Artificial Intelligence 289 (2020) 103389

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Real-time reasoning in OWL2 for GDPR compliance

Piero A. Bonatti [a,*], Luca Ioffredo [b], Iliana M. Petrova [b], Luigi Sauro [a], Ida R. Siahaan [b]

[a] *Università di Napoli Federico II, Italy*
[b] *CeRICT, Italy*

ARTICLE INFO

Article history:
Received 19 April 2019
Received in revised form 31 July 2020
Accepted 17 September 2020
Available online 18 September 2020

Keywords:
Tractable OWL2 fragments
Structural subsumption
Import-by-query
Knowledge compilation
Semantic policy languages
GDPR

ABSTRACT

This paper shows how knowledge representation and reasoning techniques can be used to support organizations in complying with the GDPR, that is, the new European data protection regulation. This work is carried out in a European H2020 project called SPECIAL. Data usage policies, the consent of data subjects, and selected fragments of the GDPR are encoded in a fragment of OWL2 called $\mathcal{PL}$ (policy language); compliance checking and policy validation are reduced to subsumption checking and concept consistency checking. This work proposes a satisfactory tradeoff between the expressiveness requirements on $\mathcal{PL}$ posed by the modeling of the GDPR, and the scalability requirements that arise from the use cases provided by SPECIAL's industrial partners. Real-time compliance checking is achieved by means of a specialized reasoner, called PLR, that leverages knowledge compilation and structural subsumption techniques. The performance of a prototype implementation of PLR is analyzed through systematic experiments, and compared with the performance of other important reasoners. Moreover, we show how $\mathcal{PL}$ and PLR can be extended to support richer ontologies, by means of import-by-query techniques. We prove novel tractability and intractability results related to $\mathcal{PL}$, and some negative results about the restrictions posed on ontology import.

1. Introduction

The new European General Data Protection Regulation[1] (GDPR), that has come into force on May 25, 2018, places stringent restrictions on the processing of personally identifiable data. The regulation applies also to companies and organizations that are not located in Europe, whenever they track or provide services to data subjects that are in the European Union.[2] Infringements may severely affect the reputation of the violators, and are subject to substantial administrative fines (up to 4% of the total worldwide annual turnover or 20 million Euro, whichever is higher). Therefore, the risks associated to infringements constitute a major disincentive to the abuse of personal data. Given that the collection and the analysis of personal data are paramount sources of innovation and revenue, companies are interested in maximizing personal data usage within the limits posed by the GDPR. Consequently, *data controllers* (i.e. the personal and legal entities that process personal data) are looking for methodological and technological means to comply with the regulation's requirements efficiently and safely.

* Corresponding author.
E-mail address: pab@unina.it (P.A. Bonatti).

[1] http://data.consilium.europa.eu/doc/document/ST-5419-2016-INIT/en/pdf.
[2] Cf. Article 3 of the GDPR.

https://doi.org/10.1016/j.artint.2020.103389

The European H2020 project SPECIAL[3] is aimed at supporting controllers in complying with the GDPR. SPECIAL is tackling several hard problems related to usability, transparency and compliance, see [13,9,34] for an overview. In this paper, we focus on SPECIAL's approach to the representation of data usage activities and consent to data processing, together with the associated reasoning tasks related to the validation of data usage policies and compliance checking.

The management of the consent to data processing granted by data subjects plays a central role in this picture. The GDPR is not concerned with anonymous data, nor data that do not describe persons (like astronomical data). The other data (hereafter called *personal data*) must be processed according to the legal bases provided by the regulation. Some examples of such legal bases include public interest, the vital interests of the data subject, contracts, and the legitimate interests of the data controller, just to name a few.[4] These legal bases are constrained by a number of provisos and caveats that restrict their applicability.[5] So, in practice, the kinds of personal data processing that are most useful for data-driven business are almost exclusively allowed by another legal basis, namely, the explicit consent of the data subjects.[6] Thus, it is important to encode consent appropriately, so as to record it for auditing, and give automated support to compliance checking.

Also the controller's usage of personal data must be appropriately represented and stored, in order to fulfill the obligation to record personal data processing activities,[7] and in order to verify that such activities comply with the available consent and with the GDPR.

SPECIAL tackles these needs by adopting a logic-based representation of data usage policies, that constitutes a uniform language to encode consent, the activities of controllers, and also selected parts of the GDPR. A logic-based approach is essential for achieving several important objectives, including the following:

- strong correctness and completeness guarantees on permission checking and compliance checking;
- ensuring the mutual coherence of the different reasoning tasks related to policies, such as policy validation, permission checking, compliance checking, and explanations;
- ensuring correct usage after data is transferred to other controllers (i.e. interoperability), through the unambiguous semantics of knowledge representation languages.

Some of SPECIAL's use cases place challenging scalability requirements on reasoning. During the execution of the controllers' data processing software, each operation involving personal data must be checked for compliance with the consent granted by the data subjects. The frequency of such compliance checks may be significantly high, so SPECIAL needs to implement the corresponding reasoning tasks in such a way that the time needed for each check does not exceed a few hundreds of μ-seconds. For example, here is a real-world scenario provided by SPECIAL's industrial partners.

Streaming Scenario Telecom providers, that nowadays are also Internet providers, receive from their base stations about 15000 call records per second, and receive about 850 millions of probing records per day from their wi-fi network (almost 10000 events per second). The data contained in the aforementioned records are of great interest for strategic applications and services, such as location-based services and taylored recommendations; however, these are personal data, and the European regulations on data protection prohibit the above usage without the data subject's consent. Without it, even storing the data temporarily, waiting for a batch process to discard the records that cannot be processed, is illegal. Then the description of how each application processes the data and why, that we will call *business policy* in the following, must be checked in real-time for compliance against the available consent for the record being processed, while the stream of data is generated. The scenario is further complicated by the fact that each data subject can withdraw or modify her consent anytime, and that she may selectively decide to opt in or out each processing option (e.g. a customer might accept only location tracking, and not internet tracking). ■

We address real-time requirements by designing a specialized reasoner for the policy language.

After recalling the notions about description logics and their properties, that will be needed in the paper, our contributions will be illustrated in the following order.

- Section 3 shows how to encode usage policies and the relevant parts of the GDPR with a fragment of $\mathcal{SROIQ}(\mathsf{D})$ (the logical foundation of OWL2-DL). The details of the encoding will be related explicitly to GDPR's requirements. Afterwards, we formally define $\mathcal{PL}$, that is, the fragment of $\mathcal{SROIQ}(\mathsf{D})$ used to encode data usage policies.
- Section 4 is devoted to the complexity analysis of reasoning in $\mathcal{PL}$. We consider concept satisfiability and subsumption checking, that constitute the core of policy validation and compliance checking. We will show that unrestricted $\mathcal{PL}$ subsumption checking is coNP-complete. However, under a restrictive hypothesis motivated by SPECIAL's use cases, subsumption checking is possible in polynomial time. Tractability is proved by means of a specialized two-stage reasoner

[3] https://www.specialprivacy.eu/.

[4] Cf. Article 6 of the GDPR.

[5] Of particular relevance here are the data minimization principle introduced in Article 5, and the limitations to the legitimate interests of the controller rooted in Article 6.1(f).

[6] Article 6.1(a).

[7] Cf. Article 30 of the GDPR.

called PLR, based on a preliminary normalization phase followed by a structural subsumption algorithm. A preliminary account of this section has been published in [8].
- Section 5 shows how to support richer ontology languages for the description of policy elements. The vocabularies for policy elements are treated like imported ontologies by means of an *import by query* (IBQ) approach, that can be implemented with a modular integration of the specialized reasoner for $\mathcal{PL}$ with a reasoner for the imported ontology. We prove that this integration method is correct and complete, and justify the restrictive assumptions on the imported ontologies, by adapting and slightly extending previous results on IBQ limitations. Moreover, we show that under hypotheses compatible with SPECIAL's application scenarios, the external ontology can be compiled into a $\mathcal{PL}$ ontology, thereby reducing the IBQ approach to plain $\mathcal{PL}$ reasoning.
- $\mathcal{PL}$ subsumption checking is experimentally evaluated in Section 6. After describing the implementation of PLR and its optimizations, PLR's performance is compared with that of other important engines, such as ELK [33], GraphDB [25], Hermit [23], and RDFox [37]. For this purpose, we use two sets of experiments. The first set is derived from the pilots of SPECIAL that have reached a sufficient development level, namely, a recommendation system based on location data and internet navigation information, designed by Proximus, and a financial risk analysis scenario developed by Thomson Reuters. The second batch of experiments is fully synthetic, instead, and contains increasingly large policies and ontologies, in order to assess the scalability of PLR.

Section 7 concludes the paper with a final discussion of our results and interesting perspectives for future work. Related work is heterogeneous (declarative policy languages, legal reasoning, tractable description logics, IBQ methods) so we distribute its discussion across the pertinent sections, rather than in a single dedicated section.

2. Preliminaries on description logics

Here we report the basics on the Description Logics (DL) needed for our work and refer the reader to [4] for further details. The DL languages of our interest are built from countably infinite sets of concept names ($\mathsf{N_C}$), role names ($\mathsf{N_R}$), individual names ($\mathsf{N_I}$), concrete property names ($\mathsf{N_F}$), and concrete predicates ($\mathsf{N_P}$). For brevity, individual names will sometimes be called *constants*. A signature Σ is a subset of $\mathsf{N_C} \cup \mathsf{N_R} \cup \mathsf{N_I} \cup \mathsf{N_F}$.[8]

We will use metavariables A, B for concept names, C, D for (possibly compound) concepts, R, S for roles, a, b for individual names, f, g for concrete property names, and p for concrete predicates. Concepts are built from concept names and from the concept constructors listed in Table 1. Similarly, roles are built from role names and from the role constructors listed in Table 1. In the following the term *expression* refers to both concepts and roles.

An *interpretation* $\mathcal{I}$ of a signature Σ is a structure $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a nonempty set, and the *interpretation function* $\cdot^{\mathcal{I}}$, defined over Σ, is such that (i) $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ if $A \in \mathsf{N_C}$; (ii) $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ if $R \in \mathsf{N_R}$; (iii) $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ if $a \in \mathsf{N_I}$; (iv) $f^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathsf{D}}$ if $f \in \mathsf{N_F}$, where Δ^{D} denotes the domain of the predicates in $\mathsf{N_P}$. The semantics of an n-ary predicate $p \in \mathsf{N_P}$ is a set of tuples $p^{\mathsf{D}} \subseteq (\Delta^{\mathsf{D}})^n$. As usual, the pair $(\Delta^{\mathsf{D}}, \mathsf{N_P})$ is called *concrete domain*.[9] 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 $\exists f.[\ell, u]$. So an individual $d \in \Delta^{\mathcal{I}}$ belongs to $(\exists f.[\ell, u])^{\mathcal{I}}$ if, for some integer $i \in [\ell, u]$, $(d, i) \in f^{\mathcal{I}}$.

The third column of Table 1 shows how to extend the valuation $\cdot^{\mathcal{I}}$ of an interpretation $\mathcal{I}$ to compound DL expressions and axioms. GCI stands for "general concept inclusion". An interpretation $\mathcal{I}$ *satisfies* an axiom α (equivalently, $\mathcal{I}$ is a *model* of α) if $\mathcal{I}$ satisfies the corresponding semantic condition in Table 1. When $\mathcal{I}$ satisfies α we write $\mathcal{I} \models \alpha$. We will sometimes use axioms of the form $C \equiv D$, that are abbreviations for the pair of inclusions $C \sqsubseteq D$ and $D \sqsubseteq C$.

A *knowledge base* $\mathcal{K}$ is a finite set of DL axioms. Its *terminological part* (or *TBox*) is the set of terminological axioms[10] in $\mathcal{K}$, while its *ABox* is the set of its assertion axioms.

If X is a DL expression, an axiom, or a knowledge base, then $\Sigma(X)$ denotes the signature consisting of all symbols occurring in X, but concrete predicates. An interpretation $\mathcal{I}$ of a signature $\Sigma \supseteq \Sigma(\mathcal{K})$ is a *model* of $\mathcal{K}$ (in symbols, $\mathcal{I} \models \mathcal{K}$) if $\mathcal{I}$ satisfies all the axioms in $\mathcal{K}$. We say that $\mathcal{K}$ *entails* an axiom α (in symbols, $\mathcal{K} \models \alpha$) if all the models of $\mathcal{K}$ satisfy α. The *subsumption problem* consists in deciding whether $\mathcal{K} \models C \sqsubseteq D$ for given $\mathcal{K}$, C, and D.

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

2.1. The description logics used in this paper

The logic $\mathcal{SRIQ}$ supports the $\mathcal{SRIQ}$ constructors and axioms illustrated in Table 1. In a $\mathcal{SRIQ}$ knowledge base, in order to preserve decidability, the set of role axioms should be *regular* and the roles S, S_1, S_2 *simple*, according to the definitions stated in [28].[11] Horn-$\mathcal{SRIQ}$ further restricts $\mathcal{SRIQ}$ GCIs as specified in [38]. For simplicity, here we illustrate

[8] Concrete predicates are deliberately left out due to their special treatment.
[9] We are assuming – for brevity – that there is one concrete domain. However, this framework can be immediately extended to multiple domains.
[10] See Table 1.
[11] The definitions are omitted because they are not needed in our results.

Table 1
Syntax and semantics of DL constructs and axioms.

Name	Syntax	Semantics
$\mathcal{SRIQ}$ concept and role constructors (the latter recognizable by the word "role" in the name)		
inverse role	R^-	$\{(y,x) \mid (x,y) \in R^{\mathcal{I}}\}$ $(R \in \mathsf{N_R})$
top	$\top$	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
bottom	$\bot$	$\bot^{\mathcal{I}} = \emptyset$
intersection	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
union	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
complement	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
existential restriction	$\exists R.C$	$\{d \in \Delta^{\mathcal{I}} \mid \exists (d,e) \in R^{\mathcal{I}} : e \in C^{\mathcal{I}}\}$
universal restriction	$\forall R.C$	$\{d \in \Delta^{\mathcal{I}} \mid \forall (d,e) \in R^{\mathcal{I}} : e \in C^{\mathcal{I}}\}$
number restrictions	$\bowtie n\ S.C$	$\left\{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in S^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \bowtie n\right\}$ $(\bowtie = \leq, \geq)$
self	$\exists S.\mathsf{Self}$	$\{x \in \Delta^{\mathcal{I}} \mid (x,x) \in S^{\mathcal{I}}\}$
Additional concept and role constructors of $\mathcal{SROIQ}$ (D) (the latter with the word "role" in the name)		
universal role	U	$U^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
nominals	$\{a\}$	$\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$ $(a \in \mathsf{N_I})$
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 $\vec{v} \in p^{\mathsf{D}}\}$
$\mathcal{SRIQ}$ terminological axioms		$\mathcal{I}$ satisfies the axiom if:
GCI	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
role disjointness	$\mathsf{disj}(S_1, S_2)$	$S_1^{\mathcal{I}} \cap S_2^{\mathcal{I}} = \emptyset$
complex role inclusions	$R_1 \circ ... \circ R_n \sqsubseteq R$	$R_1^{\mathcal{I}} \circ \ldots \circ R_n^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
$\mathcal{SRIQ}$ concept and role assertion axioms		
conc. assrt.	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assrt.	$R(a,b)$	$(a,b)^{\mathcal{I}} \in R^{\mathcal{I}}$
Other terminological axioms expressible with the above axioms and used in low-complexity DLs		
disjointness	$\mathsf{disj}(C, D)$	$C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$
functionality	$\mathsf{func}(R)$	$R^{\mathcal{I}}$ is a partial function
range	$\mathsf{range}(R, C)$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C^{\mathcal{I}}$

Table 2
The Horn restriction of $\mathcal{SRIQ}$ GCIs (normal form).

$C_1 \sqcap C_2 \sqsubseteq D$
$\exists R.C \sqsubseteq D$
$C \sqsubseteq \forall R.D$
$C \sqsubseteq \exists R.D$
$C \sqsubseteq \leq 1\ S.D$
$C \sqsubseteq \geq n\ S.D$

The above concepts C, C_1, C_2, D either belong to $\mathsf{N_C} \cup \{\bot, \top\}$, or are of the form $\exists S.\mathsf{Self}$. Symbol S denotes a *simple* role [28].

only the normal form adopted in [39], see Table 2. Like all Horn DLs, Horn-$\mathcal{SRIQ}$ is *convex*, that is, $\mathcal{K} \models C_0 \sqsubseteq C_1 \sqcup C_2$ holds iff either $\mathcal{K} \models C_0 \sqsubseteq C_1$ or $\mathcal{K} \models C_0 \sqsubseteq C_2$

The logic $\mathcal{EL}$ is a fragment of Horn-$\mathcal{SRIQ}$ that supports only atomic concepts and roles, $\top$, $\sqcap$, and existential restrictions. Supported axioms are GCIs and assertions. We will denote with $\mathcal{EL}^+$ the extension of $\mathcal{EL}$ with $\bot$ and range axioms. $\mathcal{EL}^{++}$ denotes the extension of $\mathcal{EL}^+$ with nominals, concrete domains, complex role inclusions, and range axioms.[12] Subsumption checking and consistency checking are tractable in $\mathcal{EL}$ and $\mathcal{EL}^+$. The same holds for $\mathcal{EL}^{++}$ provided that concrete domains have a tractable entailment problem and are *convex*, in the sense that $\models p_1(\vec{f}_1) \vee \ldots \vee p_n(\vec{f}_n)$ holds iff $\models p_i(\vec{f}_i)$ holds for some $i \in [1, n]$ [3]. $\mathcal{EL}^{++}$ provides the foundation for the OWL2-EL profile.[13]

The logic *DL-lite* is a fragment of Horn-$\mathcal{SRIQ}$ that supports only inverse roles, unqualified existential restrictions (i.e. concepts of the form $\exists R.\top$), GCIs and assertions. Moreover, complements ($\neg$) are allowed on the right-hand side of GCIs. *DL-lite*$_R$ extends *DL-lite* with role inclusions of the form $R \sqsubseteq S$ and $R \sqsubseteq \neg S$. *DL-lite*$_{horn}^{\mathcal{H}}$ extends *DL-lite* by supporting $\sqcap$ and role inclusions of the form $R_1 \sqsubseteq R_2$. Subsumption and consistency checking are tractable in both logics. *DL-lite* constitutes the foundation of the OWL2-QL profile.[14]

The logic $\mathcal{SROIQ}$(D) supports all the constructs and axioms illustrated in Table 1. It is the description logic underlying the standard OWL2-DL.

[12] Range axioms must satisfy the restrictions described in Sec. 2.2.6 of https://www.w3.org/TR/owl2-profiles. We do not need those details in this paper.
[13] https://www.w3.org/TR/owl2-profiles/#OWL_2_EL.
[14] https://www.w3.org/TR/owl2-profiles/#OWL_2_QL.

To fix ideas, in the following two subsections let $\mathcal{K}$ range over $\mathcal{SROIQ}(\mathsf{D})$ knowledge bases. However, the results and definitions of those subsections hold also for the DLs that are not fragments of $\mathcal{SROIQ}(\mathsf{D})$, such as the logic supported by CLASSIC [17] and the DLs with fixpoints [14].

2.2. The disjoint model union property

A knowledge base $\mathcal{K}$ such that $\Sigma(\mathcal{K}) \cap \mathsf{N_I} = \emptyset$ enjoys the *disjoint model union property* if for all disjoint models $\mathcal{I}$ and $\mathcal{J}$ of $\mathcal{K}$, their disjoint union $\mathcal{I} \uplus \mathcal{J} = \langle \Delta^{\mathcal{I}} \uplus \Delta^{\mathcal{J}}, \cdot^{\mathcal{I}\uplus\mathcal{J}} \rangle$ – where $P^{\mathcal{I}\uplus\mathcal{J}} = P^{\mathcal{I}} \uplus P^{\mathcal{J}}$ for all $P \in \mathsf{N_C} \cup \mathsf{N_R} \cup \mathsf{N_F}$ – satisfies $\mathcal{K}$, too ([4], Ch. 5). This definition is extended naturally to the union $\biguplus \mathcal{S}$ of an arbitrary set $\mathcal{S}$ of disjoint models. The disjoint model union property plays an important role in our results. It is broken by the universal role and nominals. The main problem with nominals (and the reason of the prerequisite $\Sigma(\mathcal{K}) \cap \mathsf{N_I} = \emptyset$) is that if $\mathcal{I}$ and $\mathcal{J}$ are disjoint, then for all individual constants $a \in \mathsf{N_I}$, $a^{\mathcal{I}} \neq a^{\mathcal{J}}$, so it is not immediately clear what $a^{\mathcal{I}\uplus\mathcal{J}}$ should be. This problem can be resolved for the constants occurring in ABoxes. Informally speaking, it suffices to pick the constants' interpretation from an arbitrary argument of the union.[15]

Definition 2.1 *(Generalized disjoint union).* For all sets of mutually disjoint interpretations $\mathcal{S}$ and all $\mathcal{I} \in \mathcal{S}$, let $\biguplus^{\mathcal{I}} \mathcal{S}$ be the interpretation $\mathcal{U}$ such that:

$$\begin{aligned} \Delta^{\mathcal{U}} &= \bigcup \{\Delta^{\mathcal{J}} \mid \mathcal{J} \in \mathcal{S}\} \\ P^{\mathcal{U}} &= \bigcup \{P^{\mathcal{J}} \mid \mathcal{J} \in \mathcal{S}\} \quad \text{for all } P \in \mathsf{N_C} \cup \mathsf{N_R} \cup \mathsf{N_F} \\ a^{\mathcal{U}} &= a^{\mathcal{I}} \qquad\qquad\qquad\quad \text{for all } a \in \mathsf{N_I}\,. \end{aligned}$$

If the terminological part of a knowledge base $\mathcal{K}$ has the (standard) disjoint model union property, then the generalized union of disjoint models of $\mathcal{K}$ is still a model of $\mathcal{K}$:

Proposition 2.2. *Let $\mathcal{K} = \mathcal{T} \cup \mathcal{A}$, where $\mathcal{T}$ is the terminological part of $\mathcal{K}$ and $\mathcal{A}$ is its ABox. If $\mathcal{T}$ has the disjoint model union property then for all sets $\mathcal{S}$ of mutually disjoint models of $\mathcal{K}$, and for all $\mathcal{I} \in \mathcal{S}$, $\biguplus^{\mathcal{I}} \mathcal{S} \models \mathcal{K}$.*

Proof. Let $\mathcal{S}$ and $\mathcal{I}$ be as in the statement, and let $\mathcal{U} = \biguplus^{\mathcal{I}} \mathcal{S}$. Note that $\Sigma(\mathcal{T}) \cap \mathsf{N_I} = \emptyset$, otherwise the disjoint union of $\mathcal{T}$'s models would not be defined and $\mathcal{T}$ would not enjoy the disjoint model union property, contradicting the hypothesis. For all interpretations $\mathcal{J}$, let $\mathcal{J}\backslash\mathsf{N_I}$ denote the restriction of $\mathcal{J}$ to the symbols in $\mathsf{N_C} \cup \mathsf{N_R} \cup \mathsf{N_F}$ (i.e. excluding the individual constants in $\mathsf{N_I}$). Note that for all $\mathcal{J} \in \mathcal{S}$, $\mathcal{J}\backslash\mathsf{N_I}$ is a model of $\mathcal{T}$, because $\Sigma(\mathcal{T}) \cap \mathsf{N_I} = \emptyset$. Therefore, by hypothesis, $\biguplus\{\mathcal{J}\backslash\mathsf{N_I} \mid \mathcal{J} \in \mathcal{S}\}$ is a model of $\mathcal{T}$. Clearly, $\biguplus\{\mathcal{J}\backslash\mathsf{N_I} \mid \mathcal{J} \in \mathcal{S}\} = (\biguplus^{\mathcal{I}} \mathcal{S})\backslash\mathsf{N_I}$; as a consequence, also $\biguplus^{\mathcal{I}} \mathcal{S}$ is a model of $\mathcal{T}$. We are only left to prove that $\mathcal{U}$ is a model of $\mathcal{A}$. Consider an arbitrary assertion $\alpha \in \mathcal{A}$. Since the models in $\mathcal{S}$ are disjoint, and the interpretation of constants in $\mathcal{U}$ ranges over $\Delta^{\mathcal{I}}$, it holds that $a^{\mathcal{U}} \in C^{\mathcal{U}}$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$, $(a^{\mathcal{U}}, b^{\mathcal{U}}) \in R^{\mathcal{U}}$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$, and $f^{\mathcal{U}}(a^{\mathcal{U}}) = f^{\mathcal{I}}(a^{\mathcal{I}})$ ($f \in \mathsf{N_F}$). Moreover, $\mathcal{I}$ is a model of $\mathcal{A}$ by hypothesis. It follows immediately that $\mathcal{U}$ is a model of $\mathcal{A}$. ■

2.3. Modularity and locality

A knowledge base $\mathcal{K}$ is *semantically modular* with respect to a signature Σ if each interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ over Σ can be extended to a model $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ of $\mathcal{K}$ such that $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}}$ and $X^{\mathcal{J}} = X^{\mathcal{I}}$, for all symbols $X \in \Sigma$. Roughly speaking, this means that $\mathcal{K}$ does not constrain the symbols of Σ in any way.

A special case of semantic modularity exploited in [21] is *locality*: A knowledge base $\mathcal{K}$ is *local* with respect to a signature Σ if the above $\mathcal{J}$ can be obtained simply as specified in the next definition.

Definition 2.3 *(Locality).* A knowledge base $\mathcal{K}$ is *local* with respect to a signature Σ if each interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ over Σ can be extended to a model $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ of $\mathcal{K}$ by setting $X^{\mathcal{I}} = \emptyset$ for all concept and role names $X \in \Sigma(\mathcal{K}) \setminus \Sigma$.

Locality will be needed in Section 5, for the integration of $\mathcal{PL}$ knowledge bases with imported ontologies. In particular, it is an essential ingredient of the completeness proof for IBQ reasoning.

3. Semantic encoding of data usage policies

SPECIAL's policy language $\mathcal{PL}$ – that is a fragment of OWL2-DL – has been designed to describe data usage. Such descriptions can be exploited to encode: (i) the consent to data processing given by data subjects, (ii) how the controller's

[15] The following formalization of this idea generalizes a proof technique used in [21, Lemma 1].

internal processes use data, and (iii) selected parts of the GDPR that can be used to support the validation of the controller's internal processes. Moreover, $\mathcal{PL}$ is used to encode the entries of SPECIAL's *transparency ledger*, that is a log of data processing operations that can be queried by:

- data subjects, in order to monitor how their personal data are used by the controller and where they are transferred to;
- data protection officers, in order to audit the behavior of the controller;
- the controllers themselves, in order to monitor their own internal processes.

The aspects of data usage that have legal relevance are clearly indicated in several articles of the GDPR and in the available guidelines. They are mentioned, for example, in the specification of what is valid consent, what are the legal bases for processing, what are the rights of data subjects, which aspects should be covered by national regulations, and the obligation of controllers to keep a record of the processing operations that involve personal data (see, inter alia, Articles 6.1, 6.3, 6.4, 7, 15.1, 23.2, 23.2, 30.1). See also the section titled "Records should contain" in the guidelines for SMEs published on http://ec.europa.eu/justice/smedataprotect/index_en.htm. That section describes how to fulfill the obligation to record the data subjects' consent to processing (Article 7) and, in particular, it specifies which pieces of information should be recorded. According to the above sources of requirements, the main properties of data usage that need to be encoded and archived are the following:

- reasons for data processing (purpose);
- which data categories are involved;
- what kind of processing is applied to the data;
- which third parties data are distributed to (recipients);
- countries in which the data will be stored (location);
- time constraints on data erasure (duration).

The above properties characterize a *usage policy*. SPECIAL adopts a direct encoding of usage policies in description logics, based on those features. The simplest possible policies have the form:

$$\exists \mathsf{has\_purpose}.P \sqcap \exists \mathsf{has\_data}.D \sqcap \exists \mathsf{has\_processing}.O \sqcap \exists \mathsf{has\_recipient}.R \sqcap \exists \mathsf{has\_storage}(\exists \mathsf{has\_location}.L \sqcap \exists \mathsf{has\_duration}.T)\,. \tag{1}$$

All of the above roles are functional. Duration is represented as an interval of integers $[t_1, t_2]$, representing a minimum and a maximum storage time (such bounds may be required by law, by the data subject, or by the controller itself). The classes P, D, O, etc. are defined in suitable *auxiliary vocabularies* (ontologies) that specify also the relationships between different terms. The expressiveness requirements on the vocabularies and their design are discussed later, in Section 5. Until then, *the reader may assume that the vocabularies are defined by means of inclusions* $A \sqsubseteq B$ *and disjointness constraints* $\mathsf{disj}(A, B)$, where A, B are concept *names*. Such restrictions will be lifted later.

If the data subject consents to a policy of the form (1), then she authorizes all of its instances. For example if $D = \mathsf{DemographicData}$ then the data subject authorizes – in particular – the use of her address, age, income, etc. as specified by the other properties of the policy.

It frequently happens that the data controller intends to use different data categories in different ways, according to their usefulness and sensitivity, so consent requests comprise multiple *simple usage policies* like (1) (one for each usage type). The intended meaning is that consent is requested for all the instances of all those policies; accordingly, such a compound policy is formalized with the union of its components. The result is called *full* (usage) *policy* and has the form:

$$P_1 \sqcup \ldots \sqcup P_n \tag{2}$$

where each P_i is a simple usage policy of the form (1). Symmetrically, with a similar union, data subjects may consent to different usage modalities for different categories of data and different purposes.

Example 3.1. A company – call it BeFit – sells a wearable fitness appliance and wants (i) to process biometric data (stored in the EU) for sending health-related advice to its customers, and (ii) share the customer's location data with their friends. Location data are kept for a minimum of one year but no longer than 5; biometric data are kept for an unspecified amount of time. In order to do all this legally, BeFit needs consent from its customers. The internal (formalized) description of such consent would look as follows:

$$
\begin{aligned}
&(\exists \text{has\_purpose.FitnessRecommendation} \sqcap \\
&\ \exists \text{has\_data.BiometricData} \sqcap \\
&\ \exists \text{has\_processing.Analytics} \sqcap \\
&\ \exists \text{has\_recipient.BeFit} \sqcap \\
&\ \exists \text{has\_storage.}\exists \text{has\_location.EU}) \\
&\sqcup \\
&(\exists \text{has\_purpose.SocialNetworking} \sqcap \\
&\ \exists \text{has\_data.LocationData} \sqcap \\
&\ \exists \text{has\_processing.Transfer} \sqcap \\
&\ \exists \text{has\_recipient.DataSubjFriends} \sqcap \\
&\ \exists \text{has\_storage.}(\exists \text{has\_location.EU} \sqcap \exists \text{has\_duration.}[y_1, y_5])\,.
\end{aligned}
\tag{3}
$$

Here y_1 and y_5 are the integer representation of one year and five years, respectively. If "HeartRate" is a subclass of "BiometricData" and "ComputeAvg" is a subclass of "Analytics", then the above consent allows BeFit to compute the average heart rate of the data subject in order to send her fitness recommendations. BeFit customers may restrict their consent, e.g. by picking a specific recommendation modality, like "recommendation via SMS only". Then the first line should be replaced with something like $\exists \text{has\_purpose.}(\text{FitnessRecommendation} \sqcap \exists \text{contact.SMS})$. Moreover, a customer of BeFit may consent to the first or the second argument of the union, or both. Then her consent would be encoded, respectively, with the first argument, the second argument, or the entire concept (3). Similarly, each single process in the controller's lines of business may use only biometric data, only location data, or both. Accordingly, it may be associated to the first simple policy, the second simple policy, or their union. In other words, (3) models the complete data usage activities related to the wearable device, that may be split across different processes. ■

The usage policies that are actually applied by the data controller's business processes are called *business policies* and include a description of data usage of the form (1). Additionally, each business policy is labeled with its legal basis and describes the associated obligations that must be fulfilled. For example, if the data category includes personal data, and processing is allowed by explicit consent, then the business policy should have the additional conjuncts:

$$
\begin{aligned}
&\exists \text{has\_legal\_basis.Art6\_1\_a\_Consent} \sqcap \\
&\ \exists \text{has\_duty.GetConsent} \sqcap \exists \text{has\_duty.GiveAccess} \sqcap \\
&\ \exists \text{has\_duty.RectifyOnRequest} \sqcap \\
&\ \exists \text{has\_duty.DeleteOnRequest}
\end{aligned}
\tag{4}
$$

that label the policy with the chosen legal basis, and model the obligations related to the data subjects' rights, cf. Chapter 3 of the GDPR. More precisely, the terms involving has_duty assert that the process modeled by the business policy includes the operations needed to obtain the data subject's consent ($\exists \text{has\_duty.GetConsent}$) and those needed to receive and apply the data subjects' requests to access, rectify, and delete their personal data.

Thus, business policies are an abstract description of a business process, highlighting the aspects related to compliance with the GDPR and data subjects' consent. Similarly to consent, a business policy may be a union $BP_1 \sqcup \ldots \sqcup BP_n$ of simple business policies BP_i of the form (1) ⊓ (4).

In order to check whether a business process complies with the consent given by a data subject S, it suffices to check whether the corresponding business policy BP is subsumed by the consent policy of S, denoted by CP_S (in symbols, $BP \sqsubseteq CP_S$). This subsumption is checked against a knowledge base that encodes type restrictions related to policy properties and the corresponding vocabularies, i.e. subclass relationships, disjointness constraints, functionality restrictions, domain and range restrictions, and the like. Some examples of the actual axioms occurring in the knowledge base are:

$$
\begin{aligned}
&\text{func(has\_purpose)} \\
&\text{range(has\_data, AnyData)} \\
&\text{Demographic} \sqsubseteq \text{AnyData} \\
&\text{Update} \sqsubseteq \text{AnyProcessing} \\
&\text{Erase} \sqsubseteq \text{Update} \\
&\text{disj(AnyData, AnyPurpose)}
\end{aligned}
$$

(recall that more general knowledge bases will be discussed later).

In order to verify that all the required obligations are fulfilled by a business process (as abstracted by the business policy), selected parts of the GDPR are formalized with concepts like the following. The first concept states that a business policy should either support the rights of the data subjects, or concern anonymous data, or it should fall under some of the exceptional cases mentioned by the regulation, such as particular law requirements. The remaining requirements are not listed here (they are replaced with an ellipsis):

$$
\begin{aligned}
&(\exists \text{has\_duty.GetConsent} \sqcap \exists \text{has\_duty.GiveAccess} \sqcap \ldots) \sqcup \\
&\ \exists \text{has\_data.Anonymous} \sqcup \\
&\ \exists \text{has\_purpose.LawRequirement} \sqcup \ldots
\end{aligned}
\tag{5}
$$

The second example encodes the constraints on data transfers specified in Articles 44–49 of the GDPR:

$$\begin{array}{l}\exists \mathsf{has\_storage}.\exists \mathsf{has\_location}.\mathsf{EU} \sqcup \\ \exists \mathsf{has\_storage}.\exists \mathsf{has\_location}.\mathsf{EULike} \sqcup \ldots\end{array} \tag{6}$$

It states that data should remain within the EU, or countries that adopt similar data protection regulations. The ellipsis stands for further concepts that model the other conditions under which data can be transferred to other nations (e.g. under suitable binding corporate rules). Please note that the above concepts constitute only a largely incomplete illustration of the actual formalization of the GDPR, that is significantly longer due to the special provisions that apply to particular data categories and legal bases. The purpose of the above examples is conveying the flavor of the formalization. Its usage is sketched below.

A business policy *BP* can be checked for compliance with the formalized parts of the GDPR by checking whether the aforementioned knowledge base entails that *BP* is subsumed by the concepts that formalize the GDPR.

Example 3.2. The following business policy complies with the consent-related obligations formalized in (5) since it is subsumed by it:

$$\begin{array}{l}(\exists \mathsf{has\_purpose}.\mathsf{FitnessRecommendation} \sqcap \\ \exists \mathsf{has\_data}.\mathsf{BiometricData} \sqcap \\ \exists \mathsf{has\_processing}.\mathsf{Analytics} \sqcap \\ \exists \mathsf{has\_recipient}.\mathsf{BeFit} \sqcap \\ \exists \mathsf{has\_storage}.\exists \mathsf{has\_location}.\mathsf{EU} \sqcap \\ \exists \mathsf{has\_legal\_basis}.\mathsf{Art6\_1\_a\_Consent}) \sqcap \\ \exists \mathsf{has\_duty}.\mathsf{GetConsent} \sqcap \ldots \textit{ all the remaining concepts in } (4) \ldots) \\ \sqcup \\ (\exists \mathsf{has\_purpose}.\mathsf{Sell} \sqcap \\ \exists \mathsf{has\_data}.\mathsf{Anonymous} \sqcap \\ \exists \mathsf{has\_processing}.\mathsf{Transfer} \sqcap \\ \exists \mathsf{has\_recipient}.\mathsf{ThirdParty})\,.\end{array} \tag{7}$$

In particular, the two disjuncts of (7) are subsumed by the first two lines of (5), respectively. Note that the second simple policy does not place any restrictions on location, so it allows data to flow to any country, including those that do not enjoy adequate data protection regulations. However, this is compliant with the GDPR because data are anonymous. ■

The concepts in the range of existential restrictions may themselves be a conjunction of atoms, interval constraints and existential restrictions. We have already seen in policy (3) that has_storage may contain a conjunction of existential restrictions over properties has_location and has_duration. Another example, related to SPECIAL's pilots, concerns the accuracy of locations, that can be modeled with concepts like:

$$\exists \mathsf{has\_data}.(\mathsf{Location} \sqcap \exists \mathsf{has\_accuracy}.\mathsf{Medium})\,.$$

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

Definition 3.3 *(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 property;
- $\mathsf{range}(S, A)$ 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.

Simple $\mathcal{PL}$ concepts are defined by the following grammar, where $A \in \mathsf{N_C}$, $R \in \mathsf{N_R}$, $f \in \mathsf{N_F}$, and l and u are integers:

$$C ::= A \mid \bot \mid \exists f\,.[l, u] \mid \exists R.C \mid C \sqcap C\,.$$

A *(full) $\mathcal{PL}$ concept* is a union $D_1 \sqcup \ldots \sqcup D_n$ of simple $\mathcal{PL}$ concepts ($n \geq 1$). $\mathcal{PL}$'s *subsumption queries* are inclusions $C \sqsubseteq D$ where C, D are (full) $\mathcal{PL}$ concepts.

3.1. Discussion of the encoding

The formalization of policies as *classes* of data usage modalities addresses several needs.

First, on the controller's side, each instance of a process may slightly differ from the others. For example, different instances of a same process may operate on data that are stored in different servers, possibly in different nations (this typically happens to large, international companies). The concrete data items involved may change slightly (e.g. age may be expressed directly or through the birth date; the data subject may be identified via a social security number (SSN), or an identity card number, or a passport number). By describing storage location, data, and the other policy attributes as classes, controllers can concisely describe an entire collection of similar process instances. With reference to the above examples, classes allow to express that data are stored "somewhere in the EU" and "in the controller's servers"; both age and birthdate fall under the class of demographic data; SSN and document numbers can be grouped under the class of unique identifiers.

A second advantage of classes is that they support a rather free choice of granularity. For example, the classes that model locations can be formulated at the granularity of continents, federations, countries, cities, zip-codes, down to buildings and rooms. Subsumption naturally models the containment of regions into other regions. A flexible choice of granularity helps in turning company documentation into formalized business policies, since it facilitates the import of the abstractions spontaneously used by domain experts.

The third, and perhaps most important advantage is that classes *facilitate the reuse of consent*. The GDPR sometimes allows to process personal data for a purpose other than that for which the data has been collected, provided that the new purpose is "compatible" with the initial purpose.[16] Compatibility cannot be assessed automatically, in general, because it is not defined in the regulation; only a human with specific legal background can deal reliably with the involved subtleties. However, by expressing purposes as classes, one can at least have the data subject consent upfront to a specified range of "similar" purposes. Roughly speaking, the accepted class of purposes is like an agreement – between data subjects and controllers – on which purposes are "compatible" in the given context. Also expressing the other policy properties as classes is beneficial. If data subjects consent to wider classes of usage modalities, then the need for additional consent requests tends to decrease; this may yield benefits to both parties, because:

1. data subjects are disturbed less frequently with consent requests (improved usability, better user experience);
2. the costs associated to consent requests decrease. Consider that sometimes the difficulties related to reaching out to the data subjects, and the concern that too many requests may annoy users, make controllers decide *not* to deliver a service that requires additional consent.

From a theoretical viewpoint, the class-based policy formalization adopted by SPECIAL is essentially akin to a well-established policy composition algebra [10]. The algebra treats policies as classes of authorizations (each policy P is identified with the set of authorizations permitted by P). In turn, authorizations are tuples that encode the essential elements of permitted operations, such as the resources involved and the kind of processing applied to those resources. Analogously, each $\mathcal{PL}$ policy like (1) denotes a set of reifications of tuples, whose elements capture the legally relevant properties of data usage operations.

3.2. Related policy languages

Logic-based languages constitute natural policy languages, because *policies are knowledge*. First, note that policies encode declarative constraints on a system's behavior, that depend on metadata about the actors and the objects involved (e.g. ownership, content categories), and an environment (as some operations may be permitted only in certain places, or at specified times of the day, or in case of emergency). Semantic languages and formats have been expressly designed to encode metadata, so standard knowledge representation languages can represent in a uniform way both policy constraints and the metadata they depend on.

The second important observation is that – like knowledge and unlike programs – every single policy is meant to be used for multiple, semantically related tasks, such as the following:

- *permission checking*: given an operation request, decide whether it is permitted;
- *compliance checking*: does a policy P_1 fulfill all the restrictions requested by policy P_2? (Policy comparison);
- *policy validation*: e.g. is the policy contradictory? Does it comply with a given regulation? Does a policy update strengthen or relax the previous policy?
- *policy explanation*: explain a policy and its decisions.

The terse formal semantics of logical languages is essential in validating the correctness of the policies themselves and the implementation of the above tasks, ensuring their mutual coherence. Moreover, when data are transferred under agreed policies, it is crucial that both parties understand the policies in the same way. So unambiguous semantics is essential for correct interoperability, too.

In the light of the above observations, it is clear that knowledge representation languages are ideal policy representation languages. Indeed, both rule languages and description logics have already been used as policy languages; a non-exhaustive

[16] See for example articles 5.1 (b) and 6.4.

list is [49,30,48,32,11]. As noted in [7], 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, because it is equivalent to Datalog query containment. So a DL-based policy language is a natural choice in a project like SPECIAL, where policy comparison is the predominant task.

The aforementioned works on logic-based policy languages 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 serious 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 [32,11] support logic program rules. Therefore, as we pointed out above, policy comparison is generally hard and possibly undecidable. This drawback makes such languages unsuitable to SPECIAL's purposes. Similarly, KAoS [48] is based on a DL that, in general, is not tractable, and supports role-value maps – a construct that easily makes reasoning undecidable (see [4], Chap. 5). The papers on KAoS do not discuss how the policy language is restricted to avoid this issue.

The terms used as role fillers in SPECIAL's policies are imported from well established formats for expressing privacy preferences and digital rights, such as P3P (the Platform for Privacy Preferences)[17] and ODRL (the Open Digital Right Language).[18] More general vocabularies will be discussed in Section 5. It is interesting to note that P3P's privacy policies – that are encoded in XML – are almost identical to simple $\mathcal{PL}$ policies: the tag `STATEMENT` contains tags `PURPOSE`, `RECIPIENT`, `RETENTION`, and `DATA-GROUP`, that correspond to the analogous properties of SPECIAL's usage policies. Only the information on the location of data is missing. The tag `STATEMENT` is included in a larger context that adds information about the controller (tag `ENTITY`) and about the space of web resources covered by the policy (through so-called *policy reference files*). All of these additional pieces of information can be directly encoded with simple $\mathcal{PL}$ concepts. Similar considerations hold for ODRL. The tag `RIGHTS` associates an `ASSET` (the analogue of has_data) to a `PERMISSION` that specifies a usage modality. ODRL provides terms for describing direct use (e.g. play or execute), reuse (e.g. annotate or aggregate), transfer (sell, lend, lease), and asset management operations (such as backup, install and delete, just to name a few). These terms provide a rich vocabulary for specifying the has_processing property of SPECIAL's policies. Also in the case of ODRL, the tree-like structure of XML documents can be naturally encoded with $\mathcal{PL}$ concepts.

3.3. Related work on legal reasoning

Despite superficial similarities, SPECIAL's policy framework and the many works on legal reasoning have different goals. The survey [44] lists several applications of logic and reasoning to the legal domain that can be grouped as follows:

a. Supporting the legislators in writing less ambiguous, possibly normalized legal documents.
b. Modeling legal concepts and definitions.
c. Interpreting the law.
d. Modeling the debates and pleadings that take place in courts, and deriving legal qualifications.

The work on vocabularies carried out by SPECIAL and the DPVCG can be regarded as a streamlined version of (b), while the use case of business policy validation based on GDPR's partial formalization does not really match any of the above points. Policy validation is less ambitious than legal reasoning; it is only aimed at checking whether the different properties of the policy are mutually coherent (e.g. by checking that the legal basis matches the data category), and whether all relevant parts have been included (such as the appropriate obligations in case of consent-based processing or data transfers outside the EU). The latter is a way to check whether the human responsible has "ticked all the necessary boxes", and by no means tackles the legal reasoning required to assess whether the obligations have been actually and appropriately fulfilled; according to our experience in SPECIAL, algorithms and ontologies are not yet trusted on this matter – especially due to the severe consequences in case of wrong decisions.

Both business policy validation and compliance checking w.r.t. consent policies shall verify that business policies do the right thing in *all contexts* while the literature on legal reasoning focuses on whether a legal qualification (such as an obligation, a permission, the validity of a contract, etc.) holds in a *specific situation* [44]. This difference has remarkable technical consequences: as we have already pointed out, validation in all contexts (i.e. policy comparison) is intractable in rule-based languages (that are common in the works on legal reasoning, since the seminal paper [45]), and even undecidable if rules are recursive [7]. The DL-based approach we adopted guarantees decidability and – under suitable hypotheses – tractability, as shown in the following sections.

The ambitious goals of legal reasoning have been tackled with sophisticated formalisms, such as deontic and nonmonotonic logics, see for example [31,29,1,24]. Fortunately, the different goals of SPECIAL's compliance checking make these complications unnecessary. Simplicity is strategic for the project, since the personnel that is expected to write the business

[17] http://www.w3.org/TR/P3P11.
[18] https://www.w3.org/TR/odrl/.

policies has no background on mathematical logic, deontic logic, nor nonmonotonic logic. The usability of SPECIAL's simple, form-like business policies has been successfully tested by one of the industrial partners of SPECIAL.

Formal ontologies based on DLs have been used to represent and reason over legal concepts, or as interchange formats to merge different sources of legal information. An example of legal ontology is LKIF Core [27] which has been developed in OWL1.1. LKIF Core follows a stratified approach defining an intentional layer on top of the legal layer. The intentional layer models different aspects of intentional behavior such as actions, plans, beliefs, and intentions. The legal layer concerns public acts, like norms, that have a legal relevance. Norms are further specialized in Right, Obligation, Permission, and Prohibition, that are coherently related to normative qualifications such as Allowed, Disallowed, Obliged, etc.

Recent work on legal reasoning – leveraging deontic and nonmonotonic logics – has been expressly tailored to the GDPR. The authors of [41,42] propose an ontology, PrOnto, aiming at supporting legal reasoning in general and compliance checking w.r.t. the GDPR. PrOnto defines a taxonomy of basic concepts and roles occurring in the GDPR and is organized in 5 distinct modules: Data and Documents, Agent and Role, Data Processing, Purposes and Legal Basis, and Deontic Operators.

In [40] PrOnto and LegalRuleML (a semantics-neutral interchange format) have been included in a larger architecture for developing GDPR-compliant cloud computing platforms for eGovernment. The graphic tool RAWE supports legal experts in translating legal text into formal rules which can be applied to a BPMN description of an eGov service. Compliance with the GDPR is then checked by the defeasible legal reasoning engine SPINdle. As a use case scenario, the article shows the formalization of Art. 8 of the GDPR concerning parental consent.

Differently from SPECIAL, the framework proposed in [40–42] pursues the more ambitious goals of legal reasoning and operates on a workflow-based representation of the controller's activities, more complex than business policies. These choices increase the cost of framework instantiation and rely on users with the necessary legal and logical background for editing and verifying legal rules – two assumptions that are not aligned with SPECIAL's reference scenarios. Furthermore, compliance checking is only static, and it does not address SPECIAL's need for real-time compliance checking with respect to the changing consent of data subjects. Summarizing, SPECIAL trades advanced legal reasoning capabilities for usability and scalability.

4. Reasoning with $\mathcal{PL}$

As explained in the introduction, some of the use cases of SPECIAL place challenging scalability requirements on the compliance checker, that should be able to execute over 10^4 subsumption checks per second, as in the streaming scenario. These scalability requirements have been addressed by finding a tradeoff between expressiveness and efficiency. The language $\mathcal{PL}$ – that is rich enough to encode the policies of interest – is also simple enough to be implemented very efficiently. Intervals, however, are a source of complexity and must be suitably restricted. In this section, we are first going to prove that unrestricted subsumption checking in $\mathcal{PL}$ is coNP-complete. Then we show that the structure of usage policies can be exploited to make restrictive assumptions on the occurrences of intervals. Under such assumptions, we can prove that an approach articulated in two stages – where first business policies are suitably normalized, then compliance with consent policies is checked with a structural subsumption algorithm – is correct, complete, and tractable. Its scalability will be experimentally assessed in Section 6.

We start by laying out the formal description and the theoretical properties of normalization and structural subsumption. In particular, this section deals with the correctness and completeness of the two-stages method, and discusses the computational complexity of arbitrary subsumptions and of the restricted, tractable case. We first prove the intractability of unrestricted subsumption in $\mathcal{PL}$.

Theorem 4.1. *Deciding whether $\mathcal{K} \models C \sqsubseteq D$, where $\mathcal{K}$ is a $\mathcal{PL}$ knowledge base and C, D are $\mathcal{PL}$ concepts, is coNP-hard.*[19] *This statement holds even if the knowledge base is empty and C is simple.*

Proof. 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 property names in $\mathcal{PL}$ concepts, and define a subsumption $C \sqsubseteq D$ that is valid iff S is unsatisfiable. Let $C = \left(\exists p_1.[0,1] \sqcap \ldots \sqcap \exists p_m.[0,1]\right)$ and $D = \bigsqcup_{i=1}^{n} \left(\tilde{L}_{i1} \sqcap \tilde{L}_{i2} \sqcap \tilde{L}_{i3}\right)$, where each $\tilde{L}_{ij}$ encodes the complement of L_{ij} as follows:

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

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 $I \models \mathsf{S}$ iff $(\mathcal{J}, d) \not\models D$. 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 in $\mathcal{J}$), $I \models \mathsf{S}$, which proves that if $C \sqsubseteq D$ is not valid, then S is satisfiable.

[19] As customary, we assume a positional representation of integers.

We conclude that the above reduction is correct. Moreover, it can be clearly computed in polynomial time. This proves that subsumption is coNP-hard even if the knowledge base is empty and C simple. ■

Later on we will complete the characterization of $\mathcal{PL}$ subsumption by proving its membership in coNP (Theorem 4.13).

The above intractability result does not apply to SPECIAL's usage policies because each simple usage policy contains at most one interval constraint, namely, a specification of storage duration of the form $\exists$has_storage.$\exists$has_duration.$[\ell, u]$. We are going to show that this property (actually, a slight generalization thereof) makes reasoning quite fast. More specifically, it enables an efficient treatment of interval constraints based on a suitable interval normalization method. Such normalization produces subsumption queries that satisfy the following property.

Definition 4.2 *(Interval safety).* An inclusion $C \sqsubseteq D$ is *interval safe* iff, for all constraints $\exists f.[\ell, u]$ occurring in C and all $\exists f'.[\ell', u']$ occurring in D, either $[\ell, u] \subseteq [\ell', u']$, or $[\ell, u] \cap [\ell', u'] = \emptyset$.

Roughly speaking, interval safety removes the need for treating intervals like disjunctions; it makes them behave like plain atomic concepts. Every inclusion can be turned into an equivalent, interval safe inclusion, using the following method.

Definition 4.3 *(Interval normalization, $split_D(C)$).* For each constraint $\exists f.[\ell, u]$ 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 $\exists f.[\ell, u]$ with the equivalent concept

$$\bigsqcup_{i=0}^{r} \left(\exists f.[x_i, x_i] \sqcup \exists f.[x_i + 1, x_{i+1} - 1]\right) \sqcup \exists f.[x_{r+1}, x_{r+1}]\,. \tag{8}$$

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 $split_D(C)$.

Example 4.4. Let $C = \exists f.[1, 9] \sqcap A$ and $D = \exists f.[5, 12]$. Then $r = 1$ and $x_0 = 1$, $x_1 = 5$, $x_2 = 9$ (12 falls outside $[1, 9]$ and is ignored). According to (8), the concept $\exists f.[1, 9]$ in C is replaced by the following union:

$$\exists f.[1, 1] \sqcup \exists f.[2, 4] \sqcup \exists f.[5, 5] \sqcup \exists f.[6, 8] \sqcup \exists f.[9, 9]\,.$$

Then, after applying distributivity, we obtain the concept $split_D(C)$ (that is a full $\mathcal{PL}$ concept):

$$(\exists f.[1, 1] \sqcap A) \sqcup (\exists f.[2, 4] \sqcap A) \sqcup (\exists f.[5, 5] \sqcap A) \sqcup (\exists f.[6, 8] \sqcap A) \sqcup (\exists f.[9, 9] \sqcap A)\,. \quad ■$$

The reader may easily verify that:

Proposition 4.5. *For all $\mathcal{PL}$ subsumption queries $C \sqsubseteq D$, $split_D(C)$ is equivalent to C and $split_D(C) \sqsubseteq D$ is an interval-safe $\mathcal{PL}$ subsumption query.*

In general, $split_D(C)$ may be exponentially larger than C, due to the application of distributivity (e.g. this happens with the concepts C and D in the proof of Theorem 4.1). However, as we have already pointed out, each simple policy has at most one, functional concrete property so no combinatorial explosion occurs during interval normalization. Accordingly – and more generally – the following proposition holds:

Proposition 4.6. *Let $C = C_1 \sqcup \ldots \sqcup C_n$ be a $\mathcal{PL}$ concept, and suppose that for all $i = 1, \ldots, n$, the number of concrete properties occurring in C_i is bounded by a constant c. Then, for all concepts D, the size of $split_D(C)$ is $O(|C| \cdot |D|^c)$.*[20]

Note that C as a whole may still contain an unbounded number of interval constraints, as n grows, because the bound c applies only to the individual disjuncts C_i.

The structural subsumption algorithm for $\mathcal{PL}$'s subsumption queries accepts subsumptions whose left-hand side is further *normalized with respect to the given knowledge base* $\mathcal{K}$ by exhaustively applying the rewrite rules illustrated in Table 3. Such rules make contradictions explicit and merge functional properties. They clearly preserve equivalence, as stated in the next proposition:

Proposition 4.7. *If $C \rightsquigarrow C'$ then $\mathcal{K} \models C \equiv C'$.*

[20] We denote the size of the encoding of an expression E with $|E|$.

Table 3
Normalization rules w.r.t. $\mathcal{K}$. Intersections are treated as sets (the ordering of conjuncts and their repetitions are irrelevant). $\sqsubseteq^*$ denotes the reflexive and transitive closure of $\{(A, B) \mid (A \sqsubseteq B) \in \mathcal{K}\}$.

1)	$\bot \sqcap D \rightsquigarrow \bot$	
2)	$\exists R.\bot \rightsquigarrow \bot$	
3)	$\exists f.[l, u] \rightsquigarrow \bot$	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)	$\exists f.[l_1, u_1] \sqcap \exists f.[l_2, u_2] \sqcap D \rightsquigarrow$ $\exists f.[\max(l_1, l_2), \min(u_1, u_2)] \sqcap D$	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 neither A nor $\bot$ are conjuncts of D
7)	$A_1 \sqcap A_2 \sqcap D \rightsquigarrow \bot$	if $A_1 \sqsubseteq^* A'_1$, $A_2 \sqsubseteq^* A'_2$, and $\mathsf{disj}(A'_1, A'_2) \in \mathcal{K}$

The proof is trivial and left to the reader. It is easy to see that concepts can be normalized in polynomial time:

Lemma 4.8. *Each $\mathcal{PL}$ concept C can be normalized w.r.t. a given $\mathcal{PL}$ knowledge base $\mathcal{K}$ in time $O(|C|^2 \cdot |\mathcal{K}|)$.*

Proof. We take this chance to illustrate an algorithm which is similar to the one actually used in the implementation of normalization. First C is parsed into a syntax tree T (time $O(|C|)$) where each conjunction of n concepts is modeled as a single node with n children. Then the tree is scanned in a depth-first fashion, looking for nodes labeled with an existential restriction in order to apply rule 4). For each such node ν, if R is the involved role and $\mathsf{func}(R) \in \mathcal{K}$, then the previous siblings of ν are searched looking for a node ν' with the same role R. If such a ν' is found, then the child C' of ν' is replaced with the intersection of C' itself and the child of ν, then ν is deleted. This operation (including the functionality test for R) takes time $O(|\mathcal{K}| + |C|)$ for each existential restriction. Thus, the exhaustive application of rule 4) needs time $O(|C| \cdot |\mathcal{K}| + |C|^2)$. Rule 5) is dealt with similarly (but instead of merging children, the interval associated to ν is intersected with the interval associated to ν'); the cost is the same. None of the other rules adds any new existential restrictions, so rules 4) and 5) are not going to be applicable again in the rest of the algorithm.

Next, rule 6) is applied by searching the tree T for existential restrictions whose role R occurs in an axiom $\mathsf{range}(R, A) \in \mathcal{K}$. For each of such nodes, A is added to the children as a new conjunct (if necessary). The cost for each existential restriction is $O(|\mathcal{K}| + |C|)$ (where $|C|$ is the cost of verifying whether the existential restriction already contains A or $\bot$). So the exhaustive application of rule 6) is again $O(|C| \cdot |\mathcal{K}| + |C|^2)$. The remaining rules can remove a range A only by substituting it with $\bot$, so rule 6) cannot be triggered again in the rest of the algorithm.

Finally, the nodes of T are visited in a depth-first fashion in order to apply rules 1), 2), 3), and 7).

Rule 7) is the most expensive. $\mathcal{K}$ is regarded as a labeled classification graph, where each node is labeled with an atomic concept and with the disjointness axioms in which that concept occurs. The disjointness test between A_1 and A_2 in rule 7) can be implemented by a relatively standard linear-time reachability algorithm, that climbs the classification graph from A_1 and starts descending the classification whenever it finds a node labeled with $\mathsf{disj}(A'_1, A'_2)$, searching for A_2. In the worst case, this stage involves $O(|C|^2)$ searches (one for each pair A_1, A_2 in each conjunction), so its global cost is $O(|C|^2 \cdot |\mathcal{K}|)$.

Finally, note that rules 1–3 do not need to be iteratively applied. If C contains an empty interval $[l, u]$ ($l > u$), or an occurrence of $\bot$, at any nesting level, then surely C can be rewritten to $\bot$. Therefore, it suffices to scan C once, looking for empty intervals or $\bot$.

Since the cost of rule 7 dominates the cost of the other rules, normalization can be computed in time $O(|C|^2 \cdot |\mathcal{K}|)$. ■

Normalized queries are passed over to a structural subsumption algorithm, called STS (Algorithm 1). It takes as inputs a $\mathcal{PL}$ knowledge base $\mathcal{K}$ and an *elementary* $\mathcal{PL}$ subsumption $C \sqsubseteq D$:

Definition 4.9 *(Elementary subsumptions).* A $\mathcal{PL}$ subsumption $C \sqsubseteq D$ is *elementary* (w.r.t. a $\mathcal{PL}$ knowledge base $\mathcal{K}$) if both C and D are simple, $C \sqsubseteq D$ is interval safe, and C is normalized w.r.t. $\mathcal{K}$.

The full subsumption checking procedure (that applies to *all* $\mathcal{PL}$ subsumptions) is called *$\mathcal{PL}$ Reasoner* (PLR for short). It is summarized in Algorithm 2.

PLR is correct and complete. We only state this result, whose proof is sketched in [8], since we are going to prove it in a more general form for an extended engine that supports more expressive knowledge bases (Section 5).

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

Input: A $\mathcal{PL}$ KB $\mathcal{K}$ and a $\mathcal{PL}$ subsumption $C \sqsubseteq D$ that is elementary w.r.t. $\mathcal{K}$
Output: `true` if $\mathcal{K} \models C \sqsubseteq D$, `false` otherwise

Note 1: Below, we treat intersections like sets. For example, by $C = C' \sqcap C''$ we mean that either $C = C'$ or C' is a conjunct of C (possibly not the first one).

Note 2: $\sqsubseteq^*$ denotes the reflexive and transitive closure of $\{(A, B) \mid (A \sqsubseteq B) \in \mathcal{K}\}$.

```
1 begin
2   if C = ⊥ then return true ;
3   if D = A, C = A' ⊓ C' and A' ⊑* A then return true ;
4   if D = ∃f.[l, u] and C = ∃f.[l', u'] ⊓ C' and l ≤ l' and u' ≤ u then return true ;
5   if D = ∃R.D', C = (∃R.C') ⊓ C'' and STS(K, C' ⊑ D') then return true ;
6   if D = D' ⊓ D'', STS(K, C ⊑ D'), and STS(K, C ⊑ D'') then return true ;
7   else return false ;
8 end
```

Algorithm 2: PLR($\mathcal{K}, C \sqsubseteq D$).

Input: A $\mathcal{PL}$ KB $\mathcal{K}$ and a $\mathcal{PL}$ subsumption query $C \sqsubseteq D$
Output: `true` if $\mathcal{K} \models C \sqsubseteq D$, `false` otherwise

```
1  begin
2    let C' be the normalization of C w.r.t. K (with the rules in Table 3) ;
3    let C'' = split_D(C') ;
     // assume that C'' = C_1 ⊔ ... ⊔ C_m and D = D_1 ⊔ ... ⊔ D_n
     // check whether each C_i is subsumed by some D_j
4    for i = 1, ..., m do
5      for j = 1, ..., n do
6        if STS(K, C_i ⊑ D_j) = true then skip to next i in outer loop;
7      end
8      return false
9    end
10   return true
11 end
```

Theorem 4.10. *For all $\mathcal{PL}$ knowledge bases $\mathcal{K}$ and all $\mathcal{PL}$ subsumption queries q,*

$$\mathcal{K} \models q \text{ iff } \mathsf{PLR}(\mathcal{K}, q) = \texttt{true}.$$

With this result, we can prove that subsumption checking in $\mathcal{PL}$ becomes tractable if the number of interval constraints per simple policy is bounded by a constant c (recall that in SPECIAL's policies $c = 1$). First we estimate the complexity of PLR.

Lemma 4.11. *For all $\mathcal{PL}$ knowledge bases $\mathcal{K}$ and all $\mathcal{PL}$ subsumption queries $C \sqsubseteq D$, $\mathsf{PLR}(\mathcal{K}, C \sqsubseteq D)$ can be computed in time $O(|C \sqsubseteq D|^{c+1} + |C \sqsubseteq D|^2 \cdot |\mathcal{K}|)$, where c is the maximum number of interval constraints occurring in a single simple concept of C.*

Proof. By Lemma 4.8 and Proposition 4.6, respectively, the complexity of line 2 of PLR is $O(|C|^2 \cdot |\mathcal{K}|)$ and the complexity of line 3 is $O(|C| \cdot |D|^c) = O(|C \sqsubseteq D|^{c+1})$. Now consider the complexity of the calls $\mathsf{STS}(\mathcal{K}, C_i \sqsubseteq D_j)$ in line 6. Each of them, in the worst case, scans C_i once for each subconcept of D_j, searching for a matching concept. Matching may require to solve a reachability problem on the hierarchy $\sqsubseteq^*$, so the cost of each call is $O(|D_j| \cdot |C_i| \cdot |\mathcal{K}|)$. If we focus on the outer loop (lines 4–9) then clearly each subconcept of D is matched against all disjuncts of C, in the worst case. Then the overall cost of the outer loop is $O(|D| \cdot |C| \cdot |\mathcal{K}|)$. By relating these parameters to the size of the query, it follows that the cost of the outer loop is bounded by $O(|C \sqsubseteq D|^2 \cdot |\mathcal{K}|)$. This dominates the cost of line 2. So we conclude that the overall time needed by PLR in the worst case is $O(|C \sqsubseteq D|^{c+1} + |C \sqsubseteq D|^2 \cdot |\mathcal{K}|)$. ■

Tractability immediately follows from Theorem 4.10 and Lemma 4.11:

Theorem 4.12. *Let c be an integer, and $\mathcal{Q}_c$ be the set of all $\mathcal{PL}$ subsumptions $C_1 \sqcup \ldots \sqcup C_n \sqsubseteq D$ such that each C_i contains at most c interval constraints ($i = 1, \ldots, n$). Then deciding whether a query in $\mathcal{Q}_c$ is entailed by a $\mathcal{PL}$ knowledge base $\mathcal{K}$ is in* P.

We conclude this section by completing the characterization of the complexity of unrestricted $\mathcal{PL}$ subsumptions. The following result, together with Theorem 4.1, proves that $\mathcal{PL}$ subsumption is coNP-complete.

Theorem 4.13. *Deciding whether $\mathcal{K} \models C \sqsubseteq D$, where $\mathcal{K}$ is a $\mathcal{PL}$ knowledge base and C, D are (simple or full) $\mathcal{PL}$ concepts, is in co*NP.

Proof. We prove the theorem by showing that the complement of subsumption is in NP. For this purpose, 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 $\exists f.[\ell, u]$ occurring in C_i with a nondeterministically chosen disjunct from (8). Call C'_i the resulting concept and note that it is one of the disjuncts in $split_D(C)$. Therefore, $\mathcal{K} \not\models C \sqsubseteq D$ iff $\mathcal{K} \not\models split_D(C) \sqsubseteq D$ iff, for some nondeterministic choice of C'_i, $\mathcal{K} \not\models C'_i \sqsubseteq D$. Note that $C'_i \sqsubseteq D$ is interval-safe by construction. Then this subsumption test can be evaluated in deterministic polynomial time by first normalizing C'_i w.r.t. $\mathcal{K}$ and then applying STS, that is complete for elementary queries [8, Theorem 2]. It follows immediately that the complement of $\mathcal{PL}$ subsumption can be decided in nondeterministic polynomial time, hence its membership in NP. ■

5. Supporting general vocabularies

SPECIAL has founded the "*Data Privacy Vocabularies and Controls Community Group*" (DPVCG),[21] a W3C group aimed at developing privacy-related vocabularies. The purpose of this initiative is developing ontologies for the main properties of usage policies and related GDPR concepts, with the contribution of a group of stakeholders that spans beyond SPECIAL's consortium. This group aims at developing upper ontologies, that can be later extended to meet the needs of specific application domains.

We intend to put as few constraints as possible on the development of such standardized vocabularies, since it is difficult to predict the expressiveness needs that may arise in their modeling – especially because standards usually change to include new application domains and follow the evolution of the old ones. $\mathcal{PL}$ knowledge bases are too simple to address this requirement. We already have evidence that it is useful to have roles whose domain is a vocabulary term, such as the accuracy of locations (cf. Section 3); so, in perspective, we should expect the ontologies that define privacy-related vocabularies to include at least existential restrictions (that cannot be used in $\mathcal{PL}$ knowledge bases, but are supported – say – by the tractable profiles of OWL2). It is hard to tell which other constructs will turn out to be useful.

For the above reasons, we are going to show how to integrate $\mathcal{PL}$ and its specialized reasoner with a wide range of ontologies, expressed with description logics that can be significantly more expressive than $\mathcal{PL}$.

Our strategy consists in treating such ontologies – hereafter called *external ontologies* – as *oracles*. Roughly speaking, whenever STS needs to check a subsumption between two terms defined in the external ontologies, the subsumption query is submitted to the oracle. In the easiest case, the queries to the oracle can be answered with a simple visit to the classification graph of the vocabularies. Of course this method, called *import by query* (IBQ), is not always complete [21,20]. In this section, we provide sufficient conditions for completeness.

More formally, let $\mathcal{K}$ and $\mathcal{O}$ be two given knowledge bases. The former will be called the *main KB*, and may use terms that are axiomatized in $\mathcal{O}$, that plays the role of the external ontology. For example, in SPECIAL's policy modeling scenario, $\mathcal{K}$ defines policy attributes – by specifying their ranges and functionality properties – while $\mathcal{O}$ defines the privacy-related vocabularies that provide the fillers for policy attributes. Therefore, in SPECIAL's framework, $\mathcal{K}$ is a $\mathcal{PL}$ knowledge base, while $\mathcal{O}$ could be formulated with a more expressive DL. The reasoning task of interest in such scenarios is deciding, for a given subsumption query $q = (C \sqsubseteq D)$, whether $\mathcal{K} \cup \mathcal{O} \models q$. Both C and D are $\mathcal{PL}$ concepts that usually contain occurrences of concept names defined in $\mathcal{O}$.

SPECIAL's application scenarios make it possible to adopt a simplifying assumption that makes oracle reasoning technically simpler [21,20], namely, we assume that neither $\mathcal{K}$ nor the query q shares any roles with $\mathcal{O}$. This naturally happens in SPECIAL precisely because the roles used in the main KB identify the sections that constitute a policy (e.g. data categories, purpose, processing, storage, recipients), while the roles defined in $\mathcal{O}$ model the *contents* of those sections, e.g. anonymization parameters, relationships between recipients (like ownership, employment relations), relationships between storage locations (e.g. part-of relations), and the like.

This layered structure does not require arbitrary alternations of roles coming from the main KB and from the external ontologies (more precisely, the roles occurring in the main KB need not occur within the scope of any role in $\Sigma(\mathcal{O})$). As a consequence, the roles of the external ontology can be allowed in the queries as syntactic sugar, as explained in the following.

Remark 5.1. Let q be a $\mathcal{PL}$ subsumption query, and $R_{\mathcal{O}}$ range over the roles occurring in $\mathcal{O}$. According to the above discussion, assume that for all concept of the form $\exists R_{\mathcal{O}}.C$ occurring in q, C contains only roles from $\Sigma(\mathcal{O})$ (no alternation of roles from the main KB and $\mathcal{O}$). Every such concept $\exists R_{\mathcal{O}}.C$ can be eliminated from q by replacing it with a fresh atom A, and extending $\mathcal{O}$ with the axiom $A \equiv \exists R_{\mathcal{O}}.C$, under the mild assumption that the language of $\mathcal{O}$ supports such

[21] www.w3.org/community/dpvcg/.

equivalences. Let q' and $\mathcal{O}'$ be the query and the ontology obtained by applying the above transformation for all $R_{\mathcal{O}} \in \Sigma(\mathcal{O})$. Clearly, by construction, $\mathcal{O}'$ implies that the resulting query q' is equivalent to q. Moreover, $\Sigma(\mathcal{O}') \cap \mathsf{N_R} = \Sigma(\mathcal{O}) \cap \mathsf{N_R}$ holds by the assumption that the concepts C in $\exists R_{\mathcal{O}}.C$ contain only roles from $\Sigma(\mathcal{O})$. Now it is easy to see that the requirement that the main KB should share no roles with $\mathcal{O}'$ is preserved by the transformation, since the main KB is not affected and $\Sigma(\mathcal{O}') \cap \mathsf{N_R} = \Sigma(\mathcal{O}) \cap \mathsf{N_R}$. Due to the same equality, q' contains no roles in $\Sigma(\mathcal{O}')$ because, by construction, it contains no roles in $\Sigma(\mathcal{O})$. Summarizing, every query q where the roles in $\Sigma(\mathcal{K})$ do not occur within the scope of the roles in $\Sigma(\mathcal{O})$ can be transformed in polynomial time into an equivalent q' that satisfies the requirement on role sharing, by means of a simple extension of $\mathcal{O}$.

5.1. On the completeness of IBQ reasoning

The IBQ framework was introduced to reason with a partly hidden ontology $\mathcal{O}$. For our purposes, IBQ is interesting because instead of reasoning on $\mathcal{K} \cup \mathcal{O}$ as a whole, each of the two parts can be processed with a different reasoner (so, in particular, policies can be compared with a very efficient algorithm similar to STS). The reasoner for $\mathcal{K}$ may query $\mathcal{O}$ as an oracle, using a query language $\mathcal{QL}$ consisting of all the subsumptions

$$A_1 \sqcap \ldots \sqcap A_m \sqsubseteq A_{m+1} \sqcup \ldots \sqcup A_n \tag{9}$$

such that $A_1, \ldots, A_n$ are concept names. If $n = m$, then we stipulate that the right-hand side of the inclusion is $\bot$. We will denote with $pos(\mathcal{O})$ all the queries to $\mathcal{O}$ that have a positive answer, that is:

$$pos(\mathcal{O}) = \{q \in \mathcal{QL} \mid \mathcal{O} \models q\}\,.$$

Remark 5.2. Each subsumption of the form (9) is equivalent to a concept (in)consistency check of the form:

$$A_1 \sqcap \ldots \sqcap A_m \sqcap \neg A_{m+1} \sqcap \ldots \sqcap \neg A_n \sqsubseteq \bot\,. \tag{10}$$

By [21, Theorem 2], such consistency checks (and, consequently, $\mathcal{QL}$) constitute a fully general oracle query language, under the assumption that $\mathcal{K}$ and the query q share no roles with $\mathcal{O}$. By "fully general" we mean that it can be decided whether $\mathcal{K} \cup \mathcal{O} \models q$ holds using only the axioms of $\mathcal{K}$ *and the members of* $pos(\mathcal{O})$.

The problem instances we are interested in are formally defined by the next definition.

Definition 5.3 *($\mathcal{PL}$ subsumption instances with oracles, $\mathcal{PLSO}$).* A *$\mathcal{PL}$ subsumption instance with oracle* is a triple $\langle \mathcal{K}, \mathcal{O}, q \rangle$ where $\mathcal{K}$ is a $\mathcal{PL}$ knowledge base (the *main knowledge base*), $\mathcal{O}$ is a Horn-$\mathcal{SRIQ}$ knowledge base (the *oracle*), and q is a $\mathcal{PL}$ subsumption query, such that $(\Sigma(\mathcal{K}) \cup \Sigma(q)) \cap \Sigma(\mathcal{O}) \subseteq \mathsf{N_C}$. The set of all $\mathcal{PL}$ subsumption instances with oracle will be denoted by $\mathcal{PLSO}$.

The restrictions on $\mathcal{K}$, $\mathcal{O}$ and q will be motivated in depth in Section 5.5. We anticipate only two observations. First, the restriction on the signatures is aimed at keeping the roles of $\mathcal{O}$ separated from those of $\mathcal{K}$ and q, as discussed in the previous section. The second observation is that the important properties of $\mathcal{O}$ are the absence of nominals and its convexity with respect to $\mathcal{QL}$, in the following sense:

Definition 5.4 *(Convexity w.r.t. $\mathcal{QL}$).* A knowledge base $\mathcal{O}$ is *convex w.r.t.* $\mathcal{QL}$ if for all subsumptions

$$q = A_1 \sqcap \ldots \sqcap A_m \sqsubseteq A_{m+1} \sqcup \ldots \sqcup A_n$$

in $\mathcal{QL}$, $q \in pos(\mathcal{O})$ iff there exists $i \in [m+1, n]$ such that $(A_1 \sqcap \ldots \sqcap A_m \sqsubseteq A_i) \in pos(\mathcal{O})$. A description logic is convex w.r.t. $\mathcal{QL}$ if all of its knowledge bases are convex w.r.t. $\mathcal{QL}$.

Accordingly, in Definition 5.3, we required $\mathcal{O}$ to be in Horn-$\mathcal{SRIQ}$ because, to the best of our knowledge, this is the most expressive description logic considered so far in the literature that is both nominal-free and convex w.r.t. $\mathcal{QL}$.

The next lemma rephrases the original IBQ completeness result [21, Lemma 1] in our notation. Our statement relaxes the requirements on $\mathcal{O}$ by assuming only that it enjoys the disjoint model union property (originally it had to be in $\mathcal{SRIQ}$). The proof, however, remains essentially the same.

Lemma 5.5. *Let $\mathcal{K}$ and $\mathcal{O}$ be knowledge bases and α a GCI, such that*

1. *$\mathcal{K}$ and α are in $\mathcal{SROIQ}(\mathsf{D})$ without U, where $\mathcal{D}$ is the concrete domain of integer intervals;*
2. *The terminological part of $\mathcal{O}$ enjoys the disjoint model union property;*

3. *The terminological part* $\mathcal{T}$ *of* $\mathcal{K}$ *is local w.r.t.* $\Sigma(\mathcal{O})$;
4. $(\Sigma(\mathcal{K}) \cup \Sigma(\alpha)) \cap \Sigma(\mathcal{O}) \subseteq \mathsf{N_C}$.

Then $\mathcal{K} \cup \mathcal{O} \models \alpha$ *iff* $\mathcal{K} \cup pos(\mathcal{O}) \models \alpha$.

Proof. We have to prove that under the above hypotheses $\mathcal{K} \cup \mathcal{O} \models \alpha$ iff $\mathcal{K} \cup pos(\mathcal{O}) \models \alpha$. The right-to-left direction is trivial since by definition $\mathcal{O} \models pos(\mathcal{O})$. For the other direction, by contraposition, assume that $\mathcal{K} \cup pos(\mathcal{O}) \not\models \alpha$. We shall find a model $\mathcal{N}$ of $\mathcal{K} \cup \mathcal{O}$ such that $\mathcal{N} \not\models \alpha$. Since α is of the form $C \sqsubseteq D$, this means that for some $\bar{d} \in \Delta^{\mathcal{N}}$, $\bar{d} \in (C \sqcap \neg D)^{\mathcal{N}}$. The construction is similar to that used in [21, Lemma 1].

By assumption, $\mathcal{K} \cup pos(\mathcal{O})$ has a model $\mathcal{I}$ such that $\mathcal{I} \not\models \alpha$, that is, there exists $\bar{d} \in \Delta^{\mathcal{I}}$ such that $\bar{d} \in (C \sqcap \neg D)^{\mathcal{I}}$. Now we extend the interpretation $\mathcal{I}$ over $\Sigma(\mathcal{K}) \cup \Sigma(\alpha)$ to a model $\mathcal{N}$ of $\mathcal{K} \cup \mathcal{O}$.

We need some auxiliary notation: for each $d \in \Delta^{\mathcal{I}}$, let $lit(d, \mathcal{I})$ denote the set of all the literals L in the language of $\mathcal{O}$ satisfied by d, that is,

$$lit(d, \mathcal{I}) = \{L \mid (\mathcal{I}, d) \models L \text{ and either } L = A \text{ or } L = \neg A, \text{ where } A \in \mathsf{N_C} \cap \Sigma(\mathcal{O})\}.$$

Since $\mathcal{I} \models pos(\mathcal{O})$, it follows that for all $d \in \Delta^{\mathcal{I}}$, $\mathcal{O} \not\models \bigsqcap lit(d, \mathcal{I}) \sqsubseteq \bot$ (see Remark 5.2). Then, for all $d \in \Delta^{\mathcal{I}}$, there exists a pointed interpretation $(\mathcal{J}_d, d)$ of $\Sigma(\mathcal{O})$ such that $\mathcal{J}_d \models \mathcal{O}$ and $lit(d, \mathcal{J}_d) = lit(d, \mathcal{I})$. We may assume without loss of generality that $\Delta^{\mathcal{J}_d} \cap \Delta^{\mathcal{I}} = \{d\}$ and that $\Delta^{\mathcal{J}_d} \cap \Delta^{\mathcal{J}_{d'}} = \emptyset$ if $d \neq d'$.

Let $\mathcal{J}$ be any of the above $\mathcal{J}_d$ and $\mathcal{U} = \biguplus^{\mathcal{J}} \{\mathcal{J}_d \mid d \in \Delta^{\mathcal{I}}\}$. By hypothesis 2 and Proposition 2.2, $\mathcal{U}$ is a model of $\mathcal{O}$. Moreover, by hypothesis 3, $\mathcal{U}$ can be extended to a model $\mathcal{M}$ of $\mathcal{T}$, by setting $X^{\mathcal{M}} = \emptyset$ for all predicates $X \in (\Sigma(\mathcal{K}) \cup \Sigma(\alpha)) \setminus \Sigma(\mathcal{O})$.

Finally, let $\mathcal{N}$ be the interpretation such that:

$$\Delta^{\mathcal{N}} = \Delta^{\mathcal{M}} \qquad \text{(note that } \Delta^{\mathcal{I}} \subseteq \Delta^{\mathcal{M}})$$
$$X^{\mathcal{N}} = \begin{cases} X^{\mathcal{I}} & \text{for all symbols } X \in (\Sigma(\mathcal{K}) \cup \Sigma(\alpha)) \setminus \Sigma(\mathcal{O}) \\ X^{\mathcal{M}} & \text{for all symbols } X \in \Sigma(\mathcal{O}). \end{cases}$$

The next part of the proof proceeds exactly as in [21, Lemma 1], in order to show that $\mathcal{N} \models \mathcal{K} \cup \mathcal{O}$. Note that by definition $\mathcal{M}$ and $\mathcal{N}$ have the same domain and agree on the symbols in $\Sigma(\mathcal{O})$, therefore $\mathcal{N}$ is a model of $\mathcal{O}$ because $\mathcal{M}$ is. So one is only left to prove that $\mathcal{N} \models \mathcal{K}$. For this purpose, first it is proved that

($\star$) for all C in the closure[22] of $\mathcal{K}$ and α, $C^{\mathcal{N}} = C^{\mathcal{I}} \cup (C^{\mathcal{M}} \setminus \Delta^{\mathcal{I}})$.

The proof of ($\star$) makes use of hypotheses 1 and 4. Then, using ($\star$) and the fact that $\mathcal{M}$ is a model of $\mathcal{T}$, it can be shown that $\mathcal{M}$ is a model of $\mathcal{K}$. Almost all details of the proof of ($\star$) and $\mathcal{M} \models \mathcal{K}$ can be found in [21]. Here we only have to add the details for ($\star$) concerning interval constraints (that are not considered in [21]). Let $C = \exists f.[l, u]$. By hypothesis 4, $f \in (\Sigma(\mathcal{K}) \cup \Sigma(\alpha)) \setminus \Sigma(\mathcal{O})$. Then, by definition of $\mathcal{N}$ and $\mathcal{M}$, $f^{\mathcal{N}} = f^{\mathcal{I}}$ and $f^{\mathcal{M}} = \emptyset$. Consequently, $C^{\mathcal{N}} = C^{\mathcal{I}}$ and $C^{\mathcal{M}} = \emptyset$, so ($\star$) obviously holds.

For our formulation of this theorem, we only have to add the observation that ($\star$) implies also that $\bar{d} \in (C \sqcap \neg D)^{\mathcal{I}} \subseteq (C \sqcap \neg D)^{\mathcal{N}}$, therefore $\mathcal{N} \not\models \alpha$. ■

Using the above lemma, we prove a variant of IBQ completeness for $\mathcal{PLSO}$. The locality requirement of Lemma 5.5 is removed by shifting axioms from $\mathcal{K}$ to $\mathcal{O}$.

Theorem 5.6. *For all problem instances* $\pi = \langle \mathcal{K}, \mathcal{O}, q \rangle \in \mathcal{PLSO}$, *let*

$$\mathcal{K}^- = \{\alpha \in \mathcal{K} \mid \alpha = \mathsf{range}(R, A) \text{ or } \alpha = \mathsf{func}(R)\}$$

and let $\mathcal{O}^+_{\mathcal{K}} = \mathcal{O} \cup (\mathcal{K} \setminus \mathcal{K}^-)$. *Then*

$$\mathcal{K} \cup \mathcal{O} \models q \text{ iff } \mathcal{K}^- \cup pos(\mathcal{O}^+_{\mathcal{K}}) \models q.$$

Proof. Since $\mathcal{K} \cup \mathcal{O} = \mathcal{K}^- \cup \mathcal{O}^+_{\mathcal{K}}$, it suffices to show that

$$\mathcal{K}^- \cup \mathcal{O}^+_{\mathcal{K}} \models q \text{ iff } \mathcal{K}^- \cup pos(\mathcal{O}^+_{\mathcal{K}}) \models q.$$

This equivalence can be proved with Lemma 5.5; it suffices to show that $\mathcal{K}^-$, $\mathcal{O}^+_{\mathcal{K}}$ and q satisfy the hypotheses of the lemma. First note that $\mathcal{K}^-$ is a $\mathcal{PL}$ knowledge base and $\mathcal{O}^+_{\mathcal{K}}$ is a Horn-$\mathcal{SRIQ}$ knowledge base (because, by definition of

[22] The closure of a set of axioms $\mathcal{K}$ is the set of all (sub)concepts occurring in $\mathcal{K}$.

Table 4
Normalization rules for $\mathsf{STS}^{\mathcal{O}^+_\mathcal{K}}$. Conjunctions are treated as sets (i.e. the ordering of conjuncts is irrelevant, and duplicates are removed).

1)	$\bot \sqcap D \rightsquigarrow \bot$	
2)	$\exists R.\bot \rightsquigarrow \bot$	
3)	$\exists f.[l,u] \rightsquigarrow \bot$	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)	$\exists f.[l_1,u_1] \sqcap \exists f.[l_2,u_2] \sqcap D \rightsquigarrow \exists f.[\max(l_1,l_2), \min(u_1,u_2)] \sqcap D$	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 neither A nor $\bot$ are conjuncts of D
7)	$A_1 \sqcap \ldots \sqcap A_n \sqcap D \rightsquigarrow \bot$	if $\mathcal{O}^+_\mathcal{K} \models A_1 \sqcap \ldots \sqcap A_n \sqsubseteq \bot$

$\mathcal{PLSO}$, $\mathcal{K}$ is in $\mathcal{PL}$ and $\mathcal{O}$ in Horn-$\mathcal{SRIQ}$, and the axioms shifted from $\mathcal{L}$ to $\mathcal{O}^+_\mathcal{K}$ can be expressed in Horn-$\mathcal{SRIQ}$, too). Both $\mathcal{PL}$ and Horn-$\mathcal{SRIQ}$ are fragments of $\mathcal{SROIQ}(\mathsf{D})$ without U, therefore hypothesis 1 is satisfied by $\mathcal{K}^-$ and $\mathcal{O}^+_\mathcal{K}$. Moreover, both $\mathcal{PL}$ and Horn-$\mathcal{SRIQ}$ enjoy the disjoint model union property, therefore hypothesis 2 is satisfied. Next recall that $(\Sigma(\mathcal{K}) \cup \Sigma(q)) \cap \Sigma(\mathcal{O}) \subseteq \mathsf{N_C}$ holds, by definition of $\mathcal{PLSO}$. Since the axioms $\alpha \in \mathcal{K} \setminus \mathcal{K}^-$ (transferred from $\mathcal{K}$ to $\mathcal{O}^+_\mathcal{K}$) contain no roles (they are of the form $A \sqsubseteq B$ or $\mathsf{disj}(A,B)$), it follows that

$$(\Sigma(\mathcal{K}^-) \cup \Sigma(q)) \cap \Sigma(\mathcal{O}^+_\mathcal{K}) \subseteq \mathsf{N_C}\,,$$

that is, hypothesis 4 holds. A second consequence of this inclusion is that $\mathcal{K}^-$ contains only axioms of the form $\mathsf{range}(R,A)$ and $\mathsf{func}(A)$ such that $R \notin \Sigma(\mathcal{O}^+_\mathcal{K})$. They are trivially satisfied by any interpretation $\mathcal{I}$ such that $R^\mathcal{I} = \emptyset$. Therefore $\mathcal{K}^-$ is local w.r.t. $\Sigma(\mathcal{O}^+_\mathcal{K})$ and hypothesis 3 is satisfied. ■

5.2. Extending $\mathcal{PL}$'s reasoner with IBQ capabilities

The integration of the PLR reasoner with external oracles relies on the axiom shifting applied in Theorem 5.6. Accordingly, in the following, let $\mathcal{K}^-$ and $\mathcal{O}^+_\mathcal{K}$ be defined as in Theorem 5.6.

The next step after axiom shifting consists in replacing the relation $\sqsubseteq^*$ used by the normalization rules and STS with suitable queries to the oracle. This change concerns the normalization rules (Table 3) and STS. The new set of rules is given in Table 4. We say that a $\mathcal{PL}$ concept C is *normalized w.r.t. $\mathcal{K}$ and $\mathcal{O}$* if none of the rules in Table 4 is applicable.

Hereafter, $\rightsquigarrow$ denotes the rewriting relation according to Table 4. Clearly, the new rules preserve the meaning of concepts, in the following sense:

Proposition 5.7. *If $C \rightsquigarrow C'$ then $\mathcal{K}^- \cup pos(\mathcal{O}^+_\mathcal{K}) \models C \equiv C'$.*

The notion of elementary inclusion is modified accordingly, by requiring normalization w.r.t. both $\mathcal{K}$ and $\mathcal{O}$.

Definition 5.8. A $\mathcal{PL}$ subsumption $C \sqsubseteq D$ is *elementary w.r.t. $\mathcal{K}$ and $\mathcal{O}$* if both C and D are simple, $C \sqsubseteq D$ is interval safe, and C is normalized w.r.t. $\mathcal{K}$ and $\mathcal{O}$.

Then STS is integrated with the oracle $\mathcal{O}$ by replacing its line 3 as in the following algorithm $\mathsf{STS}^{\mathcal{O}^+_\mathcal{K}}$. In the following, we call a subconcept "*top level*" if it does not occur in the scope of any existential restriction.

Algorithm 3: $\mathsf{STS}^{\mathcal{O}^+_\mathcal{K}}(C \sqsubseteq D)$.

Input: Ontology $\mathcal{O}^+_\mathcal{K}$ (as defined in Theorem 5.6) and a $\mathcal{PL}$ subsumption $C \sqsubseteq D$ that is elementary w.r.t. $\mathcal{K}$ and $\mathcal{O}$
Output: `true` if $\mathcal{K}^- \cup pos(\mathcal{O}^+_\mathcal{K}) \models C \sqsubseteq D$, `false` otherwise, where $\mathcal{K}^-$ is defined as in Theorem 5.6

```
1 begin
2   if C = ⊥ then return true ;
3   if D = A and (A_1 ⊓ ... ⊓ A_n ⊑ A) ∈ pos(O^+_K), where A_1, ..., A_n are the top-level concept names in C then return true ;
4   if D = ∃f.[l,u] and C = ∃f.[l',u'] ⊓ C' and l ≤ l' and u' ≤ u then return true ;
5   if D = ∃R.D', C = (∃R.C') ⊓ C'' and STS^{O^+_K}(C' ⊑ D') then return true ;
6   if D = D' ⊓ D'', STS^{O^+_K}(C ⊑ D'), and STS^{O^+_K}(C ⊑ D'') then return true ;
7   else return false ;
8 end
```

Finally, the reasoner for general $\mathcal{PL}$ subsumptions with oracles can be defined as follows:

Algorithm 4: $\mathsf{PLR}^{\mathcal{O}}(\mathcal{K}, C \sqsubseteq D)$.

Input: $\mathcal{O}$, $\mathcal{K}$ and $C \sqsubseteq D$ such that $\pi = \langle \mathcal{K}, \mathcal{O}, C \sqsubseteq D \rangle \in \mathcal{PLSO}$
Output: `true` if $\mathcal{K} \cup \mathcal{O} \models C \sqsubseteq D$, `false` otherwise

```
1  begin
2      construct K⁻ and O⁺_K as defined in Theorem 5.6 ;
3      let C' be the normalization of C w.r.t. K and O (with the rules in Table 4) ;
4      let C'' = split_D(C') ;
       // assume that C'' = C_1 ⊔ ... ⊔ C_m and D = D_1 ⊔ ... ⊔ D_n
       // check whether each C_i is subsumed by some D_j
5      for i = 1, ..., m do
6          for j = 1, ..., n do
7              if STS^{O⁺_K}(C_i ⊑ D_j) = true then skip to next i in outer loop;
8          end
9          return false
10     end
11     return true
12 end
```

The rest of this section is devoted to proving the soundness and completeness of $\mathsf{PLR}^{\mathcal{O}}$. We will need a set of canonical counterexamples to invalid subsumptions.

Definition 5.9. Let $C \neq \bot$ be a simple $\mathcal{PL}$ concept normalized w.r.t. $\mathcal{K}$ and $\mathcal{O}$. A *canonical model* of C (w.r.t. $\mathcal{K}$ and $\mathcal{O}$) is a pointed interpretation $(\mathcal{I}, d)$ defined as follows, by induction on the number of existential restrictions.

a. If $C = \left(\prod_{i=1}^{n} A_i\right) \sqcap \left(\prod_{j=1}^{t} \exists f_j.[l_j, u_j]\right)$ (i.e. C has no existential restrictions), then let $\mathcal{I} = \langle \{d\}, \cdot^{\mathcal{I}} \rangle$ where
   - $A^{\mathcal{I}} = \{d\}$ if $\left(\prod_{i=1}^{n} A_i \sqsubseteq A\right) \in pos(\mathcal{O}^{+}_{\mathcal{K}})$;
   - $f^{\mathcal{I}} = \{(d, u_j) \mid j = 1, \ldots t\}$;
   - 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 for each $i = 1, \ldots, m$, let $(\mathcal{I}_i, d_i)$ be a canonical model of D_i. Assume w.l.o.g. that all such models are mutually disjoint and do not contain d. Define an auxiliary interpretation $\mathcal{J}$ as follows:
   - $\Delta^{\mathcal{J}} = \{d, d_1, \ldots, d_m\}$;
   - $A^{\mathcal{I}} = \{d\}$ if $\left(\prod_{i=1}^{n} A_i \sqsubseteq A\right) \in pos(\mathcal{O}^{+}_{\mathcal{K}})$, where $A_1, \ldots, A_n$ are the top-level concept names in C; all other concept names are empty;
   - $f^{\mathcal{J}} = \{(d, u) \mid \exists f.[l, u]$ is a top-level constraint of $C\}$;
   - $R_i^{\mathcal{J}} = \{(d, d_i) \mid i = 1, \ldots, m\}$.

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

$$\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}} \cup \bigcup_i \Delta^{\mathcal{I}_i}$$

$$A^{\mathcal{I}} = A^{\mathcal{J}} \cup \bigcup_i A^{\mathcal{I}_i} \quad (A \in \mathsf{N_C})$$

$$R^{\mathcal{I}} = R^{\mathcal{J}} \cup \bigcup_i R^{\mathcal{I}_i} \quad (R \in \mathsf{N_R} \cup \mathsf{N_F})\,.$$

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

Note that each C has a unique canonical model up to isomorphism. The canonical model satisfies $\mathcal{K}^{-}$, $\mathcal{O}^{+}_{\mathcal{K}}$, and C:

Lemma 5.10. *If C is a simple $\mathcal{PL}$ concept normalized w.r.t. $\mathcal{K}$ and $\mathcal{O}$, and $C \neq \bot$, then each canonical model $(\mathcal{I}, d)$ of C enjoys the following properties:*

a. $\mathcal{I} \models \mathcal{K}^{-} \cup pos(\mathcal{O}^{+}_{\mathcal{K}})$;
b. $(\mathcal{I}, d) \models C$.

Proof. By induction on the maximum nesting level ℓ of C's existential restrictions.

If $\ell = 0$ (i.e. there are no existential restrictions) then obviously $(\mathcal{I}, d) \models C$ by construction (see Definition 5.9.a). The entailment $\mathcal{I} \models \mathcal{K}^{-}$ holds because $\mathcal{K}^{-}$ contains only range and functionality axioms, that are trivially satisfied since all roles are empty in $\mathcal{I}$. In order to prove the base case we are only left to show that $\mathcal{I} \models pos(\mathcal{O}^{+}_{\mathcal{K}})$. Suppose not, i.e. there

exists an inclusion $B_1 \sqcap \ldots \sqcap B_m \sqsubseteq A$ in $pos(\mathcal{O}^+_{\mathcal{K}})$ such that $d \in (B_1 \sqcap \ldots \sqcap B_m)^{\mathcal{I}}$ but $d \notin A^{\mathcal{I}}$ (where d is the only member of $\Delta^{\mathcal{I}}$). By construction of $\mathcal{I}$, $d \in B_j^{\mathcal{I}}$ only if $pos(\mathcal{O}^+_{\mathcal{K}})$ contains $\bigsqcap_{i=1}^n A_i \sqsubseteq B_j$ for all $j = 1, \ldots, m$, where the A_i are the top-level concept names in C. These inclusions, together with $B_1 \sqcap \ldots \sqcap B_m \sqsubseteq A$, imply by simple inferences that $\bigsqcap_{i=1}^n A_i \sqsubseteq A$ must be in $pos(\mathcal{O}^+_{\mathcal{K}})$, too. But then $A^{\mathcal{I}}$ should contain $\{d\}$ by definition (a contradiction). This completes the proof of 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 Definition 5.9.b satisfy D_i. 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 α in $\mathcal{K}^- \cup pos(\mathcal{O}^+_{\mathcal{K}})$.

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 α. Since all $\mathcal{I}_i$ satisfy α by I.H., $\mathcal{I}$ satisfies α, too.

If $\alpha = \mathsf{range}(R, A)$, then rule 6) makes sure that for each top-level concept of the form $\exists R.D_i$ in C, $D_i \equiv D_i' \sqcap A$. Then, by I.H., $(I_i, d_i) \models A$ and, consequently, α is satisfied by $\mathcal{I}$.

Finally, if α is an inclusion in $pos(\mathcal{O}^+_{\mathcal{K}})$, then d satisfies it by the same argument used in the base case, while the other individuals in $\Delta^{\mathcal{I}}$ satisfy α by I.H. ■

Another key property of the canonical models of C is that they characterize *all* the valid elementary subsumptions whose left-hand side is C:

Lemma 5.11. *If $C \sqsubseteq D$ is elementary w.r.t. $\mathcal{K}$ and $\mathcal{O}$, $C \neq \bot$, and $(\mathcal{I}, d)$ is a canonical model of C, then*

$$\mathcal{K}^- \cup pos(\mathcal{O}^+_{\mathcal{K}}) \models C \sqsubseteq D \text{ iff } (\mathcal{I}, d) \models D\,.$$

Proof. (Only If part) Assume that $\mathcal{K}^- \cup pos(\mathcal{O}^+_{\mathcal{K}}) \models C \sqsubseteq D$. By Lemma 5.10.a, we have $\mathcal{I} \models \mathcal{K}^- \cup pos(\mathcal{O}^+_{\mathcal{K}})$, so by assumption $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Moreover, by Lemma 5.10.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 $\mathcal{K}^- \cup pos(\mathcal{O}^+_{\mathcal{K}}) \models C \sqsubseteq D$ by structural induction on D.

If $D = A$ (a concept name), then $d \in A^{\mathcal{I}}$ by assumption. Then, by construction of $\mathcal{I}$, there must be an inclusion $\left(\bigsqcap_{i=1}^n A_i \sqsubseteq A\right) \in pos(\mathcal{O}^+_{\mathcal{K}})$, where $A_1, \ldots, A_n$ are the top-level concept names of C. This implies that both $\models C \sqsubseteq \bigsqcap_{i=1}^n A_i$ and $pos(\mathcal{O}^+_{\mathcal{K}}) \models \bigsqcap_{i=1}^n A_i \sqsubseteq A$ hold, hence $\mathcal{K}^- \cup pos(\mathcal{O}^+_{\mathcal{K}}) \models C \sqsubseteq D$.

If $D = D_1 \sqcap D_2$, then $(\mathcal{I}, d) \models D_i$ $(i = 1, 2)$, therefore, by induction hypothesis, $\mathcal{K}^- \cup pos(\mathcal{O}^+_{\mathcal{K}}) \models C \sqsubseteq D_i$ $(i = 1, 2)$, hence $\mathcal{K}^- \cup pos(\mathcal{O}^+_{\mathcal{K}}) \models C \sqsubseteq D$.

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 concept C_1 occurring in a top-level restriction $\exists R.C_1$ of C. It follows that $\models C \sqsubseteq \exists R.C_1$ and (by induction hypothesis) $\mathcal{K}^- \cup pos(\mathcal{O}^+_{\mathcal{K}}) \models C_1 \sqsubseteq D_1$, hence $\mathcal{K}^- \cup pos(\mathcal{O}^+_{\mathcal{K}}) \models C \sqsubseteq D$.

If $D = \exists f.[\ell, u]$, 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 $\exists f.[\ell', u']$, so by interval safety (that is implied by the assumption that $C \sqsubseteq D$ is elementary), $[\ell', u'] \subseteq [\ell, u]$. Then $\models C \sqsubseteq D$. ■

Moreover, by means of canonical models, one can prove that interval safety makes the non-convex logic $\mathcal{PL}$ behave like a convex logic.

Lemma 5.12. *For all interval safe $\mathcal{PL}$ subsumption queries $q = \left(C_1 \sqcup \ldots \sqcup C_m \sqsubseteq D_1 \sqcup \ldots \sqcup D_n\right)$ such that each C_i is normalized w.r.t. $\mathcal{K}$ and $\mathcal{O}$, the entailment $\mathcal{K}^- \cup pos(\mathcal{O}^+_{\mathcal{K}}) \models q$ holds iff for all $i \in [1, m]$ there exists $j \in [1, n]$ such that $\mathcal{K}^- \cup pos(\mathcal{O}^+_{\mathcal{K}}) \models C_i \sqsubseteq D_j$.*

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

By assumption and Lemma 5.11, 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 $(\mathcal{I}, d) \models \neg \bigsqcup_{j=1}^n D_j$. Moreover, $(\mathcal{I}, d)$ satisfies both $\mathcal{KB}$ and C_i by Lemma 5.10. Then $\mathcal{I}$ and d witness that $\mathcal{KB} \not\models C_i \sqsubseteq \bigsqcup_{j=1}^n D_j$. ■

Now that the semantic properties are laid out, we focus on the algorithms. Roughly speaking, the next lemma says that $\mathsf{STS}^{\mathcal{O}^+_{\mathcal{K}}}$ decides whether the canonical model $(\mathcal{I}, d)$ of C satisfies D.

Lemma 5.13. *If $C \sqsubseteq D$ is elementary w.r.t. $\mathcal{K}$ and $\mathcal{O}$, $C \neq \bot$, and $(\mathcal{I}, d)$ is the canonical model of C, then*

$$\mathsf{STS}^{\mathcal{O}^+_{\mathcal{K}}}(C \sqsubseteq D) = \texttt{true} \text{ iff } (\mathcal{I}, d) \models D\,.$$

Proof. By structural induction on D. If $D = A$ (a concept name), then by definition $\mathsf{STS}^{\mathcal{O}^+_\mathcal{K}}(C \sqsubseteq D) = \mathtt{true}$ iff there exists an inclusion $\bigsqcap_{i=1}^{n} A_i \sqsubseteq A$ in $pos(\mathcal{O}^+_\mathcal{K})$ such that the A_i's are the top-level concept names in C (see line 3 of Algorithm 3). 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 (see line 6 of Algorithm 3).

If $D = \exists R.D_1$, then $\mathsf{STS}^{\mathcal{O}^+_\mathcal{K}}(C \sqsubseteq D) = \mathtt{true}$ iff: (i) C has a top-level subconcept $\exists R.C_1$, and (ii) $\mathsf{STS}^{\mathcal{O}^+_\mathcal{K}}(C_1 \sqsubseteq D_1) = \mathtt{true}$ (see line 5). Moreover, by definition of $\mathcal{I}$, $(\mathcal{I}, d) \models D$ holds iff fact (i) holds and: (ii') $(\mathcal{I}_i, d_i) \models D_1$, where $(\mathcal{I}_i, d_i)$ is a canonical model of C_1. By induction hypothesis, (ii) is equivalent to (ii'), so the lemma immediately follows.

If $D = \exists f.[\ell, u]$, then $\mathsf{STS}^{\mathcal{O}^+_\mathcal{K}}(C \sqsubseteq D) = \mathtt{true}$ iff the following property holds:

$$C \text{ has a top-level subconcept } \exists f.[\ell', u'] \text{ such that } [\ell', u'] \subseteq [\ell, u] \tag{11}$$

(see line 4). We are only left to prove that (11) is equivalent to $(\mathcal{I}, d) \models D$.

Property (11) implies (by construction of $\mathcal{I}$) that $(d, u') \in f^\mathcal{I}$ and $u' \in [\ell, u]$, that is, $(\mathcal{I}, d) \models D$.

Conversely, if $(\mathcal{I}, d) \models D$, then there exists $u' \in \Delta^\mathcal{I}$ such that $(d, u') \in f^\mathcal{I}$ and $u' \in [\ell, u]$. Then, by construction of $\mathcal{I}$, C must have a top-level subconcept $\exists f.[\ell', u']$. By interval safety (that is implied by the hypothesis that $C \sqsubseteq D$ is elementary), the fact that $[\ell', u']$ and $[\ell, u]$ have u' in common implies $[\ell', u'] \subseteq [\ell, u]$. Therefore, (11) holds. This completes the proof. ■

We are now ready to prove that $\mathsf{PLR}^\mathcal{O}$ is correct and complete.

Theorem 5.14. *Let $\langle \mathcal{K}, \mathcal{O}, C \sqsubseteq D\rangle$ be any instance of $\mathcal{PLSO}$. Then*

$$\mathsf{PLR}^\mathcal{O}(\mathcal{K}, C \sqsubseteq D) = \mathtt{true} \text{ iff } \mathcal{K} \cup \mathcal{O} \models C \sqsubseteq D\,.$$

Proof. D is of the form $D_1 \sqcup \ldots \sqcup D_n$. Let $C_1 \sqcup \ldots \sqcup C_m$ be the concept C'' computed by lines 2 and 3 of $\mathsf{PLR}^\mathcal{O}$. We start by proving the following claim, for all $i = 1, \ldots, m$ and $j = 1, \ldots, n$:

$$\mathsf{STS}^{\mathcal{O}^+_\mathcal{K}}(C_i \sqsubseteq D_j) = \mathtt{true} \text{ iff } \mathcal{K}^- \cup pos(\mathcal{O}^+_\mathcal{K}) \models C_i \sqsubseteq D_j\,. \tag{12}$$

There are two possibilities. If $C_i = \bot$, then clearly $\mathcal{K}^- \cup pos(\mathcal{O}^+_\mathcal{K}) \models C_i \sqsubseteq D_j$ and $\mathsf{STS}^{\mathcal{O}^+_\mathcal{K}}(C_i \sqsubseteq D_j) = \mathtt{true}$ (see line 2 of Algorithm 3), so (12) holds in this case. If $C \neq \bot$, then note that $C_i \sqsubseteq D_j$ is elementary w.r.t. $\mathcal{K}$ and $\mathcal{O}$ by construction of C'' (which is obtained by splitting the intervals of the normalization of C w.r.t. $\mathcal{K}$ and $\mathcal{O}$). Then (12) follows immediately from Lemmas 5.11 and 5.13.

By (12) and convexity (Lemma 5.12), we have that lines 5–11 of Algorithm 4 return $\mathtt{true}$ iff $\mathcal{K}^- \cup pos(\mathcal{O}^+_\mathcal{K}) \models C'' \sqsubseteq D$. Moreover, C'' can be equivalently replaced by C in this entailment, by Proposition 5.7 and Proposition 4.5. The resulting entailment is equivalent to $\mathcal{K} \cup \mathcal{O} \models C \sqsubseteq D$ by Theorem 5.6. It follows that Algorithm 4 returns $\mathtt{true}$ iff $\mathcal{K} \cup \mathcal{O} \models C \sqsubseteq D$. ■

$\mathsf{PLR}^\mathcal{O}$ runs in polynomial time, modulo the cost of oracle queries.

Lemma 5.15. *$\mathsf{PLR}^\mathcal{O}(\mathcal{K}, C \sqsubseteq D)$ runs in time $O(|C \sqsubseteq D|^{c+1} + |C \sqsubseteq D|^2 \cdot |\mathcal{K}|)$ using an oracle[23] for $pos(\mathcal{O}^+_\mathcal{K})$, where c is the maximum number of interval constraints occurring in a single simple concept of C.*

Proof. Each query to the oracle triggered by the application of normalization rule 7 or by line 3 of $\mathsf{STS}^{\mathcal{O}^+_\mathcal{K}}$ counts as one step of computation, according to the definition of time complexity for oracle machines. Then, by the same arguments used in the proof of Lemma 4.11, the computation of the normalization steps in lines 2 and 3 of $\mathsf{PLR}^\mathcal{O}$ takes time $O(|C|^2 \cdot |\mathcal{K}| + |C| \cdot |D|^c)$, while the loops spanning over lines 5–9 take time $O(|D| \cdot |C| \cdot |\mathcal{K}|)$. The lemma follows by expressing the size of C and D in terms of $|C \sqsubseteq D|$ (as in Lemma 4.11). ■

As a consequence of the above lemma, the classes of subsumption instances where c is bounded can be decided in polynomial time, modulo the cost of oracle queries.

Definition 5.16. For all non-negative integers c, let $\mathcal{PLSO}_c$ be the set of $\mathcal{PLSO}$ instances $\langle \mathcal{K}, \mathcal{O}, C \sqsubseteq D\rangle$ such that the maximum number of interval constraints occurring in a single simple concept of C is bounded by c.

Theorem 5.17. *For all c, $\mathcal{PLSO}_c$ is in $\mathsf{P}^{pos(\mathcal{O}^+_\mathcal{K})}$.*

Computing the consequences of $\mathcal{O}$, in general, is intractable, although $\mathcal{O}$ is restricted to Horn-$\mathcal{SRIQ}$ knowledge bases. For other Horn DLs, however – like the profiles of OWL2 and their generalizations $\mathcal{EL}^{++}$ and *DL-lite*$^\mathcal{H}_{horn}$– subsumption

[23] Here we mean the notion of "oracle" used in the definition of oracle machines and related complexity classes [43].

checking is tractable. By Theorem 5.17, the tractability of convex oracles extends to reasoning in $\mathcal{PL}$ with such oracles. More precisely, it suffices to assume that membership in $pos(\mathcal{O}^+_\mathcal{K})$ can be decided in polynomial time, since in that case $\mathsf{P}^{pos(\mathcal{O}^+_\mathcal{K})} = \mathsf{P}$. This is what happens when $\mathcal{O}$ is in $\mathcal{EL}^{++}$ and *DL-lite*$^{\mathcal{H}}_{horn}$, since the axioms shifted from $\mathcal{K}$ to $\mathcal{O}$ (i.e. $\mathcal{O}^+_\mathcal{K} \setminus \mathcal{O}$) can be expressed both in $\mathcal{EL}^{++}$ and in *DL-lite*, therefore $\mathcal{O}^+_\mathcal{K}$ is in the same logic as $\mathcal{O}$. This is formalized as follows:

Definition 5.18. For all integers $c \geq 0$, let $\mathcal{PLSO}^{\mathcal{DL}}_c$ be the set of instances of $\mathcal{PLSO}_c$ whose oracle is in $\mathcal{DL}$.

Corollary 5.19. *For all $c \geq 0$, $\mathcal{PLSO}^{\mathcal{EL}^{++}}_c$ and $\mathcal{PLSO}^{DL\text{-}lite^{\mathcal{H}}_{horn}}_c$ are in* P.

It can also be proved that the normalization rules in Table 4 may be used as a *policy validation* method, to detect unsatisfiable policies.

Theorem 5.20. *Let $\langle \mathcal{K}, \mathcal{O}, q\rangle$ be a $\mathcal{PLSO}$ instance and C be a $\mathcal{PL}$ concept such that $\Sigma(C) \cap \Sigma(\mathcal{O}) \subseteq \mathsf{N_C}$.*

1. *A $\mathcal{PL}$ concept $C = C_1 \sqcup \ldots \sqcup C_n$ is unsatisfiable w.r.t. $\mathcal{K} \cup \mathcal{O}$ iff $C_i \rightsquigarrow^* \bot$ for all $i \in [1, n]$.*[24]
2. *Under the above hypotheses, $\mathcal{PL}$ concept satisfiability testing w.r.t. $\mathcal{K} \cup \mathcal{O}$ is in $\mathsf{P}^{pos(\mathcal{O}^+_\mathcal{K})}$ (hence in* P *if $\mathcal{O}$ belongs to a tractable logic).*

Proof. By Proposition 5.7 and Lemma 5.10, C is satisfiable w.r.t. $\mathcal{K}^- \cup pos(\mathcal{O}^+_\mathcal{K})$ iff $C_i \rightsquigarrow^* \bot$ does *not* hold for some $i \in [1, n]$. Moreover, by Theorem 5.6,

$$\mathcal{K}^- \cup pos(\mathcal{O}^+_\mathcal{K}) \models C \sqsubseteq \bot \text{ iff } \mathcal{K} \cup \mathcal{O} \models C \sqsubseteq \bot.$$

Point 1 immediately follows. Next, note that normalization can be computed in polynomial time using an oracle for $pos(\mathcal{O}^+_\mathcal{K})$. This can be shown with a straightforward adaptation of the proof of Lemma 4.8 that takes into account the oracle queries in rule 7 (the details are left to the reader). Then Point 2 follows from the complexity of normalization and Point 1. ■

5.3. Related tractability and intractability results

$\mathcal{PL}$ knowledge bases are in OWL2-RL, a Horn fragment of OWL2; they are also in the extensions of $\mathcal{EL}$ and *DL-lite* with functional roles. Answering $\mathcal{PL}$ subsumption queries is equivalent to solving query containment problems with respect to $\mathcal{PL}$ knowledge bases, where the queries are the translation of $\mathcal{PL}$ concepts into formulae of first-order logic.[25] Such formulae are instances of the class of queries investigated in [46], called *extended faceted queries*. Core faceted queries (or simply *faceted queries*, abbreviated with FQ) are formulae with one free variable, built from unary and binary predicates using $\vee$, $\wedge$, and $\exists$; moreover, variables are restricted so that each faceted query equals the translation of a DL concept built from $\sqcup$, $\sqcap$, and $\exists$. A faceted query is *conjunctive* if it contains no occurrences of $\vee$; the set of conjunctive faceted queries is denoted by CFQ. The class of *unions of faceted queries* (UCFQ) consists of all the faceted queries where $\vee$ may occur only at the top level (i.e. it cannot be nested inside the other constructs). *Extended* faceted queries support a class *Comp* of operators for number comparison, a class of special predicates *Agg* for computing aggregates, and predicates *Next, Next*$^+$ for traversing chains of binary relations. Different subclasses of extended FQ can be denoted by $\mathcal{L}[\mathcal{X}]$, where $\mathcal{L} \in \{\text{FQ}, \text{CFQ}, \text{UCFQ}\}$ specifies the restrictions on $\vee$, and $\mathcal{X} \subseteq \{Comp, Agg, Next, Next^+\}$ specifies which additional predicates are supported. For example, the class of all extended faceted query is denoted by FQ[*Comp,Agg,Next,Next*$^+$]. Extended faceted queries are more general than the translation of $\mathcal{PL}$ subsumptions in the following ways:

- disjunctions can be nested within conjunctions;
- queries may contain the special predicates for aggregates, *Next*, and *Next*$^+$.

The class of queries corresponding to $\mathcal{PL}$ concepts, where disjunction may occur only at the top level and the above special predicates are not allowed, is UCFQ[*Comp*] (unions of conjunctive faceted queries with comparison operators).

The results of [46] show that deciding query containment in FQ (i.e. the class of faceted queries without special predicates nor comparison operators) is coNP-complete, even if the knowledge base is empty; hardness is proved by nesting disjunctions within conjunctions.

Without such nesting (i.e. if we restrict to UCFQ), query containment may be tractable even if the knowledge base is nonempty. In particular, the containment of a query Q in Q' can be reduced to query answering as follows: first introduce a fresh individual name a, then extend the knowledge base with a set of assertions that make $Q(a)$ true (possibly by adding

[24] As usual, $\rightsquigarrow^*$ denotes the reflexive and transitive closure of $\rightsquigarrow$.

[25] The translation of concepts into first-order formulae can be found in [4, Chapter 4] and [19].

additional fresh constants); finally evaluate Q' and check whether a belongs to the answer [19]. Top-level disjunctions can be dealt with by exploiting convexity, as we do in this paper. The combined complexity of UCFQ answering is in P in many cases, see [6,5] for tractability results that apply to knowledge bases formulated in OWL2-EL and OWL2-QL, and to queries that are slightly more general than CFQ. Therefore the containment of UCFQ can be decided in polynomial time when the knowledge base belongs to these profiles.

The above reduction of containment, however, is not applicable to queries that contain special predicates.[26] Indeed, [46, Lemma 5] proves that query containment in CFQ[*Next,Next*$^+$] is coNP-complete even if the knowledge base is empty. Additionally, due to the relationships between query containment and concept subsumption, our Theorem 4.1 implies that query containment in UCFQ[*Comp*] is coNP-hard, even if the knowledge base is empty and the leftmost query is conjunctive. Moreover, Theorem 4.12 shows that a constant bound on the number of comparisons per conjunctive query suffices to restore tractability, for all nonempty $\mathcal{PL}$ knowledge bases. Theorem 5.17 extends the tractability of UCFQ[*Comp*] with bounded intervals to all the combinations of $\mathcal{PL}$ knowledge bases with oracles formulated in any tractable fragment of Horn-$\mathcal{SRIQ}$ (under the restrictions of Definition 5.3).

$\mathcal{PL}$ with oracles in $\mathcal{EL}^+$ can also be regarded as a tractable extension of $\mathcal{EL}$ with functionality axioms and non-convex concrete domains in the queries. Unrestricted combinations of such constructs are generally intractable, when the knowledge base – as in our subsumption instances – is nonempty and contains unrestricted GCIs.

More precisely, in the extension of $\mathcal{EL}$ with functional roles, subsumption checking is EXPTIME-complete, in general [3]. A tractability result for empty TBoxes is reported in [26, Fig. 4]; however, in the same paper, it is proved that even with acyclic TBoxes, subsumption is coNP-complete. Accordingly, OWL2-EL does not support functionality axioms, so $\mathcal{PL}$ knowledge bases cannot be encoded in this profile.

The tractability of an extension of $\mathcal{EL}$ with non-convex concrete domains (like intervals) has been proved in [26], under the assumption that the TBox is a set of *definitions* of the form $A \equiv C$, where each A is a concept name and appears in the left-hand side of at most one definition.

An extended analysis of the tractability threshold for the *DL-lite* family can be found in [2]. The results most closely related to our work are the following.

The data complexity of query answering raises at the first level of the polynomial hierarchy if *DL-lite*$^{\mathcal{H}}_{horn}$ is extended with functional roles. Knowledge base satisfiability becomes EXPTIME-complete (combined complexity). Under three syntactic restrictions [2, $\mathbf{A}_1$–$\mathbf{A}_3$] and the unique name assumption, both of the above reasoning tasks remain tractable. Nevertheless, OWL2-QL – that is founded on one of the simplest members of the *DL-lite* family – does not support functional roles, therefore it cannot be used to encode $\mathcal{PL}$ knowledge bases.

$\mathcal{PL}$ knowledge bases with oracles in $\mathcal{EL}^+$ or *DL-lite*$^{\mathcal{H}}_{horn}$ are in Horn-$\mathcal{SHOIQ}$ with *(reuse)-safe* roles [18,19]. This logic is tractable, and the role safety restriction replaces the modularity requirement of IBQ approaches.[27] By means of the results of [18,19], $\mathcal{PL}$ knowledge bases with oracles in $\mathcal{EL}^+$ and *DL-lite*$^{\mathcal{H}}_{horn}$ can be translated into a Datalog program in polynomial time, preserving fact entailment. Then, subsumption checking can be reduced to conjunctive query answering as explained above. Recall, however, that this reduction does not apply to subsumptions with interval constraints; so the tractability results of [18,19] do not imply our results for $\mathcal{PL}$ subsumptions.

The most expressive knowledge representation language enjoying a complete structural subsumption algorithm – to the best of our knowledge – is CLASSIC [17], 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 $\mathcal{PL}$ (by Theorem 4.1). On the other hand, CLASSIC additionally supports qualified universal restrictions (that strictly generalize $\mathcal{PL}$'s range restrictions), number restrictions, and role-value maps, therefore it is not comparable to $\mathcal{PL}$. The complexity of the extensions of $\mathcal{PL}$ with CLASSIC's constructs is an interesting topic for further research.

5.4. Compiling oracles into $\mathcal{PL}$ knowledge bases

Note that $pos(\mathcal{O}^+_{\mathcal{K}})$ might be *compiled*, i.e. computed once and for all, so as to reduce oracle queries to retrieval. After such knowledge compilation, $\mathsf{PLR}^{\mathcal{O}}$ could run in polynomial time, *no matter how complex* $\mathcal{O}$*'s logic is*, provided that the subset of $pos(\mathcal{O}^+_{\mathcal{K}})$ queried by $\mathsf{PLR}^{\mathcal{O}}$ (i.e. the part of $pos(\mathcal{O}^+_{\mathcal{K}})$ that should be pre-computed) is polynomial, too.

This is not always the case. The conjunctions of classes $\bigsqcap_i A_i$ that may possibly occur in the left-hand side of subsumption queries are exponentially many in the signature's size, and each of them may potentially occur in a query to the oracle. So, in order to limit the space of possible oracle queries and reduce the partial materialization of $pos(\mathcal{O}^+_{\mathcal{K}})$ to a manageable size, we have to limit the number of concepts that may occur in the left-hand side of subsumption queries.

Fortunately, in SPECIAL's use cases, the subsumption queries $C \sqsubseteq D$ that implement compliance checks have always a business policy on the left-hand side, and the set of business policies of a controller is rather stable and not large. So the prerequisite for applying oracle compilation is satisfied. We are further going to show that the oracle can be compiled into

[26] As far as *Comp* is concerned, the problem is that the arguments of comparison operators are numbers, so the idea of instantiating the atoms of Q with fresh individual names is not applicable.

[27] Of course, without modularity, query answering cannot be split among two specialized reasoners for the main part and the oracle, respectively.

a plain, oracle-free $\mathcal{PL}$ knowledge base, therefore the IBQ framework can be implemented with the same efficiency as pure $\mathcal{PL}$ reasoning.

We start the formalization of the above ideas by defining the restricted class of problem instances determined by the given set of business policies $\mathcal{BP}$.

Definition 5.21. For all sets of $\mathcal{PL}$ concepts $\mathcal{BP}$, let $\mathcal{PLSO}(\mathcal{BP})$ be the set of all $\langle \mathcal{K}, \mathcal{O}, C \sqsubseteq D\rangle \in \mathcal{PLSO}$ such that $C \in \mathcal{BP}$.

The first step of the oracle compilation consists in transforming business policies so as to collapse each conjunction of concept names into a single concept name. We say that the result of this transformation is in *single-atom form*, which is recursively defined as follows:

Definition 5.22. A simple $\mathcal{PL}$ concept C is in *single-atom form* if either

1. C is of the form $(\bigsqcap_{i=1}^{m} \exists f_i.[l_i, u_i]) \sqcap (\bigsqcap_{i=1}^{k} \exists R_i.C_i)$, where $m, k \geq 0$, and each C_i is in single-atom form, or
2. C is of the form $A \sqcap (\bigsqcap_{i=1}^{m} \exists f_i.[l_i, u_i]) \sqcap (\bigsqcap_{i=1}^{k} \exists R_i.C_i)$ where $m, k \geq 0$, and each C_i is in single-atom form.

A full $\mathcal{PL}$ concept $C_1 \sqcup \ldots \sqcup C_n$ is in single atom form if $C_1, \ldots, C_n$ are all in single atom form.

The given business policies can be transformed in single atom form in linear time:

Proposition 5.23. *For all finite sets of concepts $\mathcal{BP}$ there exist a set of concepts $\mathcal{BP}^*$ in single atom form, and a knowledge base $\mathcal{O}^*$ that belongs to both $\mathcal{EL}$ and DL-lite$_{horn}$, such that for all $\langle \mathcal{K}, \mathcal{O}, C \sqsubseteq D\rangle \in \mathcal{PLSO}(\mathcal{BP})$ there exists an equivalent problem instance $\langle \mathcal{K}, \mathcal{O} \cup \mathcal{O}^*, C^* \sqsubseteq D\rangle \in \mathcal{PLSO}(\mathcal{BP}^*)$, that is:*

$$\mathcal{K} \cup \mathcal{O} \models C \sqsubseteq D \text{ iff } \mathcal{K} \cup \mathcal{O} \cup \mathcal{O}^* \models C^* \sqsubseteq D\,.$$

Moreover, $\mathcal{BP}^*$ and $\mathcal{O}^*$ can be computed in time $O(|\mathcal{BP}|)$.

Proof. For all $C \in \mathcal{BP}$, we obtain the corresponding concept C^* by replacing each intersection of multiple concept names in C with a single fresh concept name, whose definition is included in $\mathcal{O}^*$. More precisely, if $C = C_1 \sqcup \ldots \sqcup C_n$ then for all $j = 1, \ldots, n$, replace each

$$C_j = (\bigsqcap_{i=}^{n} A_i) \sqcap (\bigsqcap_{i=1}^{m} \exists f_i.[l_i, u_i]) \sqcap (\bigsqcap_{i=1}^{k} \exists R_i.D_i)$$

such that $n > 1$ with

$$C_j^* = B \sqcap (\bigsqcap_{i=1}^{m} \exists f_i.[l_i, u_i]) \sqcap (\bigsqcap_{i=1}^{k} \exists R_i.D_i^*)\,,$$

where B is a fresh concept name and each D_i^* is obtained by recursively applying the same transformation to D_i.

The knowledge base $\mathcal{O}^*$ is the set of all the definitions $B \equiv (\bigsqcap_{i=}^{n} A_i)$ such that B is one of the fresh concepts introduced by the above transformations and $\bigsqcap_{i=}^{n} A_i$ is the intersection replaced by B.

Finally, let $\mathcal{BP}^*$ be the set of concepts $C^* = C_1^* \sqcup \ldots \sqcup C_n^*$ obtained with the above procedure. Clearly, by construction, $\mathcal{K} \cup \mathcal{O} \cup \mathcal{O}^* \models C \equiv C^*$, for all $C \in \mathcal{BP}$. Moreover, $\mathcal{K} \cup \mathcal{O} \cup \mathcal{O}^*$ is a conservative extension of $\mathcal{K} \cup \mathcal{O}$. Therefore

$$\begin{aligned} \mathcal{K} \cup \mathcal{O} \models C \sqsubseteq D \text{ iff } & \mathcal{K} \cup \mathcal{O} \cup \mathcal{O}^* \models C \sqsubseteq D \\ \text{iff } & \mathcal{K} \cup \mathcal{O} \cup \mathcal{O}^* \models C^* \sqsubseteq D\,. \end{aligned}$$

Concerning complexity, $\mathcal{BP}^*$ and $\mathcal{O}^*$ can be computed with a single scan of $\mathcal{BP}$; the generation of the fresh concepts B, the replacement of $\bigsqcap_{i=}^{n} A_i$ and the generation of the definition for B take linear time in $|C_j|$. Therefore $\mathcal{BP}^*$ and $\mathcal{O}^*$ can be computed in time $O(|\mathcal{BP}|)$. ■

By the above proposition, we can assume without loss of generality that $\mathcal{BP}$ is in single atom form. Note that the ontologies $\mathcal{K}$ and $\mathcal{O}$, in a typical application scenario, do not change frequently. So we can fix them and assume that the concepts in $\mathcal{BP}$ are already normalized w.r.t. $\mathcal{K}$ and $\mathcal{O}$. The set of problem instances with fixed $\mathcal{K}$ and $\mathcal{O}$ is defined as follows:

$$\mathcal{PLSO}(\mathcal{K}, \mathcal{O}, \mathcal{BP}) = \{\langle \mathcal{K}', \mathcal{O}', C \sqsubseteq D\rangle \in \mathcal{PLSO} \mid \mathcal{K}' = \mathcal{K},\ \mathcal{O}' = \mathcal{O},\ \text{and } C \in \mathcal{BP}\}\,.$$

The compilation of $\mathcal{K}$ and $\mathcal{O}$ into a single $\mathcal{PL}$ knowledge base is defined as follows:

$$comp(\mathcal{K}, \mathcal{O}) = \mathcal{K}^- \cup \{A \sqsubseteq B \mid (A \sqsubseteq B) \in pos(\mathcal{O}^+_{\mathcal{K}})\}\,.$$

The correctness of oracle compilation is proved by the next theorem.

Theorem 5.24. *Let $\mathcal{K}$ and $\mathcal{O}$ be two knowledge bases in $\mathcal{PL}$ and Horn-$\mathcal{SRIQ}$, respectively, and let $\mathcal{BP}$ be a set of $\mathcal{PL}$ concepts in single atom form and normalized w.r.t. $\mathcal{K}$ and $\mathcal{O}$. Then, for all $\langle \mathcal{K}, \mathcal{O}, C \sqsubseteq D\rangle \in \mathcal{PLSO}(\mathcal{K}, \mathcal{O}, \mathcal{BP})$,*

$$\mathsf{PLR}^{\mathcal{O}}(\mathcal{K}, C \sqsubseteq D) = \mathsf{PLR}(comp(\mathcal{K}, \mathcal{O}), C \sqsubseteq D)\,.$$

Proof. Since C is already normalized w.r.t. $\mathcal{K}$ and $\mathcal{O}$ by hypothesis, line 3 of $\mathsf{PLR}^{\mathcal{O}}$ computes the identity function (i.e. $C' = C$). It is easy to see that line 2 of PLR does the same. First, note that the two versions of rules 4 and 6 (in Table 3 and Table 4) apply to the same set of functionality and range axioms, since $\mathsf{func}(R) \in comp(\mathcal{K}, \mathcal{O}) \Leftrightarrow \mathsf{func}(R) \in \mathcal{K}^-$ and $\mathsf{range}(R, A) \in comp(\mathcal{K}, \mathcal{O}) \Leftrightarrow \mathsf{range}(R, A) \in \mathcal{K}^-$ (by definition of *comp*). So there are no additional axioms in $comp(\mathcal{K}, \mathcal{O})$ that may trigger rules 4 or 6 in PLR. Second, since C is in single atom form by hypothesis, rule 7 of Table 3 never applies. The other normalization rules are the same for $\mathsf{PLR}^{\mathcal{O}}$ and PLR. We conclude that lines 2 and 3 of $\mathsf{PLR}^{\mathcal{O}}$ and PLR produce the same concept $C'' = split_D(C)$.

Consequently, the loops in lines 5–9 of $\mathsf{PLR}^{\mathcal{O}}$ and lines 4–8 of PLR return the same result, too. To see this, it suffices to show that

$$\mathsf{STS}^{\mathcal{O}^+_{\mathcal{K}}}(C_i \sqsubseteq D_j) = \mathsf{STS}(comp(\mathcal{K}, \mathcal{O}), C_i \sqsubseteq D_j)\,. \tag{13}$$

The only difference between $\mathsf{STS}^{\mathcal{O}^+_{\mathcal{K}}}$ and STS is in their line 3. The membership tests executed by $\mathsf{STS}^{\mathcal{O}^+_{\mathcal{K}}}$ in line 3 are all of the form $(A_1 \sqsubseteq A) \in pos(\mathcal{O}^+_{\mathcal{K}})$, because C_j is in single atom form (this follows from the hypothesis that C is in single atom form). For the same reason, STS in line 3 checks whether $A_1 \sqsubseteq^* A$. The two tests are equivalent by definition of *comp*, therefore (13) holds and the theorem is proved. ■

Remark 5.25. Note that the size of $comp(\mathcal{K}, \mathcal{O})$ is at most quadratic in the size of $\mathcal{K} \cup \mathcal{O}$, and that PLR runs in polynomial time if the number of interval constraints per simple policy is bounded. Therefore, under this assumption – and after $comp(\mathcal{K}, \mathcal{O})$ has been computed – subsumption queries can be answered in polynomial time. If $\mathcal{O}$ uses expressive constructs from Horn-$\mathcal{SRIQ}$, then their computational cost is confined to the compilation phase only, that is essentially a standard classification of $\mathcal{O}^+_{\mathcal{K}}$. ■

A caveat on the size of $comp(\mathcal{K}, \mathcal{O})$ is in order, here. If the given set of policies $\mathcal{BP}$ is not in single atom form, then $\mathcal{O}$ must be replaced by $\mathcal{O} \cup \mathcal{O}^*$, as shown in Proposition 5.23, where the size of $\mathcal{O}^*$ is $O(|\mathcal{BP}|)$. Therefore the size of $comp(\mathcal{K}, \mathcal{O} \cup \mathcal{O}^*)$ may grow quadratically with $|\mathcal{BP}|$. This relationship shows the influence of $\mathcal{BP}$'s size on the complexity of the oracle compilation approach. So, unfortunately, oracle compilation is not always possible. For example, in the application of $\mathcal{PL}$ to data markets illustrated in the conclusions, we currently see no general criterion to restrict the space of possible queries as required by the compilation method.

Remark 5.26. Using the compilation approach, the soundness and completeness of PLR follow easily from the soundness and completeness of $\mathsf{PLR}^{\mathcal{O}}$, according to which $\mathcal{K} \models q$ holds if and only if $\mathsf{PLR}^{\emptyset}(\mathcal{K}, q) = \texttt{true}$. So it suffices to show that $\mathsf{PLR}(\mathcal{K}, q) = \mathsf{PLR}^{\emptyset}(\mathcal{K}, q)$. Note that $comp(\mathcal{K}, \emptyset)$ is simply the closure of $\mathcal{K}$ with respect to inclusions (that is, $comp(\mathcal{K}, \emptyset)$ preserves the relation $\sqsubseteq^*$ associated to $\mathcal{K}$). This fact and Theorem 5.24, respectively, imply that

$$\mathsf{PLR}(\mathcal{K}, q) = \mathsf{PLR}(comp(\mathcal{K}, \emptyset), q) = \mathsf{PLR}^{\emptyset}(\mathcal{K}, q)\,.$$

Similarly, the equality $\mathsf{PLR}(\mathcal{K}, q) = \mathsf{PLR}^{\emptyset}(\mathcal{K}, q)$ and the correspondence between the closure $\sqsubseteq^*$ of the inclusions in $\mathcal{K}$ and those in $comp(\mathcal{K}, \emptyset)$, immediately imply the following corollary of Theorem 5.20:

Corollary 5.27. *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.*

5.5. On the limitations posed on $\mathcal{PLSO}$

In this section we briefly motivate the restrictions posed on $\mathcal{PL}$ subsumption problems with oracles ($\mathcal{PLSO}$). We start with the requirements on the oracle. Recall that $\mathcal{O}$ should be convex w.r.t. $\mathcal{QL}$ and should not use nominals. Convexity w.r.t. $\mathcal{QL}$ is essential for tractability, as shown by the next result.

Theorem 5.28. *If $\mathcal{O}$ is not convex w.r.t. $\mathcal{QL}$ and enjoys the disjoint model union property, then there exists a $\mathcal{PL}$ knowledge base $\mathcal{K}$ such that deciding whether $\mathcal{K} \cup \mathcal{O} \models C \sqsubseteq D$ holds, given an interval-safe $\mathcal{PL}$ subsumption query $C \sqsubseteq D$, is co-*NP *hard.*

Proof. We are proving coNP-hardness by reducing 3SAT to the complement of subsumption. By hypothesis, $pos(\mathcal{O})$ contains an inclusion

$$A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B_1 \sqcup \ldots \sqcup B_m \tag{14}$$

such that none of the inclusions $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B_i$ belongs to $pos(\mathcal{O})$, for $i = 1, \ldots, m$. Without loss of generality, we can further assume that $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B_2 \sqcup \ldots \sqcup B_m$ is not in $pos(\mathcal{O})$ (if not, then discard some B_i from (14) until the right-hand side is a minimal union entailed by $A_1 \sqcap \ldots \sqcap A_n$). Now let $\mathcal{K}$ be the following set of inclusions, where A' and B' are fresh concept names:

$$A' \sqsubseteq A_i \quad (i = 1, \ldots, n)$$
$$B_j \sqsubseteq B' \quad (j = 2, \ldots, m)\,.$$

Note that $\mathcal{K} \cup \mathcal{O} \models A' \sqsubseteq B_1 \sqcup B'$, by construction of $\mathcal{K}$ and (14). We are going to represent the truth values *true* and *false* with B_1 and B', respectively.

Let S be any instance of 3SAT, and let $p_1, \ldots, p_k$ be the propositional symbols occurring in S. We assume without loss of generality that $p_1, \ldots, p_k$ do not occur in $\mathcal{K}$ nor in $\mathcal{O}$. Each positive literal p_i is encoded by $e(p_i) = \exists p_i.B_1$, while negative literals $\neg p_i$ are encoded by $e(\neg p_i) = \exists p_i.B'$. Then the negation of S is encoded by

$$D = \bigsqcup \{ e(\bar{L}_1) \sqcap e(\bar{L}_2) \sqcap e(\bar{L}_3) \mid L_1 \vee L_2 \vee L_3 \in S \}\,.$$

(where each $\bar{L}_i$ is the literal complementary to L_i). We claim that the non-entailment

$$\mathcal{K} \cup \mathcal{O} \not\models \Big(\bigsqcap_i \exists p_i.A' \Big) \sqsubseteq D \tag{15}$$

holds iff S is satisfiable (note that the above subsumption query is interval-free, hence trivially interval safe). To prove the "only if" part, assume that (15) holds, that is, there exists a pointed interpretation $(\mathcal{I}, d)$ such that $\mathcal{I} \models \mathcal{K} \cup \mathcal{O}$, $d \in \big(\bigsqcap_i \exists p_i.A'\big)^{\mathcal{I}}$ and $d \notin D^{\mathcal{I}}$. Since $\mathcal{K} \cup \mathcal{O} \models \big(\bigsqcap_i \exists p_i.A'\big) \sqsubseteq (\exists p_i.B_1) \sqcup (\exists p_i.B')$ holds for each symbol p_i, there exists $d_i \in \Delta^{\mathcal{I}}$ such that $(d, d_i) \in p_i^{\mathcal{I}}$ and either $d_i \in B_1^{\mathcal{I}}$ or $d_i \in (B')^{\mathcal{I}}$. Construct a truth assignment σ for S by setting

$$\sigma(p_i) = \begin{cases} \mathit{true} & \text{if } d_i \in B_1^{\mathcal{I}}\,, \\ \mathit{false} & \text{if } d_i \notin B_1^{\mathcal{I}} \text{ (therefore } d_i \in (B')^{\mathcal{I}})\,. \end{cases}$$

Since $d \notin D^{\mathcal{I}}$, each clause $L_1 \vee L_2 \vee L_3$ of S contains a literal p_i or $\neg p_i$ such that, respectively, $d_i \in B_1^{\mathcal{I}}$ or $d_i \in (B')^{\mathcal{I}}$, so σ satisfies the literal, by definition. It follows immediately that σ satisfies S.

Conversely, suppose that S is satisfied by a truth assignment σ. We are going to construct a pointed interpretation $(\mathcal{I}, \bar{d})$ that witnesses (15). Recall that neither $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B_1$ nor $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B_2 \sqcup \ldots \sqcup B_m$ belong to $pos(\mathcal{O})$. Then $\mathcal{O}$ has two disjoint models $\mathcal{M}_1$ and $\mathcal{M}_2$ such that for some $d_1 \in \Delta^{\mathcal{M}_1}$ and $d_2 \in \Delta^{\mathcal{M}_2}$,

$$d_i \in (A_1 \sqcap \ldots \sqcap A_n)^{\mathcal{M}_i} \quad (i = 1, 2)$$
$$d_1 \notin B_1^{\mathcal{M}_1}$$
$$d_2 \notin (B_2 \sqcup \ldots \sqcup B_m)^{\mathcal{M}_2}.$$

The union $\mathcal{U} = \mathcal{M}_1 \uplus \mathcal{M}_2$ is still a model of $\mathcal{O}$ by hypothesis, and it can be extended to a model $\mathcal{J}$ of $\mathcal{K} \cup \mathcal{O}$ by setting:

$$\Delta^{\mathcal{J}} = \Delta^{\mathcal{U}}$$
$$(A')^{\mathcal{J}} = (A_1 \sqcap \ldots \sqcap A_n)^{\mathcal{J}}$$
$$(B')^{\mathcal{J}} = (B_2 \sqcup \ldots \sqcup B_m)^{\mathcal{J}}\,.$$

Finally, we extend $\mathcal{J}$ to the witness $\mathcal{I}$ as follows. First let $\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$ and choose any $\bar{d} \in \Delta^{\mathcal{J}}$. For all symbols p_i define:

$$p_i^{\mathcal{I}} = \{(\bar{d}, d_1)\} \quad \text{if } \sigma(p_i) = \mathit{false}\,,$$
$$p_i^{\mathcal{I}} = \{(\bar{d}, d_2)\} \quad \text{otherwise}\,.$$

Note that $\bar{d}$ belongs to $\big(\bigsqcap_i \exists p_i.A'\big)$ by construction, so we are only left to prove that $\bar{d} \notin D^{\mathcal{I}}$. By assumption, each clause in S contains a literal L satisfied by σ. If $L = \neg p_i$, then $p_i^{\mathcal{I}} = \{(\bar{d}, d_1)\}$, therefore $\bar{d} \notin (\exists p_i.B_1)^{\mathcal{I}} = e(\bar{L})^{\mathcal{I}}$. Similarly, if $L = p_i$, then $p_i^{\mathcal{I}} = \{(\bar{d}, d_2)\}$, therefore $\bar{d} \notin (\exists p_i.B')^{\mathcal{I}} = e(\bar{L})^{\mathcal{I}}$. It follows immediately that $\bar{d} \notin D^{\mathcal{I}}$. ■

Note that the above theorem shows that reasoning can be intractable even if $\mathcal{K}$ and $\mathcal{O}$ are *fixed*.

The requirement that nominals must not occur in oracles is needed for completeness. Our algorithm $\mathsf{PLR}^{\mathcal{O}}$– and the other IBQ methods where oracle queries are consistency tests of the form (10), or the equivalent inclusions of the form (9) – are generalized by the following definition, that accounts for the shifting of axioms from $\mathcal{K}$ to $\mathcal{O}$.

Definition 5.29. Let $\mathcal{PI}$ be a set of problem instances of the form $\langle \mathcal{K}, \mathcal{O}, q\rangle$, where $\mathcal{K}$ and $\mathcal{O}$ are knowledge bases and q is an inclusion. A *shifting IBQ mechanism* for $\mathcal{PI}$ is a pair of functions (s, r) such that for all $\langle \mathcal{K}, \mathcal{O}, q\rangle \in \mathcal{PI}$:

1. $s(\mathcal{K}) \subseteq \mathcal{K}$,
2. $r(s(\mathcal{K}), pos(\mathcal{O} \cup (\mathcal{K} \setminus s(\mathcal{K}))), q) = \texttt{true}$ iff $\mathcal{K} \cup \mathcal{O} \models q$.

Informally speaking, s determines which axioms are shifted from $\mathcal{K}$ to $\mathcal{O}$, and r is the IBQ reasoner that decides entailment using the modified knowledge bases. Shifting IBQ mechanisms do not exist if $\mathcal{O}$ may use nominals.

Theorem 5.30. *Let $\mathcal{DL}$ be a description logic that supports nominals and disjointness axioms. Let $\mathcal{PI}$ be any set of problem instances that contains all $\langle \mathcal{K}, \mathcal{O}, q\rangle$ such that $\mathcal{K} = \emptyset$, $\mathcal{O}$ is a $\mathcal{DL}$ knowledge base, and q is an $\mathcal{EL}$ inclusion.*[28] *There exists no shifting IBQ mechanism for $\mathcal{PI}$.*

Proof. Let $\mathcal{K} = \emptyset$ and $q = \exists R.(A \sqcap B) \sqcap \exists R.(A \sqcap \bar{B}) \sqsubseteq A'$. Let

$$\mathcal{O}_1 = \{\mathsf{disj}(B, \bar{B})\},$$
$$\mathcal{O}_2 = \{\mathsf{disj}(B, \bar{B}),\ A \sqsubseteq \{a\}\}.$$

Note that both $\langle \mathcal{K}, \mathcal{O}_1, q\rangle$ and $\langle \mathcal{K}, \mathcal{O}_2, q\rangle$ belong to $\mathcal{PI}$.

It can be easily verified that $pos(\mathcal{O}_1) = pos(\mathcal{O}_2)$; in particular, the two sets contain all the inclusions of the form $A_1 \sqcap \ldots \sqcap A_m \sqsubseteq B_1 \sqcup \ldots \sqcup B_n$ such that:

- either the inclusion is a tautology (i.e. some concept name occurs both in the left-hand side and in the right-hand side),
- or both B and $\bar{B}$ occur in the left-hand side.

However, $\mathcal{K} \cup \mathcal{O}_1 \not\models q$, while $\mathcal{K} \cup \mathcal{O}_2 \models q$. The latter fact holds because due to the nominal $\{a\}$, both $\exists R.(A \sqcap B)$ and $\exists R.(A \sqcap \bar{B})$ should have the same role filler, that cannot satisfy the disjoint concepts B and $\bar{B}$ at the same time. It follows that q is trivially satisfied because its left-hand side is equivalent to $\bot$.

Now suppose that a shifting IBQ mechanism (s, r) for $\mathcal{PI}$ exists; we shall derive a contradiction. By condition 2 of Definition 5.29,

$$r(s(\mathcal{K}), pos(\mathcal{O}_1 \cup (\mathcal{K} \setminus s(\mathcal{K}))), q) = \texttt{false} \tag{16}$$
$$r(s(\mathcal{K}), pos(\mathcal{O}_2 \cup (\mathcal{K} \setminus s(\mathcal{K}))), q) = \texttt{true}. \tag{17}$$

However, $\mathcal{K} = s(\mathcal{K}) = \emptyset$ and consequently:

$$\begin{aligned} r(s(\mathcal{K}), pos(\mathcal{O}_1 \cup (\mathcal{K} \setminus s(\mathcal{K}))), q) &= r(\emptyset, pos(\mathcal{O}_1), q) \\ &= r(\emptyset, pos(\mathcal{O}_2), q) \\ &= r(s(\mathcal{K}), pos(\mathcal{O}_2 \cup (\mathcal{K} \setminus s(\mathcal{K}))), q) \end{aligned}$$

which contradicts (16) and (17). ■

Remark 5.31. The above result complements the analogous negative result [20, Theorem 4] that applies to knowledge bases $\mathcal{K}$ with infinity axioms (while $\mathcal{PL}$ knowledge bases have the finite model property). On the other hand, [20, Theorem 4] covers also more expressive oracle query languages.

The proof of the above negative result is based on the limited expressiveness of the oracle query language $\mathcal{QL}$. A similar consideration applies to the requirement that $\Sigma(\mathcal{O})$ may share only concept names with $\Sigma(\mathcal{K})$ and $\Sigma(q)$. Without this assumption, $\mathsf{PLR}^{\mathcal{O}}$ is not complete. More generally:

Theorem 5.32. *Let $\mathcal{PI}$ be a set of problem instances that contains all $\langle \mathcal{K}, \mathcal{O}, q\rangle$ such that $\mathcal{K} = \emptyset$, $\mathcal{O}$ is an $\mathcal{EL}$ knowledge base and q is an $\mathcal{EL}$ inclusion (possibly sharing roles with $\mathcal{O}$). There exists no shifting IBQ mechanism for $\mathcal{PI}$.*

[28] We use $\mathcal{EL}$ inclusions to strengthen our result, since they are a special case of $\mathcal{PL}$ subsumption queries.

Proof. Let $\mathcal{K} = \emptyset$, $q = (\exists R.A \sqsubseteq \exists S.A)$, $\mathcal{O}_1 = \emptyset$ and $\mathcal{O}_2 = \{q\}$. Note that

- $pos(\mathcal{O}_1) = pos(\mathcal{O}_2)$ (both contain all and only the tautological inclusions of the form (9));
- $\mathcal{K} \cup \mathcal{O}_1 \not\models q$;
- $\mathcal{K} \cup \mathcal{O}_2 \models q$.

Then the assumption that a shifting IBQ mechanism for $\mathcal{PI}$ exists leads to a contradiction, by the same argument used in Theorem 5.30. ■

In the light of the above negative results, a natural question is whether an oracle query language more expressive than $\mathcal{QL}$ would remove the need for the restrictions on nominals and roles. Note that IBQ mechanisms for shared roles have already been introduced in [20]. For a fragment of $\mathcal{EL}$, there exists an IBQ algorithm that terminates in polynomial time. Nominals are not allowed, but shared roles are, under suitable conditions.

In order to support more expressive oracle queries, $\mathsf{PLR}^{\mathcal{O}}$ and $\mathsf{STS}^{\mathcal{O}^+_{\mathcal{K}}}$ should be extensively changed, though. The proofs of the above negative results reveal that the simple treatment of existential restrictions in $\mathsf{STS}^{\mathcal{O}^+_{\mathcal{K}}}$ should be replaced with a more complex computation, involving oracle queries, and it is currently not clear how significantly such changes would affect the scalability of reasoning and the possibility of compiling oracles into $\mathcal{PL}$ knowledge bases. Given that scalability is one of SPECIAL's primary requirements, and that there is no evidence that shared roles are needed by SPECIAL's application scenarios (cf. Remark 5.1), we leave this question as an interesting topic for further research.

6. Experimental assessment

In this section we describe a Java implementation of PLR and compare its performance with that of other popular engines. We focus on PLR (as opposed to the more complex $\mathsf{PLR}^{\mathcal{O}}$) because SPECIAL's application scenarios are compatible with the oracle compilation into a $\mathcal{PL}$ knowledge base illustrated in Section 5.4. The implementation and experimental evaluation of $\mathsf{PLR}^{\mathcal{O}}$, that may be interesting in other applications of $\mathcal{PL}$, lie beyond the scope of this paper.

SPECIAL's engine is tested on two randomly generated sets of inputs. The first set is based on the knowledge base and policies developed for Proximus and Thomson Reuters. Consent policies are generated by modifying the business policies, mimicking a selection of privacy options from a list provided by the controller. This first set of test cases is meant to assess the performance of the engines in the application scenarios that we expect to arise more frequently in practice. The second set of experiments, that makes use of larger knowledge bases and policies, is meant to predict the behavior of the engines in more complex scenarios, should they arise in the future.

The implementation of PLR and its optimizations are described in the next subsection. Then Section 6.2 illustrates the test cases used for the evaluation. Finally, Section 6.3 reports the results of the experiments.

6.1. Prototype implementation and optimization

PLR is implemented in Java and it is distributed as a .jar file. The reasoner's class is named *PLReasoner*, and supports the standard OWL APIs, version 5.1.7. The package includes a complete implementation of PLR, including the structural subsumption algorithm STS, and the preliminary normalization phases, based on the 7 rewrite rules and on the interval splitting method for interval safety.

The interval splitting method has been refined in order to reduce the explosion of business policies. The reason for refinements can be easily seen: if a business policy contains interval $[1, 10]$ and a consent policy contains $[5, 10]$, then the method illustrated in (8) splits $[1, 10]$ into the (unnecessarily large) set of intervals

$$[1, 1],\ [2, 4],\ [5, 5],\ [6, 9],\ [10, 10]\,,$$

that cause a single simple policy to be replaced with 5 policies. Note that for interval safety the splitting $[1, 4],\ [5, 10]$ would be enough. While (8) is convenient in the theoretical analysis – because it has a simpler definition and it does not increase asymptotic complexity – a more articulated algorithm is advisable in practice. Here we only sketch the underlying idea: each interval end point is classified based on whether it occurs only as a lower bound, only as an upper bound, or both. A singleton interval is generated only for the third category of endpoints, while the others are treated more efficiently. In particular, in the above example, 1 and 5 occur only as lower bounds; this allows to generate non-singleton sub-intervals that have 1 and 5 as their lower bound. Moreover, 10 occurs only as an upper bound; this allows to create a non-singleton sub-interval where 10 is the upper bound. Accordingly, the refined splitting algorithm generates only the two intervals $[1, 4]$ and $[5, 10]$.

Several other optimizations have been implemented and assessed. The corresponding versions of PLR are described below:

PLR c

The normalization steps (lines 2 and 3 of PLR) are one of the most expensive parts of the reasoner. In order to reduce their cost, two caches are introduced. The first cache stores the business policies that have already been normalized w.r.t. $\mathcal{K}$ (line 2 of PLR). In this way, the seven rewrite rules are applied to each business policy only once; when the policy is used again, line 2 simply retrieves the normalized concept from the cache. This optimization is expected to be effective in SPECIAL's application scenarios because only business policies need to be normalized, and their number is limited. So the probability of re-using an already normalized policy is high, and the cache is not going to grow indefinitely; on the contrary its size is expected to be moderate.

Similarly, a second cache indexed by the two policies C and D stores the concepts $split_D(C)$ already computed (thereby speeding up line 3 of PLR, that is, the interval splitting step needed for interval safety).

PLR 2n, PLR c 2n

PLR 2n normalizes both C *and* D with the seven rewrite rules, before computing $split_D(C)$. Since the rewrite rules may merge and delete the intervals of D, this optimization potentially reduces the number of splitting points and, consequently, the size of $split_D(C)$. We denote with PLR c 2n the version of PLR that exploits both the caches of PLR c and applies double normalization, as PLR pre.

PLR pre, PLR pre 2n

Sometimes the two normalization phases can be pre-computed. When the set of business policies and the set of intervals that may occur in consent policies are known in advance, the seven rules and interval splitting can be applied once and for all before compliance checking starts. For example, intervals are available in advance when the minimum or maximum storage time are determined by law, or when the duration options available to data subjects when consent is requested are specified by the data controller. This version of the engine is designed for such scenarios. The given set of business policies is fully normalized before compliance checking starts, and stored in the caches supported by PLR c. During compliance checking, lines 2 and 3 only retrieve concepts from the caches. In this way the cost of a compliance check is almost exclusively the cost of STS. This version of PLR will be evaluated by measuring compliance checking time only; preliminary normalizations are not included.

6.2. Test case generation

The first set of test cases is derived from the business policies developed for the pilots of Proximus and Thomson Reuters; these policies will be denoted with P_{PXS} and P_{TR} respectively.

In each compliance check $P_B \sqsubseteq P_C$, P_B is a union of simple business policies randomly selected from those occurring in the pilots' policy (P_{PXS} or P_{TR}). Since P_B describes the activity of a business process of the data controller, the random choice of P_B essentially corresponds to a random distribution of the controller's data processing activities (abstracted by the simple policies) across its business processes.

The consent policy P_C is the union of a set of simple policies P_C^i $(i = 1, \ldots, n)$ randomly selected from the pilots' policy, and randomly perturbed by replacing some vocabulary terms with a different term. The random selection mimicks the opt-in/opt-out choices of data subjects with respect to the various data processing activities modeled by the simple policies. Similarly, the random replacement of terms simulates the opt-in/opt-out choices of the data subject w.r.t. each component of the selected simple policies. More precisely, if the modified term occurring in P_C^i is a superclass (resp. a subclass) of the corresponding term in the original business policy, then the data subject opted for a broader (resp. more restrictive) permission relative to the involved policy property (e.g. data categories, purpose, and so on).

In this batch of experiments, the knowledge base is always SPECIAL's ontology, that defines policy roles and the temporary vocabularies for data categories, purpose categories, etc. The size and number of this batch of experiments is reported in Table 5. The number of randomly generated business policies is higher in one case because P_{PXS} has more simple policies than P_{TR}: the ratio is 20 generated policies per simple policy. Queries have been obtained by generating 100 consent policies for each business policy. Table 5 reports also the average number of simple policies per generated policy and its standard deviation. The size of each policy is limited by SPECIAL's usage policy format: at most one interval constraint per simple policy, and nesting depth 2.

In the second set of experiments, both the ontologies and $\mathcal{PL}$ subsumptions are completely synthetic, and have increasing size in order to set up a stress test for verifying the scalability of SPECIAL's reasoner. Fifteen ontologies have been generated: five for each of the three sets of parameters O1–O3 reported in Table 6. The same table reports the parameters used to generate the $\mathcal{PL}$ concepts occurring in the queries, according to two size specifications: P1 and P2.

Note that approximately half of the roles and concrete properties are functional, and half of the roles have a range axiom. Ontologies have been generated by randomly distributing classes over approximately $\log(\#classes)$ layers. Then the specified number of disjointness axioms have been generated, by picking classes on the same layer. Finally, about $2 \cdot \#classes$ inclusions have been created, mostly across adjacent layers, in such a way that no class became inconsistent. The ratio

Table 5
Size of the test cases inspired by the pilots.

	Proximus (PXS)	Thomson Reuters (TR)
Ontology		
inclusions	186	186
disj	11	11
range	10	10
func	8	8
classification hierarchy height	4	4
Business policies		
# generated policies	120	100
avg. simple pol. per full pol.	2.71	2.39
std. dev.	1.72	1.86
Consent policies		
# generated policies	12,000	10,000
avg. simple pol. per full pol.	3.77	3.42
std. dev.	2.02	2.03
Test cases		
# generated queries	12,000	10,000

Table 6
Size of fully synthetic test cases.

Ontology size	O1	O2	O3
classes	100	1,000	10,000
roles	10	50	100
concrete properties	10	25	50
func	10	37	75
range	5	25	50
avg. disj	3	31	298
avg. inclusions	211	2224	23418
avg. classification hierarchy height	8	10	14

Concept size	P1	P2
max #simple pol. per full pol.	10	100
max #top-level inters. per simple subconcept	10	20
max depth (nesting)	4	9
avg. #simple pol. per full pol.	6.8	50.1
avg. depth	2.4	5
Simple policy size		
avg. #intersections	10.6	25.8
avg. #intervals	3.7	9

between the number of inclusions and the number of classes is similar to the ratio that can be observed most frequently in real ontologies, cf. [36,35,33].

We have generated 100 concepts of size P1 and 1000 of size P2, picking interval endpoints from $[0, 365]$ (one year, in days). Each set has been split into business and consent policies (resp. 30% and 70% of the generated policies), that have been paired randomly to generate test queries. The number of queries of size P1 generated for each ontology is 50. Let $\#int$ be the maximum number of interval constraints per simple policy after normalization w.r.t. the 7 rules[29] (for a given business policy). The number of queries of size P2 generated for each ontology and each business policy with $\#int \leq 5$ is 10. The maximum number of queries for each ontology and each $\#int > 5$ has been limited to 40, in order to keep the length of the experiments within a reasonable range. In this case, we maximized the number of different business policies occurring in the selected queries.

For each ontology $\mathcal{K}$, the business policies have been selected from the available $\mathcal{K}$-consistent policies. Furthermore, whenever possible, queries have been selected in such a way that the number of positive and negative answers are the same. Table 6 illustrates the average size of the generated policies for each parameter setting. We have not limited the number of interval constraints, in order to analyze the behavior of PLReasoner as the number of intervals per simple policy grows (if it is not bounded then $\mathcal{PL}$ subsumption query answering is coNP-hard). The maximum nesting level occurring in the generated policies is approximately $\lceil \log_2(\mathit{max\ disjuncts}) \rceil$.

6.3. Performance analysis

The experiments have been run on a server with an 8-cores processor Intel Xeon Silver 4110, 11M cache, 198 GB RAM, running Ubuntu 18.04 and JVM 1.8.0_181, configured with 32 GB heap memory (of which less than 700 MB have been actually used in all experiments). We have *not* exploited parallelism in the engine's implementation.

[29] The reason for measuring $\#int$ after normalization is explained later.

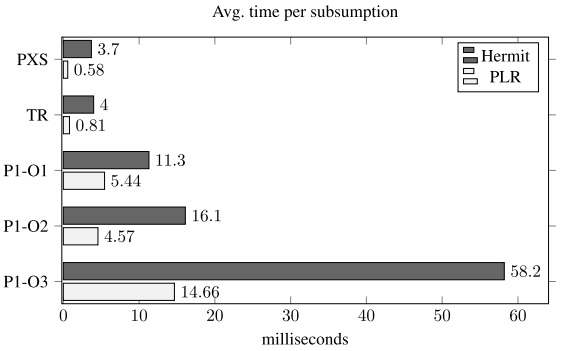

Fig. 1. Comparisons on small/medium policies.

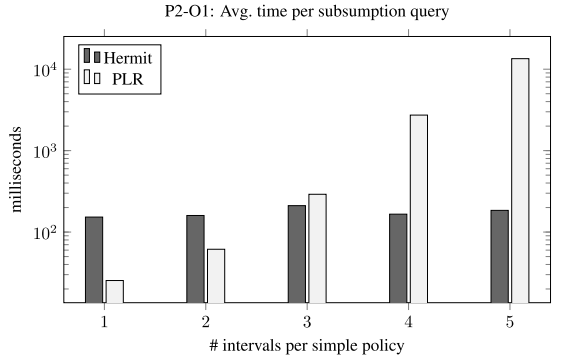

Fig. 2. Impact of interval number per simple policy – large policies.

First we compare PLR with Hermit, the only reasoner – among those we selected for comparison – that directly supports subsumptions with intervals.[30] We start by illustrating the results for the test cases with small and medium policies. Fig. 1 shows that PLR is faster than Hermit, over these test sets, even if no optimization is applied. The size of the ontology affects the performance of Hermit more than PLR's (see the results for O1, O2, and O3).

The good performance of PLR over PXS and TR had to be expected, given that the policies involved in these test sets are SPECIAL's usage policies, that by definition contain at most one interval constraint per simple policy, of the form $\exists$has_duration.$[\ell, u]$. Let $\#int$ denote the maximum number of intervals per simple policy after applying the rewrite rules, and recall that the size of $split_D(C)$ may grow exponentially with $\#int$. We have not limited $\#int$, while generating the synthetic policies in P1 and P2, to see how the number of intervals affects the performance of PLR (recall that if $\#int$ is unbounded, then $\mathcal{PL}$ subsumption is coNP-complete). We measured the value of $\#int$ after applying the rewrite rules, because they can collapse and delete intervals, thereby reducing the complexity of the subsequent interval splitting phase and the size of $split_D(C)$. After the application of the seven rules, the maximum $\#int$ over the business policies occurring in P1's queries is 9. Fig. 1 shows that the potential combinatorial explosion of $split_D(C)$ does not frequently occur with these policies. The probability of splitting a single interval into many sub-intervals is evidently not high. On the contrary, a combinatorial explosion is clearly observable in the test sets with large policies (P2); Fig. 2 illustrates the results for the smallest synthetic ontologies (O1).

Then we analyzed the effects of the optimizations described in Section 6.1. Their effectiveness over small and medium policies is illustrated by Fig. 3. The normalization of consent policies (2n) brings no benefits with small policies (actually, it slightly decreases the engine's performance, compare PLR 2n with PLR, and PLR c 2n with PLR c). Its benefits start to be visible with medium policies. The cache of normalized policies (PLR c) is the best option on small policies. On medium policies, the combination of the caches with the normalization of consent policies (PLR c 2n) is the most effective optimization.

Over large policies (P2), the normalization of consent policies (2n) is essential to mitigate the combinatorial explosion of $split_D(C)$, as shown in Fig. 4. The versions of PLR that do not normalize D become impractical already for $\#int = 3$, while the computation time of PLR 2n and PLR c 2n moderately increases. This behavior can be explained by observing the effects of normalization on this test set: after the application of the rewrite rules, the average number of intervals is about 10 times smaller, which reduces the probability of an exponential growth of $split_D(C)$.

[30] See also Remark 6.1 below.

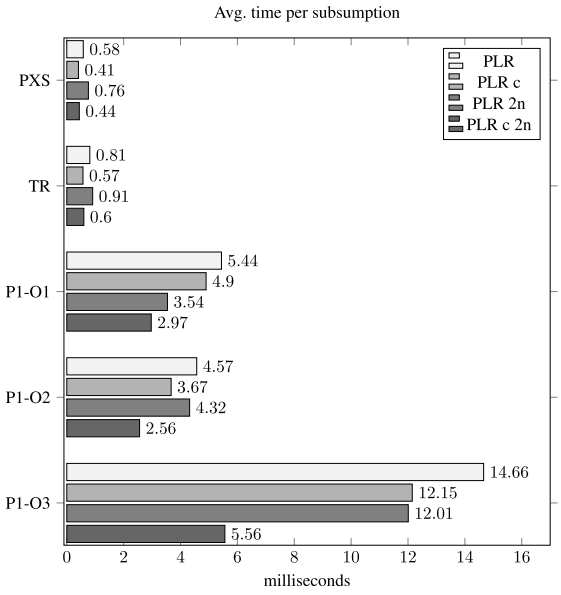

Fig. 3. Effectiveness of optimizations on small/medium policies.

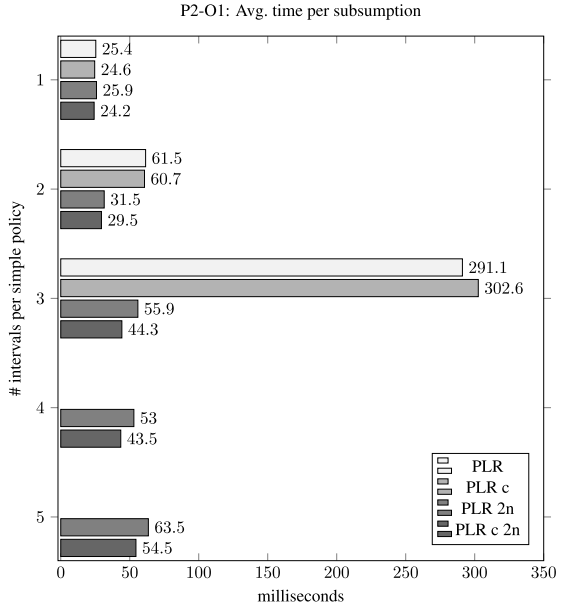

Fig. 4. Effectiveness of optimizations on large policies and small ontologies.

Next, in Fig. 5, we compare the best version of the engine over medium/large policies (i.e. PLR c 2n) with Hermit. The optimizations delay the effects of combinatorial explosions until $\#int = 7$. After this threshold, Hermit becomes faster.

Finally, we analyzed the effectiveness of business policy pre-normalization (pre). Recall that this approach is feasible in practice only if both the business policies and the intervals that may occur in consent policies are known in advance, and do not change frequently. The effects of pre-normalization on small and medium policies is remarkable: PLR pre is approximately one order of magnitude faster than Hermit, as shown in Fig. 6. Over pilot-inspired tests, pre-normalization brings the average time per subsumption query well below 500 µ-seconds.

The effects of pre-normalization quickly disappear over large policies. Fig. 7 shows that the explosion of $split_D(C)$ makes it necessary to apply also the normalization of consent policies to delay combinatorial effects (see PLR pre 2n). However, for $\#int = 8$, PLR pre 2n is slower than Hermit, so pre-normalization does not deal with the combinatorial explosion better than PLR c 2n.

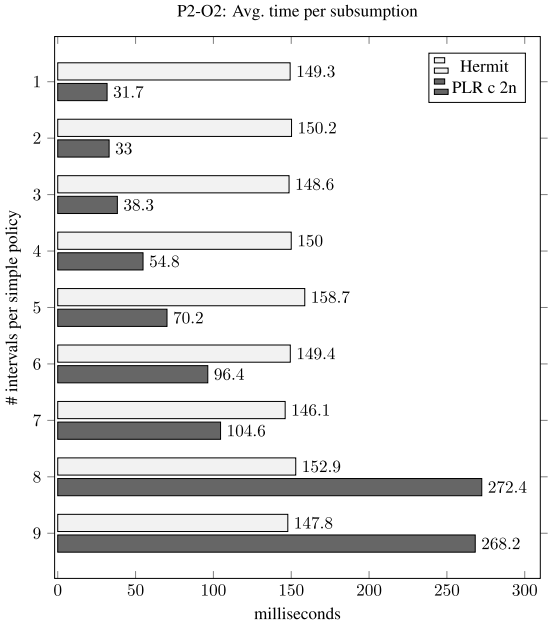

Fig. 5. Hermit vs PLR with caches and double normalization.

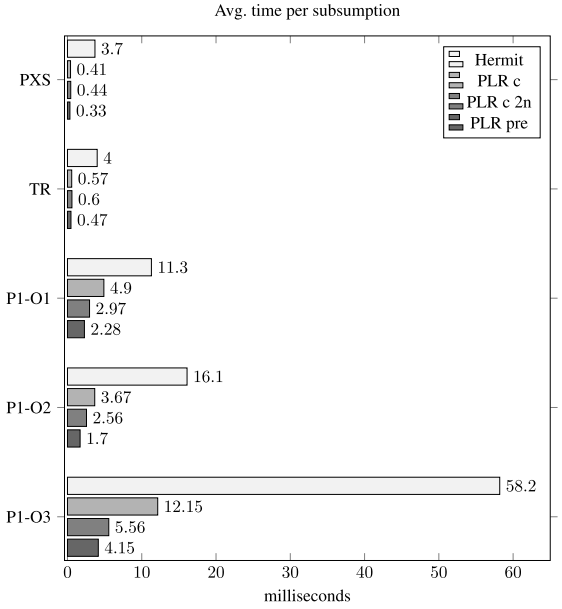

Fig. 6. Effectiveness of business policy pre-normalization on small/medium policies.

PLR can also be compared with ELK – a specialized reasoner for the tractable profile OWL2-EL – by exploiting the simple structure of PXS. The policies in this test set do not contain any intervals and are natively normalized (they never contain more than one subconcept $\exists R.C$ with the same role R). For these reasons, PXS can be correctly processed by ELK, although it supports neither intervals nor functionality axioms.

Using ELK, the average time per subsumption query is 3.11 milliseconds; therefore all versions of PLR are significantly faster. Such difference in performance may be partially due to the cost of initializing and maintaining ELK's indexing structures for the efficient application of the inference rules illustrated in [33]. Moreover, the worst-case complexity of ELK's algorithm is higher than PLR's ($O(n^3)$ vs. $O(n^2)$).

In order to test GraphDB, each subsumption query $C \sqsubseteq D$ has been translated as explained in Section 5.3, i.e. by asserting $C(a)$ in the knowledge base (where a is a fresh individual), and transforming D is into a SPARQL query to check whether $D(a)$ is entailed. Recall that this reduction is not applicable when C contains intervals, therefore GraphDB is tested on PXS

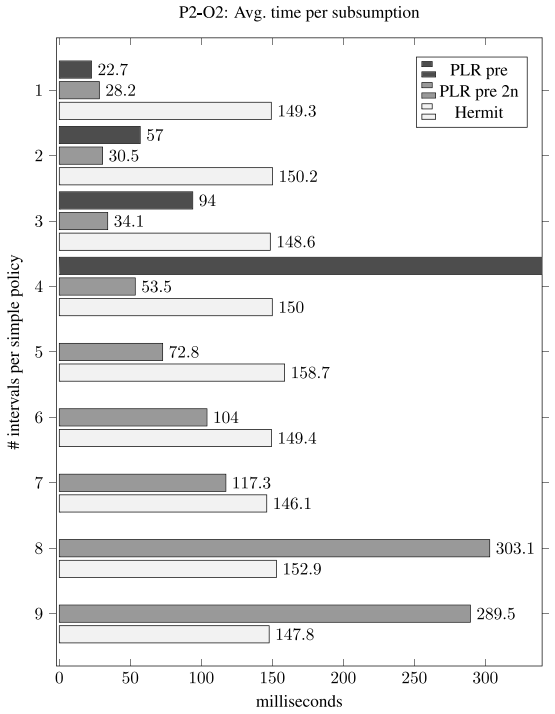

Fig. 7. Effectiveness of business policy pre-normalization on large policies.

only.[31] The cost of asserting $C(a)$ in the KB, the cost of translating D into SPARQL, and the cost of parsing the SPARQL query are *not* included in the measurement. The average time per query calculated in this way is 16.84 ms, so PLR is significantly faster on this test set. It should be considered that GraphDB is optimized for relatively small, user-generated queries on large ABoxes, while SPECIAL's scenarios involve a huge number of large, automatically generated queries on very small ABoxes.

Next we processed PXS with RDFox, by Oxford Semantic Technologies. For this purpose, $\mathcal{PL}$ subsumptions have been translated into SPARQL queries with the same reduction used for GraphDB. Computation time does not include the cost of the assertions $C(a)$ and the cost of the translation of D. On average, RDFox takes 1.03 ms per compliance check. Also in this case, it should be remarked that – similarly to GraphDB – RDFox is optimized for small, user-generated queries on large ABoxes, while SPECIAL's scenarios exhibit opposite features.

The response time of RDFox does not include the cost of computing the logical consequences of the KB, that are materialized when $C(a)$ is asserted. On the other hand, measurements include the parsing of SPARQL queries. So the performance of RDFox can be improved by caching the queries, in order to parse them only the first time they are processed.

Remark 6.1. According to Oxford Semantic Technologies, it may be profitable to leverage RDFox's generality and replace the standard reduction to query answering used in our experiments with a Datalog meta-interpreter that implements the method for $\mathcal{PL}$ subsumption checking introduced in Section 4. Such meta-interpreter would process a reified representation of $\mathcal{PL}$ concepts and make use of Datalog extensions such as comparison operators, aggregates (for interval splitting), and equality (to encode functionality axioms). A first advantage of this implementation is that it can answer all $\mathcal{PL}$ queries (while the standard reduction is not applicable to subsumptions with intervals). Another potential advantage, in terms of performance, is that RDFox can materialize and incrementally update the subsumption predicate. A potential disadvantage is the expected size of the materialization as the number of business and consent policies grows. Concerning PLR, its performance can be improved by adopting RDFox's implementation choices, in particular (i) re-engineering PLR in C^{++}, and (ii) exploiting parallelism. Independently from performance considerations, our experimental results on PLR prove that real-time subsumption checking in $\mathcal{PL}$ can be achieved without necessarily resorting to complex proprietary technology. This fact fosters adoption – a topic that is further discussed in the conclusions.

We have also considered Konclude, a general reasoner that is very competitive on standard classification benchmarks [47]. Konclude does not support intervals, therefore it has been tried on PXS only. Konclude integrates a tableau algorithm with completion-based saturation – for pay-as-you-go behavior – and adopts a wide range of optimizations. The current

[31] On the other hand, unlike GraphDB, $\mathcal{PL}$ is not able to express all SPARQL queries.

version, however, is focused on classification tasks; streams of $\mathcal{PL}$ subsumptions can be processed only at the cost of repeating the classification of the knowledge base for each query. This prevents a fair comparison with Hermit and PLR (in our tests, Konclude is slower than both).

7. Conclusions

We have introduced the description logic $\mathcal{PL}$ in order to formalize the data usage policies adopted by controllers as well as the consent to data processing granted by data subjects. Checking whether the controllers' policies comply with the available consent boils down to subsumption checking between $\mathcal{PL}$ concepts. $\mathcal{PL}$ can also formalize parts of the GDPR; then, by means of subsumption checking, one can automatically check several constraints on usage policies such as, for example:

- Are all the required policy properties specified?
- Are all the required obligations specified?
- Is the policy compatible with GDPR's constraints on cross-border data transfers?

$\mathcal{PL}$ queries supports interval constraints of the form $\exists f.[\ell, u]$ in order to model limitations on data storage duration. This feature affects convexity, and hinders a direct use of the query answering techniques for Horn DLs.

$\mathcal{PL}$ has been made as simple as possible in order to address two requirements. First, it should be usable by people with no logical or legal background. One of our industrial partners successfully assessed the usability of $\mathcal{PL}$, by verifying that its employees can write correct business policies. Second, the frequency of compliance checks can be high, so $\mathcal{PL}$ query answering should be extremely fast and scalable. Despite the simplicity of $\mathcal{PL}$, general $\mathcal{PL}$ subsumption checking is coNP-complete, due to the interplay of interval constraints and concept union. However, reasoning becomes tractable by requiring that each simple policy on the left-hand side of the subsumption query should contain a bounded number of interval constraints – a restriction that is naturally satisfied by SPECIAL's usage policies, consent policies, and in the formalization of the GDPR. Under this assumption, subsumption checking can be split into a polynomial-time normalization phase and a subsequent subsumption check that can be carried out by a fast, structural subsumption algorithm (STS).

The scalability of the complete algorithm (PLR) has been experimentally assessed. Some of the test sets consist of realistic policies and ontologies, derived from SPECIAL's pilots. Such policies and ontologies are small, so we generated also synthetic stress tests, where policies and ontologies are significantly larger than what we expect in real GDPR compliance scenarios. Our tests show that PLR is significantly faster than Hermit on small and medium policies. This had to be expected, since Hermit is not specialized on $\mathcal{PL}$ and constructs a hypertableau at each subsumption check. The performance of PLR can be further improved by caching normalized policies (PLR c). With this solution, PLR takes around 500 µseconds per subsumption check, over the test sets inspired by SPECIAL's pilots (PXS and TR). By pre-normalizing business policies (PLR pre), the average cost per subsumption check can be further reduced to 333 µsec (PXS) and 487 µsec (TR).[32]

Over large policies (P2), the probability of observing a combinatorial explosion during interval splitting grows, and the performance of PLR exhibits an exponential decrease as *#int* grows (where *#int* is the average number of intervals per simple business policy measured after applying the seven rewrite rules). This phenomenon is unavoidable, unless $\mathsf{P} = \mathsf{NP}$, because $\mathcal{PL}$ subsumption checking is coNP-hard if *#int* is unrestricted. However, by normalizing also consent policies, combinatorial effects are mitigated (because normalization may merge different intervals), and PLR c2n turns out to be faster than Hermit for $\#int < 8$.

PLR has been compared also with ELK, GraphDB, and Konclude, using only a subset of the test cases because these engines cannot answer $\mathcal{PL}$ queries with intervals. Being specialized on $\mathcal{PL}$ queries, PLR turns out to be faster than these engines, too. PLR is also faster than RDFox, when the standard reduction of subsumption to query answering is adopted.[33] Alternatively, it seems possible to implement PLR's reasoning method in Datalog, and evaluate the Datalog program with RDFox (see Remark 6.1). Investigating this approach is an interesting topic for further research.

In perspective, the expressiveness needed to encode the vocabularies of data categories, purposes, recipients, etc. is going to exceed the capabilities of $\mathcal{PL}$. For this reason, we have shown how to integrate the compliance checking method based on PLR with reasoners for logics more expressive than $\mathcal{PL}$. The integration is based on the *import by query* approach. If the "external" ontology $\mathcal{O}$ that defines vocabulary terms is in Horn-$\mathcal{SRIQ}$, and if the main knowledge base $\mathcal{K}$ and the given subsumption query share only concept names with $\mathcal{O}$, then algorithm $\mathsf{PLR}^{\mathcal{O}}$ – an adaptation of PLR that calls a reasoner for $\mathcal{O}$ – is sound and complete. If $\mathcal{O}$ additionally belongs to a tractable DL, then subsumption checking is tractable in the IBQ framework, too. The restriction on roles can be partly lifted by allowing queries to mention the roles occurring in $\mathcal{O}$, provided that if $R \in \Sigma(\mathcal{O})$, then the existential restrictions $\exists R.C$ may contain only roles in $\Sigma(\mathcal{O})$.

[32] This speed allows to process only about 20% of the base station events and 33% of the wi-fi probing events generated every second in the streaming scenario. SPECIAL addresses this issue by running multiple compliance checks in parallel, by means of a big data architecture; see for example [12,34] for more details.

[33] See also the optimization options discussed in Section 6.

We have also illustrated a different implementation strategy, based on a pre-compilation of $\mathcal{K}$ and $\mathcal{O}$ into a single $\mathcal{PL}$ knowledge base $comp(\mathcal{K}, \mathcal{O})$, whose size is polynomial in the size of $\mathcal{K} \cup \mathcal{O}$ and in the number of business policies. Compliance checks are computed in polynomial time, after compilation, even if $\mathcal{O}$ belongs to an intractable logic. Moreover, pre-compilation allows to exploit the implementation of PLR, whose scalability has been assessed in Section 6. This approach works well in SPECIAL's use cases because the number of business policies is usually small, and $\mathcal{K}$, $\mathcal{O}$, and the business policies are relatively stable and persistent. Unfortunately, the above assumptions cannot be made in general, for all potential applications of $\mathcal{PL}$.

Such applications include also the representation of licenses, which constitute a fundamental aspect of data markets. The application context is in some respect analogous to SPECIAL's: $\mathcal{PL}$ concepts should encode the usage restrictions that apply to datasets, multimedia content, and so on. In this case, however, the policies that can be reasonably assumed to belong to a limited set are those associated to sellers, that occur on the right-hand side of subsumptions, while the left-hand side can hardly be restricted. This hinders the compilation-based approach, and may require a direct implementation of $\mathsf{PLR}^{\mathcal{O}}$, that is, the general IBQ reasoner for $\mathcal{PL}$. Such implementation and its experimental assessment are interesting topics for further research.

$\mathcal{PL}$ can also naturally encode electronic health records (EHRs). In this case, the top-level properties of $\mathcal{PL}$ queries encode the sections of EHRs – according, say, to the HL7 standard – while some of the sections' contents can be specified with SNOMED terms. The IBQ framework allows to process $\mathcal{PL}$ queries with $\mathsf{PLR}^{\mathcal{O}}$, and reduce the cost of SNOMED to oracle calls, consisting of linear time visits to its classification graph. The efficiency of the structural subsumption reasoner is very promising in this context, that is challenging for all engines due to the remarkable size of SNOMED. We plan to try $\mathsf{PLR}^{\mathcal{O}}$ to increase the performance of the secure view construction reported in [15].

The simplicity of PLR makes it possible to embed $\mathcal{PL}$ reasoning in objects with limited scripting capabilities. For example, one of SPECIAL's partners has programmed $\mathcal{PL}$ compliance checking as a smart contract in an Ethereum blockchain. In this way, the creation of new entries in the blockchain is subject to compliance with a specified policy. This is also an example of how simplicity may foster adoption: SPECIAL's policy framework is not tightly bound to any specific technology or components, and it can be easily integrated in a variety of systems.

SPECIAL's deliverables comprise dashboards for controllers, data subjects, and data protection officers. We are going to support these user interfaces by developing explanation algorithms for helping data subjects in understanding policies and their decisions. The idea is leveraging the simple structure of $\mathcal{PL}$ concepts and axioms to generate high-level, user-friendly explanations.

On the theoretical side, our results on the complexity of $\mathcal{PL}$ queries are novel, as discussed in Section 5.3, and extend the available tractability and intractability results for extended faceted queries. The negative result on oracles with nominals (Theorem 5.30) extends a result of [20] to logics that (like $\mathcal{PL}$) enjoy the finite model property, and to IBQ mechanism where the axioms of the main knowledge base $\mathcal{K}$ may be shifted to the imported ontology $\mathcal{O}$.

There are further interesting topics for future work. For example, we currently do not know whether the requirement that $(\Sigma(\mathcal{K}) \cup \Sigma(q)) \cap \Sigma(\mathcal{O}) \subseteq \mathsf{N_C}$ can be relaxed without affecting tractability (under appropriate hypotheses).

Another interesting line of research consists in tracing the tractability threshold in the family of logics obtained by extending $\mathcal{PL}$ with CLASSIC's constructs, with particular attention to number restrictions, role-value maps, and nominals. Preliminary results have been published in [16].

Last but not least, from a theoretical perspective, it will be interesting to see to what extent PLR's pre-processing can be adapted to extend Horn DLs with interval constraints without affecting tractability. We expect the interplay of number restrictions and intervals to increase the complexity of reasoning.

Declaration of competing interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Acknowledgements

This research is funded by the European Union's Horizon 2020 research and innovation programme under grant agreement N. 731601. The GDPR compliance use case – here sketched with (5), (6), and Example 3.2 – is due to Benedict Whittam Smith (Thomson Reuters).

References

[1] G. Antoniou, N. Dimaresis, G. Governatori, A modal and deontic defeasible reasoning system for modelling policies and multi-agent systems, Expert Syst. Appl. 36 (2) (2009) 4125–4134.

[2] A. Artale, D. Calvanese, R. Kontchakov, M. Zakharyaschev, The DL-lite family and relations, J. Artif. Intell. Res. 36 (2009) 1–69.

[3] F. Baader, S. Brandt, C. Lutz, Pushing the EL envelope, in: IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Professional Book Center, 2005, pp. 364–369.

[4] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003.

[5] M. Bienvenu, M. Ortiz, M. Simkus, G. Xiao, Tractability guarantees for DL-lite query answering, in: Eiter et al. [22], pp. 41–52.
[6] M. Bienvenu, M. Ortiz, M. Simkus, G. Xiao, Tractable queries for lightweight description logics, in: IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China August 3-9, 2013, 2013, pp. 768–774.
[7] P.A. Bonatti, Datalog for security, privacy and trust, in: Datalog Reloaded - First International Workshop, Datalog 2010. Revised Selected Papers, Oxford, UK, March 16-19, 2010, in: Lecture Notes in Computer Science, vol. 6702, Springer, 2010, pp. 21–36.
[8] P.A. Bonatti, Fast compliance checking in an OWL2 fragment, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, ijcai.org, 2018, pp. 1746–1752.
[9] P.A. Bonatti, B. Bos, S. Decker, J.D. Fernández, S. Kirrane, V. Peristeras, A. Polleres, R. Wenning, Data privacy vocabularies and controls: semantic web for transparency and privacy, in: Proceedings of the Workshop on Semantic Web for Social Good Co-Located with 17th International Semantic Web Conference, SW4SG@ISWC 2018, in: CEUR Workshop Proceedings, vol. 2182, CEUR-WS.org, 2018.
[10] P.A. Bonatti, S. De Capitani di Vimercati, P. Samarati, An algebra for composing access control policies, ACM Trans. Inf. Syst. Secur. 5 (1) (2002) 1–35.
[11] P.A. Bonatti, J.L. De Coi, D. Olmedilla, L. Sauro, A rule-based trust negotiation system, IEEE Trans. Knowl. Data Eng. 22 (11) (2010) 1507–1520.
[12] P.A. Bonatti, S. Kirrane, Big data and analytics in the age of the GDPR, in: 2019 IEEE International Congress on Big Data, BigData Congress 2019, IEEE, 2019, pp. 7–16.
[13] P.A. Bonatti, S. Kirrane, A. Polleres, R. Wenning, Transparent personal data processing: the road ahead, in: Computer Safety, Reliability, and Security - SAFECOMP 2017 Workshops, ASSURE, DECSoS, SASSUR, TELERISE, and TIPS, Proceedings, in: Lecture Notes in Computer Science, vol. 10489, Springer, 2017, pp. 337–349.
[14] P.A. Bonatti, A. Peron, On the undecidability of logics with converse, nominals, recursion and counting, Artif. Intell. 158 (1) (2004) 75–96.
[15] P.A. Bonatti, I.M. Petrova, L. Sauro, Optimized construction of secure knowledge-base views, in: Proceedings of the 28th International Workshop on Description Logics, in: CEUR Workshop Proceedings, vol. 1350, CEUR-WS.org, 2015.
[16] P.A. Bonatti, I.M. Petrova, L. Sauro, A richer policy language for GDPR compliance, in: Proceedings of the 32nd International Workshop on Description Logics, in: CEUR Workshop Proceedings, vol. 2373, CEUR-WS.org, 2019.
[17] A. Borgida, P.F. Patel-Schneider, A semantics and complete algorithm for subsumption in the CLASSIC description logic, J. Artif. Intell. Res. 1 (1994) 277–308.
[18] D. Carral, C. Feier, B. Cuenca Grau, P. Hitzler, I. Horrocks, EL-ifying ontologies, in: Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Proceedings, Vienna, Austria, July 19-22, 2014, 2014, pp. 464–479.
[19] D. Carral, C. Feier, B. Cuenca Grau, P. Hitzler, I. Horrocks, Pushing the boundaries of tractable ontology reasoning, in: The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Proceedings, Part II, 2014, pp. 148–163.
[20] B. Cuenca Grau, B. Motik, Reasoning over ontologies with hidden content: the import-by-query approach, J. Artif. Intell. Res. 45 (2012) 197–255.
[21] B. Cuenca Grau, B. Motik, Y. Kazakov, Import-by-query: ontology reasoning under access limitations, in: IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, 2009, pp. 727–732.
[22] T. Eiter, B. Glimm, Y. Kazakov, M. Krötzsch (Eds.), Informal Proceedings of the 26th International Workshop on Description Logics, Ulm, Germany, July 23-26, 2013, CEUR Workshop Proceedings, vol. 1014, CEUR-WS.org, 2013.
[23] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, Hermit: an OWL 2 reasoner, J. Autom. Reason. 53 (3) (2014) 245–269.
[24] G. Governatori, F. Olivieri, A. Rotolo, S. Scannapieco, Computing strong and weak permissions in defeasible logic, J. Philos. Log. 42 (6) (2013) 799–829.
[25] R.H. Güting, Graphdb: modeling and querying graphs in databases, in: J.B. Bocca, M. Jarke, C. Zaniolo (Eds.), VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile, September 12–15, 1994, Morgan Kaufmann, 1994, pp. 297–308.
[26] C. Haase, C. Lutz, Complexity of subsumption in the $\mathcal{EL}$ family of description logics: acyclic and cyclic tboxes, in: ECAI 2008 - 18th European Conference on Artificial Intelligence, Proceedings, in: Frontiers in Artificial Intelligence and Applications, vol. 178, IOS Press, 2008, pp. 25–29.
[27] R. Hoekstra, J. Breuker, M.D. Bello, A. Boer, LKIF core: principled ontology development for the legal domain, in: Law, Ontologies and the Semantic Web - Channelling the Legal Information Flood, 2009, pp. 21–52.
[28] I. Horrocks, O. Kutz, U. Sattler, The even more irresistible SROIQ, in: Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, AAAI Press, 2006, pp. 57–67.
[29] J.F. Horty, Agency and Deontic Logic, Oxford University Press, 2001.
[30] S. Jajodia, P. Samarati, M.L. Sapino, V.S. Subrahmanian, Flexible support for multiple access control policies, ACM Trans. Database Syst. 26 (2) (2001) 214–260.
[31] A.J.I. Jones, M.J. Sergot, On the characterization of law and computer systems: the normative systems perspective, in: J.-J.C. Meyer, R.J. Wieringa (Eds.), Deontic Logic in Computer Science: Normative System Specification, Wiley, 1993, pp. 275–307, chapter 8.
[32] L. Kagal, T.W. Finin, A. Joshi, A policy language for a pervasive computing environment, in: 4th IEEE International Workshop on Policies for Distributed Systems and Networks, POLICY, IEEE Computer Society, June 2003, p. 63.
[33] Y. Kazakov, M. Krötzsch, F. Simancik, The incredible ELK - from polynomial procedures to efficient reasoning with EL ontologies, J. Autom. Reason. 53 (1) (2014) 1–61.
[34] S. Kirrane, J.D. Fernández, W. Dullaert, U. Milosevic, A. Polleres, P.A. Bonatti, R. Wenning, O. Drozd, P. Raschke, A scalable consent, transparency and compliance architecture, in: The Semantic Web: ESWC 2018 Satellite Events, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 11155, Springer, 2018, pp. 131–136.
[35] N. Matentzoglu, S. Bail, B. Parsia, A corpus of OWL DL ontologies, in: Eiter et al. [22], pp. 829–841.
[36] B. Motik, R. Shearer, I. Horrocks, Hypertableau reasoning for description logics, J. Artif. Intell. Res. 36 (2009) 165–228.
[37] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, J. Banerjee, Rdfox: a highly-scalable RDF store, in: M. Arenas, Ó. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P.T. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, S. Staab (Eds.), The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Proceedings, Part II, Bethlehem, PA, USA, October 11-15, 2015, in: Lecture Notes in Computer Science, vol. 9367, Springer, 2015, pp. 3–20.
[38] M. Ortiz, S. Rudolph, M. Simkus, Worst-case optimal reasoning for the horn-dl fragments of OWL 1 and 2, in: Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, AAAI Press, 2010.
[39] M. Ortiz, S. Rudolph, M. Simkus, Query answering in the horn fragments of the description logics SHOIQ and SROIQ, in: IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI/AAAI, 2011, pp. 1039–1044.
[40] M. Palmirani, G. Governatori, Modelling legal knowledge for GDPR compliance checking, in: Legal Knowledge and Information Systems - JURIX 2018: the Thirty-First Annual Conference, Groningen, the Netherlands, 12–14 December 2018, 2018, pp. 101–110.
[41] M. Palmirani, M. Martoni, A. Rossi, C. Bartolini, L. Robaldo, Legal ontology for modelling GDPR concepts and norms, in: Legal Knowledge and Information Systems - JURIX 2018: the Thirty-First Annual Conference, Groningen, the Netherlands, 12–14 December 2018, 2018, pp. 91–100.
[42] M. Palmirani, M. Martoni, A. Rossi, C. Bartolini, L. Robaldo, Pronto: privacy ontology for legal reasoning, in: Electronic Government and the Information Systems Perspective - 7th International Conference, Proceedings, EGOVIS 2018, Regensburg, Germany, September 3-5, 2018, 2018, pp. 139–152.
[43] C.H. Papadimitriou, Computational Complexity, Academic Internet Publ., 2007.

[44] H. Prakken, G. Sartor, Law and logic: a review from an argumentation perspective, Artif. Intell. 227 (2015) 214–245.
[45] M.J. Sergot, F. Sadri, R.A. Kowalski, F. Kriwaczek, P. Hammond, H.T. Cory, The British nationality act as a logic program, Commun. ACM 29 (5) (1986) 370–386.
[46] E. Sherkhonov, B. Cuenca Grau, E. Kharlamov, E.V. Kostylev, Semantic faceted search with aggregation and recursion, in: The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Proceedings, Part I, Vienna, Austria, October 21-25, 2017, 2017, pp. 594–610.
[47] A. Steigmiller, T. Liebig, B. Glimm, Konclude: system description, J. Web Semant. 27–28 (2014) 78–85.
[48] A. Uszok, J.M. Bradshaw, R. Jeffers, N. Suri, P.J. Hayes, M.R. Breedy, L. Bunch, M. Johnson, S. Kulkarni, J. 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, IEEE Computer Society, June 2003, pp. 93–96.
[49] T.Y.C. Woo, S.S. Lam, Authorizations in distributed systems: a new approach, J. Comput. Secur. 2 (2–3) (1993) 107–136.