Real time reasoning in OWL2 for GDPR compliance
(AIJ 289, 2020)

P.A. Bonatti, L. Ioffredo, I. Petrova, L. Sauro, I. Siahaan, Università di Napoli and CeRICT

30th IJCAI, August 2021
Introduction

- Work carried out by the European H2020 project SPECIAL, grant n. 731601
- Being extended by the H2020 project TRAPEZE, grant n. 883464
Introduction

- Work carried out by the European H2020 project SPECIAL, grant n. 731601
- Being extended by the H2020 project TRAPEZE, grant n. 883464
- Goal: Semantic support to GDPR compliance
  - GDPR = European General Data Protection Regulation
Introduction

- Work carried out by the European H2020 project SPECIAL, grant n. 731601
- Being extended by the H2020 project TRAPEZE, grant n. 883464
- Goal: Semantic support to GDPR compliance
  - GDPR = European General Data Protection Regulation
- Preliminary version at IJCAI’18
  - A usage policy language $\mathcal{PL}$ based on OWL2
  - NP-completeness of $\mathcal{PL}$ and tractability of a GDPR-compatible restriction
  - A structural subsumption algorithm for PTIME compliance checking
Introduction

- Work carried out by the European H2020 project SPECIAL, grant n. 731601
- Being extended by the H2020 project TRAPEZE, grant n. 883464
- Goal: Semantic support to GDPR compliance
  - GDPR = European General Data Protection Regulation
- Preliminary version at IJCAI’18
  - A usage policy language $\mathcal{PL}$ based on OWL2
  - NP-completeness of $\mathcal{PL}$ and tractability of a GDPR-compatible restriction
  - A structural subsumption algorithm for PTIME compliance checking
- New contributions
  - Tractability extended to Horn-$\mathcal{SRIQ}$ knowledge bases
  - Using Import By Query and knowledge compilation
  - Experimental scalability analysis (real time compliance checks)
\( \mathcal{P} \mathcal{L} \) Policies (BeFit Example)

Data usage policies are formalized as unions of “simple policies” i.e. \( \mathcal{E} \mathcal{L} \) concepts extended with integer intervals:

\[
(\exists purp. \text{FitnessRecommendation} \sqcap \\
\exists data. \text{BiometricData} \sqcap \\
\exists proc. \text{Analytics} \sqcap \\
\exists recip. \text{BeFit} \sqcap \\
\exists storage. \text{loc.EU})
\]

\[
\sqcap \\
(\exists purp. \text{SocialNetworking} \sqcap \\
\exists data. \text{LocationData} \sqcap \\
\exists proc. \text{Transfer} \sqcap \\
\exists recip. \text{DataSubjFriends} \sqcap \\
\exists storage. (\text{loc.EU} \sqcap [y_1, y_5](dur)).
\]
Policies (BeFit Example)

Data usage policies are formalized as unions of “simple policies” i.e. $E\mathcal{L}$ concepts extended with integer intervals:

$$(\exists purp.\text{FitnessRecommendation} \sqcap$$
$$\exists data.\text{BiometricData} \sqcap$$
$$\exists proc.\text{Analytics} \sqcap$$
$$\exists recip.\text{BeFit} \sqcap$$
$$\exists storage.\text{loc.EU})$$

$\sqcap$

$$(\exists purp.\text{SocialNetworking} \sqcap$$
$$\exists data.\text{LocationData} \sqcap$$
$$\exists proc.\text{Transfer} \sqcap$$
$$\exists recip.\text{DataSubjFriends} \sqcap$$
$$\exists storage.(loc.EU \sqcap [y_1, y_5](dur)).$$

As a privacy policy: specifies what BeFit will do with the data
Data usage policies are formalized as unions of “simple policies”
i.e. $\mathcal{EL}$ concepts extended with integer intervals:

$$(\exists purp. \text{FitnessRecommendation} \land$$
$$(\exists data. \text{BiometricData} \land$$
$$(\exists proc. \text{Analytics} \land$$
$$(\exists recip. \text{BeFit} \land$$
$$(\exists storage. \text{loc.EU}$$
$$\land$$
$$(\exists purp. \text{SocialNetworking} \land$$
$$(\exists data. \text{LocationData} \land$$
$$(\exists proc. \text{Transfer} \land$$
$$(\exists recip. \text{DataSubjFriends} \land$$
$$(\exists storage. (loc.\text{EU} \land [y_1, y_5](dur)))$$.

As a privacy policy: specifies what BeFit will do with the data

As consent to processing: specifies what can be done with the data
Data usage policies are formalized as unions of “simple policies” i.e. $\mathcal{EL}$ concepts extended with integer intervals:

\[
(\exists \text{purp}.\text{FitnessRecommendation} \sqcap \\
\exists \text{data}.\text{BiometricData} \sqcap \\
\exists \text{proc}.\text{Analytics} \sqcap \\
\exists \text{recip}.\text{BeFit} \sqcap \\
\exists \text{storage}.\text{loc.EU}) \\
\sqcap \\
(\exists \text{purp}.\text{SocialNetworking} \sqcap \\
\exists \text{data}.\text{LocationData} \sqcap \\
\exists \text{proc}.\text{Transfer} \sqcap \\
\exists \text{recip}.\text{DataSubjFriends} \sqcap \\
\exists \text{storage}.(\text{loc.EU} \sqcap [y_1, y_5](\text{dur}))) .
\]

As a privacy policy: specifies what BeFit will do with the data

As consent to processing: specifies what can be done with the data

The objective part of the GDPR can be encoded in the same way
Vocabularies and Ontologies

- $\mathcal{PL}$ is vocabulary-neutral. One may use for example:
  - W3C DPVCG group (Data Privacy Vocabularies)
    https://www.w3.org/community/dpvCG/

- Vocabularies are axiomatized by knowledge bases containing:
  (IJCAI’18 version)
  - $\text{func}(R)$ where $R$ is a role name or a concrete feature;
  - $\text{range}(S, A)$ where $S$ is a role and $A$ a concept name;
  - $A \subseteq B$ where $A, B$ are concept names;
  - $\text{disj}(A, B)$ where $A, B$ are concept names.
Policy reasoning tasks

- All the main reasoning tasks are reduced to concept subsumption
  - **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 a policy update strengthen or relax the previous policy?

- Generally intractable due to the interplay of $[l, u](f)$ and $\sqcap$

**Theorem 7** Subsumption checking in $PL$ is coNP-complete. The result holds even if the knowledge base is empty.
Tractable case (IJCAI’18)

- The number of constraints \([l, u](f)\) in simple concepts is bounded by a constant

- PTIME algorithm for checking whether \(KB \models P_1 \subseteq P_2\):
  1. normalize the intervals \([l, u]\) of \(P_1\) (offline) – \(O(|P_1| \cdot |P_2|)\)
  2. “compile” the KB into \(P_1\) (offline) – \(O(|P_1| \cdot |KB|)\)
  3. apply a structural subsumption algorithm – \(O(|P_1| \cdot |P_2|)\)
Extension to Horn-$SRIQ$ KB

- Knowledge bases are partitioned into $\mathcal{K} \cup \mathcal{O}$ where:
  - $\mathcal{K}$ is a $\mathcal{PL}$ KB that defines policy properties with “func” and “range” axioms
  - $\mathcal{O}$ is a Horn-$SRIQ$ KB that defines classes and their properties (e.g. “LocationData” and its property “precision”)
  - In the policies, the roles defined in $\mathcal{O}$ may occur within the scope of those defined in $\mathcal{K}$, but not viceversa

- Reasoning is based on “Import By Query” (IBQ):
  - Normalization and structural subsumption query $\mathcal{O}$ with subsumptions of the form $A_1 \sqcap \ldots \sqcap A_n \subseteq A$
  - This is the only difference from the algorithms of IJCAI’18
Main theoretical results

- Tractability and intractability extend to $K \cup O$, where $O$ belongs to a tractable fragment of Horn-$SRIQ$ (e.g. $\mathcal{EL}$ or $DL$-lite)
Main theoretical results

- Tractability and intractability extend to $K \cup O$, where $O$ belongs to a tractable fragment of Horn-SRIQ (e.g. $EL$ or $DL$-lite)

- Negative results: Horn-SRIQ is the best we can get
  - nominals make IBQ incomplete (no Horn-SROIQ)
  - convexity is necessary for tractability ($O$ should better be Horn)
Main theoretical results

• Tractability and intractability extend to $K \cup O$, where $O$ belongs to a tractable fragment of Horn-$SRIQ$ (e.g. $EL$ or $DL$-lite)

• Negative results: Horn-$SRIQ$ is the best we can get
  – nominals make IBQ incomplete (no Horn-$SROIQ$)
  – convexity is necessary for tractability ($O$ should better be $Horn$)

• Under suitable conditions (compatible with GDPR compliance), $O$ can be compiled into a $\mathcal{PL}$ KB
  – then the IJCAI’18 framework applies
Another view of the theoretical framework

- $\mathcal{PL}$ policies are equivalent to *unions of conjunctive faceted queries with disequalities*

- Subsumption checking is equivalent to *containment* of such queries

- Against knowledge bases in (various fragments of) Horn-$SRIQ$
Experimental evaluation

- Sequential Java implementation, supporting the OWL API
  - with several optimizations (caching of normalized policies, pre-computation of normalization)
Experimental evaluation

• Sequential Java implementation, supporting the OWL API
  – with several optimizations (caching of normalized policies, pre-computation of normalization)

• Test cases:
  – Random perturbation of SPECIAL’s use case policies
  – Fully random policies and knowledge bases of increasing size
Experimental evaluation

• Sequential Java implementation, supporting the OWL API
  – with several optimizations (caching of normalized policies, pre-computation of normalization)

• Test cases:
  – Random perturbation of SPECIAL’s use case policies
  – Fully random policies and knowledge bases of increasing size

• Some representative results:
  – On fully random policies, and medium KB \( O(10^5) \) classes and axioms: 
    \( \sim 14.7 \text{ ms (avg)} \) per compliance check/subsumption
  – On the realistic policies: from 410 to 570 \( \mu \)-sec per compliance check
  – Compares favourably with Hermit, ELK, GraphDB, and RDFox (with the standard reduction of query containment to query answering)
Summary and ongoing work

- $\mathcal{PL}$ is generally intractable, but in applications interval constraints are limited $\Rightarrow$ compliance checking is tractable
  - also when the KB is in a tractable fragment of Horn-$SRIQ$
  - and – in some sense – when it can be compiled into a $\mathcal{PL}$ KB
Summary and ongoing work

• \( \mathcal{PL} \) is generally intractable, but in applications interval constraints are limited \( \Rightarrow \) compliance checking is tractable
  
  – also when the KB is in a tractable fragment of Horn-\(SRIQ\)
  
  – and – in some sense – when it can be compiled into a \( \mathcal{PL} \) KB

• Scalability tests prove that real-time compliance checking is possible in this framework
  
  – further improvements may be possible using more efficient languages and parallelism
Summary and ongoing work

- $\mathcal{PL}$ is generally intractable, but in applications interval constraints are limited $\Rightarrow$ compliance checking is tractable
  - also when the KB is in a tractable fragment of Horn-$SRIQ$
  - and – in some sense – when it can be compiled into a $\mathcal{PL}$ KB

- Scalability tests prove that real-time compliance checking is possible in this framework
  - further improvements may be possible using more efficient languages and parallelism

- Ongoing work in TRAPEZE:
  - extending policies with negation (“my location can be tracked but not when I’m here”)
Summary and ongoing work

- $\mathcal{PL}$ is generally intractable, but in applications interval constraints are limited $\Rightarrow$ compliance checking is tractable
  - also when the KB is in a tractable fragment of Horn-$\mathcal{SRIQ}$
  - and – in some sense – when it can be compiled into a $\mathcal{PL}$ KB

- Scalability tests prove that real-time compliance checking is possible in this framework
  - further improvements may be possible using more efficient languages and parallelism

- Ongoing work in TRAPEZE:
  - extending policies with negation (“my location can be tracked but not when I’m here”)
  - recursive policies that apply to recipients in transitive data transfers (extension with greatest fixpoints)
Summary and ongoing work

- $\mathcal{PL}$ is generally intractable, but in applications interval constraints are limited $\Rightarrow$ compliance checking is tractable
  - also when the KB is in a tractable fragment of Horn-$SR\mathcal{IQ}$
  - and – in some sense – when it can be compiled into a $\mathcal{PL}$ KB

- Scalability tests prove that real-time compliance checking is possible in this framework
  - further improvements may be possible using more efficient languages and parallelism

- Ongoing work in TRAPEZE:
  - extending policies with negation ("my location can be tracked but not when I’m here")
  - recursive policies that apply to recipients in transitive data transfers (extension with greatest fixpoints)
  - Questions?
Interval normalization

intervals occurring in:

\[ P_2 : \quad [ \quad ] \quad [ \quad ] \quad [ \quad ] \quad [ \quad ] \]
\[ P_1 : \quad [ \quad ] \quad [ \quad ] \quad [ \quad ] \]

split \( P_1 \)'s
intervals:

\[ [ \quad ][ \quad ][ \quad ] \quad [ \quad ] \quad [ \quad ] \]

Afterwards, for all new \([l_1, u_1]\) and all \([l_2, u_2]\) occurring in \(P_2\), either \([l_1, u_1] \subseteq [l_2, u_2]\) or \([l_1, u_1] \cap [l_2, u_2] = \emptyset\)

Interval splitting in concepts: \([l, u](f) \sim [l, x_1](f) \sqcup \ldots \sqcup [x_n, u](f)\)

Then unions are moved to the top level using
\[ \exists R. (C_1 \sqcup C_2) \equiv \exists R. C_1 \sqcup \exists R. C_2 \]

In the tractable cases, this takes polynomial time (and space)
Second normalization phase

1) \( \bot \cap D \rightarrow \bot \)
2) \( \exists R. \bot \rightarrow \bot \)
3) \([l, u](f) \rightarrow \bot \)
4) \((\exists R.D) \cap (\exists R.D') \cap D'' \rightarrow \exists R.(D \cap D') \cap D'' \)
5) \([l_1, u_1](f) \cap [l_2, u_2](f) \cap D \rightarrow [\max(l_1, l_2), \min(u_1, u_2)](f) \cap D \)
6) \(\exists R.D \cap D' \rightarrow \exists R.(D \cap A) \cap D' \)
7) \(A_1 \cap A_2 \cap D \rightarrow \bot \)

if \( l > u \)
if \( \text{func}(R) \in \mathcal{K} \)
if \( \text{func}(f) \in \mathcal{K} \)
if \( \text{range}(R, A) \in \mathcal{K} \) and \( A \) not a conjunct of \( D \)
if \( A_1 \sqsubseteq^* A_1', A_2 \sqsubseteq^* A_2', \) and \( \text{disj}(A_1', A_2') \in \mathcal{K} \)
The structural subsumption algorithm

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

\begin{itemize}
  \item **Input:** \(\mathcal{K}\) and an elementary \(C \sqsubseteq D\) where \(C\) is normalized
  \item **Output:** \(\text{true}\) if \(\mathcal{K} \models C \sqsubseteq D\), \(\text{false}\) otherwise
  \item **Note:** Below, by \(C = C' \cap C''\) we mean that either \(C = C'\) or \(C'\) is a conjunct of \(C\) (possibly not the first one)
\end{itemize}

\begin{algorithm}
\begin{algorithmic}
  \State \textbf{1} \textbf{begin}
  \State \textbf{2} \textbf{if} \(C = \bot\) \textbf{then} \textbf{return} \textbf{true}
  \State \textbf{3} \textbf{if} \(D = A, C = A' \cap C' \text{ and } A' \sqsubseteq^* A\) \textbf{then} \textbf{return} \textbf{true}
  \State \textbf{4} \textbf{if} \(D = [l, u](f) \text{ and } C = [l', u'](f) \cap C' \text{ and } l \leq l' \text{ and } u' \leq u\) \textbf{then} \textbf{return} \textbf{true}
  \State \textbf{5} \textbf{if} \(D = \exists R. D', C = (\exists R. C') \cap C'' \text{ and } STS(\mathcal{K}, C' \sqsubseteq D')\) \textbf{then} \textbf{return} \textbf{true}
  \State \textbf{6} \textbf{if} \(D = D' \cap D'', STS(\mathcal{K}, C \sqsubseteq D'), \text{ and } STS(\mathcal{K}, C \sqsubseteq D'')\) \textbf{then} \textbf{return} \textbf{true}
  \State \textbf{7} \textbf{else} \textbf{return} \textbf{false}
  \State \textbf{8} \textbf{end}
\end{algorithmic}
\end{algorithm}