# Fast Compliance Checking in an OWL2 Fragment*

**Piero A. Bonatti**
Università degli studi di Napoli Federico II
piero.bonatti@unina.it

## Abstract

We illustrate a formalization of data usage policies in a fragment of OWL2. It can be used to encode a company's data protection policy, as well as data subjects' consent to data processing, and part of the GDPR (the forthcoming European Data Protection Regulation). Then a company's policy can be checked for compliance with data subjects' consent and with part of the GDPR by means of subsumption queries. We provide a complete and tractable structural subsumption algorithm for compliance checking and prove the intractability of a natural generalization of the policy language.

## 1 Introduction

This work stems from the EU H2020 project XYZ[1], where semantic technologies are used to help companies in complying with the new European General Data Protection Regulation (GDPR).[2] In this project, data usage policies are encoded using a fragment of OWL2-DL and the main policy-related reasoning tasks are reduced to subsumption and concept consistency checking. Such tasks include - among others:

- *permission checking*: given an operation request, decide whether it is permitted;
- *compliance checking*: does a policy $P_1$ fulfil all the restrictions requested by policy $P_2$? (Policy comparison);
- *policy validation*: e.g. is the policy contradictory? Does a policy update strengthen or relax the previous policy?

Compliance checking is the predominant task in this project: the data usage policies of the industrial partners must be compared both with a (partial) formalization of the GDPR itself, and with the consent to the usage of personal data granted by each of the *data subjects* whose data are collected and processed by the company (that is called *data controller* in the GDPR). The number of data subjects (and their policies) can be as large as the number of customers of a major communication service provider. Moreover, in the absence of explicit consent, some data cannot be stored, even temporarily; so some of the project's use cases consist in checking storage permissions against a stream of incoming data points, at the rate of hundreds of thousands per minute. Then one of the crucial project tasks is the development of scalable reasoning procedures for reasoning in the policy fragment of OWL 2.

In this paper we illustrate this fragment and introduce a structural subsumption algorithm that is a promising starting point for scalable compliance checking. Sec. 2 recalls the basics of Description Logics (DL) needed in this work. In Sec. 3 we illustrate the encoding of data usage policies in OWL2. In Sec. 4 we describe the structural subsumption algorithm for the policy language, and a policy consistency checking method. We prove correctness and completeness of these algorithms and their tractability. Moreover, we prove the intractability of a slight generalization of the policy fragment. The paper is concluded by a discussion of the results, a comparison with related work, and a description of ongoing and future work. Some proofs have been moved to Appendix A to enhance readability.

## 2 Preliminaries on Description Logics

Here we report the basics needed for our work and refer the reader to [Baader *et al.*, 2003] for further details. The DL languages of our interest are built from countably infinite sets of concept names ($\mathsf{N_C}$), role names ($\mathsf{N_R}$), concrete feature names ($\mathsf{N_F}$), and concrete predicates ($\mathsf{N_P}$). An *interpretation* $\mathcal{I}$ is a structure $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a set, and the *interpretation function* $\cdot^{\mathcal{I}}$ is such that (i) $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ for all $A \in \mathsf{N_C}$; (ii) $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for all $R \in \mathsf{N_R}$; (iii) $f^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathsf{D}}$ for all $f \in \mathsf{N_F}$.[3] The semantics of an $n$-ary predicate $p \in \mathsf{N_P}$ is a set of tuples $p^{\mathsf{D}} \subseteq (\Delta^{\mathsf{D}})^n$. In this paper we use $\Delta^{\mathsf{D}} = \mathbb{N}$ and unary concrete predicates $\mathtt{in}_{\ell,u}$, where $\ell, u \in \mathbb{N}$, such that $\mathtt{in}^{\mathsf{D}}_{\ell,u} = [\ell, u]$. To enhance readability we will abbreviate $\mathtt{in}_{\ell,u}(f)$ to $[\ell, u](f)$. Informally, $[\ell, u](f)$ means that the value of feature $f$ belongs to the interval $[\ell, u]$.

Figure 1 shows some DL operators and their semantics, that extends $\cdot^{\mathcal{I}}$ to compound DL expressions. The last four

*This research is funded by the European Unions Horizon 2020 research and innovation programme under grant agreement N. ⟨anonymized for blind review⟩.

[1] Anonymized for blind review

[2] http://data.consilium.europa.eu/doc/document/ST-5419-2016-INIT/en/pdf

[3] $\Delta^{\mathsf{D}}$ denotes the domain of the predicates in $\mathsf{NP}$. We are assuming – for brevity – that there is one concrete domain. However, our framework can be immediately extended to multiple domains.

| Name | Syntax | Semantics |
|---|---|---|
| bottom | $\bot$ | $\bot^{\mathcal{I}} = \emptyset$ |
| intersection | $C \sqcap D$ | $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$ |
| union | $C \sqcup D$ | $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$ |
| restriction | $\exists R.C$ | $\{d \in \Delta^{\mathcal{I}} \mid \exists (d,e) \in R^{\mathcal{I}} : e \in C^{\mathcal{I}}\}$ |
| complement | $\neg C$ | $(\neg D)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ |
| concrete constraints | $p(f_1, .., f_n)$ | $\{x \in \Delta^{\mathcal{I}} \mid \exists \vec{v} \in (\Delta^{\mathsf{D}})^n . (x, v_i) \in f_i^{\mathcal{I}}$ $(1 \leq i \leq n)$ and $\vec{v} \in p^{\mathsf{D}}\}$ |
| GCI | $C \sqsubseteq D$ | $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ |
| disjointness | $\mathsf{disj}(C, D)$ | $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ |
| func | $\mathsf{func}(R)$ | $R^{\mathcal{I}}$ is a partial function |
| range | $\mathsf{range}(R, C)$ | $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C^{\mathcal{I}}$ |

Figure 1: Syntax and semantics of some DL constructs and axioms.

lines illustrate some DL axioms; GCI stands for "general concept inclusion". An interpretation $\mathcal{I}$ *satisfies* an axiom $\alpha$ (equivalently, $\mathcal{I}$ is a *model* of $\alpha$) if $\mathcal{I}$ satisfies the corresponding semantic condition in Fig. 1.

A *knowledge base* $\mathcal{K}$ is a finite set of DL axioms. An interpretation $\mathcal{I}$ is a *model* of $\mathcal{K}$ ($\mathcal{I} \models \mathcal{K}$) if $\mathcal{I}$ satisfies all the axioms in $\mathcal{K}$. We say that "$\mathcal{K}$ *entails* an axiom $\alpha$" – in symbols, $\mathcal{K} \models \alpha$ – if all models of $\mathcal{K}$ satisfy $\alpha$.

A *pointed interpretation* is a pair $(\mathcal{I}, d)$ where $d \in \Delta^{\mathcal{I}}$. We say $(\mathcal{I}, d)$ *satisfies* a concept $C$ (in symbols, $(\mathcal{I}, d) \models C$) iff $d \in C^{\mathcal{I}}$.

## 3 Encoding Usage Policies in OWL2-DL

A logic-based policy language has been chosen because *policies are knowledge*. Policies encode declarative constraints on a system's behavior, that depend on metadata about *subjects* (e.g. users), *objects* (e.g. system resources), *actions* (operations), and an *environment*. Such metadata models – possibly in "semantic" terms – all the categories and attributes of the above entities that are relevant to make decisions about permissions. Moreover, like knowledge and unlike programs, every single policy is meant to be used for multiple, semantically related *tasks*, such as those listed in Sec. 1.

Then, knowledge representation (KR) languages are ideal policy representation languages. Indeed, both rule languages and description logics (DL) have already been used as policy languages, a non-exhaustive list is [Woo and Lam, 1993; Uszok *et al.*, 2003; Kagal *et al.*, 2003; Bonatti *et al.*, 2010]. As noted in [Bonatti, 2010], the advantage of rule languages is that they can express $n$-ary authorization conditions for arbitrary $n$, while encoding such conditions for $n > 2$ is challenging in DL. The advantage of DL is that all the main policy-reasoning tasks are decidable (and tractable if policies can be expressed with OWL 2 profiles), while compliance checking is undecidable in rule languages, or at least intractable, in the absence of recursion. So a DL-based policy language is a natural choice in a project where policy comparison is the predominant task.

In abstract terms, a policy is a set of tuples called *authorizations*. In traditional access control, authorizations are triples $\langle s, o, a \rangle$ whose intuitive meaning is: "subject $s$ can do action $a$ on object $o$" [Bonatti *et al.*, 2002]. In privacy and data usage policies, $o$ is a piece of data (typically encoding information about a data subject) and tuples contain further elements such as the *purpose* of the operation, the *recipients* of the results, the *duration* of storage, and so on.

Authorizations are encoded in OWL 2 by *reifying* them, i.e. representing tuples as individuals whose attributes are the tuple's elements. For example, if $S, O, A \in \mathsf{N_C}$, $\mathcal{I}$ is an interpretation, and $S^{\mathcal{I}} = \{s\}$, $O^{\mathcal{I}} = \{o\}$, $A^{\mathcal{I}} = \{a\}$, then all the instances of the following concept in $\mathcal{I}$ encode $\langle s, o, a \rangle$:[4]

$$\exists \mathtt{subj}.S \sqcap \exists \mathtt{obj}.O \sqcap \exists \mathtt{act}.A\,. \tag{1}$$

In order to make the above concept represent *only* the authorizations in the cross product of classes $S, O$, and $A$, we generally make the roles that encode tuple elements *functional*. This is probably the easiest way to prevent some likely-to-happen modelling errors.

**Example 1** Suppose that role `subj` is not functional. If the informal policy: "staff members can read confidential files" were encoded in the natural way, with the concept:

$$\exists \mathtt{subj}.\mathtt{Staff} \sqcap \exists \mathtt{obj}.\mathtt{Confidential} \sqcap \exists \mathtt{act}.\mathtt{Read}\,, \tag{2}$$

then the formalized policy may authorize also *visitors* to read confidential files, even if classes `Staff` and `Visitor` were disjoint. In particular, (2) may have an instance $x$ with two values for `subj`: a staff member $s$ and a visitor $v$. If this is not considered as an error, then the only sensible approach is taking $x$ as the representation of the two tuples with either $s$ or $v$ as a subject, and with the same object and action as $x$. But then (2) would authorize visitor $v$ to read confidential files. Instances like $x$ cause problems also in modelling the policy difference operator [Bonatti *et al.*, 2002], because deletion of authorizations for visitors (like $\langle v, o, a \rangle$) may delete also some of the authorizations for staff members (e.g. $\langle s, o, a \rangle$, that is reified by the same individual as $\langle v, o, a \rangle$). ∎

Some authorization attributes are optional, e.g. there may be no *storage* if the results of data processing are not saved in a persistent memory. Then the corresponding roles are *partial* functions, in general. There are a few exceptions to functionality. Some tuple elements are actually *sets*, e.g. a single authorization may be associated to any number of *obligations*, as in "*s can execute a on o if explicit consent is obtained from the data subjects* and *s signes a non-disclosure agreement*".

The attributes of authorizations (like data categories, purposes, etc.) range over concepts defined in simple taxonomies, derived from P3P (the Platform for Privacy Preferences)[5] and ODRL (the Open Digital Right Language).[6] Currently, such taxonomies are encoded with simple inclusions $A \sqsubseteq B$ and disjointness constraints $\mathsf{disj}(A, B)$, where $A, B$ are concept *names*.[7] Only one attribute (`dur`, for storage duration) is a concrete feature, that can be restricted to integer intervals $[l, u]$ with concrete constraints $[l, u](\mathtt{dur})$.

[4] In this paper we use the DL syntax because it is way more compact than any syntax for OWL 2.

[5] `http://www.w3.org/TR/P3P11`

[6] `https://www.w3.org/TR/odrl/`

[7] In a complementary paper we show how to support general $\mathcal{EL}$ concept inclusions by integrating the structural subsumption algorithm presented here with ELK.

Let a "simple policy concept" $P$ be an extension of (1) with additional conjuncts for purpose, storage duration (optional), obligations (non functional), etc. A single, simple policy concept $P$ cannot express different usage restrictions for different categories of entities. So we will consider *full* policy concepts, that are unions of simple policy concepts. For example, the full policy concept $P_1 \sqcup P_2$, where

$$P_1 = \exists \texttt{subj.Staff} \sqcap \exists \texttt{obj.PII} \sqcap \exists \texttt{act.Read}\,,$$

$$P_2 = \exists \texttt{subj.Other} \sqcap \exists \texttt{obj.Anonymous} \sqcap \exists \texttt{act.Read}\,.$$

represents the policy: "*Staff can read personally identifiable information, while third parties can read anonymized data*".

The main reasoning tasks are policy comparisons, that is, subsumption checks $P_1 \sqsubseteq P_2$ over full policy concepts $P_i$.

Based on the above discussion, we are now ready to specify a fragment of OWL 2 called $\mathcal{PL}$ (*policy logic*) that covers – and slightly generalizes – the language outlined above.

**Definition 1 (Policy logic $\mathcal{PL}$)** *A $\mathcal{PL}$* knowledge base $\mathcal{K}$ *is a set of axioms of the following kinds:*

- $\mathsf{func}(R)$ *where $R$ is a role name or a concrete feature;*
- $\mathsf{range}(S, A)$ *where $S$ is a role and $A$ a concept name;*
- $A \sqsubseteq B$ *where $A, B$ are concept names;*
- $\mathsf{disj}(A, B)$ *where $A, B$ are concept names.*

*A* simple $\mathcal{PL}$ concept *has the form:*

$$A_1 \sqcap \ldots \sqcap A_n \sqcap E_1 \sqcap \ldots \sqcap E_m \sqcap C_1 \sqcap \ldots \sqcap C_t \qquad (3)$$

*where each $A_i$ is either a concept name or $\bot$, each $E_j$ is an existential restriction $\exists R.C$ such that $R$ is a role and $C$ a simple $\mathcal{PL}$ concept, and each $C_k$ is a concrete constraint $[l, u](f)$. A* (full) $\mathcal{PL}$ concept *is a union $D_1 \sqcup \ldots \sqcup D_n$ of simple $\mathcal{PL}$ concepts. $\mathcal{PL}$'s* subsumption queries *are expressions $C \sqsubseteq D$ where $C, D$ are (full) $\mathcal{PL}$ concepts.*

**Example 2** The policy "*Location data can be stored for at most 1 year in a server of the data controller located in France*" can be formalized with the following concept:

$$\exists \texttt{storage}.\big([0, 365](\texttt{dur}) \sqcap \exists \texttt{loc}.(\texttt{DCServer} \sqcap \texttt{FR})\big). \quad (4)$$

The knowledge base contains the following kinds of axioms: (i) $\mathsf{func}(R)$ for each role/feature $R$, (ii) range restrictions like $\mathsf{range}(\texttt{loc}, \texttt{AnyLoc})$, for each role, and (iii) inclusions between concept names, like $\texttt{DCServer} \sqsubseteq \texttt{AnyLoc}$ and $\texttt{FR} \sqsubseteq \texttt{AnyLoc}$, that define the vocabularies. The above policy can be regarded either as a data controller's usage policy, or as a data subject's consent to data processing. The complete policy/consent includes also the subject, the purpose, the recipients, and the obligations (if any).

The GDPR requires the data of European citizens to remain in the EU, or in countries that adopt similar data protection regulations. This restriction can be encoded as follows:

$$(\exists \texttt{storage}.\exists \texttt{loc.EU}) \sqcup (\exists \texttt{storage}.\exists \texttt{loc.EULike}). \quad (5)$$

This transfer restriction is satisfied by a policy $P$ if (5) subsumes $P$. For example, policy (4) can be proved to be compliant with GDPR's transfer restrictions provided that the vocabulary contains an inclusion $\texttt{FR} \sqsubseteq \texttt{EU}$. The vocabulary contains also a term referring to what the GDPR calls "*third countries*", i.e. any country that does not belong to the EU and does not provide sufficient data protection regulations. The three classes of countries are declared to be disjoint with axioms like $\mathsf{disj}(\texttt{EULike}, \texttt{ThirdCountries})$. ∎

# 4 Compliance Checking and its Complexity

In this section we introduce a structural subsumption algorithm for deciding whether $\mathcal{K} \models C \sqsubseteq D$, for all given $\mathcal{PL}$ KB $\mathcal{K}$ and all $\mathcal{PL}$ subsumptions $C \sqsubseteq D$. We start by introducing an auxiliary algorithm for *elementary* subsumptions, that are $\mathcal{PL}$ subsumptions $C \sqsubseteq D$ where both $C$ and $D$ are simple, and $C \sqsubseteq D$ is *interval safe*, that is, for all constraints $[\ell, u](f)$ and $[\ell', u'](f')$ occurring in $C$ and $D$, respectively, either $[\ell, u] \subseteq [\ell', u']$, or $[\ell, u] \cap [\ell', u'] = \emptyset$.

The structural subsumption algorithm (Algorithm 1, hereafter $\mathsf{STS}$) takes as input a $\mathcal{PL}$ KB $\mathcal{K}$ and an elementary $\mathcal{PL}$ subsumption $C \sqsubseteq D$, where $C$ is normalized w.r.t. $\mathcal{K}$ using the rewrite rules in Table 1. There, $\sqsubseteq^*$ denotes the reflexive and transitive closure of $\{(A, B) \mid (A \sqsubseteq B) \in \mathcal{K}\}$. By a straightforward case analysis we obtain Proposition 1:

**Algorithm 1**: $\mathsf{STS}(\mathcal{K}, C \sqsubseteq D)$

**Input**: $\mathcal{K}$ and an elementary $C \sqsubseteq D$ where $C$ is normalized
**Output**: `true` if $\mathcal{K} \models C \sqsubseteq D$, `false` otherwise
**Note:** By $C = C' \sqcap C''$ we mean that either $C = C'$ or $C'$ is a conjunct of $C$ (possibly not the 1st)

**begin**
  **if** $C = \bot$ **then return** `true`
  **if** $D = A$, $C = A' \sqcap C'$ *and* $A' \sqsubseteq^* A$ **then return** `true`
  **if** $D = [l, u](f)$ *and* $C = [l', u'](f) \sqcap C'$ *and* $l \leq l'$ *and* $u' \leq u$ **then return** `true`
  **if** $D = \exists R.D'$, $C = (\exists R.C') \sqcap C''$ *and* $\mathsf{STS}(\mathcal{K}, C' \sqsubseteq D')$ **then return** `true`
  **if** $D = D' \sqcap D''$, $\mathsf{STS}(\mathcal{K}, \mathcal{O}, C \sqsubseteq D')$, *and* $\mathsf{STS}(\mathcal{K}, \mathcal{O}, C \sqsubseteq D'')$ **then return** `true`
  **else return** `false`
**end**

**Proposition 1** *The rewrite rules in Table 1 preserve concept equivalence w.r.t. $\mathcal{K}$, i.e. if $C \rightsquigarrow C'$ then $\mathcal{K} \models C \equiv C'$.*

In order to prove that $\mathsf{STS}$ is complete w.r.t. elementary subsumptions we need a canonical counterexample to invalid subsumptions.

**Definition 2** *Let $C \neq \bot$ be a simple $\mathcal{PL}$ concept normalized w.r.t. $\mathcal{K}$. A* canonical model *of $C$ is a pointed interpretation $(\mathcal{I}, d)$ defined as follows, by recursion on the nesting level of existential restrictions. Hereafter we call a subconcept of $C$ "top level" if it does not occur within the scope of $\exists$.*

*a. If $C = \big(\prod_{i=1}^{n} A_i\big) \sqcap \big(\prod_{j=1}^{t} C_j\big)$ (i.e. $C$ has no existential restrictions), then let $\mathcal{I} = \langle \{d\}, \cdot^{\mathcal{I}} \rangle$ where*
- *$A^{\mathcal{I}} = \{d\}$ if for some $i = 1, \ldots, n$, $A_i \sqsubseteq^* A$;*
- *if $C_j = [l, u](f)$, add $(d, u)$ to $f^{\mathcal{I}}$ $(1 \leq j \leq t)$;*
- *all the other predicates are empty.*

*b. If the top-level existential restrictions of $C$ are $\exists R_i, D_i$ $(i = 1, \ldots, m)$, then let $(\mathcal{I}_i, d_i)$ be a canonical model of $D_i$, for each $i = 1, \ldots, m$. Assume w.l.o.g. that all such models are mutually disjoint and do not contain $d$ (if necessary, their elements can be replaced). Define an auxiliary interpretation $\mathcal{J}$:*
- *$\Delta^{\mathcal{J}} = \{d, d_1, \ldots, d_m\}$;*

| | | |
|---|---|---|
| 1) | $\bot \sqcap D \rightsquigarrow \bot$ | |
| 2) | $\exists R.\bot \rightsquigarrow \bot$ | |
| 3) | $[l,u](f) \rightsquigarrow \bot$ | if $l > u$ |
| 4) | $(\exists R.D) \sqcap (\exists R.D') \sqcap D'' \rightsquigarrow \exists R.(D \sqcap D') \sqcap D''$ | if $\mathsf{func}(R) \in \mathcal{K}$ |
| 5) | $[l_1,u_1](f) \sqcap [l_2,u_2](f) \sqcap D \rightsquigarrow [\max(l_1,l_2), \min(u_1,u_2)](f) \sqcap D$ | if $\mathsf{func}(f) \in \mathcal{K}$ |
| 6) | $\exists R.D \sqcap D' \rightsquigarrow \exists R.(D \sqcap A) \sqcap D'$ | if $\mathsf{range}(R,A) \in \mathcal{K}$ and $A$ not a conjunct of $D$ |
| 7) | $A_1 \sqcap A_2 \sqcap D \rightsquigarrow \bot$ | if $A_1 \sqsubseteq^* A_1'$, $A_2 \sqsubseteq^* A_2'$, and $\mathsf{disj}(A_1', A_2') \in \mathcal{K}$ |

Table 1: Normalization rules w.r.t. $\mathcal{K}$. Intersections are treated as sets (the ordering of conjuncts and their repetitions are irrelevant).

- *for all concept names $A$ such that for some top-level $A_i$ in $C$, $A_i \sqsubseteq^* A$, let $A^{\mathcal{J}} = \{d\}$; all other concept names are empty;*
- *for each top-level constraint $[l,u](f)$ in $C$, add $(d,u)$ to $f^{\mathcal{J}}$;*
- *for each top-level restriction $\exists R_i.D_i$ in $C$, add a pair $(d,d_i)$ to $R_i^{\mathcal{I}}$;*
- *there are no other pairs in roles and features.*

*Finally let $\mathcal{I}$ be the union of $\mathcal{J}$ and all $\mathcal{I}_i$, that is*

$$\begin{aligned} \Delta^{\mathcal{I}} &= \Delta^{\mathcal{J}} \cup \bigcup_i \Delta^{\mathcal{I}_i} \\ A^{\mathcal{I}} &= A^{\mathcal{J}} \cup \bigcup_i A^{\mathcal{I}_i} \quad (A \in \mathsf{N_C}) \\ R^{\mathcal{I}} &= R^{\mathcal{J}} \cup \bigcup_i R^{\mathcal{I}_i} \quad (R \in \mathsf{N_R} \cup \mathsf{N_F}) \,. \end{aligned}$$

*The canonical model is $(\mathcal{I}, d)$.*

Note that each $C$ has a unique canonical model up to isomorphism. The canonical model actually satisfies $\mathcal{K}$ and $C$:

**Lemma 1** *If $C$ is a simple $\mathcal{PL}$ concept normalized w.r.t. $\mathcal{K}$, and $C \neq \bot$, then the canonical model $(\mathcal{I}, d)$ of $C$ enjoys the following properties:*

*a. $\mathcal{I} \models \mathcal{K}$;*

*b. $(\mathcal{I}, d) \models C$.*

Moreover, the canonical model of $C$ characterizes *all* the valid elementary subsumptions whose left-hand side is $C$:

**Lemma 2** *If $C \sqsubseteq D$ is elementary, and $C$ is normalized w.r.t. $\mathcal{K}$, then $\mathcal{K} \models C \sqsubseteq D$ iff $(\mathcal{I}, d) \models D$, where $(\mathcal{I}, d)$ is the canonical model of $C$.*

Basically, STS decides whether $(\mathcal{I}, d) \models D$.

**Lemma 3** *If $C \sqsubseteq D$ is elementary, $C \neq \bot$, and $C$ is normalized w.r.t. $\mathcal{K}$, then $\mathsf{STS}(\mathcal{K}, C \sqsubseteq D) = \mathtt{true}$ iff $(\mathcal{I}, d) \models D$, where $(\mathcal{I}, d)$ is the canonical model of $C$.*

The correctness and completeness of STS easily follow:

**Theorem 2** *If $C \sqsubseteq D$ is elementary and $C$ is normalized w.r.t. $\mathcal{K}$, then $\mathsf{STS}(\mathcal{K}, C \sqsubseteq D) = \mathtt{true}$ iff $\mathcal{K} \models C \sqsubseteq D$.*

**Proof.** There are two possiblities. If $C = \bot$, then clearly $\mathcal{K} \models C \sqsubseteq D$ and $\mathsf{STS}(\mathcal{K}, C \sqsubseteq D) = \mathtt{true}$ (line 2 of STS), so the theorem holds. If $C \neq \bot$, then theorem follows immediately from lemmas 2 and 3. ∎

Now, through Lemma 2, subsumption checks over *full* $\mathcal{PL}$ concepts can be reduced to elementary subsumptions.

**Theorem 3** *For all interval-safe $\mathcal{PL}$ subsumption queries $\sigma = \big(C_1 \sqcup \ldots \sqcup C_m \sqsubseteq D_1 \sqcup \ldots \sqcup D_n\big)$ such that each $C_i$ is normalized w.r.t. $\mathcal{K}$, the entailment $\mathcal{K} \models \sigma$ holds iff for all $i \in [1,m]$ there exists $j \in [1,n]$ such that $\mathcal{K} \models C_i \sqsubseteq D_j$.*

**Proof.** By simple logical inferences, these two facts hold: (i) $\mathcal{K} \models \sigma$ iff $\mathcal{K} \models C_i \sqsubseteq \bigsqcup_{j=1}^n D_j$ holds for all $i \in [1,m]$, (ii) if $\mathcal{K} \models C_i \sqsubseteq D_j$ holds for some $j \in [1,n]$, then $\mathcal{K} \models C_i \sqsubseteq \bigsqcup_{j=1}^n D_j$. So we are only left to show the converse of (ii): assuming that $\mathcal{K} \not\models C_i \sqsubseteq D_j$ for all $j \in [1,n]$, we shall prove that $\mathcal{K} \not\models C_i \sqsubseteq \bigsqcup_{j=1}^n D_j$.

By assumption and Lemma 2, the canonical model $(\mathcal{I}, d)$ of $C_i$ is such that $(\mathcal{I}, d) \models \neg D_j$ for all $j \in [1,n]$. Therefore $(\mathcal{I}, d) \models \neg \bigsqcup_{j=1}^n D_j$. Then $\mathcal{K} \not\models C_i \sqsubseteq \bigsqcup_{j=1}^n D_j$ follows by noting that $(\mathcal{I}, d)$ satisfies both $\mathcal{K}$ and $C_i$ by Lemma 1. ∎

Theorem 3 directly yields an obvious polynomial-time algorithm for checking $\mathcal{PL}$ subsumptions: it suffices to check that for each $C_i$ there exists $D_j$ such that $\mathcal{K} \models C_i \sqsubseteq D_j$. So:

**Theorem 4** *Interval-safe $\mathcal{PL}$ subsumption queries can be answered in polynomial time.*

**Proof.** Let $\sigma$ be a $\mathcal{PL}$ query of the form illustrated in Theorem 3. By that theorem, after normalizing the $C_i$, it suffices to call $\mathsf{STS}(\mathcal{K}, C_i \sqsubseteq D_j)$ at most $m \times n$ times. Each such calls scans $C_i$ for each subconcept of $D_j$, searching for a matching concept. Matching may require to visit the hierarchy $\sqsubseteq^*$, so the cost of each call is $O(|D_j| \cdot |C_i| \cdot |\mathcal{K}|) = O(|\sigma|^2 \cdot |\mathcal{K}|)$. Then the overall cost of a naive implementation – excluding normalization – is $O(|\sigma|^4 \cdot |\mathcal{K}|)$. Normalization is $O(|\sigma|^2 \cdot |\mathcal{K}|)$, due to the cost of checking pairwise disjointness of concept names, so it is dominated by the cost of STS's runs. ∎

We are left to discuss the interval-safety prerequisite needed by Theorem 3.[8] It can be satisfied as follows, with a *preliminary interval normalization phase* of the query $C \sqsubseteq D$.

For each $[\ell,u](f)$ in $C$, let $x_1 < x_2 < \cdots < x_r$ be the integers that occur as interval endpoints in $D$ and belong to $[\ell,u]$. Let $x_0 = \ell$ and $x_{r+1} = u$ and replace $[\ell,u](f)$ with the equivalent concept

$$\bigsqcup_{i=0}^{r} \big([x_i,x_i](f) \sqcup [x_i+1, x_{i+1}-1](f)\big) \sqcup [x_{r+1}, x_{r+1}](f). \quad (6)$$

Then use distributivity of $\sqcap$ over $\sqcup$ and the equivalence $\exists R.(C_1 \sqcup C_2) \equiv \exists R.C_1 \sqcup \exists R.C_2$ to move all occurrences of $\sqcup$ to the top level. Denote the result of this interval normalization phase with $C^*$. Readers may easily verify that

**Proposition 5** *For all $\mathcal{PL}$ subsumption queries $C \sqsubseteq D$, $C^*$ is equivalent to $C$ and $C^* \sqsubseteq D$ is an interval-safe $\mathcal{PL}$ subsumption query.*

[8] Theorem 2 can also be proved without assuming interval safety. The proof will be given in a forthcoming journal version.

Clearly, interval normalization may inflate $C$ exponentially, due to the application of distributivity. We have to rely on the structure of policies to get a polynomial time bound: simple ($\sqcup$-free) policies have at most one, functional concrete feature (`dur`) so no combinatorial explosion occurs during pre-normalization. Accordingly—and more generally—the following proposition holds:

**Proposition 6** *$\mathcal{PL}$ subsumption queries $C_1 \sqcup \ldots \sqcup C_m \sqsubseteq D$ can be answered in polynomial time if there exists a constant $c$ that bounds the number of concrete feature occurrences in each $C_i$.*

Note that $C$ may contain any number of interval constraints, as $m$ grows, because the bound applies to each $C_i$ individually. This may seem unsatisfactory at a first glance. Unfortunately, no general polynomial-time algorithms exist (unless P=NP) because unrestricted $\mathcal{PL}$ subsumption queries are inherently difficult due to the interplay of interval constraints and $\sqcup$:

**Theorem 7** *Subsumption checking in $\mathcal{PL}$ is co*NP*-complete. The result holds even if the knowledge base is empty.*

We conclude this section by pointing out that the normalization rules in Table 1 can be used as a *policy validation* method, to check that a full $\mathcal{PL}$ concept is satisfiable.

**Proposition 8** *Let $\mathcal{K}$ be a $\mathcal{PL}$ knowledge base.*

1. *A $\mathcal{PL}$ concept $C = C_1 \sqcup \ldots \sqcup C_n$ is unsatisfiable w.r.t. $\mathcal{K}$ iff $C_i \rightsquigarrow \bot$ for all $i \in [1, n]$.*
2. *$\mathcal{PL}$ concept satisfiability w.r.t. $\mathcal{K}$ can be checked in polynomial time.*

**Proof.** Point 1 follows immediately from Prop. 1 and Lemma 1. It is easy to see that normalization takes polynomial time, so Point 2 holds. ∎

## 5 Discussion and Future Work

The encoding of usage policies in OWL 2 – using the tractable fragment of the description logic $\mathcal{PL}$ introduced in this paper – is simple enough to reduce compliance checking to tractable structural subsumption tests. Such tests can be computed in polynomial time w.r.t. the size of the ontology and the size of the policies involved (i.e. company policies and formalized consent). For a fixed company policy and fixed vocabularies, the complexity of verifying whether the company policy complies with the explicit consent released by data subjects grows linearly with the size of formalized consent, so we expect compliance checking to scale well to large numbers of data subjects and articulated consent expressions. Tractability depends on the fact that policies have a bounded number of concrete features; otherwise, $\mathcal{PL}$ subsumption is coNP-complete.

We are planning – as part of the activities of the XYZ project – to implement and optimize the structural subsumption algorithm and perform scalability tests. The compliance checking task is well-suited for parallelization (e.g. the compliance tests with respect to different consent policies are all mutually independent and can be carried out in parallel). For this purpose we will leverage the Big Data Europe infrastructure adopted by XYZ. We are also going to integrate the structural subsumption algorithm with ELK, a specialized reasoner for the tractable profile OWL2-EL. In this way, the vocabularies of data categories, purposes, locations, etc. can be expressed with OWL2-EL, instead of the limited axioms over concept names allowed in $\mathcal{PL}$ knowledge bases.

The real-time checking of storage consent for data streams, mentioned in the introduction, most likely requires further optimization. We foresee knowledge compilation techniques as a promising approach. Their feasibility can only be verified experimentally, and will be the subject of further work.

The previous encodings of policies in KR languages, such as [Uszok *et al.*, 2003; Kagal *et al.*, 2003; Bonatti *et al.*, 2010], focus on access control and trust management, rather than data usage control. Consequently, those languages lack the terms for expressing privacy-related and usage-related concepts. A more critical drawback is that the main reasoning task in those papers is permission checking; policy comparison (which is central to our work) is not considered. Both Rei and Protune [Kagal *et al.*, 2003; Bonatti *et al.*, 2010] support logic program rules. As we already mentioned, if rules are recursive then policy comparison is generally undecidable, otherwise it is NP-hard. The comparison of $\mathcal{PL}$ policies, instead, is tractable (Proposition 6). Similarly, KAoS [Uszok *et al.*, 2003] is based on a DL that, in general, is not tractable. Actually, KAoS policies use role-value maps that in general make reasoning undecidable (see [Baader *et al.*, 2003], Chap. 5); the authors do not discuss how to avoid this issue.

Our tractability and intractability results do *not* follow directly from any previous work. The most expressive language enjoying a complete structural subsumption algorithm – to the best of our knowledge – is the description logic underlying CLASSIC [Borgida and Patel-Schneider, 1994], that supports neither concept unions ($\sqcup$) nor qualified existential restrictions ($\exists R.C$). If unions were added, then subsumption checking would immediately become coNP-hard (unless concrete domains were restricted) for the same reasons why unrestricted subsumption checking is coNP-hard in $\mathcal{PL}$ (cf. Theorem 7). On the other hand, CLASSIC additonally supports qualified universal restrictions (that strictly generalize $\mathcal{PL}$'s range restrictions), number restrictions, and role-value maps, therefore it is not comparable to $\mathcal{PL}$.

$\mathcal{PL}$ partially intersects the $\mathcal{EL}$ familiy of tractable DL, too. In particular, $\mathcal{EL}^{++}$ supports $\bot$ (hence it can express disj), $\sqcap$, qualified existential restrictions and concrete domains. It is known that role range restrictions can be added without affecting tractability [Baader *et al.*, 2008], and that union can be supported in queries. However, it has been proved that functional roles make $\mathcal{EL}$ EXP-complete [Baader *et al.*, 2005], and – to the best of our knowledge – no tractable special cases are known. Therefore, $\mathcal{PL}$ is incomparable with the known tractable logics in the $\mathcal{EL}$ family. Moreover, our coNP-completeness result is *not* entailed by the intractability result for $\mathcal{EL}$ with functional roles.

## A Proofs

**Proof of Lemma 1** By induction on the maximum nesting level $\ell$ of $C$'s existential restrictions.

If $\ell = 0$ (i.e. there are no existential restrictions) then $(\mathcal{I}, d) \models C$ by construction (cf. Def. 2.a). Disjointness axioms are satisfied, otherwise normalization would make $C = \bot$ (contradicting the hypotheses). Inclusion axioms are satisfied due to the first and third bullets of Def. 2.a. Moreover, $\mathcal{I}$ trivially satisfies all the functionality and range assertions in $\mathcal{K}$ since all roles are empty. It follows that the lemma holds for the base case.

Now suppose that $\ell > 0$. By induction hypothesis (I.H), we have that all the submodels $(\mathcal{I}_i, d_i)$ used in Def. 2.b satisfy both $D_i$ and $\mathcal{K}$. Then it is immediate to see that $(\mathcal{I}, d) \models C$ by construction. We are only left to prove that $\mathcal{I}$ satisfies all axioms $\alpha$ in $\mathcal{K}$.

If $\alpha = \mathsf{func}(R)$, then rewrite rules 4) and 5) make sure that $C$ contains at most one existential restriction for $R$, so $\mathcal{J}$ satisfies $\alpha$. Since all $\mathcal{I}_i$ satisfy $\alpha$ by I.H. (induction hypothesis), $\mathcal{I}$ satisfies $\alpha$, too.

If $\alpha = \mathsf{range}(R, A)$, then rule 6) makes sure that for each top-level $\exists R_i.D_i$ in $C$, $D_i \equiv D_i' \sqcap A$. Then, by I.H., $(I_i, d_i) \models A$.

If $\alpha = \mathsf{disj}(A, B)$, and $\alpha$ were not true in $\mathcal{J}$, then rule 7) would make $C = \bot$ (a contradiction); the other parts of $\mathcal{I}$, i.e. $(\mathcal{I}_i, d_i)$, satisfy $\alpha$ by I.H.

Finally, inclusions are satisfied by construction, cf. the second bullet of Def. 2.b. It follows that $\mathcal{I}$ satisfies $\alpha$. ∎

**Proof of Lemma 2** (Only If part) Assume that $\mathcal{K} \models C \sqsubseteq D$. By Lemma 1.a, $\mathcal{I} \models \mathcal{K}$, so $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Moreover, by Lemma 1.b, $d \in C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Therefore $(\mathcal{I}, d) \models D$.

(If part) Assume that $(\mathcal{I}, d) \models D$. We are going to prove that $\mathcal{K} \models C \sqsubseteq D$ by structural induction on $D$.

If $D = A$ (a concept name), then $d \in A^{\mathcal{I}}$. Then, by construction of $\mathcal{I}$, $C = A_i \sqcap C'$ (up to top-level subconcept reordering), and $A_i \sqsubseteq^* A$. These two facts, respectively, imply $\models C \sqsubseteq A_i$ and $\mathcal{K} \models A_i \sqsubseteq A$, hence $\mathcal{K} \models C \sqsubseteq D$.

If $D = D_1 \sqcap D_2$, then by I.H. $\mathcal{K} \models D_i$ $(i = 1, 2)$, hence $\mathcal{K} \models C \sqsubseteq D$.

If $D = \exists R.D_1$, then for some $d_i \in \Delta^{\mathcal{I}}$, $(d, d_i) \in R^{\mathcal{I}}$ and $(\mathcal{I}_i, d_i) \models D_1$, where $(\mathcal{I}_i, d_i)$ (by construction of $\mathcal{I}$) is the canonical model of a top-level restriction $\exists R.C_1$ in $C$. It follows (using the I.H.) that $\models C \sqsubseteq \exists R.C_1$ and $\mathcal{K} \models C_1 \sqsubseteq D_1$, hence $\mathcal{K} \models C \sqsubseteq D$.

If $D = [\ell, u](f)$, then for some $u' \in [\ell, u]$, $(d, u') \in f^{\mathcal{I}}$. By construction of $\mathcal{I}$, $C$ must contain a top-level constraint $[\ell', u'](f)$, and by interval safety (as $C \sqsubseteq D$ is elementary), $[\ell', u'] \subseteq [\ell, u]$. Then $\models C \sqsubseteq D$. ∎

**Proof of Lemma 3** By structural induction on $D$. If $D = A$ (a concept name), then $\mathsf{STS}(\mathcal{K}, C \sqsubseteq D) = \mathtt{true}$ iff $C$ has a top-level subconcept $A_i$ such that $A_i \sqsubseteq^* A$. By def. of $\mathcal{I}$, this holds iff $d \in A^{\mathcal{I}}$, that is, $(\mathcal{I}, d) \models D$. This proves the base case.

If $D = D_1 \sqcap D_2$, then the lemma follows easily from the induction hypothesis.

If $D = \exists R.D_1$, then $\mathsf{STS}(\mathcal{K}, C \sqsubseteq D) = \mathtt{true}$ iff (i) $C$ has a top-level subconcept $\exists R.C_1$, and (ii) $\mathsf{STS}(\mathcal{K}, C_1 \sqsubseteq D_1) = \mathtt{true}$. Moreover, by def. of $\mathcal{I}$,$(\mathcal{I}, d) \models D$ iff (i) holds and (ii') $(\mathcal{I}_i, d_i) \models D_1$, where $(\mathcal{I}_i, d_i)$ is the canonical model of $C_1$. By I.H., (ii) is equivalent to (ii'), so the lemma holds.

If $D = [\ell, u](f)$, then $\mathsf{STS}(\mathcal{K}, C \sqsubseteq D) = \mathtt{true}$ iff $C$ has a top-level subconcept $[\ell', u'](f)$ such that $[\ell', u'] \subseteq [\ell, u]$. This implies (by def. of $\mathcal{I}$) that $(d, u') \in f^{\mathcal{I}}$ and $u' \in [\ell, u]$, that is, $(\mathcal{I}, d) \models D$. Conversely, if $(d, u') \in f^{\mathcal{I}}$ and $u' \in [\ell, u]$, then $C$ must have a top-level subconcept $[\ell', u'](f)$ such that also $\ell' \in [\ell, u]$ (by interval safety, that holds by the hypothesis that $C \sqsubseteq D$ is elementary). Then $\mathsf{STS}(\mathcal{K}, C \sqsubseteq D) = \mathtt{true}$. ∎

**Proof of Theorem 7** Hardness is proved by reducing 3SAT to the complement of subsumption. Let $S$ be a given set of clauses $c_i = L_{i1} \vee L_{i2} \vee L_{i3}$ $(1 \leq i \leq n)$ where each $L_{ij}$ is a literal. We are going to use the propositional symbols $p_1, \ldots, p_m$ occurring in $S$ as feature names in $\mathcal{PL}$ concepts, and define a subsumption $C \sqsubseteq D$ that is valid iff $S$ is unsatisfiable. Let $C = \big([0,1](p_1) \sqcap \ldots \sqcap [0,1](p_m)\big)$ and $D = \bigsqcup_{i=1}^{n} \big(\tilde{L}_{i1} \sqcap \tilde{L}_{i2} \sqcap \tilde{L}_{i3}\big)$, where each $\tilde{L}_{ij}$ encodes the complement of $L_{ij}$ as follows:

$$\tilde{L}_{ij} = \begin{cases} [0,0](p_k) & \text{if } L_{ij} = p_k\,, \\ [1,1](p_k) & \text{if } L_{ij} = \neg p_k\,. \end{cases}$$

The correspondence between the propositional interpretations $I$ of $S$ and the interpretations $\mathcal{J}$ of $C \sqsubseteq D$ is the following.

Given $I$ and an arbitrary element $d$, define $\mathcal{J} = \langle \{d\}, \cdot^{\mathcal{J}} \rangle$ such that $(d, 0) \in p_i^{\mathcal{J}}$ iff $I(p_i) = \mathit{false}$, and $(d, 1) \in p_i^{\mathcal{J}}$ otherwise. By construction, $(\mathcal{J}, d) \models C$, and $I \models S$ iff $(\mathcal{J}, d) \models \neg D$. Consequently, if $S$ is satisfiable, then $C \sqsubseteq D$ is not valid.

Conversely, if $C \sqsubseteq D$ is not valid, then there exist $\mathcal{J}$ and $d \in \Delta^{\mathcal{J}}$ such that $(\mathcal{J}, d) \models C \sqcap \neg D$. Define a propositional interpretation $I$ of $S$ by setting $I(p) = \mathit{true}$ iff $(d, 1) \in p_i^{\mathcal{J}}$. By construction (and since $d$ does not satisfy $D$), $I \models S$, which proves that if $C \sqsubseteq D$ is not valid, then $S$ is satisfiable.

We conclude that the above reduction is correct. Moreover, it can be clearly computed in LOGSPACE. This proves that subsumption is coNP-hard even if the KB is empty.

Membership in coNP can be proved by showing that the complement of subsumption is in NP. Given a query $C \sqsubseteq D$, it suffices to choose nondeterministically one of the disjuncts $C_i$ in the left hand side of the query, and replace each constraint $[\ell, u](f)$ occurring in $C_i$ with a nondeterministically chosen disjunct from (6). Call $C_i'$ the resulting concept and note that it is one of the disjuncts in $C^*$. Therefore, $\mathcal{K} \not\models C \sqsubseteq D$ iff for some nondeterministic choice, $\mathcal{K} \not\models C_i' \sqsubseteq D$. The latter subsumption test can be evaluated in deterministic polynomial time with STS, so the complement of $\mathcal{PL}$ subsumption is in NP. ∎

## References


[Baader *et al.*, 2003] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[Baader *et al.*, 2005] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope. In *IJCAI-05*, pages 364–369. Professional Book Center, 2005.

[Baader *et al.*, 2008] Franz Baader, Carsten Lutz, and Sebastian Brandt. Pushing the EL envelope further. In Kendall Clark and Peter F. Patel-Schneider, editors, *Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Directions, Washington, DC, USA, 1-2 April 2008.*, volume 496 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[Bonatti *et al.*, 2002] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security*, 5, 2002.

[Bonatti *et al.*, 2010] Piero A. Bonatti, Juri Luca De Coi, Daniel Olmedilla, and Luigi Sauro. A rule-based trust negotiation system. *IEEE Trans. Knowl. Data Eng.*, 22(11):1507–1520, 2010.

[Bonatti, 2010] Piero A. Bonatti. Datalog for security, privacy and trust. In Oege de Moor, Georg Gottlob, Tim Furche, and Andrew Jon Sellers, editors, *Datalog Reloaded - First International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers*, volume 6702 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2010.

[Borgida and Patel-Schneider, 1994] Alexander Borgida and Peter F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. *J. Artif. Intell. Res.*, 1:277–308, 1994.

[Kagal *et al.*, 2003] Lalana Kagal, Timothy W. Finin, and Anupam Joshi. A policy language for a pervasive computing environment. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 63–, Lake Como, Italy, June 2003. IEEE Computer Society.

[Uszok *et al.*, 2003] Andrzej Uszok, Jeffrey M. Bradshaw, Renia Jeffers, Niranjan Suri, Patrick J. Hayes, Maggie R. Breedy, Larry Bunch, Matt Johnson, Shriniwas Kulkarni, and James Lott. KAoS policy and domain services: Towards a description-logic approach to policy representation, deconfliction, and enforcement. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 93–96, Lake Como, Italy, June 2003. IEEE Computer Society.

[Woo and Lam, 1993] Thomas Y. C. Woo and Simon S. Lam. Authorizations in distributed systems: A new approach. *Journal of Computer Security*, 2(2-3):107–136, 1993.