Presented at a meeting of the
Collegiate Institute for Values and
Science, University of Michigan,
December 9, 1981

OWNERSHIP OF COMPUTER PROGRAMS

by John N. Snapper

For the health of the computer software industry, then must be some way to assure to the developers of computer progress; any profits to be node from the sale or rent of those programs [2]. Since versatile programs can be modified to fit the needs of various clients at considerably lass expense than it takes for those clients to independently develop those programs, there is a large potential market for programs copied front those already 1n use. Though reprehensible, it is common practice to simply copy valuable programs. Anyone 1n the business will aloft that he is frequently offered pirated programs. Depending on the extent of those protections. to protect programs is to recognize that they are property. [3]

Although there is a need for protection, there are good philosophical reasons for refusing protection. We do not wish to prevent groups or individuals from carrying out research by any scientific or Intellectual neaps. I have heard it claimed (this is to fact historically inaccurate) that Newton kept his discovery of the infinitesimal calculus to himself in order to prevent competitors from making scientific discoveries with the ease that he could with his new mathematical techniques. [4] This (if true) is reprehensible. It holds back scientific growth. Morse, legal recognition of Newton's right to the new techniques would suggest an intuitively impermissible legal category of criminal thought. Imagine the prosecution telling a defendant that he could not think along certain lines reserved to Newton. We certainly do not wish to glue legal support to such foolishness. The problem with granting property rights over software is that what is essential to new programs often appears to be simply ways to think through intellectual problems. Thus, certain principles of free access to means of working out problems preclude the treatment of software as property.

A central aim of legal protections for software must be to encourage research. This has conflicting consequences for the treatment of programs as property. On the one hand, restrictions on software that reward profits to developers will encourage expenditure on research and development. On the other hand, restrictions on the use of recent discoveries my prevent other researchers from using those discoveries In further work. This is the familiar battleground for those who view the issue as utilitarians. I have little to say on this except that we should reach a balance and not (as some industry representatives my wish) push too much for private control over new discoveries. I am more interested in philosophical intuitions which must underlie any treatment of programs as property. These too conflict. On the one hand, we can argue with Locke that the labor that goes Into program development creates property rights. Although there are great differences between software discussed here and real estate discussed by Locke. there is some intuition that those who do research deserve recognition in the form of property protection. 5 on the other hand, I insist that nothing be done to restrict free access to any mode of thought or argumentation. That principle overrides all else in this discussion.

I personally view free thought as a right, such as the rights to privacy, equal treatment under the law, etc., against which particular laws are tested on the highest level of legal consideration. That view. however, has no legal history. As we shall see below, the courts have been able to reject protections for software that tend to restrict free thought by appeal to finer principles within, for Instance, patent law. The courts do not discuss

proprietary protections for software in the context of constitutional rights. There is no direct reference in the First Amendment, nor anywhere else in the constitution, to 'free thought'.

I do, all the same, believe that such a notion is intrinsic to our constitutional rights. Like privacy, which is also never explicitly mentioned, it is within the 'penumbra' of rights guaranteed by the Constitution. This view requires considerable argument, not directly related to the present topic. For present purposes. It is not necessary to treat free thought as a basic right. It is enough to recognize a social policy favoring it, without seeking a basis for the principle of free thought in the First Amendment. [6] This would not affect my key points. In any event, it is a principle of free thought, viewed as a basic right or a social policy, which gives the issue of software property Its special philosophical Interest. If programs are essentially algorithms for solving problem, they must be free to anyone who wishes to think through problem by those algorithm. It appears that we must either give up free thought, or re-define computer programs so that they are not essentially algorithms, or give up the attempt to treat them as property.

The three part dilemma is familiar in a confused mush-mash of statutory and common law governing copyrights, trade secrets, patents, and related matters that is called the law of 'Intellectual property'. The problem has been institutionalized in patent law in the somewhat misleading statement that one cannot patent 'natural laws or mathematical formulas'. Until recently, this was believed to preclude patents on computer software. But in decisions culminating in Diamond v Diehr in the spring of '81, the Supreme court has turned away from that tradition. That disputed (5-4) and complex decision has inspired lengthy disputes. Recent revisions in copyright law also leave unclear how programs may be protected by copyright. There is also a common law tradition that protects programs held as trade secrets. It is not surprising that the applications of trade secret law are better determined by the common law tradition than the alternatives are determined by statute. And, we may note certain other protections for programs, through, for instance, contracts governing the sale of programs. All in all though. I think, the legal context for a discussion of the proprietary status of software is best described as muddy.

The notion of a trade secret exemplifies exactly what, in my opinion, is improper and should be avoided in software protection. There Is, of course, no need for legal protection for programs so long as they are secret, since obviously only those privy to the secret are capable of taking economic advantage of it. There is, however, the possibility that secret information becomes public despite careful security measures. This happens when programs are taken by employees who change ,lobs, or are discovered by reverse engineering on leased programs, or are copied without authorization. Under these circumstances, both civil and criminal complaints can be brought against those who leak the secrets and those who acquire them. The law which is the basis for those complaints does indeed establish that programs are the property of those who attempt to keep them secret. And I object to that law. Let me be clear that I do not formally object to the attempt to keep secrets. Although I might encourage publication, I would not require the discoverers of natural laws and mathematical formulas to divulge their discoveries. Although I may fault a Newton for keeping the Infinitesimal calculus to himself, I do not

think that the state can force him to make it public. But trade secret law goes to the opposite extreme of encouraging secrecy. It does everything wrong.

When discoveries are kept secret, researchers not privy to those secrets are hindered and scientific and technological growth is slowed. Intellectual property law other than trade secret law is in fact designed to encourage research by discouraging secrets. Patent law, for Instance, does recognize the fact that corporations and Individuals do applied research hoping for long-range profits, and It does reward those who make basic discoveries. But patent law promotes public disclosure by granting to those who make public their discoveries the same or better economic advantages than could be gained by keeping them secret. [8] Patent law thus recognizes the need to balance the above noted utilitarian concerns.

Copyright law should similarly discourage secrecy. Until a few years ago, only public documents could be copyrighted. This places the creators of programs that incorporate real novelties In a bind. They can seek copyrights if and only if they forego the attempt to keep their algorithms as trade secrets. But present copyright law seems to leave open the possibility of copyright protection for programs that are still secret. This is still untested in the courts, and its full consequences remain unknown. I strongly disapprove of this possibility. It subverts the positive utility that may be gained when companies are forced to give up secrets for the sake of copyright. That is, I think that there is a positive utility in placing companies in certain sorts of binds.

Trade secret law, on the contrary, only protects personal use and discourages public disclosures. It does this in several ways. It defines a secret by codifying oppressive corporation security measures. For instance, research openly discussed in a common corporation cafeteria in the presence of people not working on the specific research project may not be secret according to the law. I am personally pleased that I have never had to work under the conditions that are required of corporations that wish to keep their developments as trade secrets. The law then gives additional force to those security methods by punishing anyone who takes advantage of a breakdown in security. It promotes exactly what the better legislation discourages.

That it discourages public disclosure is a criticism of trade secret law in general. When used to protect software, it not only encourages secrecy, but also creates proprietary rights that may violate free thought. Trade secret case law has proceeded to do this without adequate discussion of the basic philosophic question, whether software is the sort of thing that should be protected as property. I do not want to suggest that we separate the question of whether software is the sort of thing that can be property from questions on the sorts of protections that should be granted the software industry. To the contrary, 'property' is largely defined by the complex of laws that protect it. Since software is protected as property in trade secret law, software is property in some positive sense. [9] But that does not free us from questioning the validity of that law on the grounds that programs are simply not the sorts of things that ought to be considered as property. Since programs, viewed as mathematical abstractions, are refused protection in patent law, they are also not property in some positive sense. [10] In so far as trade secrets are property, the arguments that lead to the rejection of software patents seem applicable to software trade secrets. The issues demand resolution.

Software protected as trade secrets is property in so far as misappropriation of trade secrets is viewed as theft. [11] This is an important point. Only those things that can be owned and taken away can be stolen. Since, for instance, you cannot own another person, kidnapping is not theft. Since those who, in the ordinary sense, 'take' your ideas do not deprive you of them, ideas cannot be stolen in the sense in which cars are stolen. Software methods are like ideas and unlike cars in that to give them away is not to give them up. Yet the courts have counter-intuitively treated trade secret violations as theft. I do not think that the issue of 'removal' is particularly interesting. One can be deprived of economic benefits even if not of the methods. The more important point is that, if theft, trade secret violations are property violations. (Not all property violations are thefts, e. g., trespass is a property violation.) If software is property, the creator of the software has very special proprietary rights over the software that can be claimed against those who misappropriate it. Thus, a company whose software has been improperly discovered by a competitor can sue for the profits made from the use of that software. Since the legal structure does return profits to the original discoverer, those discoveries are certainly property in any reasonable sense of property. The point is that profits made from the use of software are legally protected by trade secret law above and beyond the company's ability to keep them for itself. When that happens we have moved from a situation where there may be contingent possession to full-fledged proprietary rights. This has all occurred in trade secret case law without addressing those arguments used in patent case law (discussed below) that reject exactly those sorts of economic protections. Trade secret law should not rush in where patent law fears to tread.

That trade secret law has rushed in is instructive. When a philosophic qualm or a slow-moving legislature prevents adequate response to rapid technical change, we can reasonably expect the courts to stretch existing statutes and common law. That legal definitions have been stretched out of shape indicates a need for software protection based on philosophically adequate foundations. If we must have trade secret law, we should seek a better foundation for it. We may. for instance, object to the means whereby programs become known without admitting that those means constitute theft. Unauthorized wire taps, for instance, are criminal regardless of the nature of the Information discovered through the wire tap. The crime is that the institution has been spied on, or invaded, not that anything has been taken. If we treat software trade secret violations this way, there need then be no decision on the status of improperly discovered information, including whether it includes modes of thought owned by the institution whose wires have been tapped. There then would be no conflict with principles of free thought.

It may appear that there is little difference between a view of the improper discovery of a software secret as theft and as an invasion of an institution--violations are punished in any event. There are, however, obvious differences. Since an improper means of discovery is improper even if no discovery is made, trade secret law would have wider scope if software violations were not treated primarily as theft. More importantly, I question the award of restitution for unjust enrichment. If it turns out that to protect software as property is to restrict modes of thought that should be free, I do not see how we can return profits made from the use of that software in so far as it is a mode of thought. [12] This will displease those who favor trade secret law (which I dislike anyway).I wonder if trade secret protection would remain effective for software if restitution for

unjust enrichment could not be demanded of unethical competitors who may be willing to pay lesser fines. For similar reasons. I disapprove of injunctions brought against an institution from use of improperly discovered software, such as can be done under current trade secret law. Even ignoring these technical differences in application of the law, I think there 1s a very important philosophical difference between treating an improper discovery as a theft of owned property and as an attack on the institution that attempted to keep the secret. In the latter case, them would be no apparent conflict with principles of free thought. (We must remember that these criticisms hold only for software trade secrets, and not for other sorts of trade secrets.)

Permissible forms of protection for intellectual property must be carefully stated so that they do not interfere with free thought. This is traditionally done in one of two ways. Copyright law distinguishes between the intellectual content of a text and its particular formulation in that text. Copyright then only protects the formulation, of the ideas. Patent law distinguishes between the intellectual content of an invention and the material machines to which the ideas are instantiated. It then only protects the construction and use of the machines. Neither distinction is unproblematic. Since complex thought apparently requires language use, it is hard to separate thought from the expression of thought. If we are to be free to think over the ideas contained in a copyrighted text, we must be free to express then, perhaps in ways that violate the copyright. This problem may be resolvable. But as far as I can see, any solution must depend on the possibility of expressing the content of a copyrighted text in ways unprotected by copyright. It is generally easier to maintain the patent law distinction between machines and the ideas on which they are based. A patent claim includes a description of the patented machine or process. That description incorporates the ideas behind the invention but is not itself protected by the patent. Thus, the basis for a distinction between the machine and its underlying idea is intrinsic to the patent claim itself. This independent formulation of the idea is not intrinsic to copyright application, where the document on file is itself protected.

We should not be in the least surprised that copyrights do not adequately protect software. A copyright protects the manner in which an idea is expressed. Programs are not written with the intention to express an idea, but to be used in processing data. Programs do, incidentally, express (to those who can read the code) the ideas incorporated in them. [13] But there is no intuitive reason to expect protections on the expression of ideas to be useful in the protection of processes. [14] Adequate protection for software must apparently protect more than the manner of expression. But when it looks as if a copyright might restrict more than one expression of an idea (because, for instance, there are no alternative expressions of the idea and thus, any copyright would restrict the use of the idea itself), then copyrights are refused. [15] Programmers clearly wish to use copyrights for just exactly that to which they may not be used: to acquire rights over the use of the complex system of interconnected calculations that are incidentally described by the program. What is surprising is that copyrights can be (and are) used effectively by the software industry. We will return to discuss this surprising success later. For the time being, however. let us observe the extent to which copyright protections are inadequate.

Any computer program can easily be rewritten so as to avoid infringing a copyright on the original program while keeping to the same basic algorithm. [15] One might, for

instance, translate the program into a different computer language. Since that is not to copy, that is not to violate copyright. It is hoped by same that the definition of 'copy' may be loosened to outlaw such simple-minded plagiarism. To some extent this is reasonable. 'Copy' is a necessarily loose notion. A close translation from French to English of a work would still be protected by a copyright on the original. But a free revision of a work is not protected. [17] It might seen that we can expand the already loose notion of copy to protect software translated from one computer language to another. But a translation from one computer language to another may result in very profound differences. It is not ,just that FORTRAN commands 'WRITE' when BASIC commands 'PRINT'. A simple change like that would be plagiarism. But, to take an extreme case, there are fundamental structural differences between a program written in FORTRAN and its counterpart in LISP. It is intrinsic to the notion of copyright that translations into architecturally different languages be possible without copyright infringement.

Copyright law is only permissible (does not violate principles of free thought) when it Is possible to present the basic ideas of the copyrighted material without making a copy. But, for theoretic reasons. the algorithms that are essential to a program cannot be written down except in a computer program. The basic point here is that computer programs are essentially effective (recursive) definitions. My adequate description of that content would Itself also have to be an effective definition.

And thus it would also be a computer program. So a copyright that fully protected a software system would cover all expressions of the system and would violate free thought. Central to the very notion of a copyright is a proscription of the use of copyrights for this purpose. Thus, we cannot so loosen the definition of copy so that all versions of a software system are protected.

The point should be emphasized that the difficulty with software copyrights is not that there are so few ways to encode a software system that a copyright on some particular coded version would generally protect all natural expressions of the system and thus violate free thought. It would, I think, be a mistake to refuse a copyright for a program on the grounds that there are few alternative natural expressions of the programmed material. [18] Generally there are many ways to encode an algorithm, and copyrights are certainly available for programs. The point is rather that copyrights are permissible ,just because there are alternative ways to encode. That is, copyrights are permissible simply because they do not provide the full protection sought by software researchers.

Valuable innovations are usually protected by patent. Patent protections in some sense complement copyright protections. Whereas copyright protects the language of a document and not the activities or innovations described by it, patents protect the innovations and not the language. The trouble with copyright protection for software is that it protects only the program code and not the essential algorithm. We should have high hopes for patents, which, if they do what they should do, would protect the algorithm itself. But, just a bit of consideration will show that this aspect of patent protection is not really all that promising. We also want our software protection to protect the code in which the program is written, but this is , just what is left unprotected by patent. This is a sign of more serious underlying problems. We observed that patent law demands a distinction between a machine and the innovative ideas behind that machine. This distinction is easy to draw in patent law because the patent claim describes the

innovation. That description presents the ideas, but is not itself part of the machine. But in a software patent, the program itself would be written in the claim and the distinction would be lost. This should make us look for violations of free thought. In fact. I do not believe that the crucial distinction can be maintained for a patent on software.

There has been official resistance to software patents. This is largely due to bureaucratic fear of the waves of Patent applications that my be expected if programs become patentable. [19] The philosophical grounds for that resistance has been one version or another of the claim that since algorithms are abstract mathematical formulas, they are unpatentable subject matter. I basically agree with those grounds. There are apparently two things to be argued here: (1) that programs are mathematical formulas in the relevant sense, and (2) that mathematical formulas in that sense are not patentable. The second point is not now an issue in the courts; it is the undisputed principle that guides argument. I, however. do not think that the argument that computers are mathematical formulas can be separated from the argument that formulas are unpatentable. Once we know what it is about formulas that is unpatentable, we can judge whether software exhibits the relevant features. I, of course, think that the basis for excluding patents on formulas is a potential for violating principles of free thought. And so far as that goes. a patent on a program would have the same potential as a patent on a formula. Software, as I argue below, is mathematical in the relevant sense.

Before 1968, that programs mere unpatentable due to their mathematical essence was expressed in the 'mental steps' doctrine. Although that doctrine has not been used since 1970. I think it showed good insight into the basic difficulties with software patents. [20] The doctrine is that any process that can be carried out as steps in the mind alone is unpatentable. Consider, for instance, a method for finding the quotient of tyro decimal expansions by the addition of successively finer increments to an initial low guess--that is, long-division. Each step in the process can be done mentally. The only thing that prevents a well-trained ten year old from mentally completing the exercise to any desired degree of accuracy is the nester of calculations that must be held in the memory. Thus, this method is unpatentable. The doctrine expresses a principle of free thought--what can be thought cannot be owned. It precludes patents of mathematical formulas. [21] And it precludes patents for programs. [22] Any algorithm can be viewed as a function on natural numbers reduced to elementary steps each of which can easily be carried out mentally.

It is irrelevant that, though each step can be performed mentally, no person has the memory for the whole series of steps performed by a computer. As already noted in the discussion of copyright, complex thoughts require language and free thought extends to those thoughts that can only be carried out with linguistic aids. That we need pencil and paper to do long-division does not make it less thought. The term 'mental steps' does unfortunately suggest that what is protected is a series of events that take place in an Incorporeal Cartesian mind. That would be the worse sort of confusion. Even if Cartesian mind-body dualism were correct, the sort of activity we wish to protect is that which 'is performed by the hand, when we think by writing'. [23] The only practical limits to thinking through the whole series of steps done by a programmed machine are time and availability of ink and paper. That limit cannot be the basis for patentability. [24]

This argument appears to conclusively preclude not only Patent protection for software, but the argument has some hidden subtleties. It may yet be possible to avoid

that conclusion. All innovations, even those most obviously patentable are based on ideas that can be perceived or run through mentally. We must not hold a mental steps doctrine that suggests that the conceptualization of a process precludes its patentability. We must remember that to conceptualize a process (e.g., a chemical synthesis) is not to carry out the process (result in the chemical).

The other side of the mental steps doctrine is a doctrine that the only patentable processes are those that transform corporeal substance. [25] Certainly the complex modes of thought that software researchers wish to protect are always carried out with physical tools, such as ink on paper and bites on computer tape. But these corporeal transformations are beside the point. All this ink and paper merely helps us think through the problem. In common parlance, programs 'press information'. And information is not corporeal. (Thought is not incorporeal because it takes place in an incorporeal substance, but because modes of thought and algorithms have a very different character than corporeal processes.)

Software could be protected by patent if we could only think of some way to describe the machine process that we wish to patent in such a way that to think through the algorithm is not to carry out the process. That is, the process defined in a software patent claim must be a machine process and not an information process. There must be a clear definitional demarcation between the abstract algorithm and the physical events that take place when that algorithm is performed by a computer. Our problems are definitional.

The definition of a process must be broad enough to preclude processes that are too similar to the patented process. It must not be so broad that it precludes conceptualization of the process. The mental steps refection of software patents centers on the fact that the definition of algorithm which is broad enough to protect the software is so broad that it precludes even thinking through the patent claim itself. Once again we must take note of the fact that an application for a software patent will more than likely include a coded version of the process that we wish to protect. (Perhaps the program is presented in a flow-chart rather than standard code. But that makes little difference. If the flow-chart is an adequate definition of the program, it is itself conceptually a program. We may soon have machines that read flow-charts.) This is very odd. The patent office could not accept an actual machine as a definition of the machine. It would fail to say what machine other than the given machine is covered by the patent. But a software patent claim, rather than defining the sort of thing protected, would be itself a synecdochic instance of the type of thing we wish to describe. For the same reason that the patent office does not file actual machines to patent machines of that type, it ought never to accept programs as applications for patents on programs of that type.

The obvious solution to our problem is the inclusion in the description of the software process a phrase such as '...when run on a digital computer.' [26] In a desperate attempt to find some way to provide patent protection for software, the courts have suggested exactly this move. The courts have suggested that the fact that a process can be performed mentally need not preclude patent protection for when it is performed otherwise. [27] For all practical purposes this is a rejection of the traditional version of the mental steps doctrine. But we still require that the definition of the patented process specify that to mentally perform the process is not a patent violation. The most blatant version of this suggestion is a district court attempt to view a programmed computer as a different, improved machine over an unprogrammed computer. [28] The result would be

that programs could be patented if the 'claims were drafted in apparatus form.' This is a brave attempt to overcome the difficulties, and I would not be surprised if the court were to ultimately settle on some variation of this. However, at present the Court remains unconvinced.

There are basically two problems with the 'apparatus form' patent. In the first place, the distinction between apparatus form and non apparatus form applications is so fine that it smacks of sophistry. Contrary to the common notion that lawyers live off minute legal niceities, the courts do not like this sort of thing. Sneaky subtleties are only introduced into the law when legal principles interfere with the clear desire of the legislatures or the courts. Everyone would be happier with a good firm basis for the law. There has been a proper protest against decision 'resting on nothing more than the way in which the patent claims had been drafted.' [29]

The more serious objection to a move to apparatus-form patents is that it ignores the nature of software innovations. To see the force of this criticism, you must understand that patents protect things, even

when those things are processes. Patent law speaks of 'patentable subject matter' with a strong, almost metaphysical sense of the independent status of the existing innovative thing that is patentable. That treatment is important for the law. We must so clearly demarcate the innovation that we can have a sense of how it may be moved about. Only then can we distinguish a new process (e.g., the use of a hair-dryer to defrost a refrigerator) from a protected process (e.g.. the use of a hair-blowing-device to dry damp surfaces, including wet hair and frosted refrigerators). A patent claim must specify the process so precisely that the courts can decide if the claim covers both hair drying and refrigerator defrosting. A process that is delimited with adequated precision for a patent claim has a special ontological status so that, like a simple physical object, we can tell when the identical object turns up in an unexpected place.

With this exalted ontological sense. Justice Stevens opens his dissent to Diehr by asking 'what the inventor claims to have discovered.' This is a demand for a description of a machine or process which is innovative and which is clearly delimited. His conclusion that software patents would protect algorithms is correct, because the high standard of definition required by that exalted ontological sense precludes a distinction between programmed and unprogrammed machines. Since those algorithm may be the basis for a method of argument, it follows that software patents would seek to protect unpatentable subject matter.

Computing machines are built to be programmed to solve large varieties of problems. An advertisement for a stove or a car with 'a Computer in it' is terribly misleading. Similarly, hand calculators and 'hand computers' use computer technology, but are not sufficiently versatile to be considered computers. A versatile computer programmed with an innovative algorithm is not used innovatively. It is used in precisely the may it is intended to be used. It follows that what the inventor claims to have discovered when he applied for a software patent is not an innovative computer use. Thinking of a programmed computer as an innovative improvement of an unprogrammed computer is like thinking of a car with gas in it as an innovative improvement on a car without gas. A programmed computer might indeed be part of an innovative machine use that is patentable. So, sore software innovations may be partly protectable by patent. [30] Without falling into science fiction, however. I cannot give examples. I wish I could; I would patent them. But then the innovation is not ,just new software. Since software innovation is not an innovative machine use, it looks like it must be something removable from the context of computing machine use. It is, in fact, the abstract algorithm. On that level it is unpatentable.

The mental steps doctrine was abandoned in the hope that a way may be found to make 'machine performances' of algorithms patentable as innovative uses of computers. Though it may still be wise to abandon the mental steps doctrine, this move will not help us protect software. It still looks as if effective software protection must restrict consideration of algorithm in ways incompatible with principles of free thought. If the mental steps doctrine is abandoned, other principles must be preserved that prevent violations of free thought. The proscription of patents on mathematical formulas will do for present purposes. We my note that Justice Rehnquist (even in an opinion that is widely thought to open the way for software patents) treats mathematical formulas just as I treat algorithms here: '...when a claim recites a mathematical formula ..., an Inquiry must be made Into whether the claim is seeking patent protection for that formula in the abstract. A mathematical formula as such is not accorded the protection of our patent

lays..., and this principle cannot be circumvented by attempting to limit the use of the formula to a particular technical environment. [31] It does not matter if the test for patentability is mental steps or mathematical abstraction, the algorithm is patentable subject matter on either test. Since the algorithm and not its performance is innovative. software is then not patentable.

My view hem is very close to the dissent in Diehr. So it may appear to the casual reader of that case that my argument is not accepted by the majority of the Court. That is not so. My general theoretic points are accepted by the whole Court. The disagreement rests with the interpretation of Diehr's patent claim. The popular press (including computer trade journals) has ignored the general theoretic agreement and hailed the decision far opening the way for software patents. That the facts of the case lend themselves to this misinterpretation indicates that indeed the facts rare correctly read by the minority and incorrectly read by the majority.

My philosophical survey of intellectual property law seems to preclude any philosophically acceptable proprietary protections for computer programs. But that is misleading. Even if there is no form of protection which works 1n general, there are situations where adequate protections are available. I will conclude with a few examples.

Patent protection may be available in certain situations. The argument against software patents turns on the versatility of computers. When program are built directly into machines that perform only a small number of functions (just play chess or ,just edit texts), machines with different programs built into them am indeed different machines. In these machines, the programs are built into ROMs (for 'read only memory'). The ROM has an odd status between software and hardware. usually it is called 'firmware'. I do not want to discuss the complexities of applying patent law to these cases nor, except to observe that they area special case. Perhaps they are patentable. Also, some firmware is rather hard to copy or reverse engineer. Thus, the more outrageous forms of copyright violations may became less common. Since specialized machines with ROMs cost less than real computers, patent protections for these machines may cover a large portion of the expanding market in computer technology.

In spite of the limitations on three discussed above, copyright protections can be very effective. For various economic reasons, programs must be written in a small number of standard or normal languages. Businesses, for instance, mostly use COBOL. So the fact that a program can be translated into a rear language without violating copyright is not important. [32] A program that is not to the right language may be useless. Also the standard look of a program may be a large part of its value and that may be protected by copyright. The language SPSS, for Instance, is nor standard in the social sciences. Part of its valor is that mast social scientists know it, can judge the effectiveness of programs written in it, and can use it on local computers. The compiler programs that read SPSS into machines are fairly well protected by trademark and copyright on the look of SPSS itself. Moreover, complex software systems are often very long, [33] containing perhaps even a hundred thousand lines of code, as long as a short Tolstoy novel. We might admit the conceptual possibility that one can recount the events and describe the characters of War and Peace without copying it in the legal sense. (In fact on my view, we must admit this very dubious possibility if we are to permit copyrights.) But the task of rewriting would be immense. Similarly, the task of rewriting a complex software system without legally copying it is immense. The sort of simple-minded copying which is

common in the industry today can be prevented by copyright. We might also note that 'operating programs' that teach hardware systems how to respond to software are largely determined by the hardware itself. It my not be possible to replace an operating program with another effective program that is not a copy of the original. Copyrights should be particularly effective protections for long programs that my take considerable labor to write but which are not so radically creative that they could qualify for patents anyway.
Finally, w may note that software may be protected by contracts that cover, for Instance, client use. Contracts may be at present the most popular means to protect software. There are, however, good reasons to doubt that contracts can be written In a form that provides adequate protection for software without treating it In ways against which I have argued here. A contract cannot create proprietary protections without some background notion of property against which the contract may be tested. One cannot, for instance. give oneself up as property (bind oneself into slavery) since w do not admit that persons can be property. My argument here apparently precludes protective contracts for software by precluding any conception of software under which it is the sort of thing that can be property. My arguments against trade secrets in general, and against full protection from copyrights and patents, leave no apparent basis for the contract. Moreover. I have generally asserted that any legal recognition of a right over software turns it into property, and when this extends to algorithms, it violates free thought. The obvious consequence is that contracts cannot validly protect software. All the same, there may be some way to avoid this conclusion.
A full discussion of contracts would have to separately consider several different forms of contracts covering various sorts of situations (e. g., client relations, employee relations, subcontractor relations, etc.) I will simply sketch some of the problems I see with one sort of contract. The obvious ray to avoid client disclosure of software is to include the software in a service contract to process data without ever permitting the client access to the software. Some such approach to contract protection my be attempted without discussion of the proprietary status of the underlying algorithms.
We should note that the philosophic qualms that I have to the contract are not at present legal obstacles to the contract. Trade secret law provides the necessary background for protective contracts. In fact, a discussion of contract protections is not logically distinct from a discussion of trade secrets. Contracts often act to create trade secrets and trade secret law provides the basis for contracts. All my qualms concerning contracts are merely extensions of my qualms concerning trade secrets. Thus, my prior discussion of trade secrets is hero shown to be inadequate to the extent that I have not discussed alternative forms of contracts. I do hold to my earlier discussion. but now note a way in which it may be attacked. I did open the way to this attack when I qualified my discussion of trade secrets with suggestions for a better basis for trade secret law.

The treatment of software as a service has consequences in such matters as company liability and tax procedures. It is interesting to note in this respect that discussions of liability for software failure closely resemble some of our discussions here.[34] Moreover, a service contract might conflict with anti-trust law that could prohibit tying software to general services. Whether companies will find it practical to protect software through service contracts will depend in part on matters such as these. More seriously, I think it is implausible to view software as a service. For one thing, it seems a bit underhanded to protect software indirectly through control of its use rather than through direct protections. Secondly, software simply does not look like a service. In those cases when a software company is called in by a client to develop a system for a client on that client's machines, the software company does apparently provide a service. But in those cases, the client normally expects a partial verification of the correctness of the system by a 'talk through' of the basic programs. If that is indeed done, the software company has lost control over the program and the protections envisioned here are of no use.

In the case when the software company provides a standard system (e.g., adapts an established inventory program to the client's needs), then the system just does not look like a service. In this case, the system should be treated as a product and the protections envisioned here are inappropriate. Of course, in so far as it looks like a product, it also looks like the sort of thing that can be property. At this point we rediscover in our discussion of contracts the same issues that arose in the discussion of patents. A more complete discussion of contracts would require a lengthy re-investigation of those issues from a new

perspective. I do believe that there is, in this direction, hope for successful software protection that I would find acceptable in spite of appearances to the contrary.

[1] Research for and preparation of this paper was partly sponsored by the Center for the Study of Ethics in the Professions at Illinois Institute of Technology.

[2] The economic situation and the consequences of the present lack of protective policies is nicely summed up in J. D. Slack, 'Programming Protection: the Problem of Software', Journal of Communication, 31:1, Winter 1981.

[3] I am interested here only in software that is in the form of programs, i.e., instructions for calculations. I do not look at attempts to own software in the form of data (e.g., customer lists, machine tolerance statistics, etc.). Some of the issues are very similar.

[4] The Newton example is popular fiction. A real example is Tartaglio's attempt to keep private his solution for cubic polynomials in the 16th century.

[5] Some software researchers have also expressed a desire for recognition for their work that would not amount to its treatment as property. A friend, Jim, for instance says that he is perfectly happy to see his algorithms used by others, but resents it when others try to sell or pass off his work as their own. Jim has an academicians sense of recognition. See note #17.

[6] Since at present patents for software are rejected on narrow grounds within the patent law, there is the possibility that patent protections be extended to software by legislation. Such legislation is being sought by various groups. In opposition to that legislation, my more fundamental criticisms could make a difference.

[7] Diamond v Diehr, 101 S.ct. 1048 (1981). Diamond is commissioner of patents and trademarks. Diehr has computerized a process for curing synthetic rubber and seeks patent protection. The majority (5) opinion delivered by Rhenquist is that the whole system including the programmed computer comprises a patentable process. The minority (4) opinion delivered by Stevens is that the only novel part of Diehr's process is the computer program and that is unpatentable.

[8] See M. Novick and H. Wallenstein, 'The Algorithm and Computer Software Patentability', Rutgers Journal of Computers, Technology and the Law, Vol. 7, 1980, note #1.

[9] See e.g., Hancock v State, 402 S.W. 2d. 906 (1966).

[10] Diehr at 1056; Parker v Flook, 437 U.S. 584 (1978); Gottschalk v Benson, 409 U.S. 63 (1973).

[11] See Hancock v State. Trade secret law is determined state by state. At the present time, all the United States treat trade secret violations as theft.

[12] A reward for damages for a trade secret violation may include a return of profits made with the use of the misappropriated secret to the original 'owner'. In Telex v I.B.M., 510 F. 2d. 844, the court considered the projected cost to Telex of independently developing the secrets taken from I.B.M. Alternative relief includes injunctions against the use of the secrets, either permanently or for an estimate of the time needed for independent development. Punitive damages may be added on.

[13] The Copyright Office Circular 61 (1975) observes that 'programs can also embody textual materials ... which can be considered literary expressions.' The copyright office here probably has in mind the explanatory 'comment' that is inserted between lines of code in most programs. The point is more generally true, holding also of the code itself.

[14] It is clearly against the intent of the patent law that it be used to protect the production and use of objects other than for the expression of ideas. See note #18. All the same, in Mazur v Stein, 347 U.S. 201 (1954), the court held that a sculptural item was protected even though it was meant as a lamp base. Copyrights are also used to protect the look of board games such as monopoly from close imitation, etc.

[15] In Morrissy v Proctor & Gamble Co., 379 F. 2d. 675 (1967), it was held that the expression of the rules for a simple game was uncopyrightable for they were the only natural way to express the rules.

[16] I carefully use the word 'program' here to refer to the sequence of coded statements that instruct the computer. 'Algorithm' is used to refer to the essential calculations, regardless of the code in which they are expressed. The program algorithm distinction is like the sentence/proposition distinction, with all its familiar difficulties. I use 'software' when I wish to be ambiguous, without regard to the distinction.

[17] What is commonly called 'plagiarism' in academia is not strictly copyright violation. If you put it in your own words, you are legally free to use the argument of this paper in the order in which they appear here without giving credit. There is a much stricter 'gentlemen's code' for academia. It is both reasonable and proper that this code be broader than the legal code. But an industry as economically competitive as the computer software industry cannot depend on a gentlemen's code. Sometimes the word 'plagiarism' is reserved for violations of the gentlemen's code. Then what is at issue here is 'piracy'.

[18] In Synercom Technology v Universal Computers, 472 F. supp. 37, a copyright for the input format of a program was questioned because the copyright would overly restrict the expression of ideas encompassed in the format. I agree with Allan E. Rezich 'Copyright Protection for Computer Formats and the Idea/Expression Dichotomy', Rutgers Journal for Computers, Technology and the Law, Vol. 8, #1, 1980, that this is a mistake for there are plenty of alternative formats. All the same, the court had reason to look askance at the copyright claim. Synercom was typically attempting to use a copyright to receive the sorts of protections for a process usually sought through patent. In this regard, we may note that in Whist Club v Foster, 42 F. 2d. 782 (1929), the play of a game is unprotected even though the expression of the rules is copyrighted.

[19] See the President's Commission on the Patent System, Report (Nov. 17, 1966).

[20] The history of this doctrine is nicely summed up by Justice Stevens in Diehr at 1060-1065.

[21] Justice Stevens notes that the mental steps doctrine is 'based upon' the principle that scientific concepts (and formulas) are unpatentable

subject matter, Diehr at 1060. Historically, this is the case. But philosophically, I think it is the reverse. The mental steps doctrine, by emphasizing free thought, indicates the basis for the principle that scientific concepts are unpatentable.

[22] This has always been obvious to the courts. The mental steps doctrine precludes patents for 'almost any conceivable program', Diehr at 1060.

[23] Ludwig Wittgenstein, Blue and Brown Books, Harper and Row, New York, 1965, p.6.

[24] An opposing view is nicely argued by H. L. Davis, 'Computer Programs and Patent Matter Subjectability', Rutgers Journal of Computers, Technology and the Law, Vol. 6, #1 (1977), p.22.

[25] See Cochrane v Deener, 94 U.S. 780 (1876). As applied to computer software, see Gottschalk v Benson, 409 U.S. 663 (1972).

[26] This is argued by H. L. Davis, op. cit., p.22.

[27] See In re Prater, at 1389.

[28] In re Bernhardt, 417 F. 2d. 1395 (1969), at 1400.

[29] Diehr, at 1065.

[30] This is exactly how Justice Rhenquist views the patent claim in Diehr.

[31] Diehr, at 1058-1059.

[32] If computer languages become more standardized, then some of the points made against copyrights here may become moot. In that case, however, there may still be obstacles to copyright protections. It will still be the case that copyrights will be used here to protect information processes and not expressions of ideas. And if the only natural expression of an algorithm is in one standard language, a copyright in that language might preclude any natural alternative expressions of the algorithm and be disallowed.

[33] Long Software systems are never viewed as single programs, but consist of programs each rarely longer than a hundred lines of code. The programs are put together into complex systems of programs that 'call upon' each other.

[34] We may note the striking similarity between the issues discussed here and the discussion in Sysan Nycum 'Liability for Malfunction of a Computer Program', Rutgers Journal of Computers, Technology and the Law, Vol. 7, 1979, pp.16-19.