Designing Open Source Licenses[*]

Kim-Sau Chung[†] Melody Lo[‡]

January 31, 2023

Abstract

Open source licenses are noted for being self-referential. The two dominant licenses at the early stage of the open source movement were 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 construct the universal space of all self-referential licenses such as GPL and BSD. We also provide a plausible explanation of why GPL and BSD stood out from other licenses as the two most natural choices for the first-generation open source developers.

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

JEL CLASSIFICATIONS: L86, O36

1 Introduction

Once a software is developed, its developer has at least two ways to distribute it. The first is to go proprietary, meaning that she sells copies of the binary code for a profit. Since the 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

[*]We thank Yi-Chun Chen, Sambuddha Gosh, Junichiro Ishida, Chia-hui Lu, Thomas Mariotti, William Taysom, and participants at various seminars and conferences for very helpful discussions. Chung and Lo are co-first authors. The work described in this paper is supported by the Research Grants Council of Hong Kong (via grant GRF12502121) and HKBU (via grant RC-IG-FNRA/17-18/01).

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

[‡]Department of Economics, National Taiwan University; peiyulo2006@gmail.com

learn from her source code. In other words, while going proprietary allows 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. 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 GPL, 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 proprietary, and you have to go open source with the same license I am using, namely GPL." This kind of open source licenses are restrictive because they restrict how future developers can distribute their softwares. In particular, the restriction is a "share-alike" requirement, requiring that future developers share in the same manner as the original developer.

One the most famous developers who went open source using GPL is Linus Torvalds. Ever since Linus Torvalds distributed the source code of the first version of Linux using GPL, all subsequent versions have to be shared alike using the same license. Even when Red Hat, a for-profit company, developed their commercial version of Linux, called Red Hat Enterprise Linux (RHEL), they were also bound by GPL to share their source code. It means that they could not make a profit by selling copies of the binary code. Instead, they could 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 BSD, the developer is telling future developers, "You can use my source code to develop a

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

[2] BSD stands for "Berkeley software distribution". We use BSD as an umbrella term referring to various BSD-like licenses including, for example, LPPL (to be discussed below), MIT (which is even more permissive than BSD by not requiring acknowledgement of the licensor), and Apache (which makes explicit certain permissions that are only implicit in BSD). See, for example, Smith (2022).

more advanced version, and you can distribute your advanced version in whichever way you like. You can go proprietary, you can also go open source. If you choose to go open source, you can use any open source license you like, including BSD."

One of the most famous developers who went open source using a BSD-like permissive license is Donald Knuth. While the core of TEX is copyrighted by Donald Knuth and no changes are permitted, add-on programs are licensed under the LATEX Project Public License (LPPL), which is very much like BSD (Gaudeul, 2007). The permissive nature of LPPL allowed subsequent developers to go proprietary and profit from their efforts. This option encouraged 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 softwares (such as Linux, LATEX, Apache, etc.) might never have been developed if developers had not learned how to share their contributions using open source licenses. The movement also had 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.

While GPL and BSD are the two dominant open source licenses, with their dominance especially pronounced at the early stage of the open source movement,[3] it is important to recognize that they were inventions of idealistic developers, who invented them more as their anti-capitalist manifestos, instead of as calculated designs that maximize productivity. Therefore, it is conceivable that a more careful design exercise can unleash even more productivity.

A prerequisite of this design exercise is, of course, the construction of the space of all possible open source licenses, which is the first contribution of this paper. Since open source licenses restrict future developers' choices of open source licenses, they are necessarily self-referential in nature. The construction of the space of all possible open source licenses hence is the construction of the universal space of self-referential licenses, where the technique of constructing universal type spaces can help.

[3] Vendome *et al*. (2017) study 16,221 Java projects on GitHub. They find that, by 2012, BSD-like licenses (including MIT and Apache) and various versions of GPL still accounted for more than 90% of all open source licenses used. Balter (2015) reports that, up to 2015, only 15% of open source projects on GitHub used a license other than MIT, Apache, or GPL.

Our construction of the universal space of self-referential licenses confirms that there indeed exist many self-referential licenses other than GPL and BSD. More importantly, it is not always possible to pigeonhole these licenses into either the restrictive or the permissive camp. For example, a license may be restrictive in the sense that it restricts what restrictions the next developer can impose on the next-next developer, but is permissive at the same time exactly because the next-next developer is less restricted.

We shall see that there indeed exist other self-referential licenses that serve purposes neither GPL nor BSD can serve. However, given the dominance of GPL and BSD at the early stage of the open source movement, we shall also provide a plausible explanation of why they stood out from other licenses as the two most natural choices for the first-generation open source developers.

We develop this explanation in two steps. First, we introduce an axiom called imposture-proofness to exclude some open source licenses that can be gamed by a developer by splitting his version of the software into two consecutive versions, with only the first version subject to the restrictions contained in the previous developer'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 essentially the same as GPL.

In the second step, we show that, with exponential discounting and a linear structure of developers, a developer going open source cannot do better than using either GPL or BSD. If she does not allow the next developer to go proprietary, then her license is essentially the same as GPL, per the result in the first step. If she does, then BSD gives the next developer maximum freedom, which is good for her as well. The key in this last step is that exponential discounting and a linear structure of developers render the interests of the current and the next developers perfectly aligned when it comes to putting restrictions on the next-next developer.

Although the open source revolution has rightfully attracted a lot of economic studies,[4] literally fewer than a handful of them have likewise studied open source *licenses*, notwithstanding the significant role these licenses played in this revolution. Lerner and Tirole (2005b) provide a stylized model that differentiates different open source licenses by a single parameter, namely their "permissiveness". Their model abstracts away the defin-

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

ing 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 mixing-and-matching 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 feature finitely many generations of developers, rendering open source licenses in her models not self-referential.[5]

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 hence the optimality of mixing-and-matching these features can be meaningfully discussed (in contrast to Lerner and Tirole, 2005b). Second, it is a model with infinitely many generations of developers, and hence open source licenses in the model are self-referential (in contrast to Gaudeul, 2004, 2005).[6]

This paper is structured as follows. The rest of this section reviews the related literature. In Section 2, we starts with a basic model that only allows for GPL and BSD. 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 single parameter called "permissiveness" as in Lerner and Tirole (2005b). We shall also demonstrate that these newly introduced licenses are not bogus, and can sometimes serve purposes that neither GPL nor BSD can serve. In Section 4, we go for a fully general model and construct the universal space of open source licenses.

In Section 5, we introduce the axiom of imposture-proofness, and show that any imposture-proof open source license that does not allow the next developer to go proprietary is essentially the same as GPL. In Section 6, we use this result from Section 5 to construct an environment where a developer going open source cannot do better than using either GPL and BSD. Section 7 concludes with some remarks on possible future research.

[5]For example, the penultimate generation's licenses do not restrict the ultimate generation's choices of licenses.

[6]See Section 7 for more discussions on Gaudeul (2004, 2005).

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, and Fershtman and Gandal (2011) a brief survey of the related economics, to economists. Subsequent studies can be roughly divided into three strands. The first focuses on the competition between commercial software companies and the open source community—what prices the companies would set, the chance that they can survive this competition, etc. The open source community is typically modelled as a group of altruistic members who both contribute to and benefit from the open source movement. How their collaboration is facilitated or jeopardised by different choices of licenses is typically not the focus.[7] Factors other than the license are instead the focus. For example, Johnson (2002) focuses on the size of the open source community, Casadesus-Masanell and Ghgemawat (2006) on demand-side learning, and Economides and Katsamakas (2006) on 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, can they serve purposes that neither GPL nor BSD can serve, and when can they be ignored without loss of generality?

The second strand is concerned with the kind of altruism that motivates members of the open source community—are they motivated by warm glow, pride, consumer surplus, or are they merely motivated by the material payoffs from signalling their competence? Athey and Ellison (2014) theoretically study how different kinds of altruism affect the competition between commercial software companies and the open source community; while Hertel, Niedner, and Herrmann (2003), Lerner, Pathak, and Tirole (2006), 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 a developer to go open source. In particular, we assume that he internalizes part of the consumer surplus that may be generated by future developers.[8] We have not explored the implications of other kinds of altruism such as warm glow.

The third strand is interested in the governance structure of the open source commu-

[7] Presumably the license being used is not BSD, as members do not have the option of going proprietary.

[8] Or, alternatively, he gains an "egoboo" everytime when a more advanced version of his software is developed. See Footnote 9 for this this alternative interpretation.

nity. 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 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) for some general insights distilled from these 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 topic of intra-generation coordination hence cannot be studied in our stylized model.

2 The Basic Model

In this section, we shall first describe a basic model that is barely rich enough to accommodate GPL and BSD. 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 $s \geq t$ 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 first draws a tuple of four variables, $\theta_t = (c_t, \pi_t, w_t, W_t)$, where c_t is his cost of developing software t, π_t and w_t are the potential profit and consumer surplus, respectively, if he develops software t and then goes proprietary, and $W_t > w_t$ is the consumer surplus if he develops software t and then goes open source instead. The tuple θ_t is drawn from the probability law $P\left(\theta_t \,|\, \theta^{t-1}\right)$, where $\theta^{t-1} = (\theta_0, \ldots, \theta_{t-1})$. We assume that the realization of θ_t is observable to all developers $s \geq t$.

Upon observing θ_t, developer t then decides whether to pay the development cost c_t and develop software t. If he decides not to (an option denoted by **Q**, which stands for quitting), 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 have, and then explain which subsets of these options are available to him given different open source licenses chosen by developer $t-1$.

P: *going proprietary*

Developer t goes proprietary, realizing potential profit π_t and consumer surplus w_t, and the game ends.

G: *going open source using 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 using GPL (i.e., choosing **G**) as well. For developer t, potential profit π_t is forgone, and the consumer surplus is W_t.

B: *going open source using 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 proprietary (i.e., choosing **P**), or go open source using any of the two licenses (i.e., choosing **G** or **B**). For developer t, potential profit π_t is forgone, and the consumer surplus is W_t.

We should hasten to emphasize that what we called the consumer surplus (either w_t or W_t) 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 $W_t > \pi_t + w_t$, which would have been a natural inequality to assume had we interpreted these as the whole consumer surplus that can potentially be generated. Indeed, allowing for $W_t < \pi_t + w_t$ is necessary to explain why developer t may sometimes go proprietary.[9]

For developer $t > 0$, which of these three options (**P**, **G**, and **B**) are available depends on what open source license he is subject to (i.e., what open source license 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.

For developer 0, we assume that all three options (**P**, **G**, and **B**) are available to him, and hence we may treat him as if he is subject to license **B**.

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, 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.

[9]While our favorite interpretation of w_t and W_t is the (internalizable part of the) consumer surplus, this is not the only possible interpretation. Alternatively, one can interpret them as what Raymond (1999) calls "egoboo"; i.e., the ego boost all developers $s \leq t$ can gain from their increased visibility and recognition when the more advanced software t is developed. If the "egoboo" is increasing in the size of the user base, and if the user base is larger when software t is available for free in an open source manner, then we will again have $W_t > w_t$.

For any developer $t < T$, his payoff is

$$U_t := -c_t + \sum_{s=t}^{T-1} \beta^{s-t} W_s + \beta^{T-t} \times \begin{cases} 0 & \text{if developer } T \text{ does not develop software } T \\ w_T & \text{if developer } T \text{ develops software } T \text{ and goes proprietary} \end{cases}, \tag{1}$$

where $\beta \in (0,1)$ is the common discount factor. In other words, we assume that if developer t ever incurs the development cost c_t but then forgoes his potential profit π_t, he does so because he is altruistic enough to care about current and future consumer surplus.[10]

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

We assume that developer t's choice is observable to all developers $s > t$. We have thus described an infinite-horizon observable-action game. The primitives of the game are the probability law $P(\cdot\,|\,\cdot)$ and the common discount factor β. Our solution concept is subgame perfect equilibrium.

It is easy to see that, depending on the probability law $P(\cdot\,|\,\cdot)$, it can be strictly optimal for developer 0 to choose each of the four options (**Q**, **P**, **G**, and **B**). For example, he would **Q**uit if c_0 is prohibitively large; goes **P**roprietary if W_0 is small compared to $\pi_0 + w_0$ and all (w_t, W_t), $t \geq 1$, are likely to be negligible; goes open source using **G**PL if $\pi_0 + w_0$ is small compared to W_0 and all w_t, $t \geq 1$, are likely to be negligible; and goes open source using **B**SD if $\pi_0 + w_0$ is small compared to W_0, w_1 is likely to be similar to W_1, and all W_t, $t \geq 2$ are likely to be negligible.

Without imposing significantly more structure on the probability law $P(\cdot\,|\,\cdot)$, it seems difficult to fully characterize the conditions under which developer 0 would choose each of these options. Since our main focus is on what *other* open source licenses are available, we shall not pursue such a full characterization here, and shall leave it for future research. The following example, however, demonstrates how, if we are willing to impose strong assumptions on the probability law $P(\cdot\,|\,\cdot)$, some interesting characterizations can be obtained.

[10]One may wonder whether the consumer surplus W_t generated by software t depends on whether software $t+1$ will be developed, as the latter may supersede the former. In that case we may reinterpret W_{t+1} as the marginal increase in total consumer surplus generated by the creation of software $t+1$, taking into account the creative destruction effect on software t (as well as all the earlier softwares).

2.1 An Example

Consider the following example. Suppose $\beta = 1/2$; and $\forall t \geq 0$, $w_t \equiv w = 1$ and $W_t \equiv W = 3$. Suppose the joint distribution of c_t and π_t is iid across t, and is as depicted in Table 2, where z indexes the correlation between c_t and π_t (with $z = 1$ means perfect positive correlation, and $z = -1$ means perfect negative correlation).

$P(c_t, \pi_t)$	$\pi_t = 1$	$\pi_t = 7$
$c_t = 1$	$(1+z)/4$	$(1-z)/4$
$c_t = 7$	$(1-z)/4$	$(1+z)/4$

Table 2: Joint distribution of c_t and π_t.

Note that $c_t = 1$ or 7 with the same marginal probability, and $\pi_t = 1$ or 7 with the same marginal probability.

Since this is a stationary environment, we shall further restrict our attention to pure-strategy Markov perfect equilibrium, in which every developer t follows the same pure Markov strategy that depends only on (*i*) the open source license he is subject to, which in turn is chosen by developer $t-1$,[11,12] and (*ii*) the realization of (c_t, π_t). More formally, let $\mathcal{O} := \{\mathbf{G}, \mathbf{B}\}$ denote the set of open source licenses. A pure Markov strategy is a function $\sigma : (c, \pi, o) \mapsto \{\mathbf{Q}, \mathbf{P}\} \cup \mathcal{O}$ that describes any developer's choice over $\{\mathbf{Q}, \mathbf{P}\} \cup \mathcal{O}$ given his draw of (c, π) and the open source license $o \in \mathcal{O}$ that he is subject to, with the restriction that $\forall (c, \pi)$, $\sigma(c, \pi, \mathbf{G}) \notin \{\mathbf{P}, \mathbf{B}\}$. A pure-strategy Markov perfect equilibrium (hereafter, an *equilibrium*) is a Markov strategy σ^* that is optimal for any developer t among all pure Markov strategies if all future developers $s > t$ follow the pure Markov strategy σ^*.

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 $W_t = W = 3$ (which is achieved if no future software is developed), and is at most

$$W_t + \beta W_{t+1} + \beta^2 W_{t+2} + \cdots = W/(1-\beta) = 3/(1-1/2) = 6$$

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

[11] 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.

[12] Recall that we can treat developer 0 as if he is subject to license **B**.

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 $3 > 2 = \pi_t + w_t$), 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 $\pi_t + w_t < 7$ (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 strategy for a developer who is subject to license **G** (and hence cannot choose **P** or **B**) and for a developer who is subject to license **B**, 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

$$V_G = W_t + (\beta/2)W_{t+1} + (\beta/2)^2 W_{t+2} + \cdots = W/(1 - 1/4) = 3/(1 - 1/4) = 4.$$

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 using BSD if future development costs and potential profits are sufficiently positively correlated (i.e., if* $z \geq 1/2$*), and using GPL 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 BSD, by allowing developer 1 to go proprietary, will help encourage developer 1 to further develop developer 0's idea even in this unfortunate event. BSD hence can be the optimal choice for a developer going open source if development cost and potential profit are positively correlated, while GPL 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, it is easy to see how anyone, by playing with Table 1, would naturally come up with the new licenses depicted in Table 5.[13] In Table 5, **R** is simply the mirror image of **G**. It refers to itself in its own definition, in exactly the same self-referential manner as **G**. But it deviates from **G** by allowing the next developer to go proprietary. License **1** is derived from **G** in a different way. It puts the same restriction on the next developer's choices of open source licenses, but deviates from **G** by also allowing the next developer to go proprietary. Finally, license **Ɩ** is derived from **R** in exactly the same way as how **1** is derived from **G**. It puts the same restriction on the next developer's choices of open source licenses, but deviates from **R** by *not* allowing the next developer to go proprietary.

what developer $t-1$ chose	what developer t can choose after developing software t
G	**G**
R	**P** and **R**
1	**P** and **G**
Ɩ	**R**

Table 5: A simple extension.

We call **R** the *recursive-BSD license*. It pushes the defining feature of BSD (its permission to go proprietary) to its limit. Not only that the next developer is allowed to go proprietary, all future developers, to the extent that they have a chance to move, are allowed to go proprietary as well. This is because, for example, the next developer cannot use GPL to forbid the next-next developer from going proprietary. In this sense, **R** is even more permissive than BSD. Of course, one can also argue that **R** is more restrictive than BSD, as it puts more restrictions on how the next developer can restrict the next-next developer. 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).

We call license **1** the *1-chance-only license*. License **1** is unambiguously more restrictive than **R**. While it, like **R**, gives the next developer the permission to go proprietary, it forbids 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,

[13]These authors confess that they had lots of fun filling out Table 5.

namely in the next period and in the next period only.

We call license $\rotatebox[origin=c]{180}{\mathbf{1}}$ the *1-time-forbiddance license*. As suggested by the symbol, license $\rotatebox[origin=c]{180}{\mathbf{1}}$ in effect turns the idea of license **1** upside-down. While license **1** allows the next developer to go proprietary but forbids him and any future developer from allowing their successors to do the same, license $\rotatebox[origin=c]{180}{\mathbf{1}}$ does not allow the next developer to go proprietary but requires that he and any future developer allow their successors to do so.

Are these new licenses bogus? Are there any reasons why an open source developer may consider using them? Are there situations where they may serve purposes that neither GPL nor BSD serve? The answer is a qualified "no". Indeed, in later sections we shall develop an environment where these new licenses can be ignored without loss of generality (see Theorem 6). This amounts to our explanation of why GPL and BSD stood out from other licenses as the two most natural choices for the first-generation open source developers.

However, that our "no" answer needs to be qualified also means that it is not universally true. In the remainder of this section, we shall provide an example where developer 0 strictly prefers to use **R** instead of either **G** or **B**. It involves developers with hyperbolic discounting, an assumption that will violate the premises of Theorem 6.

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 $\alpha_1 \leq \alpha_2 \leq \beta$.[14]

Consider the following example with hyperbolic discounting. Suppose $\alpha_1 = 1/4$, $\alpha_2 = 2/4$, and $\beta = 3/4$; and $\forall t \geq 0$, $w_t \equiv w = 15$, and $W_t \equiv W = 20$. Suppose the joint distribution of c_t and π_t is iid across t, and is as depicted in Table 6.

[14]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.

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

Table 6: Joint distribution of c_t and π_t.

Again, since this is a stationary environment, we shall restrict our attention to pure-strategy Markov perfect equilibrium. Our notations and definitions follow closely those in Subsection 2.1, except that $\mathcal{O}$ is now the set $\{\mathbf{G}, \mathbf{R}, \mathbf{1}, \mathbf{B}\}$, and $\mathbf{B}$ is appropriately redefined as the license that allows the next developer to choose either $\mathbf{P}$ or any open source license $o \in \mathcal{O}$.[15]

As in the example in Subsection 2.1, we can calculate a lower and an upper bounds for any developer t's gross expected payoff of going open source, which are

$$W_t = W = 20$$

and

$$\begin{aligned} W_t + \alpha_1 \left(W_{t+1} + \alpha_2 \left(W_{t+2} + \beta W_{t+3} + \beta^2 W_{t+4} + \cdots \right) \right) &= W + \alpha_1 \left(W + \alpha_2 \left(W + \beta W + \beta^2 W + \cdots \right) \right) \\ &= 35, \end{aligned}$$

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 $35 < 50 = \pi_t + w_t$). 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 gross expected payoff of going open source, which is approximately

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

[15] We ignore license $\mathbf{I}$, as it is probably not enforceable (see Section 5).

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 $35 = \pi_t + w_t$).

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 as depicted in Tables 7 and 8.

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

Table 7: The equilibrium Markov strategy for a developer who is subject to **G**.

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

Table 8: The equilibrium Markov strategy for a developer who is subject to **B**, **R**, or **1**.

Now consider the problem of a developer t who is subject to license **B** and has drawn $(c_t, \pi_t) = (10, 0)$. If he chooses **G**, all future developers' behavior will be described by Table 7, and 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 discounts 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 $(\alpha_1, \alpha_2, \beta)$-formulation to approximate 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 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.[16]

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.[17] We assume that going open source (using *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.[18]

[16] 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.

[17] It goes without saying that quitting, $\mathbf{Q}$, is always an option. We hence omit it from the developer's choice set for brevity.

[18] 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 using BSD. 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.

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, $|g(o)|$, 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 GPL.

Formally, for any space of open source license $\mathcal{S} = (\mathcal{O}, g)$, let $\sim$ be an equivalence relation on $\mathcal{O}$.[19] 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 $(\mu \circ g)(o) = \{[x] \in \mathcal{O}^{\sim} \cup \{\mathbf{P}\} : x \in g(o)\}$, 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 $g^{\sim}([o]) = (\mu \circ g)(o)$ for any $o \in \mathcal{O}$. 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 distinct $o, o' \in \mathcal{O}$ such that $o \sim o'$. If there exists a nontrivial equivalence relation $\sim$ compatible with g, we say that $\mathcal{S} = (\mathcal{O}, g)$ is *reducible to* $\mathcal{S}^{\sim} = (\mathcal{O}^{\sim}, g^{\sim})$ (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 $(\mu \circ g)(\mathbf{G_1}) = \{[\mathbf{G_2}]\} = (\mu \circ g)(\mathbf{G_2})$. Therefore, $\mathcal{S} = (\mathcal{O}, g)$ is reducible to $\mathcal{S}^{\sim} = (\mathcal{O}^{\sim}, g^{\sim})$, where $\mathcal{O}^{\sim} = \{[\mathbf{G_2}]\}$ and $g^{\sim}([\mathbf{G_2}]) = \{[\mathbf{G_2}]\}$.

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, $\mathcal{S}^{\mathcal{U}} = (\mathcal{O}^{\mathcal{U}}, g^{\mathcal{U}})$, we first recursively construct a sequence of nonempty sets $(\Theta_0, \Omega_0, \Theta_1, \Omega_1, \ldots)$ as follows. Let

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

$\Theta_0 = \Omega_0 = \{0, 1\}$. 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,
- $\ldots$, 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 GPL 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. GPL gives developer t only a single option—meaning that θ_1^G must be a singleton. Moreover, that single option is to use GPL as well, whose first restriction has already been encoded in θ_0^G.[20] Therefore, θ_1^G must be the singleton $\left\{\theta_0^G\right\} = \{0\}$.

[20] 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.

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. GPL gives developer t only a single option—meaning that θ_2^G must be a singleton. Moreover, that single option is to use GPL as well, whose first two restrictions have already been encoded in θ_0^G and θ_1^G.[21] 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 $\left(\theta_0^G, \theta_1^G, \ldots\right)$, 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

[21]GPL 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-next developer to go proprietary.

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 GPL, 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:

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

For any restriction hierarchy $(\theta_0, \theta_1, \ldots)$, to the extent that it represents an open source license (not yet, as we shall see shortly), we should be able to extract 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 $\Gamma : \Omega_\infty \rightrightarrows \Omega_\infty$ as follows: for any $(\theta_0, \theta_1, \ldots) \in \Omega_\infty$ and $(\theta'_0, \theta'_1, \ldots) \in \Omega_\infty$, $(\theta'_0, \theta'_1, \ldots) \in \Gamma(\theta_0, \theta_1, \ldots)$ iff

$\forall n \geq 0$,[22]

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

As an illustration, if we apply the correspondence Γ to restriction hierarchies $\left(\theta_0^G, \theta_1^G, \ldots\right)$, $\left(\theta_0^R, \theta_1^R, \ldots\right)$, and $\left(\theta_0^1, \theta_1^1, \ldots\right)$, we will obtain

$$\begin{aligned}
\Gamma\left(\theta_0^G, \theta_1^G, \ldots\right) &= \left\{\left(\theta_0^G, \theta_1^G, \ldots\right)\right\}, \\
\Gamma\left(\theta_0^R, \theta_1^R, \ldots\right) &= \left\{\left(\theta_0^R, \theta_1^R, \ldots\right)\right\}, \quad \text{and} \\
\Gamma\left(\theta_0^1, \theta_1^1, \ldots\right) &= \left\{\left(\theta_0^G, \theta_1^G, \ldots\right)\right\},
\end{aligned}$$

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 using open source license **G** (respectively, **R** and **G**).

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 $\mathrm{Proj}_{\Omega_{n-1}} \theta_{n+1} = \theta_n$.[23]

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

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

[22]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 $(\theta_0, \theta_1, \ldots)$.

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

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

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 $\theta_1 \in \Theta_1 = \mathcal{P}(\Omega_0) = \mathcal{P}(\Theta_0)$ and $\mathcal{P}(\Theta_0)$ 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 $\mathrm{Proj}_{\Omega_{n-1}} \theta_{n+1} = \theta_n$, and hence $\left(\theta'_0, \ldots, \theta'_{n-1}\right) \in \mathrm{Proj}_{\Omega_{n-1}} \theta_{n+1}$ 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 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 shall not stop here, and shall 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 will be the set of every restriction hierarchy that represents an open source license.[24]

The reader can readily check that each of $\left(\theta^G_0, \theta^G_1, \ldots\right)$, $\left(\theta^R_0, \theta^R_1, \ldots\right)$, and $\left(\theta^1_0, \theta^1_1, \ldots\right)$ belongs 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

[24] 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.

have

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

by the inductive assumption, and hence $\left(\theta_0^G, \theta_1^G, \ldots\right)$ belongs to 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 O_k for any $k \geq 1$, and hence belong to O^U. This also shows that O^U is nonempty.

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

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

We shall call the pair $\mathcal{S}^U = (O^U, g^U)$ 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 (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 that any finite space of open source licenses can be made into a compact continuous possibility structure.[25] 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 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 $o \in O^U \subseteq O_1$, by Lemma 1, $\Gamma(o)$ is a nonempty subset of Ω_∞. Moreover, for any $k \geq 1$, $\Gamma(o) \subseteq O_k$ because $o \in O^U \subseteq O_{k+1}$, and hence $\Gamma(o)$ is also a nonempty subset of $\cap_k O_k = O^U$. Therefore, $g^U(o) \cap O^U = \Gamma(o) \neq \emptyset$, and hence $\mathcal{S} = (O^U, g^U)$ is in itself a space of open source licenses. ■

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

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

called the canonical representation, such that for any $o \in \mathcal{O}$,

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

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_∞. We call this mapping the *canonical representation* of open source licenses. We then prove that the range of f actually lies inside $\mathcal{O}^{\mathcal{U}}$ (Lemma 2). Finally, we prove that, if $\mathcal{S} = (\mathcal{O}, g)$ is finite irreducible, this will be a 1-to-1 mapping that satisfies (7) (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 $o \in \mathcal{O}$,

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

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

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

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 $\mathcal{O}^{\mathcal{U}}$.*

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 $n \geq 1$,

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

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 $n \geq 0$,

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

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* *that satisfies* (7).

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 $o_1 \sim o_2$,

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

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

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

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

Since $o_1' \in g(o_1)$, we have $\left(f_0\left(o_1'\right), \ldots, f_n\left(o_1'\right)\right) \in f_{n+1}\left(o_1\right)$ by the construction of the canonical representation f. 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.

It remains to prove that $f : \mathcal{O} \to \mathcal{O}^U$ satisfies (7). By the construction of f and g^U, we have

$$\mathbf{P} \in g(o) \quad \Longleftrightarrow \quad f_0(o) = 1 \quad \Longleftrightarrow \quad \mathbf{P} \in g^U\left(f(o)\right) = \Gamma\left(f(o)\right) \cup \{\mathbf{P}\}.$$

Suppose $o' \in g(o)$. Then, by the construction of f, $\forall n \geq 0$, we have $\left(f_0\left(o'\right), \ldots, f_n\left(o'\right)\right) \in f_{n+1}(o)$. By the construction of Γ, we have $\left(f_0\left(o'\right), f_1\left(o'\right), \ldots\right) \in \Gamma\left(f_0(o), f_1(o), \ldots\right)$. Therefore, by the construction of g^U, we have $f\left(o'\right) \in g^U\left(f(o)\right)$.

Suppose $f(o') \in g^U(f(o))$. Then, by the construction of g^U, we have $f(o') \in \Gamma(f(o))$. By the construction of Γ, $\forall n \geq 0$, we have $(f_0(o'), \ldots, f_n(o')) \in f_{n+1}(o)$. By finiteness of $g(o)$, there exists $o'' \in g(o)$ such that $f(o'') = f(o')$. By the fact that f is a 1-to-1 mapping, we have $o' = o'' \in g(o)$. This completes the proof that f satisfies (7). ■

5 Imposture-Proof Open Source Licenses

To motivate the idea of imposture-proofness, let's revisit the 1-time-forbiddance license introduced in Section 3:

- **Ɪ** *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., choosing **R**), thus enabling future developers to go proprietary.

Imagine that developer $t-1$ goes open source with license **Ɪ**, 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 **Ɪ**. 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 **Ɪ** is that, while **Ɪ** precludes **P**, it allows for **R** which in turn 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 other open source licenses studied in Section 3 (i.e., **G**, **R**, and **1**) are imposture-proof open source licenses.

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 an irreducible space of open source licenses that is imposture-proof. Any open source license $o \in \mathcal{O}$ that does not allow the next developer to go proprietary (i.e., $\mathbf{P} \notin g(o)$) is identical to the GPL license* **G** *in the sense that $g(o) = \{o\}$.*

Proof: Consider any open source license $o \in \mathcal{O}$ such that $\mathbf{P} \notin g(o)$. If $o' \notin g(o)$ for any $o' \neq o$, then by the nonemptyness of $g(o) \cap \mathcal{O}$ we must have $g(o) = \{o\}$, and we are done. Therefore, let's suppose there exists $o' \neq o$ such that $o' \in g(o)$. Let's define an equivalence relation $\sim$ such that $[o] = g(o) \cup \{o\}$, and $[o''] = \{o''\}$ for any $o'' \notin g(o) \cup \{o\}$. This is a nontrivial equivalence relation because $o' \neq o$ and yet $o' \sim o$. We shall prove that g and $\sim$ are compatible, and hence $\mathcal{S} = (\mathcal{O}, g)$ is reducible. To prove compatibility, it suffices to prove that $(\mu \circ g)(o') = (\mu \circ g)(o)$ for any $o' \in g(o)$. By imposture-proofness, we have

$$g(o') \subseteq g(o) \subseteq g(o) \cup \{o\} = [o],$$

and hence $(\mu \circ g)(o') = \{[o]\} = (\mu \circ g)(o)$ as claimed. ■

In the rest of this section, we shall categorize different imposture-proof open source licenses. This categorization, however, will not be used in Section 6. Readers who are eager to learn why GPL and BSD stood out from other licenses as the two most natural choices for the first-generation open source developers can skip the rest of this section and jump to Section 6 without loss.

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 $\mathcal{S} = (\mathcal{O}, g)$ 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 Section 3, we have

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

Lemma 4 *Let $\mathcal{S} = (\mathcal{O}, g)$ be an 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, $\forall o' \in g(o) \cap \mathcal{O}$, $\mathbf{P} \notin g(o')$, 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* $\mathbf{L}$ 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 either go proprietary (i.e., choosing $\mathbf{P}$), or go open source with either $\mathbf{G}$, $\mathbf{1}$, $\mathbf{2}$, $\ldots$, **L-2**, or **L-1**.

Theorem 4 *Let $\mathcal{S} = (\mathcal{O}, g)$ be an 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* $\mathbf{L}$ *in the sense that $\exists\, \mathbf{G}, \mathbf{1}, \mathbf{2} \ldots, \mathbf{L-1} \in \mathcal{O}$, with $g(\mathbf{G}) = \{\mathbf{G}\}$, $g(\mathbf{1}) = \{\mathbf{P}, \mathbf{G}\}$, $g(\mathbf{2}) = \{\mathbf{P}, \mathbf{G}, \mathbf{1}\}$, $\ldots$, and $g(\mathbf{L-1}) = \{\mathbf{P}, \mathbf{G}, \mathbf{1}, \mathbf{2}, \ldots, \mathbf{L-2}\}$, such that $g(o) = \{\mathbf{P}, \mathbf{G}, \mathbf{1}, \mathbf{2}, \ldots, \mathbf{L-1}\}$.*

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 irreducibility, unique) $\mathbf{G}, \mathbf{1}, \mathbf{2} \ldots, \mathbf{m} \in \mathcal{O}$, with $g(\mathbf{G}) = \{\mathbf{G}\}$, $g(\mathbf{1}) = \{\mathbf{P}, \mathbf{G}\}, \ldots$, and $g(\mathbf{m}) = \{\mathbf{P}, \mathbf{G}, \mathbf{1}, \ldots, \mathbf{m}-\mathbf{1}\}$, 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 Section 3, we have

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

Lemma 5 *Let $\mathcal{S} = (\mathcal{O}, g)$ be an 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 ∞.*

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 $g(\mathbf{G}) = \{\mathbf{G}\}$, $g(\mathbf{1}) = \{\mathbf{P}, \mathbf{G}\}$, $g(\mathbf{2}) = \{\mathbf{P}, \mathbf{G}, \mathbf{1}\}, \ldots$, and $g(\mathbf{L}) = \{\mathbf{P}, \mathbf{G}, \mathbf{1}, \mathbf{2}, \ldots, \mathbf{L}-\mathbf{1}\}$, 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 an 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: Since $L(o) = \infty$, we have $\mathbf{P} \in g(o)$. If $o' \notin g(o)$ for any $o' \neq o$, then by the nonemptyness of $g(o) \cap \mathcal{O}$ we must have $g(o) = \{\mathbf{P}, o\}$, and we are done. Therefore, let's suppose there exists $o' \neq o$ such that $o' \in g(o) \cap \mathcal{O}$. Let's define an equivalence relation $\sim$ such that $[o] = (g(o) \cap \mathcal{O}) \cup \{o\}$, and $[o''] = \{o''\}$ for any $o'' \notin (g(o) \cap \mathcal{O}) \cup \{o\}$. This is a nontrivial equivalence relation because $o' \neq o$ and yet $o' \sim o$. We shall prove that g and $\sim$ are compatible, and hence $\mathcal{S} = (\mathcal{O}, g)$ is reducible. To prove compatibility, it suffices to prove that $(\mu \circ g)(o') = (\mu \circ g)(o)$ for any $o' \in g(o) \cap \mathcal{O}$. Since $l(o) = \infty$, we have $L(o') = \infty$, and hence $[\mathbf{P}]$ is contained in $(\mu \circ g)(o)$ as well as in $(\mu \circ g)(o')$. By imposture-proofness, we have

$$g(o') \cap \mathcal{O} \subseteq g(o) \cap \mathcal{O} \subseteq (g(o) \cap \mathcal{O}) \cup \{o\} = [o],$$

and hence $(\mu \circ g)(o') = \{[\mathbf{P}], [o]\} = (\mu \circ g)(o)$ 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.[26] 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 probability law $P(\cdot\,|\,\cdot)$.

[26]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 BSD.

6 Why GPL and BSD are Natural Choices for Developers

In this section, we return to the (possibly non-stationary) setting in Section 2 and consider all subgame perfect equilibria of the game (instead of restricting our attention to pure-strategy Markov perfect equilibrium). We shall present an environment where, in any subgame perfect equilibrium, if developer 0 is ever going to go open source, he cannot do better than going open source with either GPL or BSD. This result is of interest because it sheds light on why GPL and BSD stood out from other licenses as the two most natural choices for the first-generation open source developers.

Theorem 6 *Assume (i) exponential discounting and (ii) that the probability law $P(\cdot\,|\,\cdot)$ is atomless. Let $\mathcal{S} = (\mathcal{O}, g)$ be an 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 subgame perfect equilibrium, if developer 0 is ever going to go open source, he cannot do better than going open source using either $\mathbf{G}$ or $\mathbf{B}$.*

Proof: We have already seen in Subsection 2.1 that either $\mathbf{G}$ or $\mathbf{B}$ can be strictly optimal for developer 0 if these are the only two open source licenses available to him.[27] Since $\mathcal{S} = (\mathcal{O}, g)$ is irreducible and imposture-proof, by Theorem 3, any open source license $o \in \mathcal{O}$ other than $\mathbf{G}$ must have the property that $\mathbf{P} \in g(o)$. Therefore, it suffices to prove that, in any subgame perfect equilibrium, choosing $\mathbf{B}$ is weakly better than choosing any of these open source licenses for developer 0.

For any open source license $o \in \mathcal{O}$ such that $\mathbf{P} \in g(o)$. Suppose developer 0 goes open source using license o. For any $o_1 \in g(o) \cap \mathcal{O}$, let $v_1(o_1)$ be developer 1's gross expected payoff (gross of development cost c_1) if he goes open source with o_1, where the expectation is taken over the realizations of $\{\theta_t\}_{t \geq 2}$, and is taken conditional on the equilibrium strategies of developers $t \geq 2$. Note that $v_1(o_1) = W_1 + \beta \times (\cdots) \geq W_1$. Let $v_1^* = \sup_{o_1 \in g(o) \cap \mathcal{O}} v_1(o_1)$. Developer 1's optimal strategy is hence

1. not to develop software 1 if $\theta_1 \in E^0 := \{\theta_1 \,|\, c_1 > \max\{\pi_1 + w_1, v_1^*\}\}$;
2. to develop software 1 and go proprietary if $\theta_1 \in E^P := \{\theta_1 \,|\, \pi_1 + w_1 > \max\{c_1, v_1^*\}\}$; and

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

3. to develop software 1 and go open source with one of the open source licenses in $argmax_{o_1 \in g(o) \cap \mathcal{O}} v_1(o_1)$ if $\theta_1 \in E^* := \{\theta_1 \,|\, v_1^* > \max\{c_1, \pi_1 + w_1\}\}$.[28]

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

$$W_0 + \beta \left[P(E^0 \,|\, \theta_0) \times 0 + P(E^P \,|\, \theta_0) \times w_1 + P(E^* \,|\, \theta_0) \times v_1^* \right],$$

which is strictly increasing in v_1^* because $v_1^* \geq W_1 > w_1$. Since v_1^* is weakly increasing in the set $g(o) \cap \mathcal{O}$ (according to the order of set inclusion), it is maximized at **B**. This proves that, in any subgame perfect equilibrium, choosing **B** is weakly better than choosing any open source license $o \in \mathcal{O}$ such that $\mathbf{P} \in g(o)$. ∎

In the proof of Theorem 6, the assumption of an atomless probability law $P(\cdot \,|\, \cdot)$ 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 BSD 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

[28] The sup v_1^* is attainable in this last case because an optimal strategy exists in a subgame perfect equilibrium.

licenses that made this revolution possible. This paper is the first attempt to provide a rich enough model to accommodate infinitely many generations of developers so that the self-referential features of these licenses can be studied explicitly.

To make such a rich model tractable, we have imposed a linear structure on the community of developers (i.e., each developer, if he goes open source, can inspire at most one developer). It should be emphasized that our construction of the universal space of open source licenses (Section 4) and the categorization of imposture-proof open source licenses (Section 5) do *not* depend on this linear structure.

The sufficiency of GPL and BSD (Theorem 6), however, may rely on this linear structure. Imagine, for example, an alternative, "tree-like" structure where developer 0 can inspire up to two second-generation developers, A and B. In the absence of developer B, developer 0 finds it optimal to give developer A as much freedom as possible in designing A's own open source license. In the presence of developer B, however, this may no longer be true, thanks to the strategic interaction between A and B.

While it is certainly more realistic to replace our linear structure with a "tree-like" structure, the cost of doing so is a discontinuous jump in complexity, as a model with a "tree-like" structure has a lot more free parameters. For starter, softwares A and B are likely close substitutes of each other, and their substitutability likely depends on whether one or both of them are proprietary. Similarly, if developers A and B can each inspire up to two third-generation developers—A inspiring A1 and A2, and B inspiring B1 and B2—then the substitutability between softwares A1 and B2 is likely lower than that between A1 and A2, and the whole matrix of these substitutabilities likely depends on which of these softwares are proprietary. All these parameters may play a role in determining developer 0's optimal open source license. A model with a "tree-like" structure also opens up the possibility of dual licensing—the practice of offering different licenses to different downstream developers.

Such a more realistic model will be closer to that of Gaudeul (2004, 2005) than to ours. To keep this more realistic model tractable, she limits her attention to only GPL and BSD, and to only three generations of developers. The cost of doing so is that other open source licenses, together with their self-referential features, cannot be studied explicitly.

Finally, when the sufficiency of GPL and BSD (Theorem 6) breaks down (say, in a model with a "tree-like" structure), the reader will find other open source licenses studied 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 linear structure.

References

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

[2] Balter, Ben (2015), "Open Source License Usage on GitHub.com." https://github.blog/2015-03-09-open-source-license-usage-on-github-com/ (last accessed on November 23, 2022)

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

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

[5] 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.

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

[7] Fershtman, Chaim and Neil Gandal (2011), "A Brief Survey of the Economics of Open Source Software." In Palgrave Macmillan (ed.), *The New Palgrave Dictionary of Economics*, online edition.

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

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

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

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

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

[13] 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.

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

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

[16] Lerner, Josh, Parag A. Pathak, and Jean Tirole (2006), "The Dynamics of Open-Source Contributors." *American Economic Review*, 96(2): 114-118.

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

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

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

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

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

[22] 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.

[23] Raymond, Eric S. (1999), *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly.

[24] 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.

[25] Smith, P. McCoy (2022), "Copyright, Contract, and Licensing in Open Source." In Amanda Brock (ed.), *Open Source Law, Policy and Practice*, Oxford University Press.

[26] Vendome, Christopher, Gabriele Bavota, Massimiliano Di Penta, Mario Linares-Vásquez, Daniel Germán, and Denys Poshyvanyk (2017), "License Usage and Changes: A Large-Scale Study on GitHub." *Empirical Software Engineering*, 22: 1537-1577.

[27] 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.