Designing Open Source Licenses[*]

Kim-Sau Chung[†] Melody Lo[‡]

August 13, 2021

Abstract

Open source licenses are noted for often being self-referential. The two most prominent examples of open source licenses are GPL and BSD. GPL says the next developer cannot go proprietary, and can only go open source with the same license, namely GPL. BSD says the next developer can go proprietary, and can also go open source with any open source license, including BSD. We provide a framework to study other potentially self-referential open source licenses. We construct the universal space of all possible open source licenses that allow finitely many options for the next developer, and explain why GPL and BSD naturally stood out from other licenses as the two most prominent choices for developers going open source.

Keywords: open source, licenses, self-referential, GPL, BSD, universal space

JEL Classifications: L86, O36

1 Introduction

After a software developer develops a software, there are at least two ways to distribute it. The first way is to go proprietary, meaning that she sells copies of the binary code for a profit. Since binary code is difficult to interpret, it deters the others from disabling its security device and making illegal copies. But it also makes it difficult for future developers to learn from her source code. In other words, while going proprietary allows

[*]We thank Yi-Chun Chen, Junichiro Ishida, Chia-hui Lu, and William Taysom for very helpful discussions. Chung and Lo are co-first authors. The work described in this paper was supported by HKBU (via grant RC-IG-FNRA/17-18/01).

[†]Department of Economics, Hong Kong Baptist University; `kimsauchung@gmail.com`

[‡]Department of Economics, Hong Kong Baptist University; `peiyulo2006@gmail.com`

the developer to profit from her effort, it also stifles further development of her original idea.

Another way the developer can distribute her software is to go open source, meaning that she makes her source code open for everyone to see and copy. By going open source, she forgoes the profit she could have made from selling copies of the binary code, but she can gain utilities when future developers, inspired by her source code, develop more advanced versions by adding extra functionalities to her original software. Such utilities may come from pride, prestige, sheer joy of seeing her idea further developed, or the satisfaction of using a more advanced version of her original software.

There are two major kinds of license a developer may use when she decides to go open source. The first is the restrictive kind, as exemplified by the GPL license.[1] By sharing her source code using the GPL license, the developer is telling future developers, "You can use my source code to develop a more advanced version. But if you ever want to distribute your advanced version, you have to go open source instead of going propriety, and you have to go open source by using the same license I am using, namely the GPL license." This kind of open source licenses are restrictive because they restrict how future developers can distribute their software. In particular, the restriction is a "share-alike" requirement, requiring that future developers share in a manner like how the original developer shares her source code. In a sense, the GPL license is defined in a self-referential way because of this "share-alike" requirement.

One the most famous developers who went open source with the GPL license is Linus Torvalds. Ever since Linus Torvalds distributed the source code of the first version of Linux using the GPL license, all subsequent versions have to be shared alike using the same GPL license. Even when Red Hat, a for-profit company, develops their commercial version of Linux, called Red Hat Enterprise Linux (RHEL), they are also bound by the GPL license to share their source code using the same GPL license. It means that they cannot make a profit by selling copies of the binary code of RHEL. Instead, they can only make a profit by selling complementary services. Today, Linux powers millions of smart devices, including smart phones, smart TVs, tablet computers, etc.

The second kind of license a developer can use when she goes open source is the permissive kind, as exemplified by the BSD license.[2] By sharing her source code using

[1]GPL stands for "general public license".

[2]BSD stands for "Berkeley software distribution".

the BSD license, the developer is telling future developers, "You can use my source code to develop a more advanced version, and you can distribute your advanced version in whichever way you like. You can go propriety, you can also go open source. If you choose to go open source, you can use any open source license you like, including the BSD license."

One of the most famous developers who went open source with a BSD-like permissive license is Donald Knuth. While the core of TEX is copyrighted by Donald Knuth (the developer of TEX) and no changes are permitted, most add-on programs are licensed under the LATEX Project Public License (LPPL), which is very much like a BSD license (Gaudeul, 2007). The permissive nature of LPPL allows subsequent developers who developed more advanced versions to go proprietary and profit from their efforts. This option encourages many developers to participate in developing various add-on programs, with Scientific Workplace being one of the most profitable examples.

The open source movement is nothing short of a revolution in how production is organized. Many of the most valuable software (such as Linux, LATEX, Apache, etc.) might never have been developed if developers had not learned how to share their ideas using open source licenses. The movement also has impacts reaching far beyond the software industry. Projects with user-generated contents such as Wikipedia might never have been possible if contributors had not learned how to share their contributions using Creative Common licenses, which in turn were inspired by open source licenses.

It is important to recognize that open source licenses such as GPL and BSD are inventions of idealistic programmers, who invented these licenses more as manifestos of their anti-capitalism ideologies, instead of as calculated designs that maximize the productivity gain made possible by a new mode of collaboration. Therefore, it is conceivable that only part but not all of the productivity gain has been unleashed.

The goal of this paper is to investigate (*i*) what open source licenses other than GPL and BSD may be possible, (*ii*) under what circumstances these other open source licenses may serve purposes that neither GPL nor BSD can serve, and (*iii*) under what circumstances it is without loss of generality for any developer going open source to choose only between the GPL and the BSD licenses.

Although the open source revolution has rightfully attracted a lot of economic studies,[3]

[3]See Subsection 1.1 for a review of this literature.

literally fewer than a handful of them have likewise studied open source *licenses*, notwithstanding the significant role of these licenses in the open source revolution. Lerner and Tirole (2005b) provide an extremely stylized model that differentiates different open source licenses by only one parameter, namely their "permissiveness". Their model abstracts away even the most defining features of different open source licenses, such as GPL's "share-alike" requirement and BSD's permission to go proprietary, rendering it impossible to discuss the optimality of, say, a new license that mixes-and-matches these features. In two unpublished working papers, Gaudeul (2004, 2005) compares GPL and BSD using a model less stylized than that of Lerner and Tirole (2005b). In particular, she explicitly models GPL's "share-alike" requirement and BSD's permission to go proprietary. Her models can accommodate up to three generations of developers—the original developer, the second-generation developers who further develop the original developer's idea, and the third-generation developers who further develop the second-generation developers' ideas. Given her particular model setup, complexity explodes exponentially in the number of generations, rendering further extensions beyond three generations analytically intractable.

This paper presents a model of open source licenses that has the following two properties. First, it is rich enough that the defining features of GPL (its "share-alike" requirement) and BSD (its permission to go proprietary) can be described explicitly, and open source licenses other than GPL and BSD can be meaningfully discussed (in contrast to Lerner and Tirole, 2005b). Second, it is tractable enough so that more than three generations of developers can be studied (in contrast to Gaudeul, 2004, 2005).[4]

In Section 2, we first present a basic model that only allows for the GPL and the BSD licenses. Notwithstanding this restrictive choice set, the basic model contains two key components that we shall carry over to more general models later. The first key component is a sequence of software developers, each one can be inspired to develop a more advanced version of a particular software only if he has the opportunity to study the source code of the previous version, which in turn is possible only if the previous developer went open source. The second key component is that each developer is altruistic in the

[4]Gaudeul (2004, 2005) allows every generation-t software to inspire multiple generation-$(t+1)$ developers, each competing with the others in the market. As a result, the number of generation-t developers explode exponentially in t. Our model, in contrast, assumes that every generation-t software can only inspire one generation-$(t+1)$ developer. Our assumption is less realistic, but it renders our model more tractable.

sense that he cares not only about his own profit but also about consumer surplus, and hence may sometimes be willing to sacrifice his own profit by going open source if that allows future developers to further develop his idea and generate more consumer surplus.

In Subsection 2.1, we use a numerical example to illustrate that both GPL and BSD can sometimes be the optimal choice for a developer (say, developer t) going open source. Intuitively, developer t would like to see his idea further developed. However, development incurs development costs. In the unfortunate event that the next developer (i.e., developer $t + 1$) finds his development cost high, he will be discouraged from further developing developer t's idea, especially if he is also prohibited from going proprietary and making a profit out of his effort. If development cost and potential profit are positively correlated such that a higher potential profit typically accompanies a higher development cost, then the BSD license, by allowing developer $t + 1$ to go proprietary, will help encourage developer $t + 1$ to further develop developer t's idea even in this unfortunate event. The BSD license hence can be the optimal choice for a developer going open source if development cost and potential profit are positively correlated, while the GPL license can be the optimal choice in the case of negative correlation.

In Section 3, we slightly extend our basic model by introducing two new open source licenses. This exercise illustrates how our framework allows us to conceive new, never-heard-of open source licenses, and why it can be difficult to summarize an open source license by a signle parameter called "permissiveness" as in Lerner and Tirole (2005b).

To illustrate that the newly introduced open source licenses are not bogus, we use a numerical example in Subsection 3.1 to show that at least one of these two new open source licenses can sometimes serve purposes that cannot be served by either GPL or BSD.

To the extent that the way we introduce new open source licenses in Section 3 looks *ad hoc*, we show in Section 4 that one can systematically study "all" possible open source licenses by constructing what we call the universal space of open source licenses. Our construction resembles the construction of the universal type space in epistemic game theory (Mertens and Zamir, 1985; Brandenburger and Dekel, 1993; Mariotti, Meier, and Piccione, 2005), except for that we proceed without topology. We show that every space of open source licenses that permit finitely many options for the next developer can be embedded as a subspace in the universal space of open source licenses.

In Section 5, we introduce an axiom called imposture-proofness to exclude some open source licenses that can be gamed by a developer (say, developer t) who splits his more advanced version of the software (i.e., version t) into two consecutive versions: version $t.1$ and version $t.2$, with only version $t.1$ subject to the restrictions contained in developer $t-1$'s open source license. Among other implications of imposture-proofness, we show that any imposture-proof open source license that does not allow the next developer to go proprietary is identical to the GPL license.

In Section 6, we use this result from Section 5 to construct an environment where, in any equilibrium, if developer 0 is ever going to go open source, he cannot do better than going open source with either the GPL or the BSD license. This result is of interest because it sheds light on why GPL and BSD arose as the two most prominent open source licenses in the history of the open source movement.

Finally, in Section 7, we conclude with some remarks on possible future research.

1.1 Related Literature

The open source revolution has rightfully attracted a lot of economic studies. Lerner and Tirole (2002, 2005a) provide some early introduction of this revolution to economists. The subsequent studies can be roughly divided into three strands. The first strand of studies are interested in the competition between a commercial software company and an open source community—for example, what prices the commercial software company would set, the chance that it can survive this competition, etc. The open source community is typically modelled as a group of altruistic developers who both contribute to and benefit from the open source movement. How their collaboration is facilitated or jeopardised by different choices of open source licenses is typically not the focus. Instead, the license being used is typically assumed (implicitly) to be GPL, and hence no member of this community would have the option of going proprietary. These studies typically focus on factors other than the open source license. For example, Johnson (2002) is interested in the effects of the size of the open source community, Casadesus-Masanell and Ghgemawat (2006) are interested in the effects of demand-side learning, and Economides and Katsamakas (2006) are interested in the effects of network externalities.

This paper differs from this strand of studies in that it focuses explicitly on open source *licenses*—what options other than GPL and BSD do we have, when will they serve purposes that neither GPL nor BSD can serve, and when can they be ignored without loss of generality?

A second strand of studies are concerned with the exact kind of altruism that motivates many members of the open source community—are they motivated by warm glow, pride, the knowledge that someone else benefits from their efforts, or are they merely motivated by the material payoffs from being able to signal their competence? For example, Athey and Ellison (2014) theoretically study how different forms of altruism on the part of the community members affect the competition between a commercial software company and an open source community; while Hertel, Niedner, and Herrmann (2003), Roberts, Hann, and Slaughter (2006), and Fershtman and Gandal (2007) empirically study related questions.

In comparison to this strand of studies, our model makes specific assumptions on the kind of altruism that motivates any developer to go open source. In particular, we assume that he internalizes part of the consumer surplus that may be generated by future developers after he goes open source. We have not explored the implications of other kinds of altruism such as warm glow.

A third strand of studies are interested in the (mostly informal) governance structure of the open source community. Most of these studies are empirical in nature, taking the form of meticulous case studies of selected open source projects, and taking advantage of the fact that, for most such projects, the whole history of interactions among the members of the corresponding community is stored as log files in the public domain. Examples of these studies include Fielding (1999) on the Apache project, Mockus, Fielding, and Herbsleb (2002) on the Apache and Mozilla projects, and Han and Xu (2019) on the Python project. See also von Hippel and von Krogh (2003) and Johnson (2006) on some general insights distilled from these empirical studies.

In comparison to this strand of studies, our model assumes that each generation of the software is developed by one and only one developer. The very interesting question of how different developers are organized together to collaboratively upgrade the software to the next generation hence cannot be investigated in this stylized model.

2 The Basic Model

In this section, we shall first describe a basic model that is barely rich enough to accommodate the GPL and BSD licenses. We will explain how this basic model can be extended to accommodate other open source licenses in subsequent sections.

Consider a discrete-time model, where time is indexed by $t = 0, 1, 2, \ldots$. In every period t, there is one and only one developer, called developer t, who has the potential of developing a software, called software t. Note that we abuse notation by using the same index, t, for time, for developer, and for software.

We can think of developer 0 as an original developer such as Linus Torvalds or Donald Knuth, and software 0 is his original software. For any $t > 0$, we can think of software t as a more advanced version of software $t - 1$, perhaps by adding extra functionalities. Of course, indirectly via software $t - 1$, software t is also a more advanced version of any software $s < t - 1$.

Developer t will be able to develop software t only if he has an opportunity to learn from the source code of software $t - 1$. Apparently, this cannot happen if developer $t - 1$ chose to go proprietary; i.e., to sell copies of the binary code for a profit, instead of sharing the source code with the others. Therefore, we shall assume that, if developer $t - 1$ goes proprietary, none of the software t, $t + 1$, $t + 2 \ldots$, etc., can be developed. In other words, the game ends after developer $t - 1$ goes proprietary.

We assume that if developer $t-1$ did not go proprietary, then he must go open source; i.e, sharing his source code with the others using one of the open source licenses. In other words, keeping the software private is not an option. This is not a strong assumption. When a developer is not going to make a profit out of his creation, any tiny preference of sharing will prompt him to share.

If developer $t - 1$ chose to go open source, thus enabling developer t to develop software t, the game continues to period t. In period t, developer t makes two decisions in sequence. First, he draws a development cost, $c_t \geq 0$, and a potential profit, $\pi_t \geq 0$, from a joint distribution $P(c, \pi)$. We assume that these draws are IID across time.

Upon observing (c_t, π_t), he decides whether to pay the development cost c_t and develop software t. If he decides not to, the game ends.

If he decides to develop software t, he then decides whether to go proprietary or to go open source using one of the open source licenses. Some of these options may not be available, depending on the open source license chosen by developer $t-1$. We first enumerate the three different options developer t may be allowed to choose, and then explain which subsets he is allowed to choose from given different open source licenses chosen by developer $t-1$.

P: *going proprietary*

Developer t goes proprietary, realizing the potential profit π_t, and the game ends. We assume that the consumer surplus, w_t, generated from a proprietary software is w.

G: *going open source with the GPL license*

Developer t goes open source, thus enabling developer $t+1$ to develop software $t+1$. However, if developer $t+1$ chooses to develop software $t+1$, he has to go open source with the GPL license (i.e., choosing **G**) as well. Potential profit π_t is forgone, and the consumer surplus, w_t, is W, where $W > w$ (reflecting the fact that the software is distributed for free).

B: *going open source with the BSD license*

Developer t goes open source, thus enabling developer $t+1$ to develop software $t+1$. If developer $t+1$ chooses to develop software $t+1$, he can either go propriety (i.e., choosing **P**), or go open source with any of the two licenses (i.e., choosing **G** or **B**). Potential profit π_t is forgone, and the consumer surplus, w_t, is W.

We should hasten to emphasize that the consumer surplus w_t (which equals either w if developer t goes proprietary, or W if he goes open source) should more appropriately be understood as the *internalizable* part of the consumer surplus. It is the part that current and past developers (i.e., developers $s \leq t$) who went open source can internalize. It is likely only a small part of the whole consumer surplus. In particular, we do not presume that $W > \pi_t + w$, which would have been a natural inequality to assume had we interpreted w_t as the whole consumer surplus. Indeed, allowing for $\pi_t + w > W$ is necessary to explain why developer t may sometimes go proprietary.

We assume that all three options (**P**, **G**, and **B**) are available to developer 0. For developer $t > 0$, which of these three options are available depends on what developer $t-1$ chose. We enumerate these different situations in Table 1.

what developer $t-1$ chose	what developer t can choose after developing software t
P	the game ended in period $t-1$ already
G	**G**
B	**P**, **G**, and **B**

Table 1: The basic model.

Let T be the period when the game ends. The game ends in period T iff (1) $\forall t < T$, developer t decided to develop software t and then went open sourse, and (2) developer T either (2a) decides not to develop software T (in which case $w_T = 0$), or (2b) decides to develop software T and then goes proprietary. If the game never ends, let $T = \infty$.

For most of this paper (except for Subsection 3.1), we assume exponential discounting. For any developer $t < T$, his payoff is

$$U_t := \sum_{s=t}^{T} \beta^{s-t} w_s - c_t = \sum_{s=t}^{T-1} \beta^{s-t} W + \beta^{T-t} w_T - c_t, \tag{1}$$

where $\beta \in (0, 1)$ is the common discount factor. In other words, we assume that developer t incurs the development cost c_t and forgoes his potential profit π_t because he is altruistic enough to care about (current and future) consumer surplus. This is not a strong assumption, as we can always re-label anything an open source developer cares about as "consumer surplus".

As for developer T, his utility is 0 if he decides not to develop software T, or is $\pi_T + w_T - c_T = \pi_T + w - c_T$ if he decides to develop software T and then goes proprietary.

We have thus defined an infinite-horizon observable-action game. The primitives of the game are $P(\cdot, \cdot)$, w, W, and β. Our solution concept is Markov perfect equilibrium, in which every developer t follows the same Markov strategy that depends only on (*i*) the open source license he is subject to, which in turn is chosen by developer $t-1$,[5,6] and (*ii*) the realizations (c_t, π_t) of his development cost and potential profit. More formally, let $O := \{\mathbf{G}, \mathbf{B}\}$ denote the set of open source licenses, and $\mathbf{Q}$ denotes the option of not developing the software and hence quitting the game. A Markov strategy is a measurable

[5]Recall that if developer $t-1$ did not choose any open source license, then either he did not develop software $t-1$, or he did but chose to go proprietary, in either case developer t would not have a chance to move.

[6]For developer 0, since all options are available for him, we may treat him as if he is subject to license **B**.

function $\sigma : (c, \pi, o) \mapsto \Delta(\{\mathbf{Q}, \mathbf{P}\} \cup \mathcal{O})$ that describes how any developer t randomizes over $\{\mathbf{Q}, \mathbf{P}\} \cup \mathcal{O}$ given his development cost c, his potential profit π, and the open source license $o \in \mathcal{O}$ that he is subject to, with the restriction that $\forall(c, \pi)$, $\sigma(c, \pi, \mathbf{G})(\{\mathbf{P}, \mathbf{B}\}) = 0$. A Markov perfect equilibrium (hereafter, an *equilibrium*) is a Markov strategy σ^* that is optimal for any developer t among all Markov strategies if all future developers $s > t$ follow the Markov strategy σ^*.[7]

The following example shows that each of the three options ($\mathbf{P}$, $\mathbf{G}$, and $\mathbf{B}$) can be strictly optimal for developer 0 in equilibrium. In other words, all these options can be rational choices under appropriate circumstances.

2.1 An Example

Consider the following example. Suppose $w = 1$, $W = 3$, $\beta = 1/2$, $c = 1$ or 7 with equal marginal probability, and $\pi = 1$ or 7 with equal marginal probability. Suppose the joint distribution of c and π is as depicted in Table 2, where z indexes the correlation between c and π (with $z = 1$ means perfect positive correlation, and $z = -1$ means perfect negative correlation).

$P(c, \pi)$	$\pi = 1$	$\pi = 7$
$c = 1$	$(1 + z)/4$	$(1 - z)/4$
$c = 7$	$(1 - z)/4$	$(1 + z)/4$

Table 2: Joint distribution of c and π.

Consider any developer t. If he develops software t and goes open source, his gross payoff (gross of development cost c_t) is at least $W = 3$ (which is achieved if no future software is developed), and is at most

$$W + \beta W + \beta^2 W + \cdots = W/(1 - \beta) = 3/(1 - 1/2) = 6$$

(which is achieved if all future developers develop their softwares and go open source).

[7] We did not specify explicitly what history developer t can observe, other than the obvious fact that he must be able to observe the open source license $o \in \mathcal{O}$ he is subject to, which in turn is chosen by developer $t-1$. Depending on whether developer t can contingent his behavior on a richer history, the set of strategies developer t can choose from is potentially bigger than the set of Markov strategies. However, since the only part of the history that is payoff-relevant to developer t is the open source license that he is subject to, any strategy that is optimal within the set of Markov strategies will also be optimal within any bigger set of strategies.

Therefore,

1. if $c_t = 1$, he should develop software t, he should go open source if $\pi_t = 1$ (because by going open source he can get at least $3 > 2 = \pi_t + w$), and he should go proprietary if $\pi_t = 7$ (provided he has such an option);

2. if $c_t = 7$, he should never go open source (because by going open source he can get at most 6), and he should not even develop software t if $\pi_t + w < 7$ (or, equivalently, if $\pi_t < 6$); and

3. if $\pi_t = 7$ and he has the option to go proprietary, he should develop software t and then go proprietary.

These observations imply that the equilibrium Markov strategy for a developer who is subject to license **G** (and hence can only choose **G**) and for a developer who is subject to license **B** (and hence has all three options (**P**, **G**, and **B**) available to him), respectively, must be the ones depicted in Tables 3 and 4.

$\sigma(c, \pi, \mathbf{G})$	$\pi = 1$	$\pi = 7$
$c = 1$	**G**	**G**
$c = 7$	**Q**	**Q**

Table 3: The equilibrium Markov strategy for a developer who is subject to **G**.

$\sigma(c, \pi, \mathbf{B})$	$\pi = 1$	$\pi = 7$
$c = 1$	$o \in O$	**P**
$c = 7$	**Q**	**P**

Table 4: The equilibrium Markov strategy for a developer who is subject to **B**.

Now consider the problem of a developer t who is subject to license **B** when $(c_t, \pi_t) = (1, 1)$. If he chooses **G**, all future developers' behavior will be described by Table 3, and hence his gross expected payoff (gross of development cost) will be

$$V_G = W + (\beta/2)W + (\beta/2)^2 W + \cdots = 3/(1 - 1/4) = 4.$$

If he chooses **B**, his payoff will depend on future developers' license decisions in the same situation; i.e., it will depend on $\sigma(1, 1, \mathbf{B})$. Suppose $\sigma(1, 1, \mathbf{B}) = \mathbf{B}$, then his gross expected payoff will be

$$V_{BB} = W + \beta\left[\left(\frac{1+z}{4}\right)V_{BB} + \left(\frac{1-z}{4} + \frac{1+z}{4}\right)w\right] = \frac{26}{7-z}.$$

Suppose, instead, $\sigma(1,1,\mathbf{B}) = \mathbf{G}$, then his gross expected payoff will be

$$V_{BG} = W + \beta\left[\left(\frac{1+z}{4}\right)V_{G} + \left(\frac{1-z}{4} + \frac{1+z}{4}\right)w\right] = \frac{15+2z}{4}.$$

An equilibrium with $\sigma(1,1,\mathbf{B}) = \mathbf{B}$ exists iff $V_{BB} \geq V_G$; i.e., iff $z \geq 1/2$. An equilibrium with $\sigma(1,1,\mathbf{B}) = \mathbf{G}$ exists iff $V_G \geq V_{BG}$; i.e., iff $z \leq 1/2$. We summarize these with the following proposition.

Proposition 1 *In the example in this subsection, a pure-strategy Markov perfect equilibrium always exists, and is generically unique. In a pure-strategy Markov perfect equilibrium, developer 0*

1. *does not develop software 0 if development cost is high and potential profit is low (i.e., when* $(c_0, \pi_0) = (7, 1)$*);*

2. *develops software 0 and goes proprietary whenever potential profit is high (i.e., whenever* $\pi_0 = 7$*);*

3. *develops software 0 and goes open source when both development cost and potential profit are low (i.e., when* $(c_0, \pi_0) = (1, 1)$*); he goes open source with the BSD license* **B** *if future development costs and potential profits are sufficiently positively correlated (i.e., if* $z \geq 1/2$*), and with the GPL license* **G** *otherwise.*

Intuitively, developer 0 would like to see his idea further developed. However, development incurs development costs. In the unfortunate event that the next developer (i.e., developer 1) finds his development cost high, he will be discouraged from further developing developer 0's idea, especially if he is also prohibited from going proprietary and making a profit out of his effort. If development cost and potential profit are positively correlated such that a higher potential profit typically accompanies a higher development cost, then the BSD license, by allowing developer 1 to go proprietary, will help encourage developer 1 to further develop developer 0's idea even in this unfortunate event. The BSD license hence can be the optimal choice for a developer going open source if development cost and potential profit are positively correlated, while the GPL license can be the optimal choice in the case of negative correlation.

3 A Simple Extension

The basic model in Section 2 is so flexible that adding new open source licenses is easy. For example, consider the following two new open source licenses:[8]

R: *going open source with the recursive-BSD license*
Developer t goes open source. If developer $t+1$ chooses to develop software $t+1$, he can either go propriety (i.e., choosing **P**), or go open source with the recursive-BSD license (i.e., choosing **R**).[9]

1: *going open source with the 1-chance-only license*
Developer t goes open source. If developer $t+1$ chooses to develop software $t+1$, he can either go propriety (i.e., choosing **P**), or go open source with the GPL license (i.e., choosing **G**).

To better appreciate how these two licenses differ from the existing ones, we can extend the earlier table as follows:

what developer $t-1$ chose	what developer t can choose after developing software t
P	the game ended in period $t-1$ already
G	**G**
B	**P**, **G**, **B**, **R**, and **1**
R	**P** and **R**
1	**P** and **G**

In particular, license **R** in effect guarantees that each future developer, if he ever chooses to go open source, must continue to give the next developer the permission to go proprietary. In this sense, **R** is even more permissive than the BSD license (**B**), as it preserves the permission to go proprietary for all future developers. Of course, one

[8]In the following, it goes without saying that, as in Section 2, as long as developer t develops software t and goes open source, the profit generated will be $\pi_t = 0$, and the consumer surplus generated will $w_t = W$.

[9]Note that license **R**, like license **G**, is self-referential.

can also argue that **R** is more restrictive than **B**, as it forbids the next developer from forbidding the next-next developer from going proprietary. This is a perfect example of why it can be difficult to summarize an open source license by a single parameter called "permissiveness" as in Lerner and Tirole (2005b).

In comparison, license **1** is unambiguously more restrictive than **R**. While it, like **R**, gives the next developer the permission to go proprietary, it forbid the next developer from giving the same permission to the next-next developer. In other words, license **1** gives future developers one and only one chance to go proprietary, namely in the next period and in the next period only.

It can be shown that, with exponential discounting (which is what we assumed so far) and an atomless joint distribution $P(\cdot\,,\,\cdot)$, neither **R** nor **1** will ever be strictly optimal for developer 0 in equilibrium. This is a special case of Theorem 6 that will appear later in Section 6. However, **1** can be a rational choice if developers have hyperbolic discounting, as illustrated by the next numerical example.

3.1 An Example with Hyperbolic Discounting

It has become a common practice in behavioral economics to *approximate* hyperbolic discounting using an (α, β)-formulation. Here, we shall instead adopt an $(\alpha_1, \alpha_2, \beta)$-formulation. Specifically, given any infinite sequence of dated payoffs, $\{u_t, u_{t+1}, u_{t+2}, \ldots\}$, we assume that developer t's present discounted value is

$$U_t = u_t + \alpha_1 \left(u_{t+1} + \alpha_2 \left(u_{t+2} + \beta u_{t+3} + \beta^2 u_{t+4} + \cdots \right) \right),$$

where $\alpha_1 \leq \alpha_2 \leq \beta$.[10]

Consider the following example with hyperbolic discounting. Suppose $w = 15$, $W = 20$, $\alpha_1 = 1/4$, $\alpha_2 = 2/4$, and $\beta = 3/4$. Suppose the joint distribution of c and π is as depicted in Table 5.

As in the example in Subsection 2.1, we can calculate a lower and an upper bounds for the gross expected payoff of going open source, which are $W = 20$ and

$$W + \alpha_1 \left(W + \alpha_2 \left(W + \beta W + \beta^2 W + \cdots \right) \right) = 35,$$

[10]As will be explained later, α_1 actually does not play any role, and hence the restriction $\alpha_1 \leq \alpha_2$ is redundant. We maintain this restriction only to stay in sync with the hyperbolic-discounting literature.

$P(c,\pi)$	$\pi = 0$	$\pi = 20$	$\pi = 35$
$c = 10$	$\epsilon \approx 0$	0	0
$c = 20$	0	$(1-\epsilon)/2$	0
$c = 40$	0	0	$(1-\epsilon)/2$

Table 5: Joint distribution of c and π.

respectively. We can obtain a number of observations from these two bounds.

First, from the lower bound we can infer that $\sigma(10, 0, \cdot) \in \mathcal{O}$ (because the gross expected payoff of going open source is at least 20). This in turn implies that the gross expected payoff of going open source is strictly higher than 20, because the probability that the next developer developing his software is strictly positive.

Second, from the upper bound we can infer that $\sigma(40, 35, \mathbf{G}) = \mathbf{Q}$ and $\sigma(40, 35, \mathbf{B}) = \sigma(40, 35, \mathbf{R}) = \sigma(40, 35, \mathbf{1}) = \mathbf{P}$ (because by going open source the gross expected payoff is at most $35 < 50 = \pi + w$). It means the probability of any developer t going open source is at most $1 - P(40, 35) = (1 + \epsilon)/2 \approx 1/2$. This in turn implies a tighter upper bound for the payoff of going open source, which is approximately

$$W + \alpha_1 \left(\frac{W+w}{2} + \alpha_2 \left(\frac{W+w}{2} + \beta \frac{W+w}{2} + \beta^2 \frac{W+w}{2} + \cdots \right) \right) = 33\frac{1}{8} < 35.$$

From this tighter upper bound we can infer that $\sigma(20, 20, \mathbf{B}) = \sigma(20, 20, \mathbf{R}) = \sigma(20, 20, \mathbf{1}) = \mathbf{P}$ (because by going open source the gross expected payoff is strictly less than $35 = \pi + w$).

Finally, since the gross expected payoff of going open source is strictly higher than 20, we have $\sigma(20, 20, \mathbf{G}) = \mathbf{G}$.

These observations imply that the equilibrium Markov strategy must be the ones depicted in Tables 6 and 7.

$\sigma(c,\pi,\mathbf{G})$	$\pi = 0$	$\pi = 20$	$\pi = 35$
$c = 10$	**G**	–	–
$c = 20$	–	**G**	–
$c = 40$	–	–	**Q**

Table 6: The equilibrium Markov strategy for a developer who is subject to **G**.

Now consider the problem of a developer t who is subject to license **B** when $(c_t, \pi_t) = (10, 0)$. If he chooses **G**, all future developers' behavior will be described by Table 6, and

$\sigma(c,\pi,\mathbf{B}/\mathbf{R}/\mathbf{1})$	$\pi = 0$	$\pi = 20$	$\pi = 35$
$c = 10$	$o \in \bar{O}$	$-$	$-$
$c = 20$	$-$	$\mathbf{P}$	$-$
$c = 40$	$-$	$-$	$\mathbf{P}$

Table 7: The equilibrium Markov strategy for a developer who is subject to **B**, **R**, or **1**.

hence his gross expected payoff will be approximately

$$V_G = W + \frac{\alpha_1}{2}\left(W + \frac{\alpha_2}{2}\left(W + \frac{\beta}{2}W + \left(\frac{\beta}{2}\right)^2 W + \cdots\right)\right) = 23\frac{1}{2}.$$

If he chooses either **B**, **R**, or **1**, with probability close to 1 the next developer will go proprietary, and hence his gross expected payoff is approximately

$$V_{B/R/1} = W + \alpha_1 w = 23\frac{3}{4} > 23\frac{1}{2} = V_G.$$

Therefore, $\sigma(10, 0, \mathbf{B}) \in \{\mathbf{B}, \mathbf{R}, \mathbf{1}\}$. To compare these three contenders, note that they differ only in developer t's continuation payoff conditional on the event that $(c_{t+1}, \pi_{t+1}) = (10, 0)$, and hence it suffices to ask what developer t may want to allow developer $t+1$ to choose in that event (hereafter event E).

Conditional on event E, if developer $t+1$ chooses **G**, the present value of developer t's continuation payoff will be approximately

$$V_{E:G} = \alpha_1\left(W + \frac{\alpha_2}{2}\left(W + \frac{\beta}{2}W + \left(\frac{\beta}{2}\right)^2 W + \cdots\right)\right) = \alpha_1 \times 28.$$

Conditional on event E, if developer $t+1$ chooses either **B**, **R**, or **1** the present value of developer t's continuation payoff will be approximately

$$V_{E:B/R/1} = \alpha_1\left(W + \alpha_2 w\right) = \alpha_1 \times 27\frac{1}{2} < \alpha \times 28 = V_{E:G}.$$

Therefore, when choosing among **B**, **R**, and **1**, developer t would like to choose the one that only allows developer $t+1$ to choose **G**. This goal can be achieved by choosing **1**. We summarize these with the following proposition.

Proposition 2 *In the example in this subsection, a pure-strategy Markov perfect equilibrium* *always exists and is unique. In a pure-strategy Markov perfect equilibrium, developer 0*

1. *develops software 0 and goes proprietary when* $(c_0, \pi_0) = (20, 20)$ *or* $(40, 35)$*; and*
2. *develops software 0 and goes open source with the 1-chance-only license* **1** *when* $(c_0, \pi_0) = (10, 0)$.

What is going on? Intuitively, if developer $t+2$ is allowed to go proprietary, he is more likely to develop software $t+2$, but the stream of consumer surplus will also more likely terminate in period $t+2$. To decide whether to allow developer $t+2$ to go proprietary, one is hence trading off consumer surplus in period $t+2$ versus consumer surplus in periods $s \geq t+3$, which depends on that decision maker's discount rate between periods $t+2$ and $t+3$. With hyperbolic discounting, this discount rate is higher for developer $t+1$ than for developer t, meaning that developer $t+1$ is more tempted, compared to developer t, to allow developer $t+2$ to go proprietary. Developer t hence may want to choose an open source license that forbids developer $t+1$ from allowing developer $t+2$ to go proprietary. While both **G** and **1** come with this same forbiddance, **1** has the extra benefit over **G** in allowing developer $t+1$ to go proprietary. This is a benefit for developer t because, just like developer $t+1$, developer t discount the future hyperbolically and hence is equally tempted to allow the next developer to go proprietary.

The above intuition also explains why we need an $(\alpha_1, \alpha_2, \beta)$-approximation of hyperbolic discounting, instead of the more traditional (α, β) one. The key misalignment between developers t's and $t+1$'s interests lies in their discount rates between periods $t+2$ and $t+3$; i.e., between α_2 and β. The assumption that $\alpha_2 \leq \beta$ implies that developer t is more patient than developer $t+1$, and hence will like to limit the latter's ability to reap an earlier reward. In contrast, α_1 does not play any role in this story, and the assumption that $\alpha_1 \leq \alpha_2$ can be relaxed without affecting our results.

4 The Universal Space of Open Source Licenses

The two new licenses introduced in Section 3 are not the only possible new open source licenses. Indeed, a little thought would suggest that infinitely many can be generated in a similar manner. It is hence important to have a sense of how the space of all possible open source licenses looks like, and then impose some intuitive axioms to weed out the less interesting ones.

We can define a general space of open source licenses in a manner similar to that in Sections 2 and 3. For any set X, let $\mathcal{P}(X)$ be the set of all *nonempty* subsets of X.[11]

Definition 1 *Let $\mathcal{O}$ be an arbitrary set, with each element $o \in \mathcal{O}$ corresponding to an open source license. Let $g : \mathcal{O} \to \mathcal{P}(\{\mathbf{P}\} \cup \mathcal{O})$ be a nonempty correspondence such that, for any $o \in \mathcal{O}$, $g(o) \cap \mathcal{O} \neq \emptyset$. Then, $\mathcal{S} = (\mathcal{O}, g)$ is a space of open source licenses.*

Intuitively, $g(o)$ specifies the (nonempty) set of options available to a developer who is subject to open source license o. We assume that going open source (with *some* open source license) must always be an option, which translates into the requirement that $g(o) \cap \mathcal{O} \neq \emptyset$. This assumption is natural, as it seems impossible to design any open source license that forces the next developer to go proprietary.[12]

We say that a space of open source licenses $\mathcal{S} = (\mathcal{O}, g)$ is *finite* if $g(o)$ is finite for every $o \in \mathcal{O}$. Note that the set of open source licenses $\mathcal{O}$ needs not be finite even when the space $\mathcal{S} = (\mathcal{O}, g)$ is finite, as finiteness refers only to the number of options allowed by each open source license $o \in \mathcal{O}$.

A space of open source licenses may contain duplicates of otherwise identical open source licenses. For example, consider $\mathcal{O} = \{\mathbf{G_1}, \mathbf{G_2}\}$, with $g(\mathbf{G_1}) = \{\mathbf{G_2}\}$ and $g(\mathbf{G_2}) = \{\mathbf{G_2}\}$. Then the differences between $\mathbf{G_1}$ and $\mathbf{G_2}$ are superfluous, and one may for all purposes regard both as being identical to the GPL license.

Formally, for any space of open source license $\mathcal{S} = (\mathcal{O}, g)$, let $\sim$ be an equivalence relation on $\mathcal{O}$.[13] For any $o \in \mathcal{O}$, let $[o]$ denote the equivalence class of o. We shall abuse notation by sometimes writing $\mathbf{P}$ as $[\mathbf{P}]$ as well. Let $\mathcal{O}^{\sim}$ denote the corresponding quotient set (i.e., the set of equivalence classes of elements in $\mathcal{O}$). Let $\mu : \mathcal{O} \cup \{\mathbf{P}\} \to \mathcal{O}^{\sim} \cup \{\mathbf{P}\}$ be the canonical mapping such that, $\forall x \in \mathcal{O} \cup \{\mathbf{P}\}$, $\mu(x) = [x]$. For any $o \in \mathcal{O}$, let $(\mu \circ g)(o) =$

[11]In mathematics, the notation $\mathcal{P}(X)$ typically stands for the power set of X, which is the set of all subsets of X, including the empty set. Here, it will ease our notation if we re-define $\mathcal{P}(X)$ as the set of all *nonempty* subsets of X instead.

[12]It seems impossible to stop the next developer from giving up his copyright and putting his source code in the public domain, which in our model is equivalent to going open source with the BSD license. It should however be pointed out that, in reality, there are subtle differences between "putting the source code in the public domain" and "going open source with the BSD licenses", and these subtle differences are not captured by our model.

[13]A binary relation $\sim$ is an equivalence relation if it is reflexive, symmetric, and transitive.

$\{[x] \in \mathcal{O}^{\sim} \cup \{\mathbf{P}\} : x \in g(o)\}$, which is nonempty because $g(o)$ is nonempty. We say that g and $\sim$ are *compatible* if $(\mu \circ g)(o) = (\mu \circ g)(o')$ whenever $o \sim o'$. If g is compatible with $\sim$, we can define $g^{\sim} : \mathcal{O}^{\sim} \to \mathcal{P}(\mathcal{O}^{\sim} \cup \{\mathbf{P}\})$ such that $g^{\sim}([o]) = (\mu \circ g)(o)$. Then $\mathcal{S}^{\sim} = (\mathcal{O}^{\sim}, g^{\sim})$ is a space of open source licenses. We say that $\sim$ is nontrivial if there exist $o, o' \in \mathcal{O}$ such that $o \sim o'$ but $o \neq o'$. If there exists a nontrivial equivalence relation $\sim$ compatible with g, we say that $\mathcal{S} = (\mathcal{O}, g)$ is *reducible to* $S^{\sim} = (\mathcal{O}^{\sim}, g^{\sim})$ (or simply *reducible*); otherwise $\mathcal{S} = (\mathcal{O}, g)$ is *irreducible*.

Consider our earlier example where $\mathcal{O} = \{\mathbf{G_1}, \mathbf{G_2}\}$, $g(\mathbf{G_1}) = \{\mathbf{G_2}\}$, and $g(\mathbf{G_2}) = \{\mathbf{G_2}\}$. The only nontrivial equivalence relation is such that $\mathbf{G_1} \sim \mathbf{G_2}$. This equivalence relation is compatible with g, because $(\mu \circ g)(\mathbf{G_1}) = \{[\mathbf{G_1}]\} = (\mu \circ g)(\mathbf{G_2})$. Therefore, $\mathcal{S} = (\mathcal{O}, g)$ is reducible to $S^{\sim} = (\mathcal{O}^{\sim}, g^{\sim})$, where $\mathcal{O}^{\sim} = \{[\mathbf{G_1}]\}$ and $g^{\sim}([\mathbf{G_1}]) = \{[\mathbf{G_1}]\}$.

Consider the example in Section 2 where $\mathcal{O} = \{\mathbf{G}, \mathbf{B}\}$, $g(\mathbf{G}) = \{\mathbf{G}\}$, and $g(\mathbf{B}) = \{\mathbf{P}, \mathbf{G}, \mathbf{B}\}$. The only nontrivial equivalence relation is such that $\mathbf{G} \sim \mathbf{B}$. This equivalence relation is not compatible with g, however, because $(\mu \circ g)(\mathbf{G}) = \{[\mathbf{G}]\} \neq \{\mathbf{P}, [\mathbf{G}]\} = (\mu \circ g)(\mathbf{B})$. Therefore, $\mathcal{S} = (\mathcal{O}, g)$ is irreducible.

To construct the universal space of open source licenses, $\mathcal{S}^{U} = (\mathcal{O}^{U}, g^{U})$, we first recursively construct a sequence of nonempty sets $(\Theta_0, \Omega_0, \Theta_1, \Omega_1, \ldots)$ as follows. Let $\Theta_0 = \Omega_0 = \{0, 1\}$. For $n \geq 1$, let

$$\begin{aligned}
\Theta_n &= \mathcal{P}(\Omega_{n-1}) \\
\Omega_n &= \Omega_{n-1} \times \Theta_n \\
&= \Omega_{n-2} \times \Theta_{n-1} \times \Theta_n \\
&= \cdots \\
&= \Omega_0 \times \Theta_1 \times \cdots \times \Theta_n \\
&= \Theta_0 \times \Theta_1 \times \cdots \times \Theta_n.
\end{aligned}$$

A sequence $(\theta_0, \theta_1, \ldots)$ is called a *restriction hierarchy* if $\theta_n \in \Theta_n$ for any $n \geq 0$. Let Ω_∞ be the set of all restriction hierarchies. Intuitively, an open source licenses can be represented by a restriction hierarchy $(\theta_0, \theta_1, \theta_2, \ldots)$ with

- θ_0 specifying whether the next developer—say, developer t—is allowed to go proprietary (where $\theta_0 = 1$ means "yes" and $\theta_0 = 0$ means "no"),

- θ_1 specifying what restrictions developer t is allowed to impose on developer $t+1$ regarding the option of going proprietary,

- θ_2 specifying what restrictions developer t is allowed to impose on developer $t+1$ regarding *both* (*i*) the option of going proprietary *and* (*ii*) what restrictions developer $t+1$ can impose on developer $t+2$ regarding the option of going proprietary,

- ..., etc.

To illustrate how an open source license can be represented by a restriction hierarchy, let's construct the restriction hierarchy $(\theta_0^G, \theta_1^G, \ldots)$ that represents the GPL license **G**.

To determine θ_0^G, we ask whether the GPL license allows the next developer—say, developer t—to go proprietary. No, it does not. Therefore, $\theta_0^G = 0$.

To determine θ_1^G, we ask what options developer t has regarding whether to allow developer $t+1$ to go proprietary. The GPL license gives developer t only a single option—meaning that θ_1^G must be a singleton. Moreover, that single option is to use the GPL license as well, whose first restriction has already been encoded in θ_0^G.[14] Therefore, θ_1^G must be the singleton $\left\{\theta_0^G\right\} = \{0\}$.

To determine θ_2^G, we ask what options developer t has regarding (*i*) whether to allow developer $t+1$ to go proprietary, and (*ii*) whether to allow developer $t+1$ to allow developer $t+2$ to go proprietary. The GPL license gives developer t only a single option—meaning that θ_2^G must be a singleton. Moreover, that single option is to use the GPL license as well, whose first two restrictions have already been encoded in θ_0^G and θ_1^G.[15] Therefore, θ_2^G must be the singleton $\left\{\left(\theta_0^G, \theta_1^G\right)\right\} = \{(0, \{0\})\}$.

More generally, for any $n \geq 1$, θ_n^G must be a singleton, and must be the singleton $\left\{\left(\theta_0^G, \theta_1^G, \ldots, \theta_{n-1}^G\right)\right\}$.

The GPL license can hence be represented by the restriction hierarchy $\left(\theta_0^G, \theta_1^G, \ldots\right)$,

[14]The GPL license carries infinitely many restrictions. Here, we are concerned about only its first restriction, namely whether it allows the next developer to go proprietary.

[15]The GPL license carries infinitely many restrictions. Here, we are concerned about only its first two restrictions, namely (*i*) whether it allows the next developer to go proprietary, and (*ii*) whether it allows the next developer to allow the next developer to go proprietary.

where:

$$\begin{aligned}
\theta_0^G &= 0, \\
\theta_1^G &= \{0\}, \\
\theta_2^G &= \{(0, \{0\})\}, \\
\theta_3^G &= \{(0, \{0\}, \{(0, \{0\})\})\}, \\
\theta_4^G &= \{(0, \{0\}, \{(0, \{0\})\}, \{(0, \{0\}, \{(0, \{0\})\})\})\}, \\
&\vdots
\end{aligned} \tag{2}$$

Likewise we can construct the restriction hierarchy $\left(\theta_0^R, \theta_1^R, \ldots\right)$ that represents the recursive-BSD license **R**. The recursive-BSD license allows the next developer to go proprietary, and hence $\theta_0^R = 1$. If the next developer chooses to go open source, the recursive-BSD license gives him only a single option—meaning that for any $n \geq 1$, θ_n^R must be a singleton. Moreover, that single option is to use the recursive-BSD license as well, whose first n restrictions have already been encoded in θ_0^R, θ_1^R, . . . , and θ_{n-1}^R. Therefore, θ_n^R must be the singleton $\left\{\left(\theta_0^R, \theta_1^R, \ldots, \theta_{n-1}^R\right)\right\}$. The recursive-BSD license can hence be represented by the restriction hierarchy $\left(\theta_0^R, \theta_1^R, \ldots\right)$, where:

$$\begin{aligned}
\theta_0^R &= 1, \\
\theta_1^R &= \{1\}, \\
\theta_2^R &= \{(1, \{1\})\}, \\
\theta_3^R &= \{(1, \{1\}, \{(1, \{1\})\})\}, \\
\theta_4^R &= \{(1, \{1\}, \{(1, \{1\})\}, \{(1, \{1\}, \{(1, \{1\})\})\})\}, \\
&\vdots
\end{aligned} \tag{3}$$

As our final example, let's also construct the restriction hierarchy $\left(\theta_0^1, \theta_1^1, \ldots\right)$ that represents the 1-chance-only license **1**. The 1-chance-only license allows the next developer to go proprietary, and hence $\theta_0^1 = 1$. If the next developer chooses to go open source, the 1-chance-only license gives him only a single option—meaning that for any $n \geq 1$, θ_n^1 must be a singleton. Moreover, that single option is to use the GPL license, whose first n restrictions have already been encoded in θ_0^G, θ_1^G, . . . , and θ_{n-1}^G. Therefore, θ_n^1 must be the singleton $\left\{\left(\theta_0^G, \theta_1^G, \ldots, \theta_{n-1}^G\right)\right\}$. The 1-chance-only license can hence be represented by the restriction hierarchy $\left(\theta_0^1, \theta_1^1, \ldots\right)$, where:

$$\begin{aligned}
\theta_0^1 &= 1, \\
\theta_1^1 &= \{0\}, \\
\theta_2^1 &= \{(0, \{0\})\}, \\
\theta_3^1 &= \{(0, \{0\}, \{(0, \{0\})\})\}, \\
\theta_4^1 &= \{(0, \{0\}, \{(0, \{0\})\}, \{(0, \{0\}, \{(0, \{0\})\})\})\}, \\
&\vdots
\end{aligned} \tag{4}$$

While an open source license can be represented by a restriction hierarchy, however, not every restriction hierarchy can be the representation of an open source license. In order for a restriction hierarchy $(\theta_0, \theta_1, \theta_2, \ldots)$ to represent an open source license, it needs to satisfy a consistency requirement. To motivate this consistency requirement, note that both θ_1 and θ_2 contain information on what restrictions developer t is allowed to impose on developer $t+1$ regarding the option of going proprietary, and these two pieces of information must agree with each other. More generally, for any $n \geq 1$, θ_n and θ_{n+1} must be consistent in the sense that $\text{Proj}_{\Theta_n} \theta_{n+1} = \theta_n$.[16]

Definition 2 *A restriction hierarchy $(\theta_0, \theta_1, \ldots)$ is consistent if* $\text{Proj}_{\Omega_{n-1}} \theta_{n+1} = \theta_n$ *for any $n \geq 1$.*

The reader can readily check that the restriction hierarchies $\left(\theta_0^G, \theta_1^G, \ldots\right)$, $\left(\theta_0^R, \theta_1^R, \ldots\right)$, and $\left(\theta_0^1, \theta_1^1, \ldots\right)$ are all consistent, because for any $n \geq 1$,

$$\begin{aligned}
\text{Proj}_{\Omega_{n-1}} \theta_{n+1}^G &= \text{Proj}_{\Theta_0 \times \cdots \times \Theta_{n-1}} \left\{\left(\theta_0^G, \ldots, \theta_n^G\right)\right\} = \left\{\left(\theta_0^G, \ldots, \theta_{n-1}^G\right)\right\} = \theta_n^G, \\
\text{Proj}_{\Omega_{n-1}} \theta_{n+1}^R &= \text{Proj}_{\Theta_0 \times \cdots \times \Theta_{n-1}} \left\{\left(\theta_0^R, \ldots, \theta_n^R\right)\right\} = \left\{\left(\theta_0^R, \ldots, \theta_{n-1}^R\right)\right\} = \theta_n^R, \quad \text{and} \\
\text{Proj}_{\Omega_{n-1}} \theta_{n+1}^1 &= \text{Proj}_{\Theta_0 \times \cdots \times \Theta_{n-1}} \left\{\left(\theta_0^G, \ldots, \theta_n^G\right)\right\} = \left\{\left(\theta_0^G, \ldots, \theta_{n-1}^G\right)\right\} = \theta_n^1.
\end{aligned}$$

Let $\mathcal{O}_1$ be the set of all consistent restriction hierarchies.

For any consistent restriction hierarchy $(\theta_0, \theta_1, \ldots) \in \mathcal{O}_1$, to the extent that it represents an open source license (not yet, as we shall see shortly), we should be able to extract

[16]$\text{Proj}_{\Omega_{n-1}} \theta_{n+1}$ denotes the projection of $\theta_{n+1} \in \Theta_{n+1} = \mathcal{P}(\Omega_n) = \mathcal{P}(\Omega_{n-1} \times \Theta_n)$ into Ω_{n-1}, resulting in a subset of Ω_{n-1}. Consistency requires that this subset is the same as $\theta_n \in \mathcal{P}(\Omega_{n-1})$.

from θ_0 whether it allows the next developer to go proprietary, and from $(\theta_1, \theta_2, \ldots)$ what open source licenses it allows the next developer to use. The first task is easy ($\theta_0 = 1$ means "yes", and $\theta_0 = 0$ means "no"), and the second task can be done by defining a correspondence $\Gamma : \mathcal{O}_1 \rightrightarrows \Omega_\infty$ as follows: for any $(\theta_0, \theta_1, \ldots) \in \mathcal{O}_1$ and $(\theta'_0, \theta'_1, \ldots) \in \Omega_\infty$, $(\theta'_0, \theta'_1, \ldots) \in \Gamma(\theta_0, \theta_1, \ldots)$ iff $\forall n \geq 0$,[17]

$$\left(\theta'_0, \ldots, \theta'_n\right) \in \theta_{n+1} \in \Theta_{n+1} = \mathcal{P}(\Omega_n) = \mathcal{P}\left(\Theta_0 \times \Theta_1 \times \cdots \times \Theta_n\right). \tag{5}$$

As an illustration, if we apply the correspondence Γ to consistent restriction hierarchies $\left(\theta^G_0, \theta^G_1, \ldots\right)$, $\left(\theta^R_0, \theta^R_1, \ldots\right)$, and $\left(\theta^1_0, \theta^1_1, \ldots\right)$, we will obtain

$$\begin{aligned} \Gamma\left(\theta^G_0, \theta^G_1, \ldots\right) &= \left\{\left(\theta^G_0, \theta^G_1, \ldots\right)\right\}, \\ \Gamma\left(\theta^R_0, \theta^R_1, \ldots\right) &= \left\{\left(\theta^R_0, \theta^R_1, \ldots\right)\right\}, \quad \text{and} \\ \Gamma\left(\theta^1_0, \theta^1_1, \ldots\right) &= \left\{\left(\theta^G_0, \theta^G_1, \ldots\right)\right\}, \end{aligned}$$

confirming the fact that a developer who is subject to open source license **G** (respectively, **R** and **1**), if going open source, is only allowed to do so with open source license **G** (respectively, **R** and **G**).

Lemma 1 *For any consistent restriction hierarchy* $(\theta_0, \theta_1, \ldots) \in \mathcal{O}_1$*, the subset* $\Gamma(\theta_0, \theta_1, \ldots) \subset \Omega_\infty$ *is nonempty.*

PROOF: We first show that there exists $\theta'_0 \in \theta_1$. This follows from $\theta_1 \in \Theta_1 = \mathcal{P}(\Omega_0)$ and $\mathcal{P}(\Omega_0)$ does not contain the empty set.

For any $n \geq 1$, assume that there exists $\left(\theta'_0, \ldots, \theta'_{n-1}\right) \in \theta_n$. Since $(\theta_0, \theta_1, \ldots)$ is consistent, we have $\text{Proj}_{\Omega_{n-1}} \theta_{n+1} = \theta_n$, and hence $\left(\theta'_0, \ldots, \theta'_{n-1}\right) \in \text{Proj}_{\Omega_{n-1}} \theta_{n+1}$ as well. Therefore, there exists θ'_n such that $\left(\theta'_0, \ldots, \theta'_{n-1}, \theta'_n\right) \in \theta_{n+1}$. By induction, there hence exists a restriction hierarchy $(\theta'_0, \theta'_1, \ldots) \in \Omega_\infty$ such that, $\forall n \geq 0$, $(\theta'_0, \ldots, \theta'_n) \in \theta_{n+1}$, implying that $\Gamma(\theta_0, \theta_1, \ldots)$ is nonempty. ■

That the restriction hierarchy $(\theta_0, \theta_1, \ldots)$ is consistent, however, does not guarantee that those restriction hierarchies in $\Gamma(\theta_0, \theta_1, \ldots)$ are also consistent. Therefore, we shall

[17]We abuse notation by writing $\Gamma((\theta_0, \theta_1, \ldots))$ as $\Gamma(\theta_0, \theta_1, \ldots)$, although Γ has only one argument, namely the vector $(\theta_0, \theta_1, \ldots)$.

define

$$\mathcal{O}_2 = \{(\theta_0, \theta_1, \ldots) \in \mathcal{O}_1 \,:\, \Gamma(\theta_0, \theta_1, \ldots) \subset \mathcal{O}_1\},$$

and retain only restriction hierarchies in $\mathcal{O}_2$.

That every restriction hierarchy $(\theta'_0, \theta'_1, \ldots)$ in $\Gamma(\theta_0, \theta_1, \ldots)$ is consistent, however, does not guarantee that those restriction hierarchies in $\Gamma(\theta'_0, \theta'_1, \ldots)$ are also consistent. Therefore, we should not stop here, and should continue and recursively define, for any $k \geq 2$,

$$\mathcal{O}_k = \{(\theta_0, \theta_1, \ldots) \in \mathcal{O}_{k-1} \,:\, \Gamma(\theta_0, \theta_1, \ldots) \subset \mathcal{O}_{k-1}\}.$$

We shall now define

$$\mathcal{O}^U = \bigcap_{k=1}^{\infty} \mathcal{O}_k,$$

which is the set of every restriction hierarchy that represents an open source license.[18]

The reader can readily check that all of $\left(\theta_0^G, \theta_1^G, \ldots\right)$, $\left(\theta_0^R, \theta_1^R, \ldots\right)$, and $\left(\theta_0^1, \theta_1^1, \ldots\right)$ belong to $\mathcal{O}^U$. Indeed, we have already seen that they are all consistent and hence belong to $\mathcal{O}_1$. For any $k \geq 1$, assume that they have already been shown to belong to $\mathcal{O}_k$. Then we have

$$\Gamma\left(\theta_0^G, \theta_1^G, \ldots\right) = \left\{\left(\theta_0^G, \theta_1^G, \ldots\right)\right\} \subset \mathcal{O}_k$$

by the inductive assumption, and hence $\left(\theta_0^G, \theta_1^G, \ldots\right)$ belongs to $\mathcal{O}_{k+1}$ as well; and similarly for $\left(\theta_0^R, \theta_1^R, \ldots\right)$ and $\left(\theta_0^1, \theta_1^1, \ldots\right)$. Therefore, by induction, all of them belong to $\mathcal{O}_k$ for any $k \geq 1$, and hence belong to $\mathcal{O}^U$. This also shows that $\mathcal{O}^U$ is nonempty.

Define $g^U : \mathcal{O}^U \to \mathcal{P}(\{\mathbf{P}\} \cup \mathcal{O}^U)$ such that, for any consistent restriction hierarchy $(\theta_0, \theta_1, \ldots) \in \mathcal{O}^U$,

$$g^U(\theta_0, \theta_1, \ldots) = \begin{cases} \Gamma(\theta_0, \theta_1, \ldots) & \text{if } \theta_0 = 0 \\ \Gamma(\theta_0, \theta_1, \ldots) \cup \{\mathbf{P}\} & \text{if } \theta_0 = 1 \end{cases}. \tag{6}$$

We shall call the pair $\mathcal{S}^U = (\mathcal{O}^U, g^U)$ the *universal space of open source licenses*. The following two theorems justify why this terminology is appropriate. The first theorem states that $\mathcal{S}^U$ is in itself a space of open source licenses. The second theorem states that any finite irreducible space of open source licenses is a sub-space of $\mathcal{S}^U$. These two theorems

[18]Readers familiar with the classical construction of the universal type space will recognize that this step is akin to the imposition of common knowledge of coherency.

(except for the "1-to-1" part of Theorem 2) can also be proved as corollaries of Mariotti, Meier, and Piccione's (2005) Proposition 3, by recognizing any finite space of open source licenses can be made into a compact continuous possibility structure.[19] Below we provide an elementary proof without topology for each of these two theorems.

Theorem 1 *The universal space of open source licenses $\mathcal{S}^U$ is in itself a space of open source licenses.*

Proof: We have already seen that $\mathcal{O}^U$ is nonempty, as it contains, for example, $\left(\theta_0^G, \theta_1^G, \ldots\right)$, $\left(\theta_0^R, \theta_1^R, \ldots\right)$, and $\left(\theta_0^1, \theta_1^1, \ldots\right)$. For any $o \in \mathcal{O}^U$, by Lemma 1, $\Gamma(o)$ is also nonempty. For any $k \geq 1$, $\Gamma(o) \subseteq \mathcal{O}_k$ because $o \in \mathcal{O}^U \subseteq \mathcal{O}_{k+1}$. Therefore, $g^U(o) \cap \mathcal{O}^U = \Gamma(o) \subseteq \cap_k \mathcal{O}_k = \mathcal{O}^U$, and hence $\mathcal{S} = (\mathcal{O}^U, g^U)$ is in itself a space of open source licenses. ■

Theorem 2 *Any finite irreducible space of open source licenses $\mathcal{S} = (\mathcal{O}, g)$ is a sub-space of the universal space of open source licenses in the sense that there exists a 1-to-1 mapping $f : \mathcal{O} \to \mathcal{O}^U$, called the canonical representation, such that for any $o \in \mathcal{O}$,*

$$\begin{aligned} \mathbf{P} \in g(o) &\Longleftrightarrow \mathbf{P} \in g^U(f(o)) \\ \textit{and} \quad o' \in g(o) &\Longleftrightarrow f(o') \in g^U(f(o)). \end{aligned} \tag{7}$$

We prove Theorem 2 in three steps. We first explicitly construct a mapping f from $\mathcal{O}$ into the set of all restriction hierarchies Ω_∞ that satisfies (7). We call this mapping the *canonical representation* of open source licenses. We then prove that the range of f actually lies inside $\mathcal{O}^U$ (Lemma 2). Finally, we prove that, if $\mathcal{S} = (\mathcal{O}, g)$ is finite irreducible, this mapping will be 1-to-1 (Lemma 3).

We construct the canonical representation f by recursively defining a sequence of mappings $(f_0, f_1, \ldots)$, with each f_n a mapping from $\mathcal{O}$ into Θ_n. We first define f_0 such that, for any $o \in \mathcal{O}$,

$$f_0(o) = \begin{cases} 0 & \text{if } \mathbf{P} \notin g(o) \\ 1 & \text{if } \mathbf{P} \in g(o). \end{cases}$$

[19]We thank Yi-Chun Chen for pointing this out to us.

Suppose we have already defined mappings $(f_0, \ldots, f_n)$, we then define f_{n+1} such that, for any $o \in \mathcal{O}$,

$$f_{n+1}(o) = \{(f_0(o'), \ldots, f_n(o')) \,:\, o' \in g(o) \cap \mathcal{O}\}.$$

We can now define the canonical representation $f : \mathcal{O} \to \Omega_\infty$ such that, for any $o \in \mathcal{O}$,

$$f(o) = (f_0(o), f_1(o), \ldots).$$

Lemma 2 *The range of the canonical representation f lies inside $\mathcal{O}^U$.*

Proof: We first prove that, for any $o \in \mathcal{O}$, $f(o)$ is a consistent restriction hierarchy. Let $f(o) = (\theta_0, \theta_1, \ldots)$. For any $n \geq 1$,

$$\begin{aligned} \mathrm{Proj}_{\Omega_{n-1}} \theta_{n+1} &= \mathrm{Proj}_{\Omega_{n-1}} f_{n+1}(o) \\ &= \mathrm{Proj}_{\Omega_{n-1}} \{(f_0(o'), \ldots, f_n(o')) \,:\, o' \in g(o) \cap \mathcal{O}\} \\ &= \{(f_0(o'), \ldots, f_{n-1}(o')) \,:\, o' \in g(o) \cap \mathcal{O}\} \\ &= f_n(o) = \theta_n, \end{aligned} \tag{8}$$

and hence $f(o)$ is a consistent restriction hierarchy.

Given any $o \in \mathcal{O}$. Consider a sequence $\left\{\left(\theta_0^k, \theta_1^k, \ldots\right)\right\}_{k \geq 1}$ of restriction hierarchies such that

$$\begin{aligned} &\left(\theta_0^1, \theta_1^1, \ldots\right) \in \Gamma(f(o)), \quad \text{and} \\ \forall k > 1, \quad &\left(\theta_0^k, \theta_1^k, \ldots\right) \in \Gamma\left(\theta_0^{k-1}, \theta_1^{k-1}, \ldots\right). \end{aligned} \tag{9}$$

We claim that, $\forall k \geq 1$, there exists $\left\{o_0^k, o_1^k, \ldots\right\} \subset \mathcal{O}$ such that,

$$\forall n \geq 0, \quad \left(\theta_0^k, \ldots, \theta_n^k\right) = \left(f_0\left(o_n^k\right), \ldots, f_n\left(o_n^k\right)\right). \tag{10}$$

We prove this claim by induction. Consider $k = 1$. Since $(\theta_0^1, \theta_1^1, \ldots) \in \Gamma(f(o))$, we have, $\forall n \geq 0$,

$$\left(\theta_0^1, \ldots, \theta_n^1\right) \in f_{n+1}(o) = \{(f_0(o'), \ldots, f_n(o')) \,:\, o' \in g(o) \cap \mathcal{O}\},$$

and hence there exists $o_n^1 \in g(o) \cap \mathcal{O}$ such that $(\theta_0^1, \ldots, \theta_n^1) = \left(f_0\left(o_n^1\right), \ldots, f_n\left(o_n^1\right)\right)$.

Assume we have already proved that, for some $k \geq 1$, there exists $\left\{o_0^k, o_1^k, \ldots\right\} \subset \mathcal{O}$ that satisfies (10). Since $\left(\theta_0^{k+1}, \theta_1^{k+1}, \ldots\right) \in \Gamma\left(\theta_0^k, \theta_1^k, \ldots\right)$, we have, $\forall n \geq 0$,

$$\left(\theta_0^{k+1}, \ldots, \theta_n^{k+1}\right) \in \theta_{n+1}^k = f_{n+1}\left(o_{n+1}^k\right) = \left\{(f_0(o'), \ldots, f_n(o')) : o' \in g\left(o_{n+1}^k\right) \cap \mathcal{O}\right\},$$

and hence there exists $o_n^{k+1} \in g\left(o_{n+1}^k\right) \cap \mathcal{O}$ such that $(\theta_0^{k+1}, \ldots, \theta_n^{k+1}) = \left(f_0\left(o_n^{k+1}\right), \ldots, f_n\left(o_n^{k+1}\right)\right)$. By induction, the claim is hence true for all $k \geq 1$.

Given any $\left(\theta_0^k, \theta_1^k, \ldots\right)$, for any $n \geq 0$,

$$\begin{aligned}\mathrm{Proj}_{\Omega_{n-1}} \theta_{n+1}^k &= \mathrm{Proj}_{\Omega_{n-1}} f_{n+1}\left(o_{n+1}^k\right) \\ &= f_n\left(o_{n+1}^k\right) = \theta_n^k,\end{aligned}$$

where the second equality follows from (8), and the last equality follows from $(\theta_0^k, \ldots, \theta_n^k, \theta_{n+1}^k) = \left(f_0(o_{n+1}^k), \ldots, f_n(o_{n+1}^k), f_{n+1}(o_{n+1}^k)\right)$. Therefore, $\left(\theta_0^k, \theta_1^k, \ldots\right)$ is a consistent restriction hierarchy.

Since this is true for any $\left(\theta_0^k, \theta_1^k, \ldots\right)$ in any sequence $\left\{\left(\theta_0^k, \theta_1^k, \ldots\right)\right\}_{k \geq 1}$ that satisfies (9), we have $f(o) \in \mathcal{O}_k$ for every $k \geq 1$, and hence $f(o) \in \mathcal{O}^U$. Since this is true for any $o \in \mathcal{O}$, we have proved that the range of f lies inside $\mathcal{O}^U$. ■

Lemma 3 *If $\mathcal{S} = (\mathcal{O}, g)$ is finite irreducible, the canonical representation f is a 1-to-1 mapping.*

Proof: Let $\mathcal{S} = (\mathcal{O}, g)$ be a finite irreducible space of open source licenses. Define an equivalence relation $\sim$ on $\mathcal{O}$ such that $\forall o_1, o_2 \in \mathcal{O}$, $o_1 \sim o_2$ iff $f(o_1) = f(o_2)$. Suppose, by way of contradiction, $f : \mathcal{O} \to \mathcal{O}^U$ is not 1-to-1, then $\sim$ will be a nontrivial equivalence relation. To arrive at a contradiction, it suffices to prove that $\sim$ and g are compatible; i.e., it suffices to prove that, whenever $o_1 \sim o_2$,

$$(\mu \circ g)(o_1) = \{[x] : x \in g(o)\} =: A_1 = A_2 := \{[x] : x \in g(o')\} = (\mu \circ g)(o_2).$$

Since o_1 and o_2 play symmetric roles, it suffices to prove that, $[x] \in A_1 \implies [x] \in A_2$.

Suppose $[\mathbf{P}] \in A_1$, then $f_0(o_1) = 1$. But then $f_0(o_2) = 1$ as well because $f(o_1) = f(o_2)$, and hence $[\mathbf{P}] \in A_2$ as claimed.

Suppose $[o'] \in A_1 \setminus A_2$, then there exists $o_1' \in g(o_1)$ such that $f(o_1') = f(o')$, but for any $o_2' \in g(o_2)$, $f(o_2') \neq f(o')$. By finiteness of $g(o_2)$, there exists $n \geq 0$ such that

$$(f_0(o'), \ldots, f_n(o')) \notin \{(f_0(o_2'), f_1(o_2'), \ldots, f_n(o_2')) : o_2' \in g(o_2) \cap \mathcal{O}\}.$$

But then we have

$$\begin{aligned}\left(f_0\left(o'\right),\ldots,f_n\left(o'\right)\right) &= \left(f_0\left(o'_1\right),\ldots,f_n\left(o'_1\right)\right)\\ &\in f_{n+1}\left(o_1\right)\\ &= f_{n+1}\left(o_2\right)\\ &= \left\{\left(f_0\left(o'_2\right),f_1\left(o'_2\right),\ldots,f_n\left(o'_2\right)\right) : o'_2 \in g\left(o_2\right)\cap\mathcal{O}\right\},\end{aligned}$$

a contradiction. ■

To be able to make use of Lemma 3, we shall focus on finite spaces of open source licenses in our subsequent analysis.

5 Imposture-Proof Open Source Licenses

To motivate the idea of imposture-proofness, let's consider the following open source license:

$\rotatebox[origin=c]{180}{$\mathbf{1}$}$ *going open source with the 1-time-forbiddance license*
Developer t goes open source. If developer $t+1$ chooses to develop software $t+1$, he has to go open source with the recursive-BSD license (i.e., **R**), thus enabling future developers to go proprietary.

As suggested by the symbol, license $\rotatebox[origin=c]{180}{$\mathbf{1}$}$ in effect turns the idea of license **1** upside-down. While **1** allows the next developer to go proprietary but forbids him (or any future developer) to allow his successor to do the same, $\rotatebox[origin=c]{180}{$\mathbf{1}$}$ does not allow the next developer to go proprietary but requires that he (or any future developer) allows his successor to do so. We conjecture that, like **1**, license $\rotatebox[origin=c]{180}{$\mathbf{1}$}$ may also serve purposes not served by the GPL or the BSD license if developers deviate from exponential discounting.[20] However, unlike **1**, license $\rotatebox[origin=c]{180}{$\mathbf{1}$}$ is probably not enforceable.

[20]Specifically, we conjecture that if developers exhibit some kind of anti-hyperbolic discounting—in particular, if the discount rate between periods $t+2$ and $t+3$ is *smaller* for developer $t+1$ than for developer t—then developer $t+1$ may be more tempted than developer t is to forbid developer $t+2$ from going proprietary, resulting in developer t finding it optimal to use $\rotatebox[origin=c]{180}{$\mathbf{1}$}$ to limit developer $t+1$'s ability to impose such forbiddance. We have not tried to work out such an example, partly because such kind of anti-hyperbolic discounting seems unrealistic, and partly because $\rotatebox[origin=c]{180}{$\mathbf{1}$}$ is unlikely to be enforceable anyway.

Imagine that developer $t-1$ goes open source with license **I**, hoping to prohibit developer t from going proprietary. One possible way for developer t to game this license is to split his software t into two successive versions, version t.1 and version t.2, with version t.1 a more advanced version of software $t-1$, and version t.2 a more advanced version of version t.1. He can roll out version t.1 first, go open source with license **R**, thus satisfying the terms in developer $t-1$'s license **I**. He can then roll out version t.2, possibly using a different identity, and then go proprietary, which is allowed by the terms in version t.1's license **R**.

The reason why developer t can game license **I** is that, while **I** precludes **P**, it allows for **R** which allows for **P**. More generally, any license that tries to preclude option x but allows for some open source license o that allows for option x can be gamed in a similar manner. This motivates the following axiom.

Definition 3 *Let $\mathcal{S} = (\mathcal{O}, g)$ be a space of open source licenses. An open source license $o \in \mathcal{O}$ is said to be imposture-proof if*

$$o' \in g(o) \cap \mathcal{O} \quad \Longrightarrow \quad g(o') \subseteq g(o).$$

The space $\mathcal{S}$ is said to be imposture-proof if every open source license $o \in \mathcal{O}$ is imposture-proof.

Note that all the open source licenses studied in Sections 2 and 3 (i.e., **G**, **B**, **R**, and **1**) are imposture-proof open source licenses.

Given any space of open source licenses $\mathcal{S} = (\mathcal{O}, g)$ that is imposture-proof, and given any open source license $o \in \mathcal{O}$, let's recursively define the following:

$$\begin{aligned}
G_0(o) &= g(o) \\
G_1(o) &= \bigcup_{o' \in G_0(o) \cap \mathcal{O}} g(o') \\
&\vdots \\
G_n(o) &= \bigcup_{o' \in G_{n-1}(o) \cap \mathcal{O}} g(o').
\end{aligned}$$

Since the space $\mathcal{S}$ is imposture-proof, we must have

$$G_0(o) \supseteq G_1(o) \supseteq G_2(o) \supseteq \cdots .$$

Therefore, there exists a first time (possibly infinity), denoted by $L(o)$, such that the option $\mathbf{P}$ is no longer available; i.e., $\mathbf{P} \in G_n$ iff $n < L(o)$. We say that $L(o)$ is the (*upper*) *level* of open source license o. Among the open source licenses studied in Sections 2 and 3, we have

$$\begin{aligned} L(\mathbf{G}) &= 0, \\ L(\mathbf{B}) &= \infty, \\ L(\mathbf{R}) &= \infty, \quad \text{and} \\ L(\mathbf{1}) &= 1. \end{aligned}$$

The following theorem is the main result of this section that we shall use in Section 6.

Theorem 3 *Let $\mathcal{S} = (\mathcal{O}, g)$ be a finite irreducible space of open source licenses that is imposture-proof. Any open source license $o \in \mathcal{O}$ with $L(o) = 0$ is identical to the GPL license $\mathbf{G}$ in the sense that $g(o) = \{o\}$.*

Proof: We shall prove that $\mathbf{P} \notin g(o)$, and $\forall o' \in g(o)$, $f(o') = f(o)$, which by Lemma 3 will then imply that $g(o) = \{o\}$.

Since $L(o) = 0$, $\mathbf{P} \notin g(o)$, which also implies that $f_0(o) = 0$. For any $o' \in g(o)$, since $g(o') \subseteq G_1^o \subseteq G_0^o = g(o)$, we have $\mathbf{P} \notin g(o')$ as well, and hence $f_0(o') = 0 = f_0(o)$. This also implies that

$$f_1(o) = \{f_0(o') : o' \in g(o) \cap \mathcal{O}\} = \{f_0(o)\}.$$

Assume we have already proved that, for some $n \geq 1$,

1. $\forall m = 0, \ldots, n-1$, $\forall o' \in g(o) \cap \mathcal{O}$, $f_m(o') = f_m(o)$;
2. and
$$f_n(o) = \{(f_0(o'), \ldots, f_{n-1}(o')) : o' \in g(o) \cap \mathcal{O}\} = \{(f_0(o), \ldots, f_{n-1}(o))\}. \qquad (11)$$

Then, for any $o' \in g(o) \cap \mathcal{O}$,

$$\begin{aligned} f_n(o') &= \{(f_0(o''), \ldots, f_{n-1}(o'')) \,:\, o'' \in g(o') \cap \mathcal{O}\} \\ &\subseteq \{(f_0(o''), \ldots, f_{n-1}(o'')) \,:\, o'' \in G_1^o \cap \mathcal{O}\} \\ &\subseteq \{(f_0(o''), \ldots, f_{n-1}(o'')) \,:\, o'' \in G_0^o \cap \mathcal{O}\} \\ &= \{(f_0(o''), \ldots, f_{n-1}(o'')) \,:\, o'' \in g(o) \cap \mathcal{O}\} \\ &= f_n(o). \end{aligned}$$

Since $f_n(o)$ is a singleton by the inductive assumption (11), and $f_n(o') \in \mathcal{P}(\Omega_{n-1})$ is nonempty, we have $\forall o' \in g(o) \cap \mathcal{O}$, $f_n(o') = f_n(o)$.

Therefore,

$$\begin{aligned} f_{n+1}(o) &= \{(f_0(o'), \ldots, f_{n-1}(o'), f_n(o')) \,:\, o' \in g(o) \cap \mathcal{O}\} \\ &= \{(f_0(o), \ldots, f_{n-1}(o), f_n(o))\}. \end{aligned}$$

By induction, we have for any $o' \in g(o) \cap \mathcal{O}$, $\forall n \geq 0$, $f_n(o') = f_n(o)$, and hence $f(o') = f(o)$ as claimed. ■

In the rest of this section, we shall provide similar characterizations of the other open source licenses studied in Sections 2 and 3. These characterizations, however, will not be used in Section 6. Readers who are eager to learn in what environment it is without loss of generality for any developer going open source to choose only between the GPL and the BSD licenses can skip the rest of this section and jump to Section 6 without loss.

Lemma 4 *Let $\mathcal{S} = (\mathcal{O}, g)$ be a finite irreducible space of open source licenses that is imposture-proof. Any open source license $o \in \mathcal{O}$ with $L(o) = 1$ is identical to the 1-chance-only license* $\mathbf{1}$ *in the sense that $\exists \mathbf{G} \in \mathcal{O}$ with $g(\mathbf{G}) = \{\mathbf{G}\}$ such that $g(o) = \{\mathbf{P}, \mathbf{G}\}$.*

Proof: By definition, $L(o) = 1$ implies that $\mathbf{P} \in G_0(o) = g(o)$ but $\mathbf{P} \notin G_1(o) = \cup_{o' \in g(o) \cap \mathcal{O}} g(o')$. Therefore, $\forall o' \in g(o) \cap \mathcal{O}$, $L(o') = 0$, and hence by Theorem 3 is identical to the GPL license in the sense that $g(o') = \{o'\}$. Irreducibility then implies that there is only one such o', denoted by $\mathbf{G}$, and that $g(o) = \{\mathbf{P}, \mathbf{G}\}$. ■

For any $L \geq 1$, let's recursively define the *L-chances-only license* **L** as follows.

L: *going open source with the L-chances-only license*
Developer t goes open source. If developer $t+1$ chooses to develop software $t+1$, he can go propriety (i.e., **P**), or go open source with either **G**, **1**, **2**, . . . , **L-2**, or **L-1**.

Theorem 4 *Let $\mathcal{S} = (\mathcal{O}, g)$ be a finite irreducible space of open source licenses that is imposture-proof. Any open source license $o \in \mathcal{O}$ with $L(o) = L \geq 1$ is identical to the L-chances-only license* **L** *in the sense that $\exists \mathbf{G}, \mathbf{1}, \mathbf{2} \ldots, \mathbf{L-1} \in \mathcal{O}$, with $g(\mathbf{G}) = \{\mathbf{G}\}$, $g(\mathbf{1}) = \{\mathbf{P}, \mathbf{G}\}$, . . . , and $g(\mathbf{L-1}) = \{\mathbf{P}, \mathbf{G}, \mathbf{1}, \ldots, \mathbf{L-2}\}$, such that $g(o) = \{\mathbf{P}, \mathbf{G}, \mathbf{1}, \ldots, \mathbf{L-1}\}$.*

Proof: By Lemma 4, the statement is true for $L = 1$. Assume we have already proved that the statement is true for $L = 1, \ldots, m$, for some $m \geq 1$. Consider an open source license $o \in \mathcal{O}$ with $L(o) = m+1$. By definition, $L(o) = m+1$ implies that $\mathbf{P} \in G_0(o), \ldots, G_m(o)$ but $\mathbf{P} \notin G_{m+1}(o)$. Therefore, $\forall o' \in g(o) \cap \mathcal{O}$, $L(o') \leq m$, and $\exists o' \in g(o)$ such that $L(o') = m$. Therefore, by the inductive assumption there exist (and, by imposture-proofness, unique) $\mathbf{G}, \mathbf{1}, \mathbf{2} \ldots, \mathbf{m} \in \mathcal{O}$, with $g(\mathbf{G}) = \{\mathbf{G}\}$, $g(\mathbf{1}) = \{\mathbf{P}, \mathbf{G}\}$, . . . , and $g(\mathbf{m}) = \{\mathbf{P}, \mathbf{G}, \mathbf{1}, \ldots, \mathbf{m-1}\}$, such that

$$G_0(o) \subseteq \{\mathbf{P}\,\mathbf{G}, \mathbf{1}, \ldots, \mathbf{m}\} = g(\mathbf{m}) \cup \{\mathbf{m}\} \subseteq G_1(o) \cup \{\mathbf{m}\} \subseteq G_0(o),$$

and hence all inclusions are equality. By induction, the statement is true for any $L \geq 1$. ■

By imposture-proofness, if $L(o) = \infty$, then $\exists o' \in g(o) \cap \mathcal{O}$ such that $L(o') = \infty$, and hence $\max\{L(o') : o' \in g(o) \cap \mathcal{O}\} = \infty$. Let's define

$$l(o) = \min\{L(o') : o' \in g(o) \cap \mathcal{O}\}.$$

We shall call $l(o)$ the *lower level* of open source license o. Among the open source licenses studied in Sections 2 and 3, we have

$$l(\mathbf{G}) = 0,$$
$$l(\mathbf{B}) = 0,$$
$$l(\mathbf{R}) = \infty, \quad \text{and}$$
$$l(\mathbf{1}) = 0.$$

Lemma 5 *Let $\mathcal{S} = (\mathcal{O}, g)$ be a finite irreducible space of open source licenses that is imposture-proof. For any open source license $o \in \mathcal{O}$, $l(o)$ is either 0 or infinity.*

Proof: Note that, for any open source license $o \in \mathcal{O}$, $\forall o' \in g(o)$, $L(o') \geq l(o)$, and $\exists o' \in g(o)$ such that $L(o') = l(o)$. Suppose $l(o) = l < \infty$. Then, by Theorem 4, $\exists \mathbf{G}, \mathbf{1}, \mathbf{2} \ldots, \mathbf{L} \in \mathcal{O}$, with $g(\mathbf{G}) = \{\mathbf{G}\}$, $g(\mathbf{1}) = \{\mathbf{P}, \mathbf{G}\}$, $\ldots$, and $g(\mathbf{L}) = \{\mathbf{P}, \mathbf{G}, \mathbf{1}, \ldots, \mathbf{L-1}\}$, such that $\mathbf{L} \in g(o)$. By imposture-proofness, $\mathbf{G} \in g(\mathbf{L}) \subset g(o)$, and hence $l(o) \leq L(\mathbf{G}) = 0$. ■

Theorem 5 *Let $\mathcal{S} = (\mathcal{O}, g)$ be a finite irreducible space of open source licenses that is imposture-proof. Any open source license $o \in \mathcal{O}$ with $L(o) = l(o) = \infty$ is identical to the recursive-BSD license* $\mathbf{R}$ *in the sense that $g(o) = \{\mathbf{P}, o\}$.*

Proof: We shall prove that $\mathbf{P} \in g(o)$, and $\forall o' \in g(o)$, $f(o') = f(o)$, which by Lemma 3 will then imply that $g(o) = \{\mathbf{P}, o\}$.

Since $L(o) = \infty$, $\mathbf{P} \in g(o)$, which also implies that $f_0(o) = 1$. For any $o' \in g(o)$, since $L(o') \geq l(o) = \infty$, we have $\mathbf{P} \in g(o')$ as well, and hence $f_0(o') = 1 = f_0(o)$. This also implies that

$$f_1(o) = \{f_0(o') : o' \in g(o) \cap \mathcal{O}\} = \{f_0(o)\}.$$

Assume we have already proved that, for some $n \geq 1$,

1. $\forall m = 0, \ldots, n-1$, $\forall o' \in g(o) \cap \mathcal{O}$, $f_m(o') = f_m(o)$;
2. and
$$f_n(o) = \{(f_0(o'), \ldots, f_{n-1}(o')) : o' \in g(o) \cap \mathcal{O}\} = \{(f_0(o), \ldots, f_{n-1}(o))\}. \tag{12}$$

Then, for any $o' \in g(o) \cap \mathcal{O}$,

$$\begin{aligned} f_n(o') &= \{(f_0(o''), \ldots, f_{n-1}(o'')) : o'' \in g(o') \cap \mathcal{O}\} \\ &\subseteq \{(f_0(o''), \ldots, f_{n-1}(o'')) : o'' \in G_1^o \cap \mathcal{O}\} \\ &\subseteq \{(f_0(o''), \ldots, f_{n-1}(o'')) : o'' \in G_0^o \cap \mathcal{O}\} \\ &= \{(f_0(o''), \ldots, f_{n-1}(o'')) : o'' \in g(o) \cap \mathcal{O}\} \\ &= f_n(o). \end{aligned}$$

Since $f_n(o)$ is a singleton by the inductive assumption (11), and $f_n(o') \in \mathcal{P}(\Omega_{n-1})$ is nonempty, we have $\forall o' \in g(o) \cap \mathcal{O}$, $f_n(o') = f_n(o)$.

Therefore,

$$\begin{aligned} f_{n+1}(o) &= \{(f_0(o'), \ldots, f_{n-1}(o'), f_n(o')) : o' \in g(o) \cap \mathcal{O}\} \\ &= \{(f_0(o), \ldots, f_{n-1}(o), f_n(o))\}. \end{aligned}$$

By induction, we have for any $o' \in g(o) \cap \mathcal{O}$, $\forall n \geq 0$, $f_n(o') = f_n(o)$, and hence $f(o') = f(o)$ as claimed. ∎

We have, up to this point, characterized licenses **G**, **1**, ..., **L**, and **R**. By Lemma 5, all the remaining imposture-proof open source licenses have the properties of $L(o) = \infty$ and $l(o) = 0$. These include the two BSD licenses studied in Sections 2 and 3, respectively, and many more.[21] Common across these licenses are:

1. All allows the next developer to go proprietary; i.e., $\mathbf{P} \in g(o)$.
2. All allows the next developer to go open source with a license $o' \in \mathcal{O}$ that is restrictive in the sense that $L(o') < \infty$.
3. All allows the next developer to go open source with a license $o' \in \mathcal{O}$ that is permissive in the sense that $L(o') = \infty$.

We have not tried to further categorizing these licenses. Indeed, Theorem 6 in Section 6 says that many of them will be irrelevant under the assumptions of exponential discounting and atomless joint distribution $P(\cdot, \cdot)$ of development cost c and potential profit π.

6 The Sufficiency of GPL and BSD

In this section, we shall present an environment where, in any equilibrium, if developer 0 is ever going to go open source, he cannot do better than going open source with either

[21]Note that the two BSD licenses studied in Sections 2 and 3, respectively, are not exactly the same—while both have the flavor of "everything goes", the meaning of "everything" differs across the two spaces of open source licenses studied in those two respective sections—and hence should more appropriately be regarded as different variants of the BSD license.

the GPL or the BSD license. This result is of interest because it sheds light on why GPL and BSD arose as the two most prominent open source licenses in the history of the open source movement.

Theorem 6 *Assume (i) exponential discounting and that (ii) the joint distribution $P(\cdot, \cdot)$ of development cost c and potential profit π is atomless. Let $\mathcal{S} = (\mathcal{O}, g)$ be a finite irreducible space of open source licenses that is imposture-proof and contains open source licenses $\mathbf{G}, \mathbf{B} \in \mathcal{O}$ such that $g(\mathbf{G}) = \{\mathbf{G}\}$ and $g(\mathbf{B}) = \{\mathbf{P}\} \cup \mathcal{O}$. Then, in any equilibrium, if developer 0 is ever going to go open source, he cannot do better than going open source with either* **G** *or* **B**.

Proof: We have already seen in Subsection 2.1 that either **G** or **B** can be developer 0's optimal choice if these are the only two open source licenses available to him.[22] By Lemma 3, any open source license $o \in \mathcal{O}$ with $L(o) = 0$ is identical to **G**. Therefore, it suffices to prove that developer 0 cannot do better by going open source with any other open source license $o \in \mathcal{O}$ with $L(o) > 0$.

For any open source license $o \in \mathcal{O}$ with $L(o) > 0$, we have $\mathbf{P} \in g(o)$. Suppose developer 0 goes open source with o. For any $o' \in g(o) \cap \mathcal{O}$, let $v(o')$ be developer 1's gross expected payoff (gross of development cost c_1) if he goes open source with o', where the expectation is taken over the realizations of $\{(c_t, \pi_t)\}_{t \geq 2}$, and is taken conditional on the strategies of developers $t \geq 2$. Note that $v(o') = W + \beta(\cdots) \geq W$. Let $v^* = \max_{o' \in g(o) \cap \mathcal{O}} v(o')$. Developer 1's optimal strategy is hence

1. not to develop software 1 if $(c_1, \pi_1) \in E^0 := \{(c, \pi) \,|\, c > \max\{\pi, v^*\}\}$;
2. to develop software 1 and go proprietary if $(c_1, \pi_1) \in E^P := \{(c, \pi) \,|\, \pi > \max\{c, v^*\}\}$; and
3. to develop software 1 and go open source with one of the open source licenses in $\mathit{argmax}_{o' \in g(o) \cap \mathcal{O}}\, v(o')$ if $(c_1, \pi_1) \in E^* := \{(c, \pi) \,|\, v^* > \max\{c, \pi\}\}$.

Since $P(\cdot, \cdot)$ is atomless, the boundaries of events E^0, E^P, and E^* have probability 0 and hence can be ignored.

By going open source with o, developer 0's gross expected payoff (gross of development cost c_0) is hence

$$W + \beta \left[P(E^0) \times 0 + P(E^P) \times w + P(E^*) \times v^* \right],$$

[22]While the example in Subsection 2.1 involves a joint distribution $P(\cdot, \cdot)$ that contains atoms, it can be easily modified into one with an atomless joint distribution without affecting this conclusion.

which is strictly increasing in v^* because $v^* \geq W > w$. Since v^* is weakly increasing in the set $g(o) \cap \mathcal{O}$ (by the order of set inclusion), it is maximized by **B**. This proves that developer 0, if he is ever going to go open source, cannot do better than going open source with either **G** or **B**. ■

In the proof of Theorem 6, the assumption of an atomless joint distribution $P(\cdot, \cdot)$ of development cost c and potential profit π implies that developer 1 is indifferent (between developing software 1 or not, and between going proprietary or going open source) with probability 0. If developer 1 is indifferent with strictly positive probability, how he breaks ties has non-trivial implications on developer 0's gross expected payoff. If developer 1 breaks ties in a manner that depends on some payoff-irrelevant details—such as the name of the license, or whether the license allows for certain irrelevant options—then it is possible that the BSD license does not fare as well as another license simply because the former leads developer 1 to break ties in a way that is unfavorable to developer 0. If we alternatively assume that, say, developers always break ties in favor of developing the software, and in favor of going open source, then we can relax the assumption of atomless $P(\cdot, \cdot)$ in Theorem 6.

7 Concluding Remarks

The open source movement is nothing short of a revolution in how production is organized. It has rightfully attracted a lot of economic studies, but very few on the very licenses that made this revolution possible. This paper is the first attempt to provide a model that is at the same time tractable enough to accommodate more than three generations of developers, and yet rich enough that the defining features of GPL (its "share-alike" requirement) and BSD (its permission to go proprietary) can be studied explicitly.

To build such a tractable yet rich model, we have made a number of assumptions, most notably

1. the linear structure of the community of developers (i.e., each developer, if he goes open source, can inspire at most one developer),

2. the *i.i.d.* assumption imposed on the joint distribution $P(\cdot\,,\,\cdot)$ of development cost c and potential profit π, and

3. the assumption that consumer surplus (either W or w) is deterministic.

It should be pointed out that our construction of the universal space of open source licenses (Section 4) and the characterizations of imposture-proof open source licenses (Section 5) do not depend on these assumptions. If anything, relaxing these assumption helps overturning the sufficiency of the GPL and the BSD licenses (Theorem 6), rendering other open source licenses characterized in Sections 4 and 5 relevant again. We hence expect that our results in Sections 4 and 5 will be even more useful for future research that studies more general environments than what is allowed by our assumptions.

References

[1] Athey, Susan and Glenn Ellison (2014), "Dynamics of Open Source Movements." *Journal of Economics and Management Strategy*, 23(2): 294-316.

[2] Brandenburger, Adam and Eddie Dekel (1993), "Hierarchies of Beliefs and Common Knowledge." *Journal of Economic Theory*, 59: 189-198.

[3] Casadesus-Masanell, Ramon and Pankaj Ghemawat (2006), "Dynamic Mixed Duopoly: A Model Motivated by Linux vs. Windows." *Management Science*, 52(7): 1072-1084.

[4] Economides, Nicholas and Evangelos Katsamakas (2006), "Two-Sided Competition of Proprietary vs. Open Source Technology Platforms and the Implications for the Software Industry." *Management Science*, 52(7): 1057-1071.

[5] Fershtman, Chaim and Neil Gandal (2007), "Open Source Software: Motivation and Restrictive Licensing." *International Economics and Economic Policy*, 4: 209-225.

[6] Fielding, Roy T. (1999), "Shared Leadership in the Apache Project." *Communications of the ACM*, 42(4): 42-43.

[7] Gaudeul, Alex (2004), "Open Source Software Development Patterns and License Terms." Toulouse working paper.

[8] Gaudeul, Alex (2005), " Public Provision of a Private Good: What is the Point of the BSD License?" Toulouse working paper.

[9] Gaudeul, Alex (2007), "Do Open Source Developers Respond to Competition? The (La)TEXCase Study." *Review of Network Economics*, 6(2): 239-263.

[10] Han, Xintong and Lei Xu (2019), "Technology Adoption in Input-Output Networks." Bank of Canada Staff Working Paper 2019-51.

[11] Hertel, Guido, Sven Niedner, and Stefanie Herrmann (2003), "Motivation of Software Developers in Open Source Projects: An Internet-Based Survey of Contributors to the Linux Kernel." *Research Policy*, 32: 1159-1177.

[12] Johnson, Justin P. (2002), "Open Source Software: Private Provision of a Public Good." *Journal of Economics and Management Strategy*, 11(4): 637-662.

[13] Johnson, Justin P. (2006), "Collaboration, Peer Review and Open Source Software." *Information Economics and Policy*, 18: 477-497.

[14] Lerner, Josh and Jean Tirole (2002), "Some Simple Economics of Open Source." *Journal of Industrial Economics*, 50(2): 197-234.

[15] Lerner, Josh and Jean Tirole (2005a), "The Economics of Technology Sharing: Open Source and Beyond." *Journal of Economic Perspectives*, 19(2): 99-120.

[16] Lerner, Josh and Jean Tirole (2005b), "The Scope of Open Source Licensing." *Journal of Law, Economics, and Organization*, 21(1): 20-56.

[17] Mariotti, Thomas, Martin Meier, and Michele Piccione (2005), "Hierarchies of Beliefs for Compact Possibility Models." *Journal of Mathematical Economics*, 41: 303-324.

[18] Mertens, Jean-François and Shmuel Zamir (1985), "Formulation of Bayesian Analysis for Games with Incomplete Information." *International Journal of Game Theory*, 14: 1-29.

[19] Mockus, Audris, Roy T. Fielding, and James D. Herbsleb (2002), "Two Case Studies of Open Source Software Development: Apache and Mozilla." *ACM Transactions on Software Engineering and Methodology*, 11(3): 309-346.

[20] Roberts, Jeffrey A. Il-Horn Hann, and Sandra A. Slaughter (2006), "Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects." *Management Science*, 52(7): 984-999.

[21] von Hippel, Eric and Georg von Krogh (2003), "Open Source Software and the 'Private-Collective' Innovation Model: Issues for Organization Science." *Organization Science*, 14(2): 209-223.