects along the lines of the TMA-model are fuzzy because they abstract from real objects, instead of focusing on the relation between an ideal object and other ideal objects that occupy a higher position in its genus-species tree.

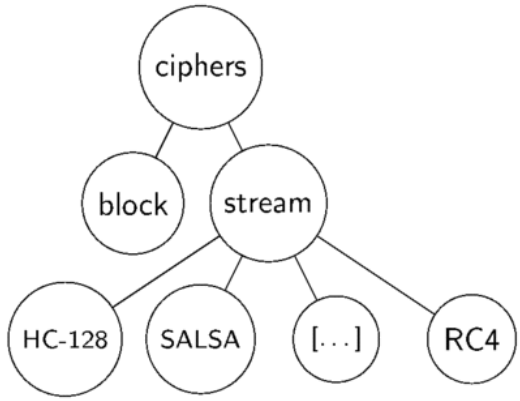

Illustration 1: Genus species tree for RC4

(2) Wieland (1999) claims that Plato emphasises throughout all of his writings that knowledge (which for Plato is always the knowledge of forms) has two sides: on the one hand, knowing something about *x* implies having a belief about *x*, that is, knowledge is structured similar to propositions (know-that); on the other hand, knowing *x* implies being capable of making judgements about *x*, that is, knowledge also requires a certain kind of competence (know-how). Thus, knowledge cannot be reduced to being aware of the truth of certain propositions, but also requires that one is competent in how these propositions relate to different contexts. Wieland (pp. 118–124),

therefore, argues that the TMA ensues from employing the principles of the theory of forms in the wrong way; there is no meaningful answer to the question what two red flowers and the idea of redness must have in common in order for the two flowers to count as exemplifying redness, because this confuses the normative and the descriptive function of ideas.

Neither of these two proposals, however, can be readily applied to software patent litigation. (1) Of course, appreciating the *pros ta alla/pros heato* distinction allows for more accurate descriptions of ideal objects, but doing so requires some knowledge about the nature of the ideal object described, and that knowledge may simply be unavailable. For example, the relation between a key and the stream of pseudo-random numbers that RC4 generates from that key is part of the nature of RC4, that is, the way in which RC4 operates necessitates this relation, yet the fact that this stream of pseudo-random numbers can easily be distinguished from truly random numbers had not been realised until eight years after RC4 had been leaked to the general public (see Mantin & Shamir 2002). Put simply, algorithms are a complex subject matter, so that understanding their nature will require a fair amount of research, but that research takes time and can begin only after the algorithm has been published—and the patent claim already been filed.

(2) Making the 'right' use of ideas is easier said than done, especially if you are a patent examiner or judge and giving grounds for which some piece of software should or should not count as infringing on a patent claim counts as 'wrong'. Plato, at least according to Wieland, would argue that such a giving of grounds must come to an end, and that people who truly *know* the idea under consideration *must* at that point accept good grounds simply by virtue of being competent to make good judgements regarding that idea. To put this differently, Plato would argue that patent examiners as well as judges (and lawyers for that matter) should refrain from discussing algorithms, unless they are competent to make good judgements about their nature. That, of course, is quite high a demand to make, after all, most judges and lawyers, are not computer scientists or mathematicians; put bluntly, this is *not* part of their job description.

6. CONCLUSION

To sum up, it was shown above that algorithms appear as fuzzy, context-sensitive, and trivial if they are described in accordance with the TMA-model of ideal objects (sec. 3) and that there are good reasons to suppose

that patent examiners and judges rely on that very model (sec. 4). Plato's proposals for avoiding the TMA-model, by contrast, were found to be difficult to apply to patent litigation (sec. 5). To be sure, refining the principles of patent law, for instance, the doctrine of equivalents, in accordance with the *pros ta alla/pros heauto* distinction or setting up specialist courts for software patents may improve upon the status quo, but the way in which the patent system works limits the prospects of those measures. Ultimately, Samuel Beckett's famous advice may be all that is in store for any attempt to reform the software patent regime: 'Ever tried. Ever failed. No matter. Try again. Fail again. Fail better.'

REFERENCES

[1] Abraham, S. E., 2009. Software patents in the United States: A balanced approach. *Computer Law & Security Review*, 25(6), pp. 554–562.

[2] Bergstra, J. A. & Klint, P., 2007. About 'trivial' software patents: The IsNot case. *Science of Computer Programming*, 64(3), pp. 264–285.

[3] Bessen, J. & Hunt, R. M., 2007. An empirical look at software patents. *Journal of Economics & Management Strategy*, 16(1), pp. 157–189.

[4] Bessen, J. & Meurer, M. J., 2008. *Patent failure: How judges, bureaucrats, and lawyers put innovators at risk*, Princeton, NJ: Princeton University Press.

[5] Bessen, J. & Meurer, M. J., 2007. What's wrong with the patent system? Fuzzy boundaries and the patent tax. *First Monday*, 12(6). Available at: http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1867.

[6] Chin, A., 2009. On abstraction and equivalence in software patent doctrine: A response to Bessen, Meurer and Klemens. *Journal of Intellectual Property Law*, 16(2), pp. 197–240.

[7] Cohen, S. M., 1971. The Logic of the Third Man. *The Philosophical Review*, 80(4), pp. 448–475.

[8] Graeser, A., 2003. *Platons Parmenides*, Stuttgart: F. Steiner.

[9] Klemens, B., 2008. The rise of the information processing patent. *Boston University Journal of Science and Technology Law*, 14, pp. 1–38.

[10] Lefstin, J. A., 2008. The formal structure of patent law and the limits of enablement. *Berkeley Technology Law Journal*, 23, pp. 1141–1225.

[11] Mantin, I., 2001. *Analysis of the stream cipher RC4*. Master thesis. Rehovot, Israel: Weizmann Institute of Science.

[12] Mantin, I. & Shamir, A., 2002. A practical attack on broadcast RC4. *Lecture Notes in Computer Science*, 2355, pp. 87–104.

[13] Meinwald, C. C., 1992. Good-bye to the Third Man. In R. Kraut, ed. *The Cambridge Companion to Plato*. Cambridge: Cambridge University Press, pp. 365–396.

[14] Meinwald, C. C., 1991. *Plato's Parmenides*, New York, NY: Oxford University Press.

[15] Peritz, R. J. R., 2008. Freedom to experiment: Toward a concept of inventor welfare. *Journal of the Patent and Trademark Office Society*, 90, pp. 245–267.

[16] Pilch, H., 2004. Why are software patents so trivial? In *Patents, innovation and economic performance*. Paris: OECD, pp. 289–294.

[17] Plato, 1997. Parmenides. In J. M. Cooper & D. S. Hutchinson, eds. *Complete Works*. Indianapolis, IN: Hackett, pp. 359–397.

[18] Powers, T. M., 2005. Ideas, Expressions, Universals, and Particulars: Metaphysics in the Realm of Software Copyright Law. In R. Spinello & H. T. Tavani, eds. *Intellectual Property Rights in a Networked World: Theory and Practice*. Hershey, PA: Information Science Publishing, pp. 99–111.

[19] Thayer, R. & Kaukonen, K., 1999. A Stream Cipher Encryption Algorithm "Arcfour". *IETF*. Available at: http://tools.ietf.org/id/draft-kaukonen-cipher-arcfour-03.txt [Accessed October 22, 2010].

[20] Vlastos, G., 1954. The Third Man Argument in the Parmenides. *The Philosophical Review*, 63(3), pp. 319–349.

[21] Wieland, W., 1999. *Platon und die Formen des Wissens* 2nd ed., Göttingen: Vandenhoeck & Ruprecht.