Harvard Journal of Law & Technology
Volume 10, Number 2 Winter 1997

COMPUTER SCIENCE CONCEPTS IN COPYRIGHT CASES: THE PATH TO A COHERENT LAW

Marci A. Hamilton[*]
Ted Sabety[**]

TABLE OF CONTENTS

---

* Professor of Law, Benjamin N. Cardozo School of Law. We owe a large debt of gratitude to Pamela Samuelson, whose comments (and former writings) were very helpful. We also thank Mark Lemley, Bill Patry, and Stewart Sterk for their comments on an earlier draft and Cynthia Cook and Roni Jacobson for research assistance.

** J.D., Class of 1997, Columbia University School of Law.

I. INTRODUCTION

The law surrounding computer software copyright[1] is on a collision course with computer science. One of the main reasons for this is that legal terms of art in copyright case law do not reflect accepted computer science terminology. For example, the cases have coined phrases like "structure, sequence, and organization"[2] and "program structure,"[3] but neither term accurately reflects computer science reality. The use of these court-made terms has tended to make the application of copyright law to computer software frustrating for judges and litigants and confusing for the billion-dollar computer software industry.

Because the Copyright Act[4] unambiguously protects computer programs,[5] but the discourse has not yet accurately identified the noncopyrightable elements bound up within programs, there is a serious likelihood that authors of computer programs are receiving and will receive more protection than copyright law and policy justify. This inadvertent overprotection of computer software introduces powerful negative externalities into the software industry.

For example, Sun Microsystems requires that users enter into a detailed licensing agreement before writing a program that implements their programming language, "Java." A dispute between Sun and one of their licensees, Microsoft, is emerging because Microsoft has modified its version of Java to increase compatibility with their Windows

---

1. Computer programs have been recognized as copyrightable by the U.S. Copyright Office since 1964. *See* George D. Cary, *Copyright Registration and Computer Programs*, 11 BULL. COPYRIGHT SOC'Y 362 (1964); Second Supplementary Report of the Register of Copyrights on the General Revision of the U.S. Copyright Law: 1975 Revision Bill 2 (1975) (explaining that the definition of literary works was intended to be broad enough to encompass computer programs); *see also* Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 1249 (3d Cir. 1983) (holding that the Apple system program object code embodied in read-only memory ("ROM") was copyrightable subject matter).

"Software" and "program" are two terms that are used interchangeably in the computer science field. However, the connotations in certain contexts may be different. For example, a program is the sequence of instructions that are executed by the computer when it performs a desired task. Software, on the other hand, sometimes refers to a finished program product ready for retail sale, e.g., a copy of the program as well as instruction manuals, help files, tutorial programs, and other aspects that make the program consumable by the public. Often a software product is a collection of several programs that together constitute a single product for sale. This Article follows computer science convention and uses the terms essentially interchangeably.

2. Whelan Assocs., Inc. v. Jaslow Dental Lab., Inc., 797 F.2d 1222, 1224 (3d Cir. 1986).

3. Computer Assocs. Int'l, Inc. v. Altai, Inc., 982 F.2d 693, 702, 707 (2d Cir. 1992).

4. 17 U.S.C. §§ 101-121 (1994).

5. *See* 17 U.S.C. § 101 (1994) (defining "computer program" and "literary work"); *see also* Sega Enters. v. Accolade, Inc., 977 F.2d 1510, 1519 (9th Cir. 1992).

operating system without Sun's permission.[6] This dispute may erupt into litigation centering on the question of whether computer languages are copyrightable subject matter — and it may turn out that Sun's rigorous licensing requirements are unenforceable.

To be fair, courts, Congress, and the commentators have not known what they did not know. The tests proffered in the leading software cases, *Whelan Associates, Inc. v. Jaslow Dental Laboratories, Inc.*[7] and *Computer Associates International, Inc. v. Altai, Inc.*,[8] draw on two different branches of settled copyright doctrine, but both suffer from a lack of accurate terminology for discrete aspects of the software. On the one hand, the *Whelan* court protects the structure, sequence, and organization of the program, drawing upon well-settled copyright doctrine that structure and organization are copyrightable even when the component pieces of a work are not.[9] On the other hand, the *Altai* decision tells courts to excise the noncopyrightable aspects of the program before assessing the program's overall copyrightability, drawing upon well-settled copyright precedent regarding infringement analysis.[10] The approaches in these two leading computer software copyright cases threaten to protect noncopyrightable elements in computer programs because they fail to use scientifically relevant terms to identify those noncopyrightable elements.

While the courts have been crafting new legal terms to deal with computer software cases, important computer science terms that could contribute to more accurate application of copyright law have not found their way into the courts' copyright lexicon. The use of computer science terms such as "data structure," "algorithm," and "computer language" would assist courts as they attempt to determine that which is copyrightable in a computer program and that which is not. Although some commentators have offered compelling arguments that computer programs, and other types of "know-how," do not fit neatly within existing intellectual property paradigms,[11] the fact is that the 1976

---

6. *See Sun Wakes Up and Smells the Java*, BUS. WK., Dec. 23, 1996, at 46.
7. 797 F.2d 1222 (3d Cir. 1986).
8. 982 F.2d 693 (2d Cir. 1992).
9. *See infra* note 51 and accompanying text.
10. *See Altai*, 982 F.2d at 707; *see also infra* notes 52-55 and accompanying text.
11. *See* Pamela Samuelson et al., *A Manifesto Concerning the Legal Protection of Computer Programs*, 94 COLUM. L. REV. 2308 (1994) (proposing a sui generis legal protection for computer software); Jerome Reichman, *Legal Hybrids Between the Patent and Copyright Paradigms*, 94 COLUM. L. REV. 2432 (1994) (surveying hybrid legal regimes for protecting industrial arts) [hereinafter Reichman, *Legal Hybrids*]; Jerome Reichman, *Charting the Collapse of the Patent-Copyright Dichotomy: Premises for a Restructured International Intellectual Property System*, 13 CARDOZO ARTS & ENT. L.J. 475 (1995) (suggesting that a new intellectual property paradigm is needed to replace a proliferation of hybrid legal regimes).

Copyright Act explicitly covers computer programs and therefore requires courts to grapple with the application of copyright concepts to programs.

This Article's thesis is that it is time to accurately name the copyrightable elements of computer software using computer science terminology so that copyright analysis can go forward on more solid footing. The monopoly created by imprecise application of copyright law to computer software is leading to anti-competitive effects that could be corrected through the use of more exact computer science terminology in the case law.[12] This Article takes a step toward mitigating the case law's unintended externalities by bringing the two discrete worlds of copyright law and computer science directly into contact with one another. Now is the time for such a project because of the current explosion in software authorship and the widespread computer use attendant upon the arrival of the global information infrastructure.[13]

Part I provides an overview of the computer software copyright controversy. Part II suggests that the terms "structure, sequence, and organization" and "program structure" obscure the question of whether the program's data structure is copyrightable. By employing the appropriate computer language terminology, we illustrate that many computer software copyright cases have coined terminology that leads courts to provide copyright protection for elements of programs unworthy of protection. In particular, we conclude in Part II that data structures necessary to a particular algorithm should not receive copyright protection because such protection would confer a de facto monopoly over the uncopyrightable algorithm. Part III makes the point that computer languages should not be protected because such protection confers a priori copyright protection on unfixed expression. The same reasoning leads to the conclusion that computer language grammars also fail to qualify for copyright protection.

---

12. Professor Randall Davis initiated this important project with his article, *The Nature of Software and Its Consequences for Establishing and Evaluating Similarity*, 5 SOFTWARE L.J. 299 (1992).

International law would also benefit from more specificity. *See* Multilateral Negotiations, Marrakesh Agreement Establishing the World Trade Organization, signed at Marrakesh (Morocco), Apr. 15, 1994, Appendix F: Agreement on Trade-Related Aspects of Intellectual Property Rights, Including Trade in Counterfeit Goods [hereinafter *TRIPS Agreement*]. The *TRIPS Agreement* is as vague on this score as American law.

13. Fourth quarter 1995 worldwide revenues in software were $2.01 billion. *See* Don Clark, *Sales for Large-Computer Software Surge*, WALL ST. J., Apr. 4, 1996, at B2.

II. THE COMPUTER SOFTWARE COPYRIGHT CONTROVERSY: THE NEED TO SPEAK WITH COMPUTER SCIENCE WORDS AND TO THINK WITH COMPUTER SCIENCE CONCEPTS

The application of copyright law to computer programs has produced a prodigious amount of commentary and controversy for over twenty years.[14] Even though the copyright statute has been written and re-written in order to accommodate emerging technologies,[15] and computer programs in particular,[16] there is still considerable uncertainty when it comes to the copyrightability of computer program elements.

In general, any creative work that is "fixed in any tangible medium of expression" may be protected by copyright law.[17] Fixation means that the expression[18] is somehow recorded in some kind of medium. Canvas, paper, magnetic tape, and now computer memory and computer disk storage are considered mediums of expression. An important aspect of copyright law doctrine, and one that fundamentally distinguishes it from patent law, is that expression may be afforded copyright protection but the underlying idea embodied in that expression is not copyrightable.[19] This means that processes and procedures described by a text (or a computer program) may not be protected by copyright even though the text (or the program code) itself can be.

In addition, the Supreme Court has held that only work that meets some de minimis threshold of originality and creativity can be protected. "Sweat of the brow" alone does not justify copyright protection for a work.[20] The Court decided that the act of compiling phone numbers in alphabetical order for a phone book did not meet the de minimis

---

14. *See generally* Anthony L. Clapes et al., *Silicon Epics and Binary Bards: Determining the Proper Scope of Copyright Protection for Computer Programs*, 34 UCLA L. REV. 1493 (1987); Steven R. Englund, *Idea, Process, or Protected Expression: Determining the Scope of Copyright Protection of the Structure of Computer Programs*, 88 MICH. L. REV. 866 (1990); Morton D. Goldberg & John F. Burliegh, *Copyright Protection for Computer Programs: Is the Sky Falling?*, 17 AM. INTEL. PROP. L. ASS'N Q. J. 294 (1989); Peter S. Menell, *Computer Copyright*, 41 STAN. L. REV. 1045 (1989); Arthur R. Miller, *Copyright Protection for Computer Programs, Databases, and Computer Generated Works: Is Anything New Since CONTU?*, 106 HARV. L. REV. 977 (1993); Samuelson et al., *supra* note 11.

15. "Copyright protection subsists, in accordance with this title, in original works of authorship fixed in any tangible medium of expression, now known or later developed . . . ." 17 U.S.C. § 102(a) (1994).

16. *See supra* note 1 and accompanying text.

17. 17 U.S.C. § 102 (1994).

18. *See id.* § 101 (giving the statutory definition of "fixed").

19. *See, e.g.*, Baker v. Selden, 101 U.S. 99, 103 (1879).

20. *See* Feist Publications, Inc. v. Rural Tel. Serv. Co., 499 U.S. 340, 359-60 (1991).

creativity standard regardless of how much work was required.[21] However, collections of unprotected elements that are compiled in a creative way can be protected as compilations: the protection extends to the compilation aspect but not to the unprotected underlying elements.

A literal copy of a work is infringing.[22] But a non-literal copy can infringe as well. "[Copyright] cannot be limited literally to the text, else a plagiarist would escape by immaterial variations."[23] The test as to whether one work is a non-literal copy of another requires a determination of whether they are substantially similar.[24] To that end, courts will compare the sequence, structure, and organization of the underlying elements that make up the work.[25]

Although these broad principles of U.S. copyright law are well settled, applying them in the context of computer programs has not been straightforward. That the two leading approaches, *Whelan* and *Altai*, are as different as they are is a testimony to the landscape of confusion.[26]

Because of this confusion, computer software entrepreneurs have been placed in the position of not knowing the scope of copyright protection (and therefore the market value) for the works they create and simultaneously not knowing what they can legitimately borrow from existing programs. Furthermore, the consumers of computer programs have been left at a disadvantage whenever new software technology remains unavailable to them because one vendor cannot provide a "migration path" to users of a competitor's program for fear of copyright infringement.[27] Naturally, these are serious externalities in the computer market, which is driven to both standardize and to revolutionize its

---

21. *See id.* at 362.

22. *See* 17 U.S.C. § 106 (1994).

23. Nichols v. Universal Pictures Corp., 45 F.2d 119, 121 (2d Cir. 1930).

24. *See* Warner Bros. v. American Broadcasting Cos., 720 F.2d 231, 245 (2d Cir. 1983).

25. *See Nichols*, 45 F.2d at 121-22.

26. *See infra* text accompanying notes 51-52; *cf.* Miller, *supra* note 14, at 1001-02 (arguing that the *Altai* court merely modified the *Whelan* approach, but noting that some observers, and the majority in *Altai*, viewed the *Altai* opinion as a distinct break with the Third Circuit's reasoning in *Whelan*).

27. A migration path is typically a translation program that allows users of program A to convert their own work to a format useable by program B. The vendor of program B will write this translator to attract users of program A. If vendor B cannot produce a translator without infringing on vendor A's copyright, then the users of program A will not be able to use program B unless they re-enter their data by hand — a prohibitively expensive process. *See* Clapes et al., *supra* note 14, at 1503. A migration path from Lotus 1-2-3 to Borland Quattro Pro is the center of the dispute in *Lotus Dev. Corp. v. Borland Int'l, Inc.*, 831 F. Supp. 223 (D. Mass. 1993), *rev'd*, 49 F.3d 807 (1st Cir. 1995), *aff'd by an equally divided court*, 116 S.Ct. 804 (1996).

products.[28] Uncertainty impedes the risk-taking inherent to innovation, and this impedes the purpose of the Copyright Clause of the Constitution "to promote the Progress of Science and the useful Arts."[29]

The legal world's studious avoidance of computer science terminology arises from the vexing reality that computer programs are a form of expression not intended to be "consumed" by a human.[30] Although a human is the ultimate consumer of the program's results, the program itself is written for the machine to read and act upon.[31] A reasonable person employing five senses is not adequate to analyze such works. Rather, the determination of the extent of copyright protection for a computer program requires recourse to experts who are familiar enough with the oeuvre to identify elements that are not in fact expression and then to identify which expression is sufficiently original to justify copyright protection. Some courts and commentators draw analogies between computer programs and existing copyrightable works, such as literary works, in an attempt to simplify or avoid this problem.[32] Yet, the analogies frequently create more problems than they solve. They lead courts and theorists away from computer science — with its more precise definitions — into a discourse unsupported by scientific reality that obscures the underlying copyright issues.

Because computer programs are not typical expression and require a fair degree of sophistication to analyze for copyright purposes, precise terminology is required. The terms of art used in the law to define the non-literal aspects of a computer program fit uncomfortably with

---

28. *See, e.g.*, S.J. Liebowitz & Stephen E. Margolis, *Should Technology Choice Be a Concern of Antitrust Policy?*, 9 HARV. J.L. & TECH. 283, 290 (1996) (discussing the economic impact of selecting standards in high-tech industries).

29. U.S. CONST. art. I, § 8, cl. 8; *see* Menell, *supra* note 14, at 1049 (describing software copyright as the solution to the public goods problem).

30. *See, e.g.*, White-Smith Music Publ'g. Co. v. Apollo Co., 209 U.S. 1, 16 (1908) (superseded by statute as stated in Apple Computer, Inc. v. Franklin Computer Corp., 714 F.2d 1240, 1248 (3d Cir. 1983)) (stating that player piano rolls posed a similar problem in trying to divine how something not easily read by a human could be copyrightable expression).

31. Commissioner Hersey of the National Commission on New Technological Uses of Copyrighted Works proposed that computer programs be precluded from copyright and that only their product for human consumption be copyrightable. NATIONAL COMMISSION ON NEW TECHNOLOGICAL USES OF COPYRIGHTED WORKS, FINAL REPORT 29 (1978) [hereinafter CONTU]. That is, copyright law would be restricted to expression directly perceived for humans. His view was rejected. *See id.* at 27.

32. *See* 17 U.S.C. § 101 (1994) ("A computer program is a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result."); H.R. REP. NO. 94-1476, at 53, *reprinted in* 1976 U.S.C.C.A.N. 5659, 5666-69, § 102 (explaining that the term "literary works" includes computer programs); *see also TRIPS Agreement*, *supra* note 12 ("Computer Programs, whether in source or object code, shall be protected as literary works under the Berne Convention (1971).").

computer science. The court-created phrases such as "structural similarities," do not have precise analogues in computer science. For example, the phrase "structural similarity" between two programs could refer broadly to how the data is organized as well as how the program code is organized, or more narrowly to the program code itself.[33] There is a tremendous factual difference among these meanings. A more precise definition will be found by integrating computer science terms of art into the legal discourse.

The legal community should not shy away from coming to terms with "computerese." Computers and their programs are plainly here to stay. The lack of a precise set of terms to describe program components has unnecessarily complicated application of copyright doctrine to software. Without labels for the various parts of a program, courts cannot accurately identify where the program's copyrighted expression ends and its non-copyrightable aspects begin, or distinguish its expressive aspects from its purely utilitarian ones. Always a difficult line to draw with any work, the precise identification of the work's protectable expression poses daunting problems where the work is not naturally readable and its components are unnamed.

Although plainly applicable to new technological forms of expression, the copyright statute itself is not specific about how to separate protected expression from non-protected aspects of these emerging technologies.[34] For traditional copyright goals to be achieved, reference must be made to the new meanings arising within the new technological world. Careful use of computer science terms of art and the proper integration of that lexicon into the orbit of copyright law will provide a firmer foundation upon which to build a coherent jurisprudence of computer program copyright. A more coherent doctrine derived from more accurate computer science terminology will, in turn, produce a more predictable legal environment conducive to furthering the computer

---

33. *See, e.g.*, Englund, *supra* note 14, at 871 (defining "structure" in terms of the organization of programming subtasks). Many cases, specifically *Computer Assocs. Int'l, Inc. v. Altai, Inc.*, 982 F.2d 693, 698 (2d Cir. 1992) and *Lotus Development, Inc. v. Paperback Software Int'l*, 740 F. Supp. 37, 53 (D. Mass. 1990), cite Englund's article as an explanation of "program structure"; however, his work does not directly address the issue raised in this article. Englund's explanation and analytical test are aimed at program code organization ("control flow") rather than data organization ("data structure"). *See* Englund, *supra* note 14, at 901-08.

34. *See* 17 U.S.C. § 102(a) (1994) (extending protection to "any tangible medium of expression now known or later developed").

software art and the "progress" encouraged by the Constitution.[35] The industry and consumers should benefit.

III. "PROGRAM STRUCTURE," DATA STRUCTURES, AND ALGORITHMS

Large computer programs can be divided into three basic parts. First, there is the "user interface,"[36] which includes all means by which a user can interact with the software and the hardware.[37] User interfaces address the human side of the human-computer relationship.

Second, within the program itself, there are low-level hardware and software drivers. They are short, fast, and compact code modules that directly command the computer hardware. For example, one driver controls the disk drive while another controls the computer display screen. The driver control modules directly perform the most elemental level of computer input and output functions.

Third, there is a data processing section — the heart of any program. The data processing portion of the program solves the computational problem that the human has input into the computer via the user interface. While solving the problem posed, the data processing section may request the hardware drivers to provide it with information stored on disk. When the data processing section needs to notify the human operator regarding its status, it will call on the video display drivers to display the result.

The easiest copyright cases involving computer software are the ones that naturally draw upon existing copyright doctrine. For example,

---

35. *See* Twentieth Century Music Corp. v. Aiken, 422 U.S. 151, 156 (1975) ("'The sole interest of the United States and the primary object in conferring the [copyright] monopoly' . . . 'lie in the general benefits derived by the public from the labors of authors.'") (quoting Fox Film Corp. v. Doyal, 286 U.S. 123, 127 (1932)); *see also* Menell, *supra* note 14, at 1058 ("The Supreme Court has interpreted the law implementing this language to mean that the author's benefit is 'secondary' to advancement of the arts and sciences for society's benefit.") (citing United States v. Paramount Pictures, Inc., 334 U.S. 131, 158 (1948)).

36. "User interface" is a computer science term that describes the method or process by which a human interacts with a computer program. Aside from the display on the computer screen, the user interface may also include paper output and pictures, and the input can range from inserting a credit card into a cash machine at a bank to some kind of advanced brain wave detection in the future. In typical usage, however, user interface refers more narrowly to the screen display. We hew to this convention because the hardware aspects of user interfaces fall under the purview of patent law. *See generally* RAY E. EBERTS, USER INTERFACE DESIGN (1994).

37. Examples of user interface include everything from screen displays to verbal commands received by the computer. *See generally* Pamela Samuelson, *Computer Programs, User Interfaces, and Section 102(b) of the Copyright Act of 1976: A Critique of* Lotus v. Paperback, 6 HIGH TECH. L.J. 209, 264 (1991) (discussing user interfaces).

the user interface portion of a program is copyrightable as part of an audiovisual work.[38] The principles to be applied to this aspect of computer software are fairly well-settled and therefore will not be addressed in this Article.[39]

The hardware drivers are not likely candidates for copyright protection under existing copyright principles. Less than de minimis creativity is involved in the creation of these small programs.[40] Typically, they are written within strict hardware and software engineering constraints.[41] These constraints limit the possibilities of expression, making it improbable that the expression chosen is original.[42] Computer programs do not pose new problems for copyright law on this score and therefore will not be addressed in this Article.[43]

More difficult computer software copyright questions involve the data processing section. Inside the data processing section lies the collection of algorithms and data structures that actually perform the

---

38. We use user interface in its narrow sense: referring to the screen display. *See supra* note 36; Registration and Deposit of Computer Screen Displays, 53 Fed. Reg. 21, 817-23 (1988); *see also* Atari Games Corp. v. Oman, 888 F.2d 878 (D.C. Cir. 1989) (deciding that the visual appearance of a video game was proper subject matter of copyright); Apple Computer, Inc. v. Microsoft Corp., 821 F. Supp. 616 (N.D. Cal. 1993) (denying "look and feel" protection for Apple Macintosh user interface); Apple Computer, Inc. v. Microsoft Corp., 799 F. Supp. 1006 (N.D. Cal. 1992) (modifying previous order to protect specific icons that were identically copied); Broderbund Software, Inc. v. Unison World, Inc., 648 F. Supp. 1127 (N.D. Cal. 1986) (holding that copyright protection extends to a program's audiovisual display). Public domain elements and scènes à faire act as limitations on user interface copyright. *See, e.g., Atari*, 888 F.2d at 886.

39. *See, e.g., Atari*, 888 F.2d at 885-86; *Apple*, 799 F. Supp. at 1020; *Apple*, 821 F. Supp. at 619, 626.

40. *See* NEC Corp. v. Intel Corp., 10 U.S.P.Q. 2d 1177, 1178 (N.D. Cal. 1989) ("It may well be that, considered alone, several of the microsequences in Intel's microcode consist of forms of expression directed solely by functional considerations lacking even minimal creativity.") (internal quotations omitted); *cf.* Feist Publications, Inc. v. Rural Tel. Serv. Co., 499 U.S. 340, 348 (1991) (discussing requirement that work exceed de minimis standard of creativity to achieve copyrightability).

41. When writing a hardware driver, the programmer must use as few instructions as possible in order to achieve peak execution speed. Also, the sequence of events that the driver must request of the hardware are exacting requirements with no room for variation: the requirements are set by the hardware itself, not by the expressive choice of the programmer.

42. *See Feist*, 499 U.S. at 348 (discussing the relationship between choices and originality; where there are a limited number of expressive choices, there is a presumption of less originality); Menell, *supra* note 14, at 1086; Marci A. Hamilton, *Justice O'Connor's Intellectual Property Opinions: Currents and Crosscurrents*, 13 WOMEN'S RTS. L. REP. 71, 75-76 (1991) (stating that where there is "no choice of arrangement . . . there can be no creativity" in a compilation).

43. *See* Computer Assocs. Int'l, Inc. v. Altai, Inc., 982 F.2d 693, 708 (2d Cir. 1992); *see also* Miller, *supra* note 14, at 1005 n.134; Samuelson et al., *supra* note 11, at 2358 n.197.

computations that users demand. For example, when a program that simulates an airplane cockpit is running, the data processing section calculates all of the geometry necessary to present a picture to the user and when finished, presents the picture by using the video display hardware driver. Thus, the data processing section of a program is where most of the creativity and critical advancement in the programming art occurs.

The courts have created their own terminology for computer software cases in an attempt to deal with the difficulty of applying the law of copyright infringement to programs. Generally, copyright infringement encompasses both literal and non-literal copying.[44] Literal copying is comparatively easy to identify, even in the computer software context. It occurs when either the source code or the object code are copied verbatim.[45] Non-literal copying in the literary works cases can also be fairly easy to identify. For example, an author of a play could copy certain elements from another play without taking the verbatim expression of the former play, thereby giving rise to a finding of substantial similarity and infringement.[46] Specific terms are available to identify the noncopyrightable elements that should be exempted from a finding of infringement between two plays, such as scènes à faire,[47] plot, and theme.[48] The music industry offers many examples of non-literal copying. In those cases, musicologists testify to the non-literal similarities between two musical compositions.[49]

---

44. *See* Nichols v. Universal Pictures Corp., 45 F.2d 119, 121 (2d Cir. 1930).

45. Source code is the human-readable version of a program that is then translated into a machine-readable format called object code using another program called a compiler. *Apple Computer, Inc. v. Franklin Computer Corp.*, 714 F.2d 1240 (3d Cir. 1983), established that copyright in source code covers its object code translation. *See* 1984 Compendium of Copyright Office Practices § 321.03 ("The Copyright Office considers source code and object code as two representations of the same computer program."). A computer itself can compare two files to see if they are exactly the same. *See* THE PROTECTION OF COMPUTER SOFTWARE — ITS TECHNOLOGY AND APPLICATIONS 122-23 (Derrick Grover ed., 1989).

46. *See, e.g., Nichols*, 45 F.2d at 121.

47. In general, if the structure and organization of two plays are too similar, it may be found that one infringes the other. *See, e.g.*, Sheldon v. Metro-Goldwyn Pictures Corp., 81 F.2d 49 (2d Cir. 1936) However, some stories share aspects that are considered essential components for any original story in a similar setting, called scènes à faire. For example, any play about pirates would likely have images of revelry at some exotic port of call. Thus, the fact that two plays contained a similar scene is not in itself evidence of copying when comparing the sequence and organization of both plays. *See Nichols*, 45 F.2d at 121.

48. *See Altai*, 982 F.2d at 709.

49. *See* Bright Tunes Music Corp. v. Harrisongs Music, Ltd., 420 F. Supp. 177 (S.D.N.Y. 1976), *aff'd*, 722 F.2d 988 (2d Cir. 1983); *see also* Miller, *supra* note 14, at 1009 ("Additionally, a court is free to appoint an expert to analyze the more technical issues in a case, and thus need not confront complex issues unaided.").

In computer cases, non-literal infringement has been harder to identify. It is difficult enough for courts to compare the source or object codes of two programs without resorting to software tools to determine if they are identical. The difficulty in comparing two non-identical but similar programs has driven courts to devise two tests: the structure, sequence, and organization test and the abstraction-filtration test.

The legal notions of "structure, sequence, and organization" and "program structure" were introduced in an attempt to define what could be copied non-literally.[50] These terms seem to be a neutral, uncontroversial choice, but they obscure more than they clarify. Both tests employ a legal description of computer software concepts that glosses over the fundamental question: what are the components of any program that may or may not be copyrightable? The legal terms of art must be refined in light of the underlying computer science so that they do not overprotect or underprotect copyrightable aspects of computer software.

The idea of comparing computer programs in a non-literal infringement analysis was first introduced by the Third Circuit in *Whelan v. Jaslow*, where all aspects of program "structure, sequence, and organization" were considered copyrightable expression.[51] The Second Circuit, in *Computer Associates International, Inc. v. Altai, Inc.*, found program structure worthy of protection but took a more restrictive approach than *Whelan*.[52] Drawing upon the abstractions test devised in *Nichols v. Universal Pictures Corp.*,[53] the court began its analysis of program

---

50. *See, e.g., Altai*, 982 F.2d 693 (introducing "program structure" as a legal term); Whelan v. Jaslow, 797 F.2d 1222 (3d Cir. 1986) (introducing "structure, sequence, and organization" as a legal term).

51. *Whelan*, 797 F.2d at 1224-45.

> The [district] court also found that Rand Jaslow had not created the Dentcom system independently, and that the Dentcom system, although written in a different computer language from the Dentalab, and although not a direct transliteration of Dentalab, was substantially similar to Dentalab because its structure and overall organization were substantially similar.

Id. at 1228-29. In its affirmance, the Third Circuit went on to quote the district court opinion: "The conclusion is thus inescapable that the detailed structure of the Dentalab program is part of the expression, not the idea, of that program . . . . Because there are a variety of program structures through which that idea can be expressed, the structure is not a necessary incident to that idea." *Id.* at 1239-40.

52. The court in *Altai* stated:

> As we have already noted, a computer program's ultimate function or purpose is the composite result of interacting subroutines. Since each subroutine is itself a program, and thus, may be said to have its own "idea," *Whelan's* general formulation that a program's overall purpose equates with the program's idea is descriptively inadequate.

Altai, 982 F.2d at 705.

53. 45 F.2d 119, 119 (2d Cir. 1930) (holding that background characteristics of setting and general aspects are not considered in assessing similarity).

structure by attempting to filter out uncopyrightable elements of the program. Unfortunately, the court did not incorporate the computer science terms that could define what the filtered elements ought to be. Instead, the court in *Altai* employed existing copyright terminology and doctrine applicable to other types of works, such as merger, scènes à faire, and public domain restrictions.[54] The analogy once again, however, proved to be rough and likely misleading. The court's attempt to explain what scènes à faire might be in the computer software context is inevitably unsuccessful.[55]

Both courts would have been better served by piercing the surface of computer science to understand the components comprising a program, which include "data structures," "algorithms," and "languages."[56] By failing to identify these components, protection for the program structure threatens to provide a de facto monopoly over unnamed elements that may not deserve copyright protection.

To make coherent copyright protection in computer programs, Professor William Patry has proposed that programs be understood as compilations of uncopyrightable elements.[57] He has made an important contribution to the discourse on computer programs and copyright. Yet, like *Whelan*, this approach runs the risk of treating computer programs as though they have no underlying components that must be judged separately for copyright purposes. The risk of this approach is that it too does not draw a precise line between what is uncopyrightable and what is copyrightable *within* a program. Even if two expressions are compilations, it is often necessary to determine whether specific aspects of the compilations are expression worthy of protection. Unless the courts acquire more accurate terminology and understanding of the components of computer programs, they are likely to sweep less deserving aspects of the program under a ruling for copyright protection of the program as a whole. Thus, we turn to a brief primer on computer science terms that would assist courts in their copyright decisions.

---

54. *See Altai*, 982 F.2d at 707-10.

55. The court in *Altai* correctly points out that external factors like mechanical considerations of the computer hardware may limit programmer expression, see *supra* note 39, but then includes "widely accepted programming practices within the computer industry" as an example of scènes à faire, without giving further explanation. *Id.* at 709-10.

56. *Cf.* Davis, *supra* note 12, at 329 ("[R]eferences to 'structure, sequence, and organization' as used in *Whelan* are inherently technically defective and often unanswerable.").

57. *See* William F. Patry, *Copyright and Computer Programs: It's All in the Definition*, 14 CARDOZO ARTS & ENT. L.J. 1 (1996). Note that the use of the word "compilation" here is in the copyright law sense, not the computer science sense.

A. A Brief Primer on Computer Science Relevant to Copyright Law

The computer science concepts of "algorithm" and "data structure" are highly relevant to the software copyright discourse. Algorithms and data structures are two elements of computer software that should receive separate consideration in software cases. Algorithms are not protected by copyright because they are procedures. A data structure, explained in more detail below, is an organization of data in computer memory which permits a given algorithm to work. Data contained in a data structure might be eligible for copyright as a compilation.[58] However, the dependence of algorithms on data structures means that copyright protection that limits the use of a given data structure may in turn limit the use of a corresponding algorithm. Therefore, a data structure necessary to use a particular algorithm should not be copyrightable. To fully understand this argument, algorithms and data structures will be defined and their relationship examined.

1. What Is an Algorithm?

An algorithm is "[a] procedure or set of rules for calculation or problem-solving."[59] Because the 1976 Copyright Act explicitly excludes procedures from protection, algorithms are not copyrightable.[60] Algorithms are fundamental to computer science because they are the calculation procedures that a computer can follow at an extremely high speed to reach a desired result. The procedure may require a human to develop it and prove how it works, but it requires only a calculating machine to perform it and reach the result. This is the "magic" that makes the computer such an extraordinary tool.

Take, for example, an algorithm called Mailsort, consisting of a simple set of procedures for alphabetizing a stack of mail.[61] Direct

---

58. *See infra* note 86 and accompanying text.

59. THE NEW SHORTER OXFORD ENGLISH DICTIONARY 50 (Lesley Brown ed., 1993).

60. The 1976 Copyright Act excludes processes, procedures, and methods from copyright protection. *See* 17 U.S.C. § 102(b) (1994); *see also* Donald S. Chisum, *The Patentability of Algorithms*, 47 U. PITT. L. REV. 959 (1986); Pamela Samuelson, *Brief* Amicus Curiae *of Copyright Law Professors in* Lotus Development Corp. v. Borland International, Inc., 3 J. INT. PROP. L. 103 (1995); Pamela Samuelson, *Benson Revisited: The Case Against Patent Protection for Algorithms and Other Computer Program-Related Inventions*, 39 EMORY L.J. 1025 (1990) [hereinafter Samuelson, *Benson Revisited*].

61. Here is an algorithm to complete the task:

START MAILSORT:

Step 1. If the out-pile is empty, place the top envelope of the in-pile on the out-pile, and go to Step 5.

Step 2. Read the name on the top envelope of the in-pile and call that the "in"

copyright protection of this algorithm is barred by the Copyright Act's express elimination of processes or procedures from its zone of protection.[62] In addition, the Copyright Office expressly forbids registering any algorithm for copyright protection.[63] However, expression describing that algorithm, whether it is in English or a computer language, is protected by copyright.[64] As a result, an originator could not exert a monopoly over the Mailsort algorithm process, but could obtain copyright protection for an original description of it.

Every program involves an algorithm of some kind.[65] Obviously, more difficult problems are solved by using more sophisticated algorithms, and simple problems can often be solved more quickly using sophisticated algorithms as well. In the case of the simple mail sorting algorithm described above, the larger the pile of mail gets, the longer it takes to sort. In fact, given N pieces of mail, the time it would take to sort using the Mailsort algorithm would be proportional to N^2, which means that it is very slow. One proposed solution to this problem has been a better general purpose sorting algorithm called "Quicksort," which has been widely described in computer literature.[66]

---

envelope.

Step 3. Starting at the top of the out-pile, read through names until you reach an envelope whose name alphabetically follows the "in" envelope.

Step 4. Insert the "in" envelope into the out-pile right before that envelope.

Step 5. If the in-pile still has an envelope in it, then go to Step 2.

Step 6. If the in-pile has no envelopes left, then the algorithm has FINISHED.

This is a sorting algorithm. It may take a human a minute to understand that this algorithm will work, but having done so, it takes no additional thought to execute it. Except for the first envelope, which is placed on the empty out-pile to start, this algorithm works by taking each successive piece of mail, running through the out-pile until the correct place is found, inserting that envelope, and then picking up the next envelope from the in-pile. The algorithm is finished when the in-pile is empty.

62. *See* 17 U.S.C. § 102(b) (1994).

63. *See* 1984 Compendium of Copyright Office Practices § 325.02(c) (1990).

64. The expression describing a system is copyrightable, but the system itself resides in the public domain. *See* Baker v. Selden, 101 U.S. 99, 103 (1879).

65. The typical software product is a program that contains a number of different algorithms. Each algorithm is used to solve a particular aspect of the functionality problem presented by the software application.

66. *See* ALFRED V. AHO ET AL., DATA STRUCTURES AND ALGORITHMS 260 (1983). It was originally reported by Charles A.R. Hoare. *See* Charles A.R. Hoare, *Quicksort*, 5 COMPUTER J. 1, 10-15 (1962).

Quicksort employs an algorithm that sorts at a much faster speed. It requires a running time proportional to N times the logarithm of N (algebraically Nlog(N)). As N gets very large, Nlog(N) becomes a small fraction of N^2. Consider N=1, 10, 100, or 1000. N^2 would be 1, 100, 10,000, or 1,000,000 while NLog(N) would be 0, 10, 200, or 3000. Therefore, for 1000 pieces of mail, Quicksort would run over 300 times faster than Mailsort. The fact that there are better sorting algorithms than Mailsort, however, is inconsequential to the legal argument. One can imagine a sophisticated algorithm that is the only known solution to an important problem.

From the viewpoint of a software vendor, a faster algorithm can make one software product outperform another at the same function, providing a major competitive advantage. Therefore, there is a great incentive for software vendors to protect their algorithms in whatever way possible, including by characterizing something close to the algorithm itself as expression about the algorithm and hence protected. Yet, regardless of whether the algorithm is complicated or simple, it is not deemed worthy of copyright protection. Protection for significant advances in algorithm science may, however, reside under patent law.[67]

2. What Is a Data Structure?

Data structures pose a different problem for computer software copyright analysis, and reveal an inherent anomaly in how the 1976 Copyright Act applies to computer programs. Even if courts properly exclude algorithms from copyright protection, they may inadvertently provide copyright protection for algorithms by protecting certain data structures under the aegis of program structure. This untoward result should be avoided by refusing to protect any data structure that is necessary to a particular algorithm.

Data resides in computer memory. It is easiest to visualize computer memory as being like a wall of post office boxes. Each box has a unique numerical "address," but the content of each box can be changed at will. The computer's central processing unit ("CPU")[68] can look at a location in memory specified by the address and retrieve the piece of data residing there. Alternatively, the CPU can store a piece of data at a given address. One part of a program organizes the data in such a way that the other part, which embodies the algorithm, can work with it. The more sophisticated the data organization, the more opportunity there is to employ sophisticated algorithms that run faster and do more. This organization is referred to as a data structure.[69]

---

67. The law surrounding algorithm patents has been in flux. *See* RAYMOND NIMMER, THE LAW OF COMPUTER TECHNOLOGY § 2.06 at 2-28 (1992); Chisum, *supra* note 60; Samuelson, *Benson Revisited*, *supra* note 60. *Compare* Gottschalk v. Benson, 409 U.S. 63, 73 (1972) (denying a patent for an algorithm to convert binary-coded decimal numbers into pure binary numerals), *with* Diamond v. Diehr, 450 U.S. 175, 187 (1981) (holding that a computer program designed to work in conjunction with a rubber molding machine was patentable), *and* U.S. Patent No. 4,744,028, issued to N. Karmarkar and assigned to AT&T Bell Labs (for an improvement on the "simplex method" of solving simultaneous linear equations).

68. In a desktop computer, the CPU is embodied in an integrated circuit, or "chip." The Intel Pentium chip and Motorola Power PC chip are examples. *See generally* DAVID A. PATTERSON & JOHN L. HENNESSEY, COMPUTER ORGANIZATION AND DESIGN 14, 270 (1994).

69. *See* AHO ET AL., *supra* note 66, at 13.

The Mailsort algorithm explained above employs a data structure known as an array. Each envelope in either pile physically follows the other in succession. This is the simplest way to organize data in computer memory. In terms of the post office box visualization, the in-pile would initially be a row of full boxes ("in-boxes") and the out-pile would be a row of empty boxes ("out-boxes"). When running Mailsort, each successive in-box would be emptied and its contents placed somewhere in the row of out-boxes. Obviously, for each insertion into the middle of the row of out-boxes, the contents of all subsequent out-boxes would have to be shifted down by one box to make room. Figure 1 illustrates how a data structure is transformed by the Mailsort algorithm.

Figure 1: Simple Data Arrays[70]

	Before Sorting	
	Address	**Data**
TOP→	1	Rehnquist
	2	Scalia
	3	O'Connor
	4	Kennedy
	5	Ginsburg
	6	Souter
	7	Thomas
	8	Breyer
	9	Stevens

	After Sorting	
	Address	**Data**
TOP→	1	Breyer
	2	Ginsburg
	3	Kennedy
	4	O'Connor
	5	Rehnquist
	6	Scalia
	7	Souter
	8	Stevens
	9	Thomas

All algorithms depend on data structures. In fact, certain data structures are uniquely necessary to certain algorithms. For example, to run a different algorithm such as Quicksort,[71] the array organization described above is insufficient; a different data structure is required.

---

70. Note that TOP remains Address 1 after sorting.
71. *See* AHO ET AL., *supra* note 66, at 260.

Quicksort runs on a data structure called a "linked list,"[72] and can only work on data organized into a linked list data structure.

The linked list is a common data structure that is significantly more complicated than the array described above. First, it requires grouping the post office boxes in pairs and calling each pair a "cell." Physically, there is still a unique numerical address for each box, but the contents are handled differently depending on whether the box is the first or the second constituent of the cell. Returning to mail sorting, an envelope may be placed (with a name on it) without regard to order in the first box of each cell — that is the data. A slip of paper that has a numerical address of another box written on it called a "pointer" is placed in the second box. That box is the data box of the next cell in the list. This results in two boxes being used for each envelope. Each odd box has an envelope in it while each even one contains a pointer. Figure 2 illustrates how our data set of the Justices would be handled as a linked list data structure under Quicksort.

In Figure 2, there is a list of nine names, each having a pointer to the next cell in the list. Given a particular cell, the pointer is the address of the next cell in the list. Therefore, to store the nine names in a linked list, eighteen memory locations are required. One "follows the pointers" to read out the list. Starting at TOP (box 1), look at the data in that cell (Rehnquist), then go to the address referenced by the cell's pointer (box 3), look at the data contained there (Scalia) and then follow its pointer (box 5), and continue until null is reached. In so doing, we have moved down a linked list and read the names out of alphabetical order. When the Quicksort algorithm is run, the names stay in their current boxes. Instead of moving them, Quicksort reassigns the pointers, including TOP. When Quicksort is finished, it is possible to start at the new value for TOP (box 15), follow the pointers, and read the names out in alphabetical order. See Figure 2. None of this is possible without the linked list data structure.

---

72. *See generally* DONALD E. KNUTH, THE ART OF COMPUTER PROGRAMMING, VOL. I: FUNDAMENTAL ALGORITHMS (1968).

Figure 2: Linked Lists

	Before Sort: Top is Address 1	
	Address	Data
TOP→	1	Rehnquist
	2	3
	3	Scalia
	4	5
	5	O'Connor
	6	7
	7	Kennedy
	8	9
	9	Ginsburg
	10	11
	11	Souter
	12	13
	13	Thomas
	14	15
	15	Breyer
	16	17
	17	Stevens
	18	null

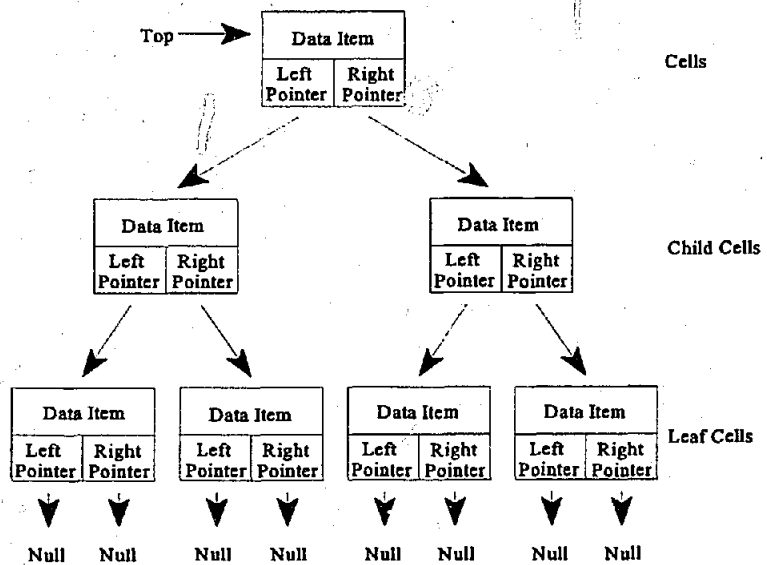
	After Sort: Top is Address 15	
	Address	Data
Data Item→	1	Rehnquist
Pointer to Next Cell→	2	3
	3	Scalia
	4	11
	5	O'Connor
	6	1
	7	Kennedy
	8	5
	9	Ginsburg
	10	7
	11	Souter
	12	17
	13	Thomas
	14	null
New TOP after sort→	15	Breyer
	16	9
	17	Stevens
	18	13

Note that the data structures presented here are so basic that they are most likely part of the public domain, but that does not change the legal argument. Advances in programming technique and algorithm development involve creating complex data structures to facilitate equally complex algorithms.

Computer science has developed more complex data structures in order to enable computers to solve more complicated problems. For example, a "tree" data structure is used for running very fast database

algorithms.[73] This data structure groups boxes by threes, with each cell consisting of a data box and two pointer boxes. One pointer points to a "left child" cell and the other to a "right child" cell, as shown in Figure 3. Another type of data structure is the "graph" structure.[74] Graph data structures are useful for running pattern-matching algorithms, such as those used in speech recognition or artificial intelligence.[75] Although the examples of data structures provided here may not be original enough to be copyrightable, that does not mean that, a priori, all data structures lack originality. Any restrictions on the use of a given data structure will necessarily restrict the use of any new algorithms designed to take advantage of that particular data structure.

Figure 3: "Tree" Data Structure[76]



73. *See* AHO ET AL., *supra* note 66, at 75.

74. *See id.* at 3.

75. *See* Y. ANZAI, PATTERN RECOGNITION AND MACHINE LEARNING 3 (1992).

76. Note that the tree structure can be many levels deep, and that the Leaf Cells always point to null.

Though this clarifies the importance of data structure to the algorithm copyright question, it is unfortunately still unclear whether current law has effectively carried out congressional intent to prohibit the monopolization of algorithms through copyright.[77] Even though algorithms have been explicitly excluded from copyright protection, defining program structure in a manner that does not effectively exclude the embedded algorithm from protection may have the unintended effect of defeating that exclusion. Furthermore, even if courts understand that algorithms may not be copyrighted, they may not understand that the protection of a data structure that is necessary to a particular algorithm may provide a de facto monopoly over the algorithm.

B. The Legal Implications of the Dependence of Algorithms on Data Structures

If algorithms should not receive copyright protection, neither should any data structures necessary to particular algorithms. As demonstrated above, algorithms are dependent on data structures because data structures organize data within computer memory for manipulation by an algorithm. Creating an algorithm in a computer programming language requires organizing the data in an appropriate data structure.[78] The implications of this fact are enormous: restrictions on the use of a given data structure will necessarily restrict the use of dependent algorithms.

If a court holds that two programs are substantially similar in program structure because the data structures are the same, that ruling will confer a de facto monopoly over any algorithm that requires that particular data structure to run.[79] For example, consider the linked list of

---

77. *See supra* notes 62-64 and accompanying text.

78. A computer science reference text explains:

> [W]e shall design algorithms in terms of ADT's [abstract data types, e.g., integer, real, character, boolean], but to implement an algorithm in a given programming language we must find some way of representing the ADT's in terms of the data types and operators supported by the programming language itself. To represent the mathematical model underlying an ADT we use data structures, which are collections of variables, possibly of several different data types, connected in various ways.

AHO ET AL., *supra* note 66, at 13.

79. The data structures described in this Article are extremely common and may be part of the public domain, but no case law has authoritatively said so other than *Altai* in dicta. *See* Computer Assocs. Int'l, Inc. v. Altai, Inc., 982 F.2d 693, 710 (2d Cir. 1992) (referring to "common programming practices"). More importantly, future algorithm development will go hand in hand with future data structure development. A newly crafted data structure that permits a new algorithm to run could be sufficiently original; however, it should not be copyrightable since doing so would restrict the use of the newer algorithm designed to run on it.

the names of the Justices described above. Assuming for argument's sake that the linked list structure is copyrightable, then the only party able to use the Quicksort algorithm — or any other algorithm dependent upon the linked list data structure — is the one holding the copyright to the linked list data structure. This result may dramatically impede the development of computer applications by permitting a lengthy de facto monopoly to attach to a particular algorithm.[80]

By introducing simple computer science terms of art such as "data structure" into the legal discourse, a court can make clear that separate analysis must be given to the data structure when examining the copyrightable aspects of computer programs. A conclusion that a given program is in some respects copyrightable should not automatically compel the same legal conclusion with respect to the data structure.

Data structures are potentially protectable under the 1976 Copyright Act's provisions for the protection of compilations.[81] The Act designates the "selection" and "arrangement" of data as the indicia for determining when copyright protection is appropriate and further requires that the selection or arrangement of the information be original.[82] The data structure may be expressed in code in any one of a number of ways, which — absent merger[83] — is protected from illicit copying, whether literal or nonliteral. Data itself is never protected.[84] Nor does the effort and investment expended in collecting data justify protection.[85] But the selection of what data to include and the chosen arrangement of that data are legitimate candidates for protection under a compilation theory.[86]

A data structure has two components: the data residing in the cells and a structure that defines how the cells are arranged. The arrangement of the cells poses thorny problems for copyright law because of the data

---

80. *See* 17 U.S.C. § 302(a),(c) (1994) (setting the duration of copyright protection to the life of the author plus 50 years or 75 years if a work for hire).

81. *See* 17 U.S.C. § 103(a) (1994).

82. *See* William F. Patry & Shira Perlmutter, *Fair Use Misconstrued: Profit, Presumptions, and Parody*, 11 CARDOZO ARTS & ENT. L.J. 667 (1993). *Compare* Feist Publications, Inc. v. Rural Telephone Serv. Co., 499 U.S. 340, 350, 354 (1991) (finding that a telephone book did not meet the modicum of originality), *with* Key Publ'g, Inc. v. Chinatown Today Publ'g Enters., Inc., 945 F.2d 509, 514 (2d Cir. 1991) (finding the selection process for Chinese yellow pages phone directory sufficiently original).

83. "Merger" is the doctrine holding when expression of an idea is inseparable from the idea itself, no copyright protection is given. *See Altai*, 982 F.2d at 707. Thus, if there are only a few ways to code a particular data structure, merger would preclude copyright protection, even from literal copying. *See id.* at 708.

84. *See Feist*, 499 U.S. at 347-48.

85. *See id.* at 349 (rejecting "sweat of the brow" as justification for copyright).

86. *See id.* at 348-49.

structure's potentially unique fit with particular algorithms.[87] If the arrangement is the only one that works with a particular algorithm, copyright protection for the arrangement confers monopoly power over the algorithm. Ironically, protecting a data structure in this manner uses the compilation doctrine to dominate the market in a nonprotectable subject matter — the algorithm. Applying the plain terms of the statute thus leads to a logical conundrum: protection for a compilation confers protection on an unprotectable process. Courts are left with two options: either find the structural aspect of the data structure protected under the compilation provisions of the Act and thus confer a monopoly on the algorithm, or find the structural aspect unprotectable because it confers an unacceptable monopoly on the algorithm.

The solution to this conundrum was not provided by Congress in the 1976 Copyright Act. Rather, courts are left to resolve the conflict, using several possible guides including the Constitution, legislative history, copyright policy, and the relative fit between copyright and patent protection for computer software elements. Taken together, these criteria strongly suggest that courts should choose the second solution: the structural aspect of data structures (considered separately from the data itself) that are necessary to particular algorithms should not be granted copyright protection.

The Constitution grants Congress the power to make laws "[t]o promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries."[88] Contrary to common perceptions regarding monopolies, the copyright monopoly is, as a constitutional matter, supposed to spur scientific and literary development.[89] Indeed, the

---

87. That an author may select the data to be arranged in a particular data structure may imply original authorship with regard to the selection step. *See supra* note 71 and accompanying text. The selection of data that resides within the cells does not pose the same potential for algorithm monopoly because algorithms are dependent on the arrangement of the data and not on any particular instance of the data itself. Considering the linked list example, it may be original authorship to have selected the names of justices, but that is separate from arranging them in a linked list data structure. By analogy, selecting particular judges for a specialized phone book may satisfy the requirement of originality, but the use of numbered pages or an alphabetical listing does not. The focus here is upon the arrangement of the data.

88. U.S. CONST. art. I, § 8, cl. 8.

89. The Supreme Court has noted:

> Creative work is to be encouraged and rewarded, but private motivation must ultimately serve the cause of promoting broad public availability of literature, music, and the other arts. The immediate effect of our copyright law is to secure a fair return for an "author's" creative labor. But the ultimate aim is, by this incentive, to stimulate artistic creativity for the general public good.

Twentieth Century Music Corp. v. Aiken, 422 U.S. 151, 156 (1975). For a discussion of

congressionally-created Commission on New Technological Uses of Copyrighted Works has spoken on the importance of furthering innovation in the software industry.[90] Courts may presume that Congress intended to follow constitutional directives, and therefore that the statute is intended to effect a regime whereby innovative development of computer software is enhanced rather than diminished. Because the progress of computer science development is impeded by conferring a de facto monopoly protection on algorithms through protection of necessary data structures, this consideration weighs against protecting the data structures necessary to certain algorithms.[91]

Copyright policy generally favors extending the copyright monopoly only to the extent that doing so will spur further original creations.[92] The statute does not provide authors absolute rights in copyrighted works, but instead explicitly recognizes certain exceptions to the copyright holder's monopoly where exercise of those rights would not induce more authorship in the field. Most notably, the "fair use" affirmative defense permits individuals to use portions of copyrighted works for certain socially valuable purposes even if the copyright owner were to object.[93]

---

the importance of encouraging the software industry to the United States economy, see *Computer Software Rental Act of 1988: Hearing on S. 2727 Before the Subcomm. on Patents, Copyrights and Trademarks of the Senate Comm. on the Judiciary*, 100th Cong. 2 (1988).

90. *See* CONTU, *supra* note 31, at 10-11.

91. *See supra* text accompanying notes 65-66 (explaining how better algorithms are the key to better software products).

92. For example, the fair use doctrine allows a critic to quote portions of a copyrighted work he is reviewing without obtaining the author's permission. *See* 17 U.S.C. § 107 (1994). This type of approach was explicitly recommended by CONTU in the case of computer software copyright. *See* CONTU, *supra* note 31, at 12; *see also* Menell, *supra* note 14, at 1049 (finding copyright to be a solution to the public goods problem); *see also* Computer Assocs. Int'l, Inc. v. Altai, Inc., 982 F.2d 693 (2d Cir. 1992) (discussing monopoly problem with protecting data structures).

93. *See* 17 U.S.C. § 107 (1994); *see also* Pierre N. Leval, *Toward a Fair Use Standard*, 103 HARV. L. REV. 1105, 1108 (1990) (finding fair use doctrine justified by the purposes of the Copyright Clause); H.R. REP. NO. 102-836, at 3 (1992), *reprinted in* 1992 U.S.C.C.A.N. 2553, 2555 (quoting Leval's article). Also note that 17 U.S.C. § 117 explicitly allows owners of an authorized copy of a program to make a "backup" copy in case the computer accidentally erases the first copy.

Other restraints on the author's rights over the work also exist. For example, American copyright law recognizes the "first sale" doctrine which limits copyright holders' rights over the corporeal versions of copyrighted works. However, the U.S. has been extremely slow to recognize the moral rights mandated by the Berne Convention, which would increase the scope of the copyright author's monopoly.

> [Adherence to the Berne Convention] will not, and should not, change the current balance of rights between American authors and proprietors, modify current copyright rules and relationships, or alter the precedential effect of prior decisions . . . . The provisions are intended neither to reduce nor expand any rights that may now exist,

Such a defense has been accepted in American copyright law with the understanding that such uses do not appreciably decrease the author's opportunity to benefit from her work. To the contrary, it spurs even further creative development by fostering a competitive environment. Likewise, prohibiting the author of a data structure from exercising monopoly control over an algorithm through control of the data structure copyright spurs creative development in the field.

Copyright protection for data structures that are necessary to particular algorithms also creates a potential conflict between the copyright and patent spheres.[94] If the algorithm tied to a particular data structure is patented, then the copyright protection afforded the structure permits the author to control the use (and profitability) of the patented algorithm. Thus, copyright protection of a data structure necessary to a particular algorithm can impede innovation in algorithms by diminishing the power of the algorithm's patent holder during the comparatively short patent term of protection to exploit the work.[95] This devalues the patent

---

nor to create any new rights under federal or state statutes or the common law.

BERNE IMPLEMENTATION ACT, S. REP. NO. 100-352, at 10 (1988), *reprinted in* 1988 U.S.S.C.A.N. 3706, 3715. *But see* 17 U.S.C. § 109(b)(1)(A) (1994) (prohibiting rental of lawfully purchased copies of phonorecords and computer programs).

94. A collision between patent law and copyright law would also occur if both laws considered data structures as subject matter. Patent law has far more stringent requirements than copyright law before conferring a monopoly. It is important to note that a patent was issued for a "data structure" to Lowry, who then assigned it to Digital Equipment Corp. *See In re* Lowry, 32 F.3d 1579 (Fed. Cir. 1994). This was a specific organization of data for use within computer-aided design and manufacturing systems. The patent examiner initially rejected the patent application as improper subject matter, but was reversed by the Federal Circuit. The Federal Circuit held that because the "data structure" resided in computer memory, it was a method to be used with a machine and therefore the case fell under the holding of *Diamond v. Diehr*, 450 U.S. 175 (1981). *See id.* Patentability of data organization methods is beyond the scope of this Article, although the same relationship between data structures and algorithms exists regardless of what law is applied. *See supra* note 71 and accompanying text.

95. Note that there are many cases of one programmer making an improvement to another's existing algorithm. Those improvements would be less profitable if the use of the data structure they both use were restricted by copyright.

For discussion of the conflict between patent and copyright coverage of computer software, see Reichman, *Legal Hybrids*, *supra* note 11, at 2486; Samuelson, *Benson Revisited*, *supra* note 60; John Swinson, *Copyright or Patent or Both: An Algorithmic Approach to Computer Software Protection*, 5 HARV. J.L. & TECH. 145, 212 (1991); see also Dennis S. Karjala, *Recent United States and International Development in Software Protection (Part 1)*, 16 EUR. INTELL. PROP. REV. 13 (1994); *Recent United States and International Developments in Software Protection (Part 2)*, 16 EUR. INTELL. PROP. REV. 58 (1994) (arguing that copyright law should be applied to computer programs in a way that avoids distortion of the distinctions between patent and copyright law).

Computer languages as intellectual property may be better governed by patent law. An argument can be made that computer languages are patentable subject matter. A

protection and results in disincentives to produce newer and more sophisticated algorithms which are dependent on the data structure and a deceleration in the progress of computer science. Given the high cost of bringing software to market[96] — in the millions of dollars — such a devaluation of patent protection and the resulting disincentives are likely to cause considerable harm to the progress of computer science.[97] Thus, courts must be extremely careful to recognize the algorithms and data structures that exist within the program structure and also be sensitive to the implications of approving copyright protection for a particular data structure.

In sum, all of the enumerated indicators weigh against the copyrightability of data structures necessary to particular algorithms. Although Congress would do well to make this rule clear in the statute, the legal analysis is available which can result in the proper allocation of property rights in data structures and algorithms — data structures necessary to particular algorithms should not be given copyright protection even if they satisfy the requirements of a compilation.[98]

---

computer language can be considered an algorithmic process for converting one set of symbols ("source code") into another set ("object code"). Algorithms have been found to be patentable subject matter under 35 U.S.C. § 101. *See* Karmarkar, U.S. Patent No. 4,744,028. Languages developed in corporate R&D labs, like C, C++, and Java, are sufficiently novel upon their release to satisfy the priority requirements of patent law under 35 U.S.C. § 102. Some languages, like LISP, C, C++, and Java, were radical departures from the prior programming art and therefore satisfy the statutory requirement for non-obviousness. *See* 35 U.S.C. § 103 (1994). Furthermore, for those kinds of languages the objective indicators of market success and long-felt need might also support a finding of non-obviousness. *Cf.* Hybritech, Inc. v. Monoclonal Antibodies, Inc., 802 F.2d 1367, 1382 (Fed. Cir. 1986) (discussing market success as an indicator of non-obviousness). Some of the problems associated with a monopoly over a computer language (see Part IV.B) may be mitigated in the patent sphere through the doctrine of patent misuse. Although controversial, this aspect of patent law could prevent a patent holder in a computer language from using that monopoly to block consumers from migrating toward competing languages or program applications. *Cf.* Windsurfing Int'l v. AMF, Inc., 782 F.2d 995, 1001 (Fed. Cir. 1986) (discussing the patent misuse doctrine).

96. "[S]oftware development now costs five to 15 times more than hardware development on a typical embedded processor project." Bernard Cole, *Embedded Systems: Part I: Chips and Tools*, ELECTRONIC ENGINEERING TIMES, Feb. 5, 1996, at 45, *available in* 1996 WL 7976419.

97. *See Computer Software Rental Amendments Act of 1988: Hearing on S. 2727 Before the Subcomm. on Patents, Copyrights and Trademarks of the Senate Comm. on the Judiciary*, 100th Cong. 2 (1988) (statement of Sen. Orrin G. Hatch); *see also* Peter S. Menell, *An Analysis of the Scope of Copyright Protection for Application Programs*, 41 STAN. L. REV. 1045, 1058-71 (1989) (discussing economic analysis of computer program copyright issues).

98. The decision to include particular data may still be copyrightable. It is only the architecture of the data structure that should be prohibited from protection under this theory.

IV. "PROGRAM STRUCTURE" AND COPYRIGHT OF A COMPUTER LANGUAGE

The other noncopyrightable component of computer programs that should not be swept within the protection of a program structure is computer language and its attendant grammar. This Part will demonstrate that computer languages do not qualify for copyright protection, but if the legal term program structure is given too much breadth, de facto monopoly over both computer languages and their attendant grammars will follow. Once again, computer science can help us refine the meaning of the legal term "program structure" so that more straightforward copyright analysis can ensue. In particular, the term program structure should exclude computer language grammar to prevent copyright in a computer language. Before arguing that copyright of a computer language is not legally justified, we turn our attention to a definition of computer language.

A. What is a Computer Language?

Computer languages are composed of a set of grammar rules and a set of symbols.[99] The typical computer language grammar is "context-free,"[100] which means that a sentence written in the computer language can be analyzed to find its grammatical construction without any need to understand the meaning of the words. This is essential because computers do not understand meanings: they are simply machines that manipulate symbols.[101]

To illustrate how context-free languages work, consider how an English speaker, without knowing what a "smorg" is, nor what it is to "vit," can parse the silly sentence: "The smorg vitted the blag." First, we know where the sentence begins and ends, and using one grammatical rule, we decompose the sentence into a subject and an object phrase: "smorg" is the subject, and "vitted the blag" is the object phrase. We use another grammatical rule to decompose the object phrase into its constituent pieces: "vit" is the verb, and "blag" is the object. Thus, we

---

99. *See* JEAN-PAUL TREMBLAY & PAUL G. SORENSON, THE THEORY AND PRACTICE OF COMPILER WRITING 30-31 (1985).

100. *See* JOHN E. HOPCROFT & JEFFREY D. ULLMAN, INTRODUCTION TO AUTOMATA THEORY, LANGUAGES AND COMPUTATION 233 (1979) ("[M]odern compiler writing systems usually require that the syntax of the language for which they are to produce a compiler be described by a context-free grammar of restricted form.").

101. Natural languages, like English, are distinct from typical computer languages because they are context-sensitive: the meaning of the words can affect the grammatical structure of a sentence.

have deduced the components of the sentence without understanding the meaning of the words. This sort of formalism makes computer languages work.

A context-free computer language is useful because its grammar can be used mechanically to generate correct sentences in the computer language or to check that a given sentence is within the set of acceptable sentences in that computer language. Understanding this mechanical function depends on one further distinction: the symbols used by a computer language are either "terminal" or "non-terminal." Those which are terminal cannot be decomposed into other symbols. Considering the context-free example in English above, "smorg," "vit," and "blag" are terminal symbols. "Subject phrase" and "object phrase" are also symbols in English, but they are non-terminal because they can be decomposed further using grammatical rules. In formal language theory, then, a computer language G with a context-free grammar is defined as the quadruple V, E, R, S, where V is the entire set of symbols used by language G, E is the set of all terminal symbols, R is the finite set of grammatical rules that transform non-terminal symbols into constituent terminal and non-terminal symbols, and S is the start symbol that tells the computer there is a sentence to parse.[102]

Using this formalism, a computer program called a "parser" can be written that accepts strings of symbols consistent with the grammar and rejects those that are grammatically incorrect.[103] The acceptance of a string of symbols by a parser is the first step any computer program takes when it responds to symbolic input. If the string is accepted, the parser will have constructed a tree data structure[104] that represents the grammatical construction of the sentence it was presented.

A computer mechanically responds to instructions that are presented to it in binary code through combinations of ones and zeros.[105] Humans cannot easily read binary instructions; therefore, we prefer to write programs in an understandable computer language ("source code") and then use a compiler program to translate that computer language into binary-coded instructions ("object code"). These are the instructions that the computer hardware can respond to without interpretation.

For example, consider the statement $X = A + B$. A computer will parse this statement in the following manner. First, the parser uses a grammatical rule that the value of any expression after the "=" sign must be assigned to memory location X. Another rule specifies that expres-

---

102. *See* TREMBLAY & SORENSON, *supra* note 99, at 31.

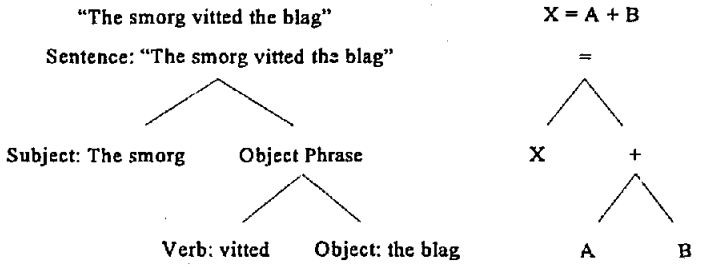
103. *See* ROBERT SEDGEWICK, ALGORITHMS 307 (1988).

104. *See supra* notes 73-76 and accompanying text.

105. *See generally* PATTERSON & HENNESSEY, *supra* note 68, at 270; *see also supra* Part II.

sion's value to be the result of an addition. Finally, the inputs to the addition will be specified as the contents of locations *A* and *B*. The resulting parse trees for both this example and the silly sentence example are presented in Figure 4.[106]

Figure 4: Parse Trees[107]



Any computer program that is supposed to generate or accept sentences in a computer language requires some kind of parser.[108] The computer program that parses input sentences (or generates sentences) from a given computer language *G* must contain within it some version of the quadruple *V, E, R, S*.[109] This typically requires[110] that the program

---

106. Once the statement is parsed, the compiler generates object code starting at the bottom of the parse tree and working up. This process is called "bottom-up" parsing. *See* TREMBLAY & SORENSON, *supra* note 99, at 52. It first produces a computer instruction that moves the content of a memory location to the inputs of the addition unit of the CPU. *See supra* note 8 (explaining CPU). This is done twice: once for the content of *A* and once for that of *B*. An instruction for the computer to add is then produced. Finally, the compiler produces an instruction that moves the resulting sum to memory location *X*.

As a result of compiling, the single statement $X = A + B$ is translated into these four (hypothetical) 8-bit binary instructions that tell the CPU to perform these four steps in sequence:

1. "10011101" (move contents of Memory Location *A* to the Addition Unit Input #1)
2. "10011110" (move contents of Memory Location *B* to the Addition Unit Input #2)
3. "10101110" (execute the sum of the inputs)
4. "10111100" (move contents of the sum into Memory Location *X*)

107. Each "leaf" is an indivisible symbol, therefore a "terminal" symbol. Each non-leaf node represents a non-terminal symbol. Each node in the tree represents the application of a grammar rule.

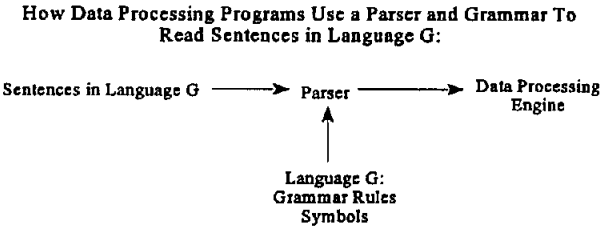
108. *See* SEDGEWICK, *supra* note 103, at 269.

109. *See id.*

110. The other choice is to create a list of all possible sentences in G, a task that is likely to be impossible. *See* TREMBLAY & SORENSON, *supra* note 99, at 31.

access a file listing all of *G*'s symbols and all of *G*'s grammatical rules.[111] Without such a list, the human equivalent would be to hand someone a German dictionary but no grammar text and demand she translate sentences. Figure 5 illustrates how an input sentence is parsed generally.

Figure 5: Parsing an Input Sentence



The application of parser programs is not limited to converting human readable source code into machine executable object code. Parsers are needed almost anytime two programs intend to communicate by sharing command sequences or data files. Parsing is essential when computer files are to be translated from one format to another: it is the first step along a migration path.[112]

If it were possible to prevent a programmer from using a language specification *G* (i.e., some *V*, *E*, *R*, *S*), or even just the grammar *R* of language *G*, it would be impossible for that programmer to write a parser for *G*. If a programmer is prevented from writing a parser in *G*, she will be unable to use language *G* to communicate between her programs or data files and whatever other programs use language *G* to communicate. This is an example of blocking a migration path by preventing translation of the language *G*.

---

111. *See* SEDGEWICK, *supra* note 103, at 270. Another type of parser program builds the grammatical rules directly into an algorithm. These programs are known formally as "finite automata." The grammar rules are represented inside the automata as a list of "IF-THEN-ELSE" conditions that select program behavior based on each new symbol presented to the program. In other words, for each new symbol presented, the program will decide which state to go into based on its current state and the next new symbol in the sentence. Nonetheless, both methods of parsing (using a table of rules or a finite automata) are equivalent. *See* HARRY R. LEWIS & CHRISTOS H. PAPADIMITRIOU, ELEMENTS OF THE THEORY OF COMPUTATION 102 (1981).

112. *See supra* note 27 (defining migration path and explaining its importance in copyright law).

B. Computer Languages Should Not Be Copyrightable

The Copyright Act of 1976 does not directly address the copyrightability of computer languages.[113] Computer languages should not be copyrightable subject matter for two reasons.[114] First, language copyright is doctrinally suspect because that would provide copyright protection for expressions not yet fixed. Fixation, of course, is a prerequisite for copyright protection.[115] If the author of a computer language sought to claim copyright protection for a computer language, either of two files would have to be covered: a list of all possible sentences in that language, or an expression of its specification (i.e., a copy of the quadruple *V, E, R, S*) that fully describes the language. The first choice is likely to be impossible: a computer language contains a huge number of possible sentences, possibly of unbounded size. Although more practical, the second choice is problematic: the specification only tells us how to decide whether a given sentence is within the language and is not fixation of the sentence itself. If the given sentence has been fixed for the first time by a party other than the hypothetical copyright holder

---

113. The text of the statute does not mention computer language. *See* 17 U.S.C. § 102(b) (1994). CONTU does not address the copyrightability of computer languages either. *See* CONTU, *supra* note 31 The only inkling that the Copyright Office has given thought to this is in the 1984 Compendium of Copyright Practices § 325.02(c) (including under the heading "Noncopyrightable elements" the entry "language (alone)" without any further explanation).

The cases are no more illuminating. The court in Bull HN Info. Sys., Inc. v. American Express Bank Ltd., 1990 WL 48098 (S.D.N.Y. Apr. 6, 1990), studiously avoids confronting the defendant's claim that a computer language is not subject to copyright. *See id.* at *2 n.2. The only other cases that consider copyright of a language are the so-called "code book" cases. *See* Brief English Sys., Inc. v. Owen, 48 F.2d 555 (2d Cir. 1931) (finding that shorthand technique, as a language, was not copyrightable); Reiss v. National Quotation Bureau, Inc., 276 F. 717 (S.D.N.Y. 1921) (holding that a book of meaningless code words was copyrightable); *but see* NIMMER, *supra* note 68, at § 1.06, 1-47 ("A programming language constitutes potentially copyrightable subject matter. [They are] compilations of commands and terms developed by a particular author . . . .").

114. *See* Richard H. Stern, *Copyright in Computer Programming Languages*, 17 RUTGERS COMPUTER & TECH. L.J. 322 (1991) (finding language uncopyrightable for the following two reasons: that language is a process, and that some languages are ideal to solve certain programming problems and, therefore, merger applies); *see also* Elizabeth G. Lowry, *Copyright Protection for Computer Languages: Creative Incentive or Technological Threat?*, 39 EMORY L.J. 1293, 1296 (1990) (reasoning that languages are a system and hence unprotectable by copyright); John P. Sumner & Steven W. Lundberg, *Patentable Computer Program Features as Uncopyrightable Subject Matter*, 17 AM. INTELL. PROP. L. ASS'N Q.J. 253 (1989) (finding a computer language uncopyrightable because it is a system, but possibly patentable). Lowry's treatment of computer languages lacks mathematical rigor yet raises a host of legitimate and interesting issues, most notably that the Lotus 1-2-3 macro language satisfies the definition of a real computer language. *See* Lowry, *supra*, at 1296.

115. *See* 17 U.S.C. § 102(a) (1994).

in the language, the quadruple *V, E, R, S* can be used to determine if that sentence is within the supposedly copyrighted language. If that determination supports a claim of infringement, then it would be infringement of an expression not previously fixed. This is contradictory to the statutory requirement that an expression must be fixed to be copyrighted.

Second, even if language copyright were consistent with copyright principles, it would violate First Amendment principles.[116] By authorizing protection for languages, the Act would be authorizing prior restraint of any expression in that language.[117] If a programmer cannot lawfully include a copy of the grammar in a parser program, the parser cannot lawfully create expression in that language. Similarly, a parser could not lawfully read any sentence in that language. It is not clear that just because such expression is machine-generated it is not protected by the First Amendment: the law already recognizes that machine-generated expression — the output of a video game, for example — is protected by the Copyright Clause.[118]

Although copyright law does not violate the First Amendment when it permits individuals to preclude others from substantially copying their original expression, it exceeds First Amendment boundaries when it permits authors to own the rights to form any expression from the building blocks of language.[119] Indeed, protection of linguistic building blocks impedes the progress of originality sanctioned by the Copyright Clause in addition to violating the First Amendment rule against suppressing speech before it has been expressed.

The implications of copyright in a computer language are extreme: if a language can be protected through copyright in its specification (i.e., the quadruple *V, E, R, S*), then any use of the language would have to be

---

116. Computer program source code has been found protected speech. *See* Bernstein v. U.S. Dep't. of State, 922 F. Supp. 1426, 1436 (N.D. Cal. 1996) ("For purposes of First Amendment analysis, this court finds that source code is speech."); *see also* Karn v U.S. Dep't. of State, 925 F. Supp. 1, 9 (D.D.C. 1996) ("[T]he Court will assume that the protection of the First Amendment extends to the source code . . .").

117. For examples of prior restraint cases see, for example, Nebraska Press Ass'n v. Stuart, 427 U.S. 539, 556 (1976); New York Times Co. v. United States, 403 U.S. 713, 713 (1971); Near v. Minnesota, 283 U.S. 697 (1931).

118. *See* Atari Games Corp. v. Oman, 888 F.2d 878 (D.C. Cir. 1989).

119. The restriction of expression to protect an idea has been found unconstitutional by the Supreme Court: "[C]opyright's idea/expression dichotomy 'strike[s] a definitional balance between the First Amendment and the Copyright Act by permitting free communication of facts while still protecting an author's expression.' No author may copyright his ideas or the facts he narrates." Harper & Row, Publishers, Inc. v. Nation Enters., 471 U.S. 539, 556 (1985) (citations omitted).

licensed since use of a language requires a parser program which must contain the specification of the language.[120]

The dependence of a parser program on the quadruple V, E, R, S is strong: the same problem would arise even if only the set of grammar rules R is protected while the symbols (elements of V) are in the public domain. As explained in Part IV.C., it is no solution to say that there could be two different expressions of a given language specification or grammar. Any two distinct expressions of the same language specification will always contain the same inherent hierarchy among the symbols that under traditional non-literal similarity tests will support an infringement claim.

That "major" computer languages are already in the public domain does not solve the conundrum. For example, Sun Microsystems has made a major investment in creating Java, which is designed to make it easier to write Internet-based graphical user interfaces. Sun has recognized the large potential value of a computer language as intellectual property and currently requires programmers to sign a licensing agreement if they wish to use Java or to write a compiler that implements Java.[121] Microsoft, a Java licensee, has created its own extensions to Java that optimize Java for use with their Windows operating system. Such language extensions are the natural adaptations of existing computer languages to new computing environments.[122] Microsoft's extensions to the Java language could make their Java compiler more attractive than Sun's compiler. Sun is now positioning itself to prevent Microsoft from running away with the language standard by promoting an industry standards committee including itself but not Microsoft.[123] If computer languages really are copyrightable, Sun could sue Microsoft

---

120. *See supra* Part IV.A. For program A to communicate in language L, it must contain its own copy of L's grammar. If that grammar has already been copyrighted, then distributing copies of program A will infringe on the existing copyright in L's grammar.

121. *See* Sun Microsystems Inc., *Technology License and Distribution Agreement* 1 (April 1996) ("WHEREAS Sun wishes to license its Java programming language . . .") (on file with author). Interestingly, the contract refers to the "Java Language Specification" as part of the technology documentation but not directly as the technology. Instead, the technology is a set of files that contain object class descriptions for implementation of Java. These files would play the role of a language grammar file analogous to the menu tree in *Lotus Dev. Corp. v. Borland Int'l, Inc.*, 831 F. Supp. 223 (D. Mass. 1993), *rev'd*, 49 F.3d 807 (1st Cir. 1995), *aff'd by an equally divided court*, 116 S. Ct. 804 (1996). *See supra* Part IV.D.

122. For example, the very popular C++ computer language was developed as an extension of the now less fashionable C language. *See* BJARNE STROUSTRUP, THE C++ PROGRAMMING LANGUAGE 627-28 (2d ed. 1991). Competing software companies now release C++ compilers that include new extensions to the language to attract programmers to buy their product.

123. *See supra* text accompanying note 6.

for creating an unauthorized derivative work. Even more disturbing is that the company could demand licenses from other application programmers who intend to use their compiler to write other software products: they would be using a copyrighted language. This would impede the progress of computer programming because entrepreneurial software ventures that might become a threat to a large manufacturer could be instantly shut down if the large manufacturer terminated their license to use the computer language extensions.[124]

The existence of other programming languages or language extensions does not solve the problem because the software entrepreneur will be forced to select a compiler not on the basis of its technical merits or consumer demand but rather based solely on the terms offered by the copyright monopoly holder. Larger companies with larger research and development staffs will have better — or at least more widely employed — compilers and at the same time will likely offer more onerous licensing terms. The new venture must either use inferior compilers or succumb to the demands of the larger company. This is an undesirable externality resulting from computer language copyright.[125] In sum, copyright in computer languages is at odds with both the statute and the First Amendment, and uses the copyright monopoly to impede the progress of computer science.

C. "Program Structure" and Non-Literal Infringement of a Computer Language Grammar

A key observation about a set of grammatical rules is that they must necessarily create a hierarchy among the symbols contained in the language. This is so because one characteristic of grammatical rules is that they specify how the non-terminal symbols of the language can be decomposed into their constituent symbols.

In the context-free English example, we know that the following sentence does not make sense because the word order is incorrect: "Vitted the blag the smorg." That observation is a result of our recogni-

---

124. Such an action would theoretically be proscribed under the antitrust laws, but depending on antitrust laws to fix problems in copyright doctrine is foolish. *Cf.* Alfred Bell & Co. v. Catalda Fine Arts, Inc., 191 F.2d 99, 106 (2d Cir. 1951) ("We have here a conflict of policies: (a) that of preventing piracy of copyrighted matter and (b) that of enforcing the anti-trust laws. We must balance the two . . . [on these facts] we think the enforcement of the first policy should outweigh enforcement of the second."); *see also* Rosemont v. Random House, 366 F.2d 303 (2d Cir. 1966) ("Thus, it is not the fact of a constitutional and statutory monopoly which is disfavored, only abuses of the lawful monopoly.")

125. *See* CONTU, *supra* note 31, at 23 ("One of the hallmarks of a competitive industry is the ease with which entrepreneurs may enter into competition with firms already doing business.").

tion that there is no rule that decomposes a sentence with the verb "vit" first.[126] In other words, we must successfully decompose the sentence into a subject phrase followed by an object phrase before we decompose the object phrase itself. This hierarchy results from the grammatical rules of English and would be inherent in any expression of those rules. In the computer language example above, the sentence $A + B = X$ is incorrect because the "+" rule cannot be exercised before the "=" rule. That hierarchy is why the parser would reject the sentence $A + B = X$ as not belonging to the language.

At some level of abstraction, all possible expressions of a given grammar will contain the same hierarchy among the symbols because that is the hierarchy inherent in the relationship between the non-terminal and terminal symbols of the language. If the legal term program structure is deemed to include the grammatical hierarchy in a given language G, then any implementation of a parser for language G, which necessarily will contain that hierarchy, will infringe on any other instance of that hierarchy through the non-literal similarity test. Therefore, no program could be written to read or write language G without infringing on the copyright in the grammar of G.

In conclusion, if grammatical rules and symbols are copyrightable, then copyright law has once again introduced a conundrum. As demonstrated earlier, providing copyright protection to grammatical rules and symbols confers a de facto monopoly over any sentence within that computer language whether fixed or not.[127] Therefore, the legal term program structure should exclude from copyright protection any expression of a language specification or the inherent hierarchy among the symbols contained in it. This prevents de facto monopoly over computer languages as a result of copyright in language specification.[128]

---

126. In English, this is because of the tense of the verb "vit." If the sentence started "Vitting carefully, the smorg . . . ," then a rule would allow decomposition with the verb first. Similarly, the verb "is" would be acceptable at the beginning: "Is this your smorg?" This, however, moves beyond the scope of the example.

127. This aspect of formal language theory is the essence of Lotus's goal in its lawsuits against Paperback and Borland with respect to the "Key Reader" macro language discussed below. *See* Lowry, *supra* note 114, at 1294. Lotus's original complaint included infringement of the macro language.

128. We are not advocating excluding copyright of human readable specifications of a language, i.e., textbooks that teach humans how to understand and write in a language. This legal proposition is limited to expressions of language grammar for use by the computer itself.

D. Computer Language Grammar, Copyright, and Lotus v. Borland

Examination of the *Lotus Development Corp. v. Borland International, Inc.*[129] case is an excellent way to observe how inaccurate terminology introduces unintended externalities into the software copyright doctrine. By revisiting *Lotus v. Borland* with accurate computer science terminology, we can demonstrate how improved software copyright analysis will result.

Considering *Lotus v. Borland* in light of our understanding of computer languages leads to the conclusion that in one portion of the suit, Lotus could have obtained a monopoly over a computer language they devised.[130] The two spreadsheet programs at issue do not express the idea of a spreadsheet program using the same expression but rather are only similar in their purposes and results. To enhance the utility of Lotus 1-2-3, a Lotus 1-2-3 user typically writes her own small programs (using the "macro language") that encompass repetitive sequences of Lotus 1-2-3 commands.[131] Each sequence is assigned to a single key on the computer keyboard: when that key is pressed, the corresponding sequence of commands is sent to Lotus 1-2-3. This aspect of the dispute centers on the fact that Borland's Quattro Pro program contains a "Key Reader" program that accepts the same symbolic sentences that are accepted by the Lotus 1-2-3 program.[132] Having parsed these sentences, it is able to control the Quattro Pro program to produce the same result as if Lotus 1-2-3 had been running. A diagram of the process is shown in Figure 6. In human terms, this is equivalent to having a language translator work between two individuals with different native languages.

---

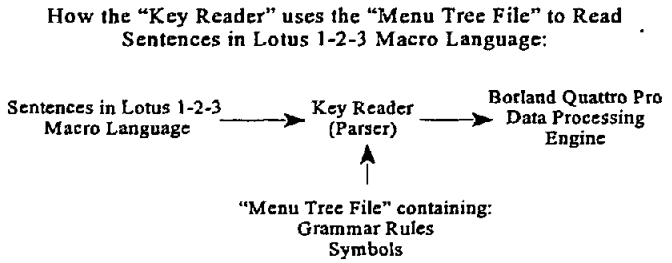
129. 831 F. Supp. 223 (D. Mass. 1993), *rev'd*, 49 F.3d 807 (1st Cir. 1995), *aff'd by an equally divided court*, 116 S. Ct. 804 (1996).

130. Lotus also raised other issues not addressed in this Article, most importantly whether a program that had a user interface substantially similar in organization to Lotus 1-2-3 infringed on the copyright in its user interface display. *See* Lotus Dev. Corp. v. Borland Int'l, Inc. 788 F. Supp. 78, 81 (D. Mass. 1992).

131. *See Lotus*, 49 F.3d at 809.

132. *See id.*

Figure 6: Reading Sentences in the Lotus 1-2-3 Macro Language



If the Key Reader parsing program cannot read Lotus 1-2-3 macro language sentences, then users who want to switch to Quattro Pro from Lotus 1-2-3 would have to rewrite their macro programs to conform with Quattro Pro's macro language.[133] The significance of this result is that consumers of commercial software will be forced to choose between re-writing their macro command programs by hand or passing up the opportunity to use a competitor's preferred product because there is no migration path between the new product and their existing software. More generally, it opens the door to the possibility that a software consumer cannot simply apply a translation program to his work to switch from using one vendor's product to using another's.[134] This capacity to block a migration path for software consumers is anti-competitive rather than the result of a legitimate copyright monopoly.[135]

Judge Keeton found infringement in the Borland Key Reader program because it contained a file that described the entire menu hierarchy of Lotus 1-2-3:

> Put another way, the point is that to implement Key Reader[,] Borland used a program file containing the same copy of the 1-2-3 menu tree structure and commands that Borland had used in its emulation interface,

---

133. *See id.* at 821.

134. Translation would be accomplished with a program quite similar to a compiler. *See supra* Part IV.A.

135. *See* CONTU, *supra* note 31, at 23 (finding that anti-competitive practices are beyond the scope of patent and copyright protection); *see also* Clapes et al., *supra* note 14, at 1560 ("[Narrow protection is warranted to prevent] giving authors of original programs the power to preclude others from writing programs that interact with those original programs. If valid, such concerns would be serious indeed."); Paul Goldstein, *Infringement of Copyright in Computer Programs*, 47 U. PITT. L. REV. 1119, 1129 (1986) (reasoning that in cases not rising to the level of misuse, courts could seek to resolve the compatibility issue through the doctrine of fair use).

> but with each menu command name stripped of everything after the first letter. Borland then appended this copy of the "stripped menu tree" to its quattro.mu file .... In sum, to interpret macros, Borland's programs use a file with phantom menus consisting of a virtually identical copy of the Lotus menu tree that Borland used for its emulation interface, but with only the first letter of each menu command name where the complete menu command name previously appeared.[136]

According to Judge Keeton, Borland's Key Reader infringed not because of literal copying of the Lotus menu screen, but as a result of the non-literal copy of the Lotus command menu hierarchy within the hidden Borland file.[137] He did not address the question of the menu hierarchy as a computer language grammar. The Court of Appeals reversed and held that the Lotus 1-2-3 menu command hierarchy was a method of operation and hence uncopyrightable.[138] Although this decision ultimately prevents an anti-competitive monopoly over a grammar, the court did not face the macro language grammar issue head-on. Once the menu hierarchy was found uncopyrightable, Borland's non-literal copy of the menu was non-infringing as a matter of law.[139] The Supreme Court did not confront the computer language grammar issue because it was equally divided on the case and therefore affirmed without opinion the decision of the Court of Appeals.[140] In examining this history, the intention is not to reargue the case but rather to consider Judge Keeton's district court opinion as a symptom of the problem presented: that to ignore basic computer science concepts when deciding software copyright cases produces bad case law.

The "stripped menu tree" file identified by Judge Keeton is a representation of the symbols and grammatical rules that define the Lotus 1-2-3 macro language.[141] This list is necessary for the Borland Key

---

136. Lotus Dev. Corp. v. Borland Int'l, Inc., 831 F. Supp. 223, 228-29 (D. Mass. 1993).

137. *See id.* at 224.

138. *See* Lotus Dev. Corp. v. Borland Int'l, Inc., 49 F.3d 807, 815 (1st. Cir. 1995).

139. *See id.* at 819.

140. *See* Lotus Dev. Corp. v. Borland Int'l, Inc., 116 S. Ct. 804 (1996).

141. An irony is that the judge had been presented the language issue before in *Lotus Development, Inc. v. Paperback Software International*, 740 F. Supp. 37, 53 (D. Mass. 1990), and regarded the argument as a "word game": "An even more striking word-game argument is defendant's contention that in copying the 1-2-3 user interface, they have only copied a language, and that languages are not copyrightable. ... I conclude that defendant's language argument about the macro facility of Lotus 1-2-3 ... is totally without merit." *Id.* at 73. *See* Samuelson, *supra* note 37 at 236.

Reader to parse and accept Lotus 1-2-3 macro language sentences.[142] The hierarchical listing of the macro commands is a grammar because the hierarchy determines which symbols may follow which other symbols. By doing so, it performs the function of a formal language grammar: to determine which sentences are within the language and which are not. For example, the sentence "WORKSHEET, FORMAT, FIXED" is acceptable only because FORMAT appears under the WORKSHEET heading and FIXED under the FORMAT heading.[143] The sentence "WORKSHEET, RANGE, LABEL" is syntactically incorrect because RANGE is not under the WORKSHEET heading: it is its own heading. Thus, the hierarchical table within the menu tree file is a representation of the macro language grammar and its symbols. The grammar represented by the menu tree file is used by the Key Reader to parse and accept the Lotus 1-2-3 macro language.[144]

According to Judge Keeton, anyone who expresses the Lotus 1-2-3 menu command hierarchy in any way infringes on Lotus's copyright. However, because the Lotus 1-2-3 macro language itself is derived from the menu commands presented on the computer screen to the Lotus 1-2-3 user, any expression of the macro language grammatical rules will always contain the hierarchy inherent in the Lotus 1-2-3 menu commands that appear on the screen.[145] To put it another way, no one can express the grammatical rules of the Lotus 1-2-3 macro language without expressing in some form the menu command hierarchy. Therefore, following Judge Keeton's reasoning, the only programmers that can write a parser that will read or write the Lotus 1-2-3 macro language sentences are those that are allowed to express the menu command hierarchy inherent in the grammatical rules of the Lotus 1-2-3 macro language: presumably Lotus and its licensees. By misunderstanding the computer language grammar aspect of the Key Reader dispute, Judge Keeton accidentally opened the door to a de facto monopoly of the Lotus 1-2-3 macro language in finding copyright protection for its grammatical rules and symbols through non-literal infringement. As a result, he disserved

---

142. A grammar specifies which sentences are in a language and which are not. *See supra* Part IV.A.

143. *See* Lotus Dev. Corp. v. Borland Int'l, Inc., 831 F. Supp. 202, 210 (D. Mass. 1993).

144. *See supra* Figure 5.

145. *See supra* Part IV.C (explaining the hierarchy in grammatical rules). In the Lotus 1-2-3 program, the menu display itself had a hierarchy: some menu choices had to be selected to reach other choices. The Key Reader technology allows users to make the same sequence of menu choices by inserting a sentence of symbols (a macro) directly into the program instead of making the selections on the screen. Each symbol represents a menu choice. *See* Samuelson, *supra* note 37, at 236.

copyright policy and created the potential for a system which impedes the optimal progress of computer software development.[146]

Given that computer language grammars are not copyrightable, the menu tree file used by Quattro Pro's Key Reader should be found non-infringing. This is because the menu tree file is merely a representation of the Lotus 1-2-3 macro language grammar. That the macro language grammar was derived by Lotus from the appearance of the Lotus 1-2-3 command menu is immaterial. On balance, this is a limit to copyright protection from non-literal copying only, not literal copying of the Lotus 1-2-3 menu appearance. This relative loss of protection would be slight compared to the greater advancement of the policies behind copyright protection and greater doctrinal coherence overall.

V. SUMMARY AND CONCLUSION

The integration of computer science terminology into the legal discourse increases the precision with which judges can analyze the software copyright cases in front of them. To that end, fundamental questions should be asked of the plaintiff in a non-literal software copyright infringement suit to be sure that the infringed portion of their program is clearly within the sphere of copyright protection. Furthermore, that any infringement claim must survive these threshold tests establishes a clear boundary line for programmers seeking to discern what is theirs and what is in the public domain.

A. What a Judge Should Ask the Plaintiff

Once a judge has learned what a data structure and a computer language grammar are, she has the tools to step around some of the pitfalls inherent in a computer program copyright case. If the plaintiff is arguing that the defendant's program is infringing because it organizes the subject data in a substantially similar way, the judge should ask the plaintiff: "Is this organization a data structure that is uniquely necessary to run your algorithm?" If yes, then that organization is excluded from copyright protection in order to ensure the free use of the algorithm. If no, then the judge can decide whether the organization of the data meets the de minimus standard for copyrightability without concern that a monopoly over the organization will prohibit public use of the algorithm.[147] Of course, the two parties will litigate whether the

146. *See supra* Part IV.B.

147. One can imagine an algorithm that could run on two different data structures, one being practical and efficient and the other introducing some gross degradation in algorithm performance. The latter data structure is not copyrightable because of the limitations in

algorithm requires the underlying data structure. Note that the judge must determine that based on the usual "battle of the experts"; however, the dependence of the algorithm on a particular data structure can be determined with mathematical precision.[148] As a result of employing the appropriate computer science terminology, the court's reasoning will hew to traditional copyright reasoning while giving full consideration to cutting edge software technology. The court's attention will be focused right at the center of the issue.

In a case where the plaintiff is claiming that some kind of hierarchy of commands or input symbols that controls the plaintiff's program has been infringed, the judge must ask: "Is this hierarchy of symbols equivalent to a formal language grammar?" To pin the plaintiff down, she can ask whether it would be possible to write a program to accept the same sentences usable by the plaintiff's program without relying on a literal or non-literal copy of the symbols and hierarchy in dispute. If the answer is yes, then the hierarchy in dispute is not essential to represent a grammar. If the answer is no, then the hierarchy in dispute is essential to represent a grammar. Again, the answer to that question can be determined with mathematical precision.

B. Computer Terms of Art and Copyright Analysis

Non-literal copying of a computer program is inherently difficult to analyze without understanding relatively simple computer science concepts and integrating them into the legal discourse. By using computer science terms of art, the risk of overprotection posed by the existing computer program jurisprudence is significantly reduced and more efficient progress in computer software development is assured. The *Whelan* and the *Altai* courts were correct to the extent that they both recognized that copyright protection for computer programs cannot extend to noncopyrightable elements. Their approaches fall short, however, to the extent that they fail to acknowledge precisely those program components that deserve independent consideration under copyright analysis. As long as the courts considering computer software copyrightability persist in crafting new terms of art and ignoring the more precise and relevant computer science terms, we will have

---

expression arising from the practicalities involved. This reasoning is the same as the reasoning used in *Kern River Gas Transmission Co. v. Coastal Corp.*, 899 F.2d 1458 (5th Cir. 1990). With respect to computer programs, this point is raised in *Computer Associates, International, Inc. v. Altai, Inc.*, 982 F.2d 693, 708 (2d Cir. 1992). *See also supra* note 83 and accompanying text (discussing merger doctrine and compilation protection for data structures). *But see* Miller, *supra* note 14, at 1009 n.156.

148. Either an algorithm is dependent on a data structure or it is not. That dependence will be apparent from the algorithm itself, separate from any instant expression of it.

unpredictability in computer software copyright law, which in turn will impede full throttle progress in the development of computer software. That result is bad for the economy, bad for consumers, and contrary to the directive of the Constitution's Copyright Clause.