Lisping Copyleft: A Close Reading of the Lisp LGPL

Eli Greenbaum [a]

(a) Attorney, Yigal Arnon & Co. Jerusalem

DOI: 10.5033/ifosslr.v5i1.75

Abstract:

The idioms of both the General Public License (the "**GPL**") and the Lesser General Public License (the "**LGPL**") seem to be grounded in the C programming language. This article analyses the Lisp Lesser General Public License (colloquially and here referred to as the "**LLGPL**"), a specific attempt to apply the LGPL to a language with a programming paradigm and method of building and distributing programs that traditionally differs substantially from the approach of C. In addition, this article attempts to understand whether the LLGPL actually succeeds in its stated goal of translating the LGPL to the Lisp context or whether the LLGPL changes the requirements and philosophical moorings of the LGPL.

Keywords:
Law; information technology; Free and Open Source Software; copyleft, copyright; derivation; compilation; Lisp; LGPL;

Introduction

The idioms of both the General Public License (the "**GPL**") and the Lesser General Public License (the "**LGPL**")[1] seem to be grounded in the C programming language. The licenses refer to "compiling", "linking" and "header files", features of the C programming languages which may not be present in other languages that are not traditionally compiled. Similarly, the licenses do not expressly include provisions relating to features of object-oriented programming languages.[2] Do the GNU licenses work as intended when applied in these other contexts? [3] This article analyses the Lisp Lesser General Public License (colloquially and here referred to as the "**LLGPL**"), a specific attempt to apply the LGPL to a language with a programming paradigm and method of

1 The LLGPL license is drafted as a preamble to version 2.1 of the LGPL. As such, in this article, unless states otherwise references to the GPL and LGPL are references to version 2.0 of the GPL and version 2.1 of the LGPL.

2 In contrast, version 3.0 of the LGPL does relate to features of object oriented languages. For example, the definition of "Application" in that license discusses the effect of defining a subclass of a class defined by the Library.

3 The Free Software Foundation has strongly asserted that the LGPL may be applied to all known programming languages. *See* David Turner, The LGPL and Java, *available at* http://www.gnu.org/licenses/lgpl-java.html (stating that "FSF's position has remained constant throughout: the LGPL works as intended with all known programming languages, including Java.").

building and distributing programs that traditionally differs from the approach of C.[4]

Lisp is one of the oldest programming languages still in use. Lisp was invented in 1958 by John McCarthy at the Massachusetts Institute of Technology. The language was first implemented when one of McCarthy's graduate students hand-compiled the Lisp *eval* function into machine code, and created the first Lisp interpreter. Following in this history, while implementations of Lisp can allow for the compilation and distribution of executables, Lisp was traditionally developed and distributed as an interpreted rather than a compiled language. Lisp was closely connected to research in the field of artificial intelligence, and the popularity of the language declined in the late 1980s together with interest in that field. Nevertheless, Lisp seems to have enjoyed somewhat of a resurgence in recent years, and currently there are several open source and commercial implementations of the language.

Open source programs are not frequently written in Lisp.[5] Nevertheless, certain features of Lisp have inspired a broader family of "dynamic languages" that can be considered to include popular languages such as PHP or Python. As with Lisp, for example, those languages are typically interpreted rather than compiled into executables. As such, programs written in those languages will also generally require the distribution of an interpreter together with the application. To the extent the LLGPL's claim that the GNU licenses are not appropriate for Lisp is justified, the suitability of the GNU licenses for these other languages will also be implicated.

This article presents a close reading of the LLGPL license. In analysing the license, this article attempts to understand whether, as the LLGPL claims, another document is necessary to apply the LGPL to the Lisp context. In addition, this article attempts to understand whether the LLGPL succeeds in its stated goal of translating the LGPL to the Lisp context or whether, in making the transition, the LLGPL moves away from the requirements and philosophical underpinnings of the LGPL. Before concluding, this article briefly discusses some issues raised by Lisp that were not expressly addressed by the LLGPL.[6]

History and Philosophy of the Lisp LGPL

The LLGPL was authored by Franz, Inc. ("**Franz**"), a leading commercial Lisp vendor based in California. Franz is the corporate developer of "Allegro Common Lisp", one of several commercial implementations of the "ANSI Common Lisp" standard.[7] The ANSI Common Lisp

4 In a somewhat ironic twist, the history of the GNU licenses began with Richard Stallman's distribution of Emacs, a text-editing program written in Lisp. Stallman initially distributed Emacs under the Emacs General Public License, out of which grew the first version of the General Public License. For an early history of the GNU licenses, see Chapter 2 of Glyn Moody, Rebel Code (2002).

5 According to Black Duck, Lisp is not one of the top fifteen languages used in open source projects. *See* http://www.blackducksoftware.com/osrc/data/projects/. C is the most popular language, used in 44.95% of releases of open source projects. For a not-up-to date list of some commercial software projects in Lisp, *see* http://www.pchristensen.com/blog/lisp-companies/.

6 Aside from the LLGPL, there are a number of other licenses that have been drafted to apply to specific programming languages. For example, PHP is distributed under a permissive license similar to the BSD. *See* http://www.php.net/license/index.php#code-lic. Python is also distributed under a permissive license. *See* http://docs.python.org/2/license.html. These licenses are generic and do not have any technical provisions that apply to features of specific languages. A number of other licenses contain provisions expressly adapted for particular programming languages. For example, the GNAT Modified General Public License is a version of the GPL which has been adapted for the "generic" feature of the Ada programming language. *See* http://libre.adacore.com/tools/gnat-gpl-edition/faq/. In addition, the Falcon Programming Language License is "specifically designed around the concept of an open source scripting language engine." See http://www.falconpl.org/index.ftd?page_id=licensing. An analysis of these latter two licenses is beyond the scope of this article.

7 Commercial implementations of Lisp also includes LispWorks. Open source implementations of Lisp include Steel Bank Common Lisp, which is licensed under BSD-style licenses and also includes code in the public domain (*See*

standard was developed in the early 1980s as an attempt to unify the several dialects of the language.[8] Franz initially began distributing Allegro Common Lisp in 1986, and authored the LLGPL in 2000. Franz has shown a commitment to the open source development of software, and has licensed a number of open source software projects under the terms of the LLGPL.[9] Unfortunately, there is a dearth of commentary regarding the interpretation of the LLGPL. As such, and as the provisions of the license are not always completely clear, the application of the LLGPL to software may not always be completely straightforward.

The LLGPL is a short document, consisting of five not-lengthy paragraphs, and by its terms is intended to be read as a “preamble” to the LGPL.[10] Generally, the LGPL permits proprietary applications to be combined and distributed with LGPL-licensed libraries, and does not require that the source code of the proprietary application be disclosed. This is in contrast to the stronger “copyleft” requirements of the GPL, which generally requires that all works “based on” a GPL-licensed work also be distributed under the same license terms. The LLGPL is intended to adapt the weaker copyleft provisions of the LGPL to the Lisp setting. In the words of the first paragraph of the LLGPL, the “LGPL uses terminology more appropriate for a program written in C than one written in Lisp” and, as such, some “clarifications” are necessary to apply the LGPL in the Lisp context.

The first paragraph of the LLGPL implies that the application of the LLGPL results in licensing terms that are not very different than the LGPL itself, even though they have been translated to the Lisp context.[11] Even so, several provisions of the LLGPL seem to belie this understanding of the license. For example, the LLGPL provides that a “Lisp application may include the same set of Lisp objects as does a Library, but this does not mean that the application is necessarily a ‘work based on the Library’ it contains.”[12] In contrast, the LGPL expressly provides that a work containing portions of the Library should be considered a derivative work of the Library under copyright law, and a “work based on the Library” under the LGPL.”[13] The clause in the LLGPL seems to contradict express provisions of the LGPL. Unfortunately, the LLGPL does not explain the motivation for making this fundamental change in the terms of the LGPL.

Other clauses of the LLGPL also seem to diverge from the provisions of the LGPL. For example, the LLGPL provides that “[i]t is permitted to add proprietary source code to the Library, but it must be done in a way such that the Library will still run without that proprietary code present.”[14] This seems to restrict a user’s ability to modify the licensed work. Interpreting this sentence in light of the LGPL is quite difficult, as Section 2 of the LGPL expressly provides a user of the Library with the right to modify and copy the Library, without any requirement to ensure that it can still run without those modifications.[15] Again, the LLGPL does not describe the reasons for

http://www.sbcl.org/history.html), GNU Common Lisp, available under the LGPL (*See* http://savannah.gnu.org/projects/gcl), and GNU Clisp, available under the GPL (See http://www.clisp.org/).

8 See the history of Lisp at http://www.dreamsongs.com/Files/Hopl2.pdf

9 See http://opensource.franz.com/

10 The concept of presenting the LLGPL as a preamble to the GNU license seems to be inspired by the structure of the LGPL itself, which begins with a preamble that explains the goals of the license.

11 On its website, Franz itself provides a somewhat more ambiguous description of the LLGPL, stating that the document is a “new license” which is intended to take the special features of dynamic programming languages into consideration. See http://opensource.franz.com/

12 See the last sentence of the second paragraph of the LLGPL.

13 For example, the definition of “Library” in the LGPL provides that a “work based on the Library” includes a “work containing the Library or a portion of it. Similarly, section 5 of the LGPL provides that “linking a ‘work that uses the Library’ with the Library creates … a derivative of the Library (because it contains portions of the Library).”

14 See first sentence of the fourth paragraph of the LLGPL.

15 Indeed, one of the core freedoms advocated by the Free Software Foundation is the freedom to modify software. See http://www.gnu.org/philosophy/free-sw.html. The LGPL also evidences a similar concern that a modified Library should be able to operate even without the application that it is linked with. *See* Section 2(d) of the LGPL. Even so, that concern is with regard to the use of the licensed work when distributed to third parties, but the literal reading of the

adding this requirement to the provisions of the LGPL.

These examples demonstrate that it is difficult to reconcile certain provisions of the LLGPL with the original GNU license. Indeed, this article shows that the LLGPL does clarify certain provisions of the LGPL in the Lisp setting, but also substantially modifies the provisions of the original license. Unfortunately, the LLGPL is frequently not explicit regarding whether a specific provision should be seen as a "translation" to the Lisp context or as an intentional change in the licensing terms of the LGPL. Furthermore, the LLGPL does not always explain its motivation for making certain clarifications or changes, and this can make it difficult to interpret and apply the license.

Definitions and Redefinitions

Several provisions of the LLGPL seem to be motivated by an attempt to clarify the provisions of the LGPL in the Lisp setting. For example, the second paragraph of the LLGPL changes several definitions of the LGPL, such as the definitions of "library", "function" and "data", making them more amenable to the Lisp context. The second paragraph reads in full:

> *A "Library" in Lisp is a collection of Lisp functions, data and foreign modules. The form of the Library can be Lisp source code (for processing by an interpreter) or object code (usually the result of compilation of source code or built with some other mechanisms). Foreign modules are object code in a form that can be linked into a Lisp executable. When we speak of functions we do so in the most general way to include, in addition, methods and unnamed functions. Lisp "data" is also a general term that includes the data structures resulting from defining Lisp classes. A Lisp application may include the same set of Lisp objects as does a Library, but this does not mean that the application is necessarily a "work based on the Library" it contains.*

These revised and generalised definitions are to some extent useful in clarifying LGPL terminology for Lisp. At the same time, however, the revisions appear to focus on certain aspects of the technical distinctiveness of Lisp which would not seem to materially affect the interpretation of the LGPL.

For example, the first sentence of the second paragraph provides that "[a] 'Library' in Lisp is a collection of Lisp functions, data and foreign modules."[16] The purpose of this definition seems to be the subtle modification of the definition of a "library" (not capitalised)[17] in the LGPL, which provides that a library means "a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to

LLGPL provides no suggestion that it should be read as anything but a restriction of a user's right to modify the program, regardless of whether the modified version is distributed to third parties. In addition, the LGPL only requires that users make a "good faith effort" to ensure the operation of the modified library. In contrast, the LLGPL's requirement is formulated as an absolute requirement.

16 A "foreign module", according to the LLGPL, is "object code in a form that can be linked into a Lisp executable". Briefly, Lisp data types often differ from data types in other languages. As such, Lisp requires a "foreign function interface" in order to link code written in a different language. *See generally* Peter Seibel, Practical Common Lisp, 467 (2005). The existence of foreign modules and the need for a "foreign function interface" is not unique to Lisp. For example, the "Java Native Interface" enables a Java Virtual Machine to invoke (and be invoked by) the native code of libraries and applications written in other languages. The Perl XS interface also allows Perl to use C libraries. The Python extension module API allows the calling of library functions and system calls.

17 The LGPL contains a definition of "library" (not capitalised) and "Library" (capitalised). The former provides a generic description of a software library, while the latter refers to the specific work licensed under the LGPL. The need for the former generic definition of a software library is not clear in the document and, indeed, version 3 of the LGPL omits this generic definition. In version 3 of the LGPL, a "Library" simply means a "covered work governed by this License."

form executables." It seems that the authors of the LLGPL did not believe that this definition was completely appropriate for the Lisp context. First, Lisp programs are traditionally interpreted rather than compiled and, as such, the LLGPL definition omits the provision that libraries are intended to be "linked … to form executables." Second, the LLGPL definition adds that a library may include a foreign module.[18]

It is unclear whether these changes to the definition of "library" are necessary for the application of the LGPL to Lisp applications. First, the LLGPL's change in the definition of "library" does not broaden the application of the license, since in any event the LGPL expressly provides that the license may be applied to "any software library or other program". In other words, the application of the LGPL is not restricted to works that meet the LGPL definition of "library". Indeed, the definition of "Library" (capitalised) in the LGPL refers generically to "any software library or work which has been distributed under these terms." Second, it is in any event doubtful that a court would interpret the LGPL's definition of "library" with a level of specificity that would exclude similar linguistic structures of Lisp. For example, it is unlikely that the word "function" in the LGPL would be interpreted to exclude a "foreign module", since both terms essentially refer to software modules that provide a level of functionality.

The second paragraph continues with several other clarifications of the LGPL terminology for the Lisp context. First, the document provides that "functions" should be understood to include "methods" and "unnamed functions". These two terms refer to syntax that is not part of the C programming language. In brief explanation (with more to come later in this article): First, in Lisp, a "method" is the specific implementation of an abstract operation, where the generalised operation is referred to as a "generic function";[19] second, Lisp (as well as other programming languages) offers the opportunity to create "unnamed functions", a way of creating functions without actually providing the function with a defined name.[20] At the same time, however, it is difficult to see why it is necessary to clarify these points in order to apply the LGPL to Lisp programs. It is difficult to conceive of a legitimate legal claim that the word "function" in the LGPL should not naturally be extended to constructions (such as methods and unnamed functions) that act as functions even though they differ in their syntactic expression.

The next clause of the LLGPL also attempts to interpret the LGPL in the Lisp context, and provides that "Lisp 'data' is also a general term that includes the data structures resulting from defining Lisp classes". This provision seems to be an attempt to apply the LGPL to the abstract data types (such as "classes") that form part of an object-oriented language. Indeed, it is possible to interpret the term "data" in the LGPL as referring to "information" rather than "data structures that contain information" in the abstract sense of the word.[21] Nevertheless, as with the word "function", it is not likely that the term "data" in the LGPL would be interpreted with a level of specificity that would exclude appropriate and similar structures in the Lisp context.

In sum, it does not seem that the changes made to the definition of "library" by the LLGPL are

18 It is possible to opine that the LLGPL broadens the defined term "function" in order to include Lisp "macros" within that defined term. This possibility is not expressly acknowledged by the text of the LLGPL and, as such, the effect of the LLGPL on Lisp "macros" remains unclear. Lisp "macros" are further discussed below in Section "**Of Macros**".

19 Other object oriented programming languages (such as Java) also provide for "methods". In Java, however, methods are typically incorporated into the definition of a class, while Lisp methods are defined outside of a class and rather as part of "generic functions." The implications of these syntactical distinctions are beyond the scope of this article. *See generally* Seibel, *supra* note 16, at 191.

20 In Lisp, "unnamed" functions are typically referred to as "lambda" functions. Lambda functions are useful, among other things, for creating functions that can use the local variables of the environment in which they were created. *See generally*, Seibel, *supra* note 16, at 62-63. "Unnamed" functions are also supported by other "dynamic" languages such as Ruby, Javascript, Perl and Python.

21 For example, Section 2(d) of the LGPL refers to a "table of data", which seems to imply that word "data" is used to mean "information". On the other hand, Section 5 of the LGPL refers to "data structure layouts".

necessary for the application of the GNU license to Lisp. Indeed, it seems that the changes made by the LLGPL are grounded in an appreciation of the technical distinctiveness of Lisp rather than an analysis of whether these differences should change the interpretation of the LGPL or the application of copyright law.

What is a Derivative Work in Lisp?

As shown above, the second paragraph of the LLGPL aims only to generalise certain terminology of the LGPL. The third paragraph, however, seems to supersede several core provisions and principles of the LGPL. Indeed, as shown below, the third paragraph is best interpreted as an abrupt re-alignment of the thrust of the LGPL. Unfortunately, the LLGPL does not clarify the motivation for these changes. As such, it is not clear whether the provisions of the third paragraph are dictated by the technical aspects of Lisp or by philosophical differences with the LGPL.

The GNU licenses are built on the copyright law concept of the "derivative work." In very general outline, a derivative work incorporates and builds on a pre-existing copyrighted work. Under copyright law, one may generally not reproduce or distribute "derivative works" of a copyrighted work without an appropriate license. The GNU licenses leveraged this idea into the "copyleft": a license to modify and distribute an original copyrighted work, on the condition that any derivative works of the original work be distributed pursuant to specified license terms.[22] This idea sets the boundaries of the requirements of the GNU licenses and, generally, the GNU licenses are not intended to impose restrictions on works that are not "derivative works".[23]

The third paragraph of the LLGPL, however, seems to take a rather different approach. This section will individually examine each sentence of the paragraph, showing that the ideas underlying these provisions differ from the motivating principles of the LGPL. The third paragraph states in full:

The Library consists of everything in the distribution file set before any modifications are made to the files. If any of the functions or classes in the Library are redefined in other files, then those redefinitions ARE considered a work based on the Library. If additional methods are added to generic functions in the Library, those additional methods are NOT considered a work based on the Library. If Library classes are subclassed, these subclasses are NOT considered a work based on the Library. If the Library is modified to explicitly call other functions that are neither part of Lisp itself nor an available add-on module to Lisp, then the functions called by the modified Library ARE considered a work based on the Library. The goal is to ensure that the Library will compile and run without getting undefined function errors.

The first example in the third LLGPL paragraph provides that "[i]f any of the functions or classes in the Library are redefined in other files, then those redefinitions ARE considered a work based on the Library." In other words, if a LLGPL-licensed work contains a defined and named function, a licensee of the work may redefine that function in a separate and different file to provide for a

22 As per the explanation of the Free Software Foundation: "*To copyleft a program, we first state that it is copyrighted; then we add distribution terms, which are a legal instrument that gives everyone the rights to use, modify, and redistribute the program's code, or any program derived from it, but only if the distribution terms are unchanged*." See "Free Software Foundation, What is Copyleft?", available at http://www.gnu.org/copyleft/copyleft.html

23 For example, The Free Software Foundation has stated that it considers the phrase "works based on the Program" in the GPL to be similar though perhaps not identical to the definition of a derivative work under copyright law. See Opinion of the Denationalization of Terminology, Free Software Foundation, *available at* http://gplv3.fsf.org/denationalization-dd2.html. Whether the actual provisions of the GNU licenses respect this boundary, or try to impose restrictions on works that are not derivative works under copyright law, has been the subject of much commentary. See LAWRENCE ROSEN, OPEN SOURCE LICENSING 119-128 (2004).

different operation – however, such redefinition will be considered a "work based on the Library".[24] It is difficult to see how the act of redefining an existing function should in itself create a derivative work under copyright law. Of course, to the extent any redefinition incorporates material from the original definition, the redefinition could be seen as a derivative of the original. The LLGPL, however, considers any redefinition of the original function to be a "work based on the library", regardless of whether it incorporates material from the original definition. This provision is not based on the understanding of copyright law but, as will be shown below, on the rather different principle of ensuring the functionality of the original licensed work.

A good illustration of the philosophy of the LLGPL is provided by the last clause of the third paragraph. That provision states that "[i]f the Library is modified to explicitly call other functions that are neither part of Lisp itself nor an available add-on module to Lisp, then the functions called by the modified Library ARE considered a work based on the Library." As noted earlier, Lisp allows users to call "foreign modules" written in a different programming language.[25] This clause provides that such called foreign modules will be deemed a "work based on the library" – in other words, a derivative work of the library which may only be distributed under the terms of the LLGPL.[26] This provision is at odds with the LGPL in several ways. First, subject to certain restrictions, the LGPL typically allows third party modules to be linked to the licensed work, without requiring that such third party modules themselves be licensed under the LGPL.[27] This LLGPL provision, on the other hand, provides that certain third party modules, even though they do not incorporate code of the licensed work, must also be licensed under the terms of the LLGPL. Second, the LGPL generally only imposes restrictions on modules to the extent they are compiled or linked with the original library, but this LLGPL provision seems to impose restrictions on such linked modules regardless of whether they are actually linked or compiled with the library. Third, the LLGPL provision relaxes the restriction in respect of functions that are "part of Lisp itself" or "an available add-on module to Lisp". There is no equivalent in the LGPL to the relaxing of restrictions solely in respect of modules written in a particular programming language.

What is the LLGPL's motivation for providing different requirements than the LGPL? The last clause of the third paragraph sets forth the underlying philosophy of these provisions, stating that: "[t]he goal is to ensure that the Library will compile and run without getting undefined function errors." In other words, these provisions of the LLGPL are not based on the copyright principles of the LGPL. Rather, they are motivated by the goal of ensuring that a modified licensed work "continues to compile and run."[28] This philosophy is reflected in the provisions of the LLGPL discussed above. These provisions state that certain redefinitions and foreign modules are subject to the full restrictions of the LLGPL, even though they would not be considered "derivative works" under ordinary circumstances.

24 Redefining an existing system function is permitted under the Common Lisp standard, though not generally recommended because of the unintended consequences that such redefinitions can generate. For example, the Allegro Common Lisp 8.2 documentation states that "Lisp permits already-defined functions to be redefined dynamically. However, redefining system-defined functions … is almost always a bad idea." *See* http://www.franz.com/support/documentation/7.0/doc/packages.htm. For a discussion of some problems associated with the redefinition of functions in Lisp, see SEIBEL, *supra* note 16, at 274-75.

25 *See supra* note 15.

26 An interesting question not expressly addressed by the LLGPL is whether a derivative of an LLGPL-licensed work must be distributed under the terms of the LLGPL itself or may rather be distributed under the terms of the LGPL. The LLGPL is not clear on this point.

27 It should be noted that the Allegro Common Lisp 8.2 documentation states that foreign functions are "linked" to a running Lisp process. *See* http://www.franz.com/support/documentation/7.0/doc/foreign-functions.htm#ff-intro-1. This is distinct from other Lisp code which is actually loaded into the memory of a running Lisp image rather than linked.

28 The LLGPL is not clear as to why applying the LGPL obligations to redefinitions and foreign functions ensures that they will continue to "compile and run". It is possible that the LLGPL believes that requiring the source code of these elements to be distributed will allow the modified library to be debugged.

Moving back to earlier provisions of the LLGPL, the second and third clauses of the third paragraph are also inspired by the same goals of ensuring functionality. The third clause provides that "[i]f Library classes are subclassed, these subclasses are not considered a work based on the Library." The question of whether subclassing creates a derivative work has been raised in other contexts. For example, in the GPL FAQs, the Free Software Foundation takes the position that subclassing creates a derivative work, without offering an explanation for that position.[29] Other commentators have taken different positions.[30] The LLGPL provides that subclasses will not be considered derivative works.[31] Again, the reasoning of the LLGPL seems to be based on whether the subclass could possibly interfere with the functionality of the original library. As simply adding the subclass would not interfere with the functionality of the original defined class, the LLGPL takes the position that the subclass should not be considered a "work based on the library".

The second clause takes the same approach, stating that "[i]f additional methods are added to generic functions in the Library, those additional methods are NOT considered a work based on the Library." Again and briefly, in Common Lisp, "methods" are various implementations of an abstract definition of a generic function in a variety of circumstances. For example, a generic function may state that it operates to draw shapes, without actually providing an implementation of that functionality. The specific methods of that generic function, however, provide the functionality for actually drawing a variety of shapes.[32] The question of whether adding additional methods to a generic function creates a derivative work is an interesting question, and beyond the scope of this article. However, it should be noted that again the approach of the LLGPL is not to ask whether the addition of methods to a generic function creates a derivative work, but rather to ask whether the modifications will preserve the functionality of the original library. Methods may be added or removed to a generic function without affecting the functionality of the generic function itself. As such, according to the LLGPL, the additional methods are not considered a work based on the library.

This section has shown that with regard to the question of what constitutes a "work based on the library," the LLGPL takes a very different approach than the original LGPL license. While the restrictions of the original license were based on an understanding of a "derivative work" under copyright law, the obligations of the LLGPL seek to ensure the functionality of the licensed program. In implementing these goals, the LLGL provides for very different requirements and obligations than the original LGPL.

Distribution

Another distinctive feature of Lisp – albeit a feature that has since been adopted by other languages – is the fact that Lisp programs are traditionally constructed within a dynamic run-time environment. Lisp programs may be developed incrementally by composing or loading program statements into the run-time environment, and the run-time environment will interpret or compile

29 See http://www.gnu.org/licenses/gpl-faq.html#OOPLang (stating that "[s]ubclassing is creating a derivative work. Therefore, the terms of the GPL affect the whole program where you create a subclass of a GPL'ed class."). The FSF takes the same position in its article "The LGPL and Java", where it states that " [i]nheritance creates derivative works in the same way as traditional linking". See http://www.gnu.org/licenses/lgpl-java.html. Version 3 of the LGPL expressly addresses the question of subclassing, *see* infra note 31.

30 *See, e.g.*, Derivative Works, http://www.law.washington.edu/lta/swp/law/derivative.html (arguing against the position of the Free Software Foundation regarding subclasses).

31 Unfortunately, the LLGPL does not actually clarify whether subclasses will be subject to the obligations imposed by the LGPL with regard to work linked with the licensed work. In contrast, Version 3 of the LGPL clarifies that "defining a subclass of a class defined the Library is deemed a mode of using an interface provided by the Library." As such, under Version 3 of the LGPL, a work that defines a subclass is subject to the usual LGPL obligations in respect of works that link with the licensed library. As such, it must be distributed under terms that "do not restrict modification […] and reverse engineering for debugging such modifications."

32 This example is taken from Seibel, *supra* note 16, chapter 6.

such forms as they are entered.[33] The Lisp run-time environment also impacts how programs are distributed. Unlike the C programming language, Lisp implementations do not generally offer the possibility of compiling source code into executables. Rather, Lisp programs may be distributed as a Lisp run-time environment together with an "image file", which is a saved representation of the state of the Lisp program. Alternatively, Lisp programs may also be distributed as a run-time environment together with compiled FASL files.[34] In other words, the distribution of a Lisp program often requires the distribution of files together with a run-time environment that will execute those files.

The fourth paragraph of the LLGPL addresses this distinct process of building and distributing Lisp applications. According to the LLGPL, applying the LGPL to these aspects of Lisp requires fundamental changes in the requirements and obligations of the GNU license. The fourth paragraph begins by providing an interpretation of the linking provisions of the LGPL:

Section 5 of the LGPL distinguishes between the case of a library being dynamically linked at runtime and one being statically linked at build time. Section 5 of the LGPL states that the former results in an executable that is a "work that uses the Library." Section 5 of the LGPL states that the latter results in one that is a "derivative of the Library", which is therefore covered by the LGPL.

Unfortunately, these provisions paint an inaccurate picture of the LGPL's requirements. Section 5 of the LGPL – notwithstanding the interpretation presented in the sentences above – does not distinguish between works that are statically or dynamically linked to an LGPL-licensed library. Rather, Section 5 clarifies that certain independent works which use an LGPL-licensed library can under certain circumstances become derivative works of that library. According to Section 5, a work may become a derivative work of the library even though the source code of that work does not contain portions of the library: the act of linking or compiling the work with the library will cause portions of the library to be incorporated in the linked or compiled work, and this linked or compiled work will then be seen as a derivative work of the library. Again, in providing that works may become a derivative work of the library, the LGPL does not distinguish between statically or dynamically linked works.[35] Indeed, Section 6 of the LGPL expressly contemplates that works may be either statically or dynamically linked with the library, and provides obligations for a party distributing both kinds of linked works.[36]

The next sentences of the LLGPL apply the previous (incorrect) interpretation of the LGPL to the Lisp context:

> *Since Lisp only offers one choice, which is to link the Library into an executable at build time, we declare that, for the purpose applying*

33 Third party Lisp libraries may similarly be loaded into the run-time environment, either as source code or as compiled files. See SEIBEL, *supra* note 16, at 17, 475.

34 Compiled Lisp files are referred to a FASL files, which stands for "fast-load file". Loading compiled Lisp files into the run-time environment can result in a faster and more efficient program. FASL files can be implementation dependent and may not be compatible between different implementations of Lisp. SEIBEL, *supra* note 16, at 475, n. 8.

35 On the other hand, certain commentators have differentiated between static and dynamic linking in determining the effect of the licenses. *See generally* "Working Paper on the legal implications of certain forms of Software Interactions (a.k.a linking)", which is available online at http://www.ifosslr.org/public/LinkingDocument.odt .

36 Section 6(a) of the LGPL addresses a situation where the "work that uses the library" is distributed as an executable linked with the library, and requires that the source or object code of the "work that uses the library" also be provided along with the linked work. This situation is colloquially referred to as statically linking the work with the library. Section 6(b) of the LGPL addresses a situation where the "work that uses the library" is linked to the library through a "shared library mechanism", which uses a copy of the library at run-time. This situation is colloquially referred to as dynamically linking the work with the library. In other words, both static and dynamically linking are governed by Section 6 of the LGPL. The preamble of the LGPL expresses the same when it states that "[w]hen a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library."

> *the LGPL to the Library, an executable that results from linking a "work that uses the Library" with the Library is considered a "work that uses the Library" and is therefore NOT covered by the LGPL.*

Because of this declaration, section 6 of LGPL is not applicable to the Library.

These provisions raise several problems. First, the statement that Lisp offers only the possibility of linking "the Library into an executable at build time" is not representative of all implementations of Lisp. It is correct that Allegro Lisp offers commercial licensees the possibility of creating an executable application – essentially a directory that contains the Allegro Lisp run-time environment, an image file, and a "license file" which together allow execution of the program.[37] It is not correct, however, that this is the only available method for distributing a Lisp application. Other commercial and free implementations of Lisp offer other alternatives for distribution. Lispworks, for example, provides the possibility of delivering an application as a dynamic library.[38] Allegro Lisp itself also offers additional options for distributing Lisp programs. For example, Allegro users that do not wish to build an executable may also distribute Lisp source code or compiled FASL files, and these files can be used by a user that already has an Allegro Lisp run-time system.[39] In addition, compiled FASL files may sometimes be distributed separately as patches to an application already executing on a run-time environment.[40] Free Lisp implementations also offer the possibility of saving a memory image of a running Lisp system, which can then be loaded by another user of the free implementation.[41] In other words, Lisp implementations offer a wide variety of distribution methods that are not addressed by the LLGPL.

Aside from the question of how a Lisp library may be distributed, the fourth paragraph of the LLGPL raises questions regarding the objectives and ambitions of the license. Indeed, the effect of the fourth paragraph of the LLGPL is to almost eviscerate the obligations of the LGPL. If Lisp programs can by definition only be distributed as an executable (an assumption that, as shown above, is not completely accurate), and such executables are stipulated as not being subject to the obligations of Section 6 of the LGPL, then the copyleft obligations of the LGPL will by definition never apply to any Lisp program. The weaker copyright obligations of the LGPL generally require linked applications to be distributed "in a form that allows for modification and relinking of the library," or pursuant to terms that "allow modification of the work … and reverse engineering for debugging such modifications." Under the LLGPL, however, even these weak copyleft obligations would never apply.

In other words, under the LLGPL, the copyleft provisions of the LGPL are essentially replaced with a rather permissive license.[42] Indeed, the copyleft obligations of the LLGPL may be

37 See http://www.franz.com/support/documentation/8.2/doc/runtime.htm.

38 *See* http://www.lispworks.com/documentation/lw61/DV/html/delivery-42.htm#pgfId-865189. Corman Lisp provides similar functionality. *See* Corman Lisp Common Lisp Development Environment*, available at* http://www.cormanlisp.com/CormanLisp/CormanLisp.pdf, page 73

39 *Id.* (providing that "[n]ote that because your source files and compiled versions of those files can be distributed without restriction, the way to distribute an application to another licensed Allegro CL customer without worrying about license agreement restrictions is to distribute your source files (and/or compiled versions of your source files), along with a file which creates the application."

40 Currently, however, not all commercial licenses to Allegro Lisp offer the rights to distribute a run-time environment together with a compiler that can read FASL files. See Franz's description of various runtime environment options, infra note 35. See also the short discussion regarding non-free runtime environments, *infra* text accompanying notes 42 - 43.

41 See http://www.sbcl.org/manual/index.html#Saving-a-Core-Image and http://www.clisp.org/impnotes.html#image

42 The LLGPL also seems to do away with several other obligations of the LGPL. For example, Section 6 of the LGPL also requires the provision of notices that the library is included in the work and that the library is covered by the LGPL. Section 6 also requires the retention of copyright notices. By broadly providing that Section 6 of the LGPL is not applicable to Lisp programs, the LLGPL seems to eliminate these requirements. In addition, to the extent the LLGPL can be applied to programs that are covered by Version 3 of the LGPL, the LLGPL may also eliminate the

summarised in the last sentence of that license: "[h]owever, in connection with each distribution of this executable, you must also deliver, in accordance with the terms and conditions of the LGPL, the source code of Library (or your derivative thereof) that is incorporated into this executable." In other words, under the LLGPL a licensee's copyleft obligations are limited to delivering the source code of the library itself.[43]

It should be noted that the permissive nature of the LLGPL stands in contrast to the fact that commercial Lisp applications are often developed using non-free platforms. For example, despite the fact that a particular application may be available under a permissive license, a commercial license to Allegro Lisp may nevertheless be necessary to run or modify that application. In addition, libraries developed with one implementation of Lisp are often not portable to another implementation.[44] As such, even though a particular application may be licensed under open source terms, a commercial license to a specific Lisp run-time environment may also be required to use that application. As such, the development of free and open source software in Lisp may require attention to both the license applicable to a particular program as well as the platform for which the software is developed.[45]

Of Macros

This section discusses two distinctive features of Lisp which are not clearly addressed by the LLGPL. First, Lisp contains "macros" – methods of defining new syntactical structures in Lisp – a feature not available in C or most other languages. Second, unlike C, Lisp programs have traditionally been constructed within a run-time interactive environment that interprets Lisp expressions. Neither of these features is expressly addressed by the LLGPL, and both raise issues regarding the interpretation and application of the LGPL. This section provides a brief overview of these features and the concerns they may raise in an open source license.

Macros are a distinctive feature of Lisp. In brief, macros are program snippets which take in Lisp code as input, manipulate that code, and return different Lisp code that is executed at runtime in place of the original code. Through such manipulations, Lisp macros allow users to extend the syntax of the language and create new constructions that can clarify and shorten code. Lisp macros differ from functions. Functions take arguments, and these function arguments are evaluated when the functions are executed. In contrast, the arguments in macros are not evaluated when the macro in executed. Instead, the macro returns code containing the unevaluated arguments, and these arguments are evaluated when the returned, macro-manipulated code is executed. Lisp macros also differ from C macros: while a C macro is essentially a textual search-and-replace mechanism, a Lisp macro provides a more general mechanism for generating code that preserves the data structures of the original code.[46]

How should the obligations of the LGPL affect the use of macros? On the one hand, it is not clear why macros should be treated differently than functions. Why should a Lisp program that uses the

requirement to provide installation information as required by the "Tivo" clause of the LGPL.

43 One ambiguity in this final, limited obligation of the LLGPL is the requirement to disclose the source code of the library and "your derivative thereof". It is difficult to clearly define what the requirement to disclose derivative works of the library refers to, since the LLGPL previously provided that works linked to the library do not constitute derivative works of the library and are not covered by the LGPL. This last requirement to disclose derivative works could either be seen as conflicting with the prior provisions of the LLGPL, as a requirement to disclose modifications to the library files themselves, or as some other undefined intermediate copyleft obligation.

44 Seibel, *supra* note 16, at 465, 475 n.8.

45 The Free Software Foundation described a similar problem with Java before Sun relicensed its Java implementation under the GPL. *See* http://www.gnu.org/philosophy/java-trap.html.

46 Much more complete explanations of the use and functionality of Lisp macros can be found in Seibel, *supra* note 16, ch. 7-8; Paul Graham, On Lisp, ch.7-8 (1993)

macros of a third party library be any less of a derivative work of that library than a Lisp program that uses the functions of that third party library? While macros do not constitute functions in the technical Lisp sense of the word, they do provide "functionality" and perform "operations" as those words are commonly understood.[47] On the other hand, the code generated by the macro may bear little resemblance to the text of the macro itself.[48] As such, it may not always be possible to say that a program that calls a macro incorporates the textual code of the macro. Rather, it may sometimes be more correct to say that the program that calls a macro incorporates code generated by the macro – and the LGPL itself provides that the output of a program need not necessarily constitute a derivative work of that program.[49] To make matters more confusing, Lisp macros are not expanded at either compile-time or run-time, but rather at an intermediate stage called macro-expansion time. Would this complicate the application of Section 6(b) of the LGPL, which defines a "suitable shared library mechanism" as a mechanism that uses a copy of the library already present on the user's system at "run-time"?

Answering the previous questions requires the untangling of complex legal and technical threads, and it is not the aim of this article to present a detailed analysis of these questions. However, any license tailored for Lisp should take a position on these questions in order to provide for legal clarity. It is unfortunate that the LLGPL does not provide any express guidance on the effect of the LGPL on macros.

As discussed earlier, the use of runtime environments is another distinctive feature of Lisp, a feature that has been adopted by other languages. A Lisp program may be developed incrementally by composing or loading functions into the run-time environment. Third party Lisp libraries may similarly be loaded into the run-time environment, either as source code or as compiled files. The Lisp run-time environment also impacts how programs are distributed. Lisp programs are generally distributed as a Lisp run-time environment together with either compiled FASL files or an "image file". Depending on the specific implementation, it may not be possible to distribute a single executable file for a Lisp program.

How should the LGPL relate to two functions loaded into the same Lisp run-time environment? Would the two functions be considered linked in the LGPL sense of that word? On one hand, linking two code files in the standard sense involves both the creation of links between the two files and the copying of the linked file (whether at build-time to create an executable or at run-time into memory to execute the program) into a larger program structure. In contrast, Lisp libraries present in an environment are already loaded into memory and do not need to be copied when a library function is called.[50] As such, it may be possible to assert that a program which uses a Lisp library already loaded into a runtime environment should not be considered "linked" to that function, and should not be subject to any obligations of the LGPL. On the other hand, the libraries present in the Lisp environment seem to satisfy the literal LGPL definition of "shared library mechanism", which is defined by the LGPL as using "at run time a copy of the library already present on the user's computer system".[51] As such, perhaps a program that uses a function loaded

47 See Graham, *supra* note 44 at 82 ("Since macros can be called and return values, they tend to be associated with functions. Macro definitions sometimes resemble function definitions, and speaking informally, people call do, which is actually a macro, a "built-in function." But pushing the analogy too far can be a source of confusion."); *Id.* at 84 ("Indeed, a macro is really a Lisp function –one which happens to return expressions").

48 For examples of how much a macro text can differ from the expanded macro program, see Paul Graham, *supra* note 46 at 97-98.

49 The preamble of the LGPL states that "[t]he act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does."

50 See Gary D. Knott, Interpreting Lisp, available at http://www.civilized.com/files/lispbook.pdf for a fuller explanation of how a Lisp interpreter stores library functions in memory.

51 The Free Software Foundation takes the position that if an interpreter includes certain facilities and "the interpreter is

into a runtime environment should be subject to the same obligations as programs that use a "shared library mechanism".

As with the question of macros, a license made for Lisp should provide a ready answer to the questions raised by the run-time environment. Again, it is unfortunate that the LLGPL does not provide easily applied rules for these questions.

Conclusion

One of the central assumptions of the LLGPL is that the GNU licenses, having been drafted with attention to a specific programming language, may need to be clarified for other programming languages. Indeed, the LGPL makes reference to technical details, such as the header files, linking and compilation, which are not applicable to all programming languages or to all situations. Even so, this article has shown that the clarifications made by the LLGPL to the original GNU license are largely unnecessary, and that the LGPL would probably be interpreted in a similar fashion without the clarifications proposed by the LLGPL. This is not to say, of course, that the LGPL comprehensively and expressly addresses all issues – as discussed, it does not expressly address the issues raised by Lisp macros or the Lisp runtime environments.

Licenses are legal documents, and chances are that their definitive legal interpretation will be made by persons with legal training but only a limited technical background. As such, to some extent, it is comforting that the interpretation of the GNU license does not depend on the details of specific programming languages. The technical detail necessary to draft similar license provisions for each and every technical context might prove too jargon-filled for the average court to apply.[52] On the other hand, the fact that such technical detail is not expressly included in the license text will not discharge a court from its obligation to understand such detail in order to properly interpret the license. In applying any software license, a court will need to understand the technical background regardless of whether it is clearly expressed in the text of the license.

In drafting software licenses – especially copyleft licenses that often refer to technical detail – it may be useful to keep these principles in mind. A well drafted license should not include an amount of technological detail that overwhelms the non-technical reader. On the other hand, it should to the extent possible provide for rules that are easy to interpret and apply in specific technical contexts. Balancing these often competing objectives is not a simple task. Nevertheless, having clear and easy to apply license terms will only increase the attractiveness of using open source software.

linked statically with these libraries or if it is designed to link dynamically with these specific libraries" then interpreted programs can be considered derivative works of those facilities. See GPL FAQs. This statement does not answer the questions raised in this section. First, the statement only addresses libraries that are statically linked or if the interpreter is designed to link dynamically with specific libraries – but not the situation of a library loaded into the run-time environment.

52 See Rosen, *supra* note 23, at 123-24 (stating that Section 2(d) of the LGPL is "an impenetrable maze of technological babble. They should not be a general-purpose software license.")

About the author

Eli Greenbaum *is an attorney at Yigal Arnon & Co. in Jerusalem, Israel, specialising in intellectual property law and transactions.*