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)

$\mathcal{PL}$ Policies (BeFit Example)

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

$$
\begin{array}{l}
(\exists \mathit{purp}.\mathrm{FitnessRecommendation} \sqcap \\
\quad \exists \mathit{data}.\mathrm{BiometricData} \sqcap \\
\quad \exists \mathit{proc}.\mathrm{Analytics} \sqcap \\
\quad \exists \mathit{recip}.\mathrm{BeFit} \sqcap \\
\quad \exists \mathit{storage}.\mathit{loc}.\mathrm{EU}) \\
\sqcup \\
(\exists \mathit{purp}.\mathrm{SocialNetworking} \sqcap \\
\quad \exists \mathit{data}.\mathrm{LocationData} \sqcap \\
\quad \exists \mathit{proc}.\mathrm{Transfer} \sqcap \\
\quad \exists \mathit{recip}.\mathrm{DataSubjFriends} \sqcap \\
\quad \exists \mathit{storage}.(\mathit{loc}.\mathrm{EU} \sqcap [y_1, y_5](\mathit{dur}))\,.
\end{array}
$$

$\mathcal{PL}$ Policies (BeFit Example)

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

$$
\begin{aligned}
&(\exists purp.\mathrm{FitnessRecommendation} \sqcap \\
&\quad \exists data.\mathrm{BiometricData} \sqcap \\
&\quad \exists proc.\mathrm{Analytics} \sqcap \\
&\quad \exists recip.\mathrm{BeFit} \sqcap \\
&\quad \exists storage.loc.\mathrm{EU}) \\
&\sqcup \\
&(\exists purp.\mathrm{SocialNetworking} \sqcap \\
&\quad \exists data.\mathrm{LocationData} \sqcap \\
&\quad \exists proc.\mathrm{Transfer} \sqcap \\
&\quad \exists recip.\mathrm{DataSubjFriends} \sqcap \\
&\quad \exists storage.(loc.\mathrm{EU} \sqcap [y_1, y_5](dur))\,.
\end{aligned}
$$

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

$\mathcal{PL}$ Policies (BeFit Example)

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

$$
\begin{aligned}
&(\exists purp.\mathrm{FitnessRecommendation} \sqcap \\
&\quad \exists data.\mathrm{BiometricData} \sqcap \\
&\quad \exists proc.\mathrm{Analytics} \sqcap \\
&\quad \exists recip.\mathrm{BeFit} \sqcap \\
&\quad \exists storage.loc.\mathrm{EU}) \\
&\sqcup \\
&(\exists purp.\mathrm{SocialNetworking} \sqcap \\
&\quad \exists data.\mathrm{LocationData} \sqcap \\
&\quad \exists proc.\mathrm{Transfer} \sqcap \\
&\quad \exists recip.\mathrm{DataSubjFriends} \sqcap \\
&\quad \exists storage.(loc.\mathrm{EU} \sqcap [y_1, y_5](dur))\,.
\end{aligned}
$$

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

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

$\mathcal{PL}$ Policies (BeFit Example)

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

$$
\begin{aligned}
&(\exists purp.\mathrm{FitnessRecommendation} \sqcap \\
&\quad \exists data.\mathrm{BiometricData} \sqcap \\
&\quad \exists proc.\mathrm{Analytics} \sqcap \\
&\quad \exists recip.\mathrm{BeFit} \sqcap \\
&\quad \exists storage.loc.\mathrm{EU}) \\
&\sqcup \\
&(\exists purp.\mathrm{SocialNetworking} \sqcap \\
&\quad \exists data.\mathrm{LocationData} \sqcap \\
&\quad \exists proc.\mathrm{Transfer} \sqcap \\
&\quad \exists recip.\mathrm{DataSubjFriends} \sqcap \\
&\quad \exists storage.(loc.\mathrm{EU} \sqcap [y_1, y_5](dur))\,.
\end{aligned}
$$

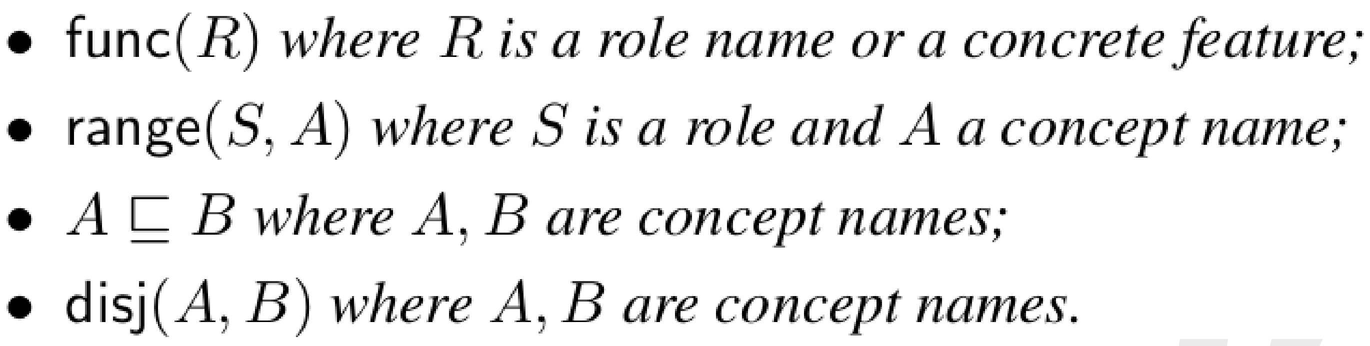
As a *privacy policy*: specifies what BeFit will do with the data

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

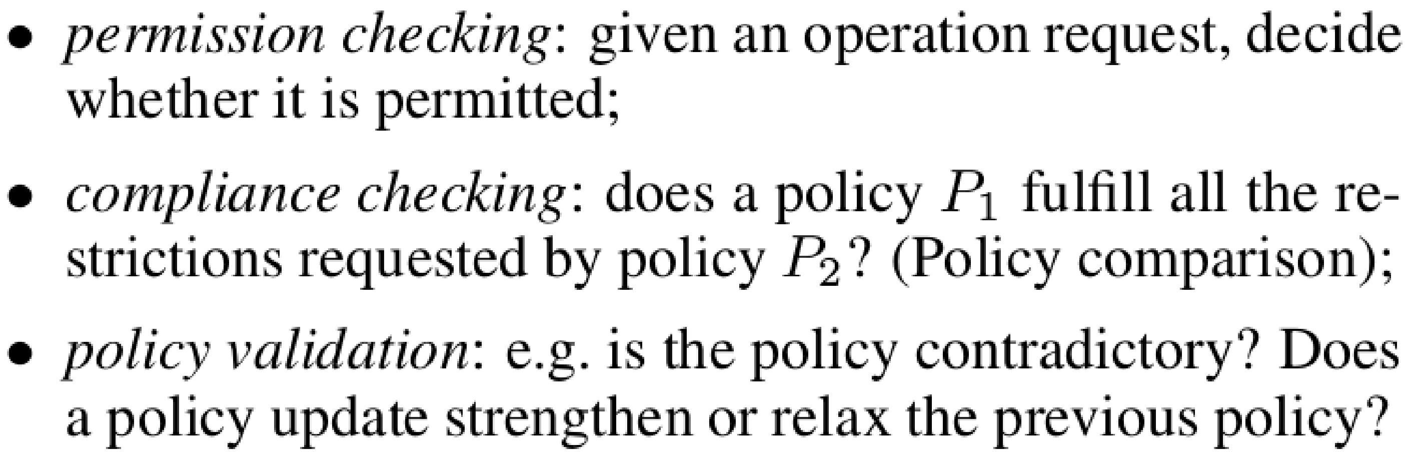
- $\mathcal{PL}$ is vocabulary-neutral. One may use for example:
  - W3C DPVCG group (Data Privacy Vocabularies) `https://www.w3.org/community/dpvcg/`
- Vocabularies are axiomatized by knowledge bases containing: (IJCAI'18 version)



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

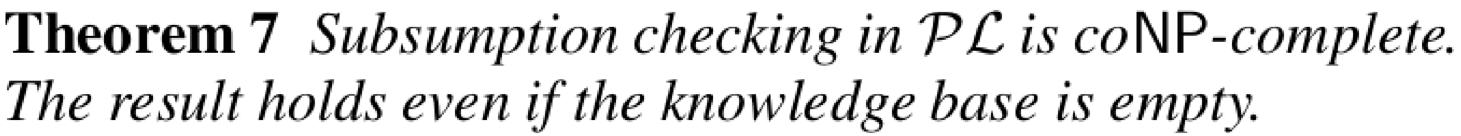
Policy reasoning tasks

- 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 $\sqcup$



Theorem 7 *Subsumption checking in $\mathcal{PL}$ is co*NP*-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 $KB \models P_1 \sqsubseteq P_2$:
  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-$\mathcal{SRIQ}$ 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-$\mathcal{SRIQ}$ 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 $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq A$
    - This is the only difference from the algorithms of IJCAI’18

Main theoretical results

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

Main theoretical results

- Tractability and intractability extend to $\mathcal{K} \cup \mathcal{O}$, where $\mathcal{O}$ belongs to a tractable fragment of Horn-$\mathcal{SRIQ}$ (e.g. $\mathcal{EL}$ or *DL-lite*)
- Negative results: Horn-$\mathcal{SRIQ}$ is the best we can get
  - nominals make IBQ incomplete (no Horn-$\mathcal{SROIQ}$)
  - convexity is necessary for tractability ($\mathcal{O}$ should better be *Horn*)

Main theoretical results

- Tractability and intractability extend to $\mathcal{K} \cup \mathcal{O}$, where $\mathcal{O}$ belongs to a tractable fragment of Horn-$\mathcal{SRIQ}$ (e.g. $\mathcal{EL}$ or *DL-lite*)
- Negative results: Horn-$\mathcal{SRIQ}$ is the best we can get
  - nominals make IBQ incomplete (no Horn-$\mathcal{SROIQ}$)
  - convexity is necessary for tractability ($\mathcal{O}$ should better be *Horn*)
- Under suitable conditions (compatible with GDPR compliance), $\mathcal{O}$ 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-$\mathcal{SRIQ}$

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): ~ 14.7 ms (avg) per compliance check/subsumption
  - On the realistic policies: from 410 to 570 μ-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-$\mathcal{SRIQ}$
  - 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-$\mathcal{SRIQ}$
  - 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

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”)

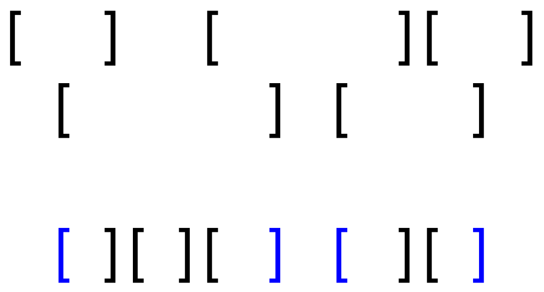
Summary and ongoing work

- $\mathcal{PL}$ 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-$\mathcal{SRIQ}$
  - 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)
  - Questions?

Interval normalization



intervals occurring in:

P_2 : [ ] [ ][ ]

P_1 : [ ] [ ]

split P_1's

intervals : [ ][ ][ ] [ ][ ]

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: $[l, u](f) \rightsquigarrow [l, x_1](f) \sqcup \ldots \sqcup [x_n, u](f)$

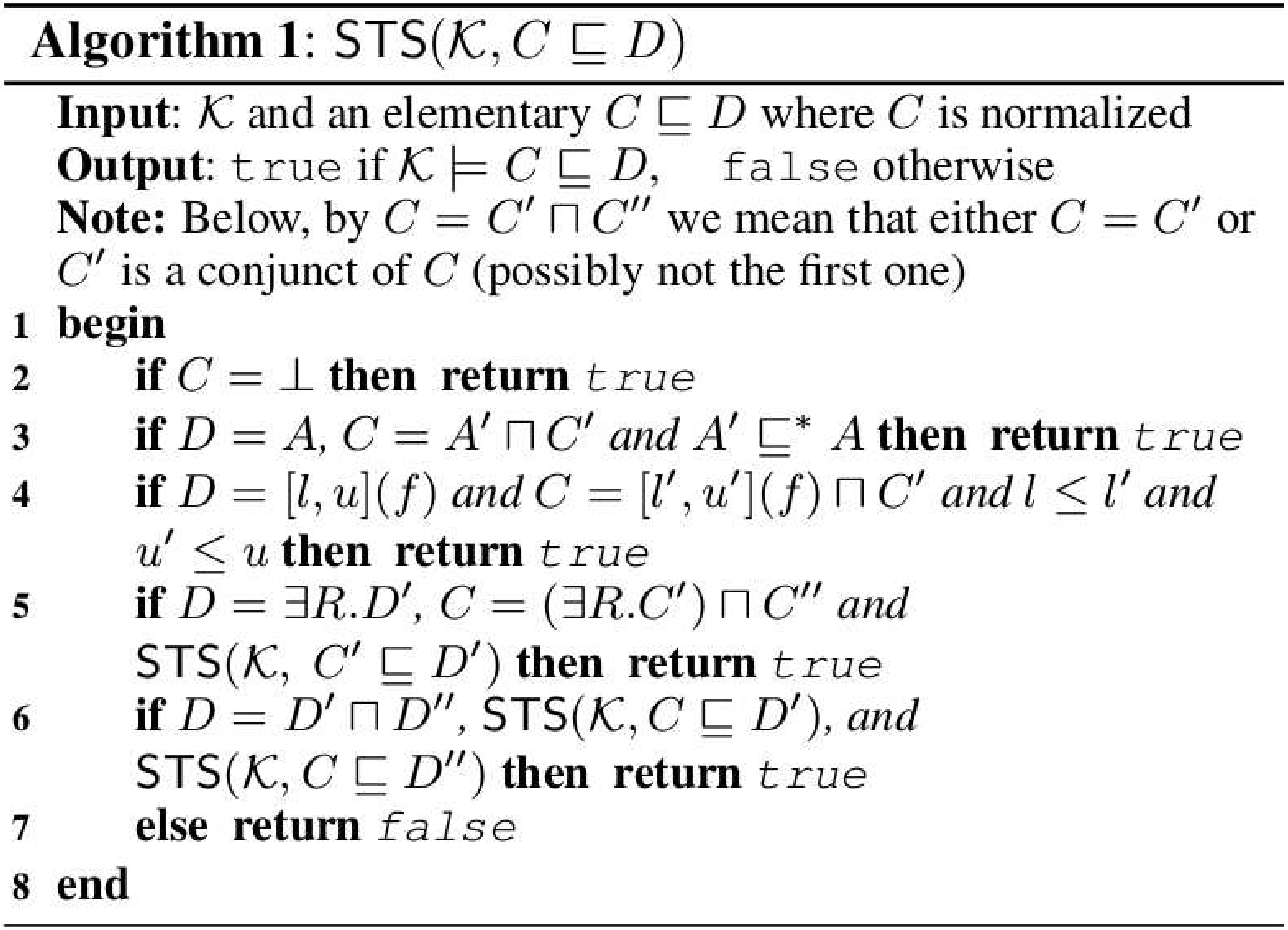
Then unions are moved to the top level using
$\exists R.(C_1 \sqcup C_2) \equiv \exists R.C_1 \sqcup \exists R.C_2$

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

Second normalization phase

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

The structural subsumption algorithm



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

```
Input: $\mathcal{K}$ and an elementary $C \sqsubseteq D$ where $C$ is normalized
Output: true if $\mathcal{K} \models C \sqsubseteq D$,    false otherwise
Note: Below, by $C = C' \sqcap C''$ we mean that either $C = C'$ or
$C'$ is a conjunct of $C$ (possibly not the first one)
begin
    if $C = \bot$ then return true
    if $D = A$, $C = A' \sqcap C'$ and $A' \sqsubseteq^* A$ then return true
    if $D = [l, u](f)$ and $C = [l', u'](f) \sqcap C'$ and $l \leq l'$ and
    $u' \leq u$ then return true
    if $D = \exists R.D'$, $C = (\exists R.C') \sqcap C''$ and
    $\mathsf{STS}(\mathcal{K},\ C' \sqsubseteq D')$ then return true
    if $D = D' \sqcap D''$, $\mathsf{STS}(\mathcal{K}, C \sqsubseteq D')$, and
    $\mathsf{STS}(\mathcal{K}, C \sqsubseteq D'')$ then return true
    else return false
end
```