A Tool to Define Step by Step the Compatibility of Licenses as Partial Orders with CaLi

Benjamin Moreau[1,2], Bastien Confais[1], Patricia Serrano-Alvarado[1], and Damien Vintache[1]

[1] Nantes University, LS2N, CNRS, UMR6004, 44000 Nantes, France
{Name.LastName@}univ-nantes.fr
[2] OpenDataSoft {Name.Lastname}@opendatasoft.com

Abstract. Licenses specify precisely the conditions of reuse of resources, i.e., what actions are permitted, obliged, and prohibited when using resources. Knowing the compatibility of a license allows knowing to what extent the protected resource is reusable. CaLi is a lattice-based model that partially orders licenses in terms of compatibility and compliance. It uses restrictiveness relations that are refined with constraints to take into account the semantics of actions existing in licenses. In this demonstration, we propose an online Web tool that interactively shows how to define the compatibility of licenses as partial orders with the CaLi model.

1 Introduction and motivation

Before sharing or publishing their resources, producers should systematically associate them with licenses. Licenses specify precisely the conditions of reuse of resources, i.e., what actions are *permitted*, *obliged*, and *prohibited* when using the resource. Examples of well-known licenses are the family of Creative Commons licenses[3], which are widely used to protect creative works. There are licenses like Open License, ODbL, etc., to protect open data. Licenses like GPL, Apache, EPL, MIT, etc. are frequently used to protect source code. Not to mention the hundreds of ad-hoc licenses defined by data producers.

Relations of compatibility, compliance, and restrictiveness on licenses could be very beneficial in a wide range of applications. For instance, platforms of data storage, repositories of source code, or any enterprise managing different licenses for their resources, could beneficiate of a compatibility order of their licenses. Knowing the compatibility of a license allows knowing to what extent the protected resource is reusable.

We consider that *a license l_j is compliant with a license l_i if a resource licensed under l_i can be licensed under l_j without violating l_i*. If a license l_j is compliant with l_i then we consider that l_i is compatible with l_j and that resources licensed under l_i are reusable with resources licensed under l_j. Usually but not always, when l_i is less restrictive than l_j then l_i is compatible with l_j.

[3] https://creativecommons.org/licenses/

In [1], we propose a model to order licenses in terms of compatibility and compliance automatically. In [3], we show the usability of CaLi with a prototype of a search engine based on a CaLi ordering of licenses.[4] Such a search engine can find resources whose licenses are compatible or compliant with a specific license. In this demonstration, we propose a Web tool that interactively shows how to define a CaLi ordering of licenses which can be used by an application that needs to verify the compatibility of licences. Such ordering can be downloaded in RDF.

The next section overviews the CaLi model, and Section 3 describes how to define CaLi orderings with our Web tool.

2 Modelling the compatibility of licenses

CaLi passes through a restrictiveness relation to partially order licenses in terms of compatibility and compliance. In a license, actions can be distributed in what we call *status*, e.g., permissions, obligations, and prohibitions. To decide if a license l_i is less restrictive than l_j, it is necessary to know if an action in a status is considered less restrictive than the same action in another status.

We remark that if two licenses have a restrictiveness relation, then they may have a compatibility relation too. The restrictiveness relation between licenses can be obtained automatically, according to the status of actions without taking into account the semantics of the actions. Thus, based on lattice-ordered sets [2], we define a restrictiveness relation among licenses.

To identify the compatibility among licenses, we refine the restrictiveness relation with constraints. The goal is to take into account the semantics of actions. Constraints also distinguish valid licenses from non-valid ones. We consider a license l_i as non-valid if a resource can not be licensed under l_i, e.g., an inconsistent license that simultaneously permits the *Derive*[5] action and prohibits *DerivativeWorks*[6].

A CaLi ordering is a tuple $\langle \mathcal{A}, \mathcal{LS}, C_{\mathcal{L}}, C_{\rightarrow} \rangle$, such that [1]:

1. $\mathcal{A}$ is a set of *actions* (e.g., *read, modify, distribute*, etc.);
2. $\mathcal{LS}$ is a *restrictiveness lattice of status* that defines (i) all possible status of an action (i.e., permission, obligation, prohibition, recommendation, undefined, etc.) and (ii) the restrictiveness relation among status, denoted by $\leqslant_S$;
3. $C_{\mathcal{L}}$ is a set of *license constraints* to identify non-valid licenses; and
4. $C_{\rightarrow}$ is a set of *compatibility constraints* to identify if a restrictiveness relation between two licenses is also a compatibility relation.

In CaLi, $\mathcal{L}_{\mathcal{A},\mathcal{LS}}$ defines the set of all licenses that can be expressed with $\mathcal{A}$ and $\mathcal{LS}$. $(\mathcal{L}_{\mathcal{A},\mathcal{LS}}, \leqslant_{\mathcal{R}})$ is the restrictiveness lattice of licenses that defines the restrictiveness relation $\leqslant_{\mathcal{R}}$ over the set of all licenses $\mathcal{L}_{\mathcal{A},\mathcal{LS}}$. We say that l_i is less restrictive than l_j, denoted $l_i \leqslant_{\mathcal{R}} l_j$, if for all actions $a \in \mathcal{A}$, the status of a in

[4] `http://cali.priloo.univ-nantes.fr`
[5] `https://www.w3.org/TR/odrl-vocab/#term-derive`
[6] `https://www.w3.org/TR/odrl-vocab/#term-DerivativeWorks`

l_i is less restrictive than the status of a in l_j. That is, $l_i \leqslant_{\mathcal{R}} l_j$ if $\forall a \in \mathcal{A}, l_i(a) \leqslant_S l_j(a)$. If two valid licenses have a restrictiveness relation, then they may have a a compatibility relation too. For two licenses $l_i \leqslant_{\mathcal{R}} l_j \in \mathcal{L}_{\mathcal{A},\mathcal{LS}}$, we say that l_i *is compatible with* l_j, denoted by $l_i \rightarrow l_j$, if $\forall \omega_{\mathcal{L}} \in C_{\mathcal{L}}, \omega_{\mathcal{L}}(l_i) = \omega_{\mathcal{L}}(l_j) = True$ and $\forall \omega_{\rightarrow} \in C_{\rightarrow}, \omega_{\rightarrow}(l_i, l_j) = True$.

Notice that contrary to the restrictiveness relation, the compatibility relation is a partial order but not necessarily a lattice-based partial order.

With CaLi it is possible to answer the question, *given a license l_i, how to automatically position l_i over a set of licenses in terms of compatibility and compliance?* Knowing the compatibility of a license allows knowing to what extent the protected resource is reusable. On the other hand, knowing the compliance of a license allows knowing to what extent other licensed resources can be reused.

3 Demonstration

Our Web tool to interactively define the compatibility of licenses with the CaLi model is available at `https://saas.ls2n.fr/cali/`. In the next, we show how to construct a CaLi ordering step by step.

Set of actions $\mathcal{A}$. Our tool uses the set of actions considered by ODRL[7]. ODRL proposes 72 actions, including actions used in Creative Commons licenses. When defining a license, users can distribute all these actions among defined status. By default, actions not distributed in a status, are assigned to the *Undefined* status.

Restrictiveness lattice of status $\mathcal{LS}$. A CaLi ordering is based on a restrictiveness lattice of status. That is, every pair of licenses has a least and a most restrictive license. The most restrictive license is the *supremum* and the least restrictive one is the *infimum*. Users define a restrictiveness lattice of status in two steps. First, the set of possible status for actions in a license should be chosen.

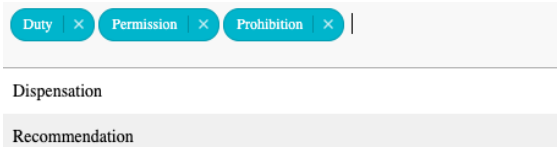


Fig. 1: Choosing the possible status for actions in a license.

Our tool proposes three status, *Permission, Prohibition*, and *Duty*, but *Recommendation* or *Dispensation* can be included, see Figure 1. The *Undefined* status is always present; it will contain all actions not distributed in another status. Second, once the set of status is saved, users should define the restrictiveness

[7] https://www.w3.org/TR/odrl-vocab/#actionConcepts

lattice of status. The *Undefined* status should be the least restrictive status, i.e., the *infimum* of any restrictiveness lattice of status. Figure 2 shows an example of restrictiveness lattice of status.

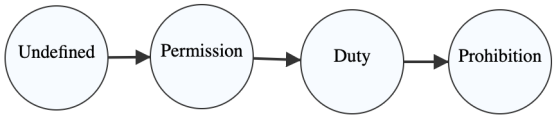


Fig. 2: Defining the restrictiveness lattice of status.

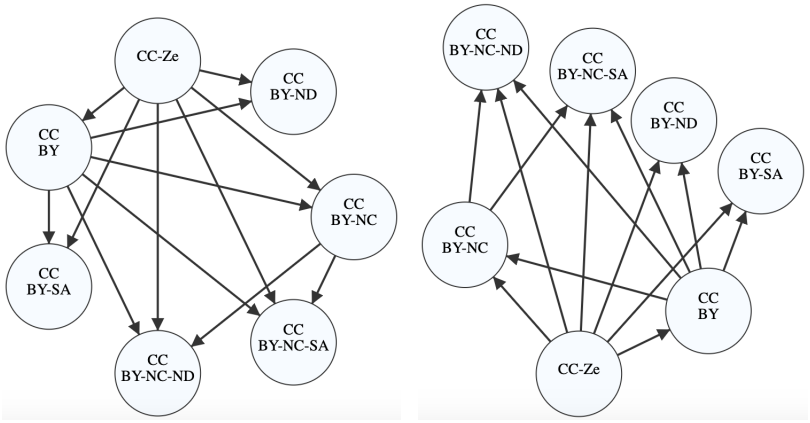
License constraints $C_{\mathcal{L}}$. A user can define several license constraints ($\omega_{\mathcal{L}}$). A license is valid if all license constraints are verified. A constraint can be defined as a set of conditions. Inside a constraint, if one condition is true, then the constraint is true. A condition of a license constraint defines if an action should exist or not in a status. For instance the condition (*cc:CommercialUse* $\notin$ *Duty*) means that a valid license should not have the action *cc:CommercialUse* as a *Duty*. The condition (*cc:ShareAlike* $\notin$ *Prohibition*) means that a valid license should not prohibit the *cc:ShareAlike* action. Both constraints are proposed by our tool. Conjunctive constraints should be defined in several constrains. For instance, the license constraint (*read* $\in$ *Prohibition* AND *modify* $\in$ *Permission*) has two conditions. To verify both conditions they should be defined in two constraints, one condition per constraint.

Compatibility constraints $C_{\rightarrow}$. To verify if a restrictiveness relation between two licenses is also a compatibility relation, users can define compatibility constraints ($\omega_{\rightarrow}$). A compatibility constraint concerns two licenses where one is more restrictive than another ($l_i \leqslant_{\mathcal{R}} l_j$).
For instance, consider the action *cc:ShareAlike* which requires that the distribution of derivative works be under the same license only. The compatibility constraint (*cc:ShareAlike* $\notin$ *Duty*) $\notin l_i$ means that l_i is compatible with l_j if the action *cc:ShareAlike* is not a Duty in l_i. In another example, the compatibility constraint (*cc:DerivativeWorks* $\notin$ *Prohibition*) $\notin l_i$ means that l_i is compatible with l_j if l_i does not prohibit the distribution of a derivative resource, regardless of the license. These two constraints are proposed by our tool.

Creation of licenses. The Web tool gives users the possibility of distributing actions into status. The same action can not be attributed to more than one status. Our tool proposes several licenses like these of Creative Commons[8] if the restrictiveness lattice of status is (*Undefined* $\leqslant_S$ *Permission* $\leqslant_S$ *Duty* $\leqslant_S$ *Prohibition*), cf. Figure 2.

[8] CC Zero, CC BY, CC BY-NC, CC BY-ND, CC BY-SA, CC BY-NC-ND and CC BY-NC-SA.

Compatibility partial order of licenses. Finally, the user can define the compatibility order of licenses. The Web tool generates an acyclic graph that represents the compatibility order of licenses.
Figure 3(a) shows an example of partial order as returned by our tool. Users can manipulate this graph to facilitate their reading, as shown in Figure 3(b). Users can download the generated partial order and concerned licenses in the RDF format.



(a) (b)

Fig. 3: Example of the graphical representation of the generated compatibility order of licenses.

An improvement for this prototype is to allow users to introduce a CaLi ordering in RDF so that they can add other licenses.

This demonstrator is based on the python package pycali[9] that implements several CaLi functions.

References

1. Benjamin, M., Serrano-Alvarado, P., Perrin, M., Desmontils, E.: Modelling the Compatibility of Licenses. In: Extended Semantic Web Conference (ESWC) (2019)
2. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order. Cambridge university press (2002)
3. Moreau, B., Serrano-Alvarado, P., Perrin, M., Desmontils, E.: A License-Based Search Engine. In: Extended Semantic Web Conference (ESWC), Poster&Demo (2019)

[9] `https://pypi.org/project/pycali/`