Computing Repairs for Inconsistent DL-programs over $\mathcal{EL}$ Ontologies

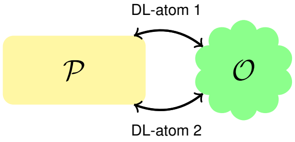
Thomas Eiter Michael Fink Daria Stepanova

Knowledge-Based Systems Group,
Institute of Information Systems,
Vienna University of Technology
http://www.kr.tuwien.ac.at/

JELIA 2014–September, 26, 2014

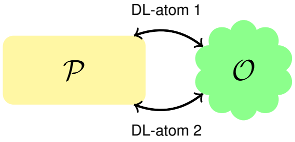
Motivation

- DL-program: rules $\mathcal{P}$ + consistent ontology $\mathcal{O}$ (loose coupling combination approach)
- DL-atoms serve as query interfaces to $\mathcal{O}$
- Possibility to add info from $\mathcal{P}$ to $\mathcal{O}$ prior to querying it: bidirectional data flow



Motivation

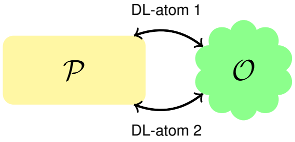
- DL-program: rules $\mathcal{P}$ + consistent ontology $\mathcal{O}$ (loose coupling combination approach)
- DL-atoms serve as query interfaces to $\mathcal{O}$
- Possibility to add info from $\mathcal{P}$ to $\mathcal{O}$ prior to querying it: bidirectional data flow



However, information exchange between $\mathcal{P}$ and $\mathcal{O}$ can cause **inconsistency** of the DL-program (absence of answer sets).

Motivation

- DL-program: rules $\mathcal{P}$ + consistent ontology $\mathcal{O}$ (loose coupling combination approach)
- DL-atoms serve as query interfaces to $\mathcal{O}$
- Possibility to add info from $\mathcal{P}$ to $\mathcal{O}$ prior to querying it: bidirectional data flow



However, information exchange between $\mathcal{P}$ and $\mathcal{O}$ can cause **inconsistency** of the DL-program (absence of answer sets).

✓ Repair answer sets [E_ *et* al, *IJCAI* 2013]

✓ Algorithm based on complete support families [E_ *et* al, *ECAI* 2014]

Effective for *DL-Lite*$_\mathcal{A}$ (few small support sets per DL-atom)
Not well suited for $\mathcal{EL}$ (might be many / large support sets . . .)

In this work: algorithm for repairing DL-programs over $\mathcal{EL}$ ontologies

Overview

$\mathcal{EL}$ Description Logic

- Lightweight DL, widely used in biology, medicine and other domains
- Concepts and roles model sets of objects and their relationships
- $\mathcal{EL}$-concept is formed according to the rule

$$C ::= A \mid \top \mid C \sqcap C \mid \exists R.C$$

- An $\mathcal{EL}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ consists of
  - TBox $\mathcal{T}$ specifying inclusions/equivalence between $\mathcal{EL}$-concepts
    $$C \sqsubseteq D \quad C \equiv D$$
  - ABox $\mathcal{A}$ specifying facts that hold in the domain
    $$A(b) \qquad R(a,b)$$

Example

$$\mathcal{T} = \left\{ \begin{array}{l} Blacklisted \sqsubseteq Staff \\ BLStaffRequest \equiv StaffRequest \sqcap \exists hasSubject.Blacklisted \end{array} \right\}$$

$$\mathcal{A} = \{\ StaffRequest(r1) \quad hasSubject(r1, john) \quad Blacklisted(john) \}$$

$\mathcal{EL}$ Description Logic

- Lightweight DL, widely used in biology, medicine and other domains
- Concepts and roles model sets of objects and their relationships
- $\mathcal{EL}$-concept is formed according to the rule

$$C ::= A \mid \top \mid C \sqcap C \mid \exists R.C$$

- An $\mathcal{EL}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ consists of
  - TBox $\mathcal{T}$ specifying inclusions/equivalence between $\mathcal{EL}$-concepts

    $$C \sqsubseteq D \quad C \equiv D$$

  - ABox $\mathcal{A}$ specifying facts that hold in the domain

    $$A(b) \qquad R(a, b)$$

- Normalized TBox $\mathcal{T}_{norm}$ contains only inclusions of the form

$$A_1 \sqsubseteq A_2 \quad A_1 \sqcap A_2 \sqsubseteq A_3 \quad \exists R.A_1 \sqsubseteq A_2 \quad A_1 \sqsubseteq R.A_2$$ [1]

[1] A_i is an atomic concept

Example: DL-program

$\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is a DL-program

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAct.Actqcap} \exists \mathit{hasSubj.Staff} \sqcap \exists \mathit{hasTarg.Proj} \\ (3)\ \mathit{BLStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubj.Blacklisted} \\ (4)\ \mathit{StaffRequest}(\mathit{r1}) \quad (5)\ \mathit{hasSubj}(\mathit{r1}, \mathit{john}) \quad (6)\ \mathit{Blacklisted}(\mathit{john}) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{projfile}(\mathit{p1});\quad (8)\ \mathit{hasowner}(\mathit{p1}, \mathit{john}); \\ (9)\ \mathit{grant}(\mathit{r1}) \leftarrow \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](\mathit{r1}), \mathit{not\ deny}(\mathit{r1}) \\ (10)\ \mathit{deny}(\mathit{r1}) \leftarrow \mathrm{DL}[; \mathit{BLStaffRequest}](\mathit{r1}) \end{array} \right\}$$

Example: DL-program

$\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is a DL-program

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ Blacklisted \sqsubseteq Staff \\ (2)\ StaffRequest \equiv \exists hasAct.Actqcap\exists hasSubj.Staff \sqcap \exists hasTarg.Proj \\ (3)\ BLStaffRequest \equiv StaffRequest \sqcap \exists hasSubj.Blacklisted \\ (4)\ StaffRequest(r1) \quad (5)\ hasSubj(r1, john) \quad (6)\ Blacklisted(john) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ projfile(p1); \quad (8)\ hasowner(p1, john); \\ (9)\ grant(r1) \leftarrow \text{DL}[Proj \uplus projfile; StaffRequest](r1), not\ deny(r1) \\ (10)\ deny(r1) \leftarrow \text{DL}[; BLStaffRequest](r1) \end{array} \right\}$$

- Interpretation: $I = \{projfile(p1), hasowner(p1, john), deny(r1)\}$
- Satisfaction relation: $I \models^{\mathcal{O}} projfile(p1)$;
  $I \models^{\mathcal{O}} \text{DL}[Proj \uplus projfile; StaffRequest](r1)$
  $I \models^{\mathcal{O}} \text{DL}[; BLStaffRequest](r1)$
- Semantics: in terms of answer sets, i.e. founded models (weak, flp, ...)
- I is a weak and flp answer set

Example: DL-program

$\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is a DL-program

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAct}.\mathit{Actqcap} \exists \mathit{hasSubj}.\mathit{Staff} \sqcap \exists \mathit{hasTarg}.\mathit{Proj} \\ (3)\ \mathit{BLStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubj}.\mathit{Blacklisted} \\ (4)\ \mathit{StaffRequest}(r1) \quad (5)\ \mathit{hasSubj}(r1, john) \quad (6)\ \mathit{Blacklisted}(john) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{projfile}(p1); \quad (8)\ \mathit{hasowner}(p1, john); \\ (9)\ \mathit{grant}(r1) \leftarrow \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1), \mathit{not}\ \mathit{deny}(r1) \\ (10)\ \mathit{deny}(r1) \leftarrow \mathrm{DL}[; \mathit{BLStaffRequest}](r1) \end{array} \right\}$$

- Interpretation: $I = \{\mathit{projfile}(p1), \mathit{hasowner}(p1, john), \mathit{deny}(r1)\}$
- Satisfaction relation: $I \models^{\mathcal{O}} \mathit{projfile}(p1)$;
  $I \models^{\mathcal{O}} \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1)$
  $I \models^{\mathcal{O}} \mathrm{DL}[; \mathit{BLStaffRequest}](r1)$
- Semantics: in terms of answer sets, i.e. founded models (weak, flp, . . .)
- I is a weak and flp answer set

Example: DL-program

$\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is a DL-program

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAct.Actqcap} \exists \mathit{hasSubj.Staff} \sqcap \exists \mathit{hasTarg.Proj} \\ (3)\ \mathit{BLStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubj.Blacklisted} \\ (4)\ \mathit{StaffRequest}(r1) \quad (5)\ \mathit{hasSubj}(r1, john) \quad (6)\ \mathit{Blacklisted}(john) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{projfile}(p1); \quad (8)\ \mathit{hasowner}(p1, john); \\ (9)\ \mathit{grant}(r1) \leftarrow \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1), \mathit{not\ deny}(r1) \\ (10)\ \mathit{deny}(r1) \leftarrow \mathrm{DL}[; \mathit{BLStaffRequest}](r1) \end{array} \right\}$$

- Interpretation: $I = \{\mathit{projfile}(p1), \mathit{hasowner}(p1, john), \mathit{deny}(r1)\}$
- Satisfaction relation: $I \models^{\mathcal{O}} \mathit{projfile}(p1)$;
  $I \models^{\mathcal{O}} \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1)$
  $I \models^{\mathcal{O}} \mathrm{DL}[; \mathit{BLStaffRequest}](r1)$
- Semantics: in terms of answer sets, i.e. founded models (weak, flp, . . .)
- I is a weak and flp answer set

Example: DL-program

$\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is a DL-program

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAct}.\mathit{Act} qcap \exists \mathit{hasSubj}.\mathit{Staff} \sqcap \exists \mathit{hasTarg}.\mathit{Proj} \\ (3)\ \mathit{BLStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubj}.\mathit{Blacklisted} \\ (4)\ \mathit{StaffRequest}(r1) \quad (5)\ \mathit{hasSubj}(r1, john) \quad (6)\ \mathit{Blacklisted}(john) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{projfile}(p1); \quad (8)\ \mathit{hasowner}(p1, john); \\ (9)\ \mathit{grant}(r1) \leftarrow \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1), \mathit{not}\ \mathit{deny}(r1) \\ (10)\ \mathit{deny}(r1) \leftarrow \mathrm{DL}[; \mathit{BLStaffRequest}](r1) \end{array} \right\}$$

- Interpretation: $I = \{\mathit{projfile}(p1), \mathit{hasowner}(p1, john), \mathit{deny}(r1)\}$
- Satisfaction relation: $I \models^{\mathcal{O}} \mathit{projfile}(p1)$;
  $I \models^{\mathcal{O}} \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1)$
  $I \models^{\mathcal{O}} \mathrm{DL}[; \mathit{BLStaffRequest}](r1)$
- Semantics: in terms of answer sets, i.e. founded models (weak, flp, . . .)
- I is a weak and flp answer set

Example: Inconsistent DL-program

$$\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$$

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAct}.\mathit{Act} \sqcap \exists \mathit{hasSubj}.\mathit{Staff} \sqcap \exists \mathit{hasTarg}.\mathit{Proj} \\ (3)\ \mathit{BLStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubj}.\mathit{Blacklisted} \\ (4)\ \mathit{StaffRequest}(r1) \quad (5)\ \mathit{hasSubj}(r1, john) \quad (6)\ \mathit{Blacklisted}(john) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{projfile}(p1) \quad (8)\ \mathit{hasowner}(p1, john) \\ (9)\ \mathit{grant}(r1) \leftarrow \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1), \mathit{not\ deny}(r1) \\ (10)\ \mathit{deny}(r1) \leftarrow \mathrm{DL}[; \mathit{BLStaffRequest}](r1) \\ (11)\ \bot \leftarrow \mathit{projfile}(p1), \mathit{hasowner}(p1, john), \\ \qquad\qquad \mathrm{DL}[; \mathit{hasSubj}](r1, john), \mathit{not\ grant}(r1) \end{array} \right\}$$

Example: Inconsistent DL-program

$$\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$$

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAct}.\mathit{Act} \sqcap \exists \mathit{hasSubj}.\mathit{Staff} \sqcap \exists \mathit{hasTarg}.\mathit{Proj} \\ (3)\ \mathit{BLStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubj}.\mathit{Blacklisted} \\ (4)\ \mathit{StaffRequest}(r1) \quad (5)\ \mathit{hasSubj}(r1, john) \quad (6)\ \mathit{Blacklisted}(john) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{projfile}(p1) \quad (8)\ \mathit{hasowner}(p1, john) \\ (9)\ \mathit{grant}(r1) \leftarrow \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1), \mathit{not}\ \mathit{deny}(r1) \\ (10)\ \mathit{deny}(r1) \leftarrow \mathrm{DL}[; \mathit{BLStaffRequest}](r1) \\ (11)\ \bot \leftarrow \mathit{projfile}(p1), \mathit{hasowner}(p1, john), \\ \qquad\qquad \mathrm{DL}[; \mathit{hasSubj}](r1, john), \mathit{not}\ \mathit{grant}(r1) \end{array} \right\}$$

Example: Inconsistent DL-program

$\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAct}.\mathit{Act} \sqcap \exists \mathit{hasSubj}.\mathit{Staff} \sqcap \exists \mathit{hasTarg}.\mathit{Proj} \\ (3)\ \mathit{BLStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubj}.\mathit{Blacklisted} \\ (4)\ \mathit{StaffRequest}(r1) \quad (5)\ \mathit{hasSubj}(r1, \mathit{john}) \quad (6)\ \mathit{Blacklisted}(\mathit{john}) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{projfile}(p1) \quad (8)\ \mathit{hasowner}(p1, \mathit{john}) \\ (9)\ \mathit{grant}(r1) \leftarrow \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1), \mathit{not}\ \mathit{deny}(r1) \\ (10)\ \mathit{deny}(r1) \leftarrow \mathrm{DL}[; \mathit{BLStaffRequest}](r1) \\ (11)\ \bot \leftarrow \mathit{projfile}(p1), \mathit{hasowner}(p1, \mathit{john}), \\ \qquad\qquad \mathrm{DL}[; \mathit{hasSubj}](r1, \mathit{john}), \mathit{not}\ \mathit{grant}(r1) \end{array} \right\}$$

Example: Inconsistent DL-program

$$\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$$

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAct}.\mathit{Act} \sqcap \exists \mathit{hasSubj}.\mathit{Staff} \sqcap \exists \mathit{hasTarg}.\mathit{Proj} \\ (3)\ \mathit{BLStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubj}.\mathit{Blacklisted} \\ (4)\ \mathit{StaffRequest}(r1) \quad (5)\ \mathit{hasSubj}(r1, john) \quad (6)\ \mathit{Blacklisted}(john) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{projfile}(p1) \quad (8)\ \mathit{hasowner}(p1, john) \\ (9)\ \mathit{grant}(r1) \leftarrow \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1), \mathit{not}\ \mathit{deny}(r1) \\ (10)\ \mathit{deny}(r1) \leftarrow \mathrm{DL}[; \mathit{BLStaffRequest}](r1) \\ (11)\ \bot \leftarrow \mathit{projfile}(p1), \mathit{hasowner}(p1, john), \\ \qquad \mathrm{DL}[; \mathit{hasSubj}](r1, john), \mathit{not}\ \mathit{grant}(r1) \end{array} \right\}$$

Example: Inconsistent DL-program

$$\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$$

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAct}.\mathit{Act} \sqcap \exists \mathit{hasSubj}.\mathit{Staff} \sqcap \exists \mathit{hasTarg}.\mathit{Proj} \\ (3)\ \mathit{BLStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubj}.\mathit{Blacklisted} \\ (4)\ \mathit{StaffRequest}(r1) \quad (5)\ \mathit{hasSubj}(r1, \mathit{john}) \quad (6)\ \mathit{Blacklisted}(\mathit{john}) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{projfile}(p1) \quad (8)\ \mathit{hasowner}(p1, \mathit{john}) \\ (9)\ \mathit{grant}(r1) \leftarrow \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1), \mathit{not}\ \mathit{deny}(r1) \\ (10)\ \mathit{deny}(r1) \leftarrow \mathrm{DL}[; \mathit{BLStaffRequest}](r1) \\ (11)\ \bot \leftarrow \mathit{projfile}(p1), \mathit{hasowner}(p1, \mathit{john}), \\ \qquad\qquad \mathrm{DL}[; \mathit{hasSubj}](r1, \mathit{john}), \mathit{not}\ \mathit{grant}(r1) \end{array} \right\}$$

Example: Inconsistent DL-program

$\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is inconsistent!

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAct}.\mathit{Act} \sqcap \exists \mathit{hasSubj}.\mathit{Staff} \sqcap \exists \mathit{hasTarg}.\mathit{Proj} \\ (3)\ \mathit{BLStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubj}.\mathit{Blacklisted} \\ (4)\ \mathit{StaffRequest}(r1) \quad (5)\ \mathit{hasSubj}(r1, \mathit{john}) \quad (6)\ \mathit{Blacklisted}(\mathit{john}) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{projfile}(p1) \quad (8)\ \mathit{hasowner}(p1, \mathit{john}) \\ (9)\ \mathit{grant}(r1) \leftarrow \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1), \mathit{not}\ \mathit{deny}(r1) \\ (10)\ \mathit{deny}(r1) \leftarrow \mathrm{DL}[; \mathit{BLStaffRequest}](r1) \\ (11)\ \bot \leftarrow \mathit{projfile}(p1), \mathit{hasowner}(p1, \mathit{john}), \\ \qquad\qquad \mathrm{DL}[; \mathit{hasSubj}](r1, \mathit{john}), \mathit{not}\ \mathit{grant}(r1) \end{array} \right\}$$

Example: Inconsistent DL-program

$\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is consistent!

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAct}.\mathit{Act} \sqcap \exists \mathit{hasSubj}.\mathit{Staff} \sqcap \exists \mathit{hasTarg}.\mathit{Proj} \\ (3)\ \mathit{BLStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubj}.\mathit{Blacklisted} \\ (4)\ \mathit{StaffRequest}(r1) \qquad (6)\ \mathit{Blacklisted}(john) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{projfile}(p1) \quad (8)\ \mathit{hasowner}(p1, john) \\ (9)\ \mathit{grant}(r1) \leftarrow \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1), \mathit{not}\ \mathit{deny}(r1) \\ (10)\ \mathit{deny}(r1) \leftarrow \mathrm{DL}[; \mathit{BLStaffRequest}](r1) \\ (11)\ \bot \leftarrow \mathit{projfile}(p1), \mathit{hasowner}(p1, john), \\ \qquad \mathrm{DL}[; \mathit{hasSubj}](r1, john), \mathit{not}\ \mathit{grant}(r1) \end{array} \right\}$$

$I = \{\mathit{projfile}(p1), \mathit{hasowner}(p1, john), \mathit{grant}(r1)\}$ is a repair answer set of Π.

Example: Inconsistent DL-program

$\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is consistent!

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAct}.\mathit{Act} \sqcap \exists \mathit{hasSubj}.\mathit{Staff} \sqcap \exists \mathit{hasTarg}.\mathit{Proj} \\ (3)\ \mathit{BLStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubj}.\mathit{Blacklisted} \\ (4)\ \mathit{StaffRequest}(r1) \quad (5)\ \mathit{hasSubj}(r1, john) \quad (6)\ \mathit{Blacklisted}(john) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (7)\ \mathit{projfile}(p1) \quad (8)\ \mathit{hasowner}(p1, john) \\ (9)\ \mathit{grant}(r1) \leftarrow \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1), \mathit{not}\ \mathit{deny}(r1) \\ (10)\ \mathit{deny}(r1) \leftarrow \mathrm{DL}[; \mathit{BLStaffRequest}](r1) \\ (11)\ \bot \leftarrow \mathit{projfile}(p1), \mathit{hasowner}(p1, john), \\ \qquad\qquad \mathrm{DL}[; \mathit{hasSubj}](r1, john), \mathit{not}\ \mathit{grant}(r1) \end{array} \right\}$$

$I = \{\mathit{projfile}(p1), \mathit{hasowner}(p1, john), \mathit{grant}(r1)\}$ is a repair answer set of Π.

Further repair for I: e.g. remove $\mathit{Blacklisted}(john)$

Support Sets for DL-atoms

$$\mathcal{O} = \Big\{ \overbrace{(1)\ StaffRequest \equiv \exists hasTarg.Proj}^{\mathcal{T}} \quad \overbrace{(2)\ hasTarg(r1, p1)}^{\mathcal{A}} \Big\}$$

$$d = \mathrm{DL}[Proj \uplus projfile; StaffRequest](r1)$$

- Support Sets encode partial info about $\mathcal{O}$, relevant for value of d

Support Sets for DL-atoms

$$\mathcal{O} = \Big\{ \overbrace{(1)\ \mathit{StaffRequest} \equiv \exists \mathit{hasTarg}.\mathit{Proj}}^{\mathcal{T}} \quad \overbrace{(2)\ \mathit{hasTarg}(r1, p1)}^{\mathcal{A}} \Big\}$$

$d = \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1)$

- Support Sets encode partial info about $\mathcal{O}$, relevant for value of d
- Ground Support Set for d is set S of atoms over *projfile* such that for all interpretations $I \supseteq S$, it holds that $I \models d$ ("*implicant*")

  E.g., $S = \{\mathit{projfile}(p1)\}$

Support Sets for DL-atoms

$$\mathcal{O} = \Big\{ \overbrace{(1)\ \mathit{StaffRequest} \equiv \exists \mathit{hasTarg}.\mathit{Proj}}^{\mathcal{T}} \quad \overbrace{(2)\ \mathit{hasTarg}(r1, p1)}^{\mathcal{A}} \Big\}$$

$$d = \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](r1)$$

- Support Sets encode partial info about $\mathcal{O}$, relevant for value of d
- Ground Support Set for d is set S of atoms over *projfile* such that for all interpretations $I \supseteq S$, it holds that $I \models d$ ("*implicant*")

  E.g., $S = \{\mathit{projfile}(p1)\}$
- Complete Support Family $\mathcal{S}$ for d consists of S's s.t. whenever $I \models d$, some $S \in \mathcal{S}$ with $I \supseteq S$ exists

Support Sets for DL-atoms

$$\mathcal{O} = \Big\{ \overbrace{(1)\ \mathit{StaffRequest} \equiv \exists \mathit{hasTarg}.\mathit{Proj}}^{\mathcal{T}} \quad \overbrace{(2)\ \mathit{hasTarg}(r1, p1)}^{\mathcal{A}} \Big\}$$

$d = \mathrm{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](\mathbf{X})$

- Support Sets encode partial info about $\mathcal{O}$, relevant for value of d
- Ground Support Set for d is set S of atoms over *projfile* such that for all interpretations $I \supseteq S$, it holds that $I \models d$ ("*implicant*")

  E.g., $S = \{\mathit{projfile}(p1)\}$
- Complete Support Family $\mathcal{S}$ for d consists of S's s.t. whenever $I \models d$, some $S \in \mathcal{S}$ with $I \supseteq S$ exists

- Nonground Support Set for $d(\mathbf{X})$ is $S = \langle N, \gamma \rangle$, where
  N: set of nonground literals over input predicates of $d(\mathbf{X})$

  γ : "guard" function, selects from groundings of N support sets for $d(c)$

Support Sets for DL-atoms

$$\mathcal{O} = \Bigg\{ \overbrace{(1)\ \mathit{StaffRequest} \equiv \exists \mathit{hasTarg}.\mathit{Proj}}^{\mathcal{T}} \quad \overbrace{(2)\ \mathit{hasTarg}(r1, p1)}^{\mathcal{A}} \Bigg\}$$

$$d = \text{DL}[\mathit{Proj} \uplus \mathit{projfile}; \mathit{StaffRequest}](\mathbf{X})$$

- Support Sets encode partial info about $\mathcal{O}$, relevant for value of d
- Ground Support Set for d is set S of atoms over *projfile* such that for all interpretations $I \supseteq S$, it holds that $I \models d$ ("*implicant*")

  E.g., $S = \{\mathit{projfile}(p1)\}$

- Complete Support Family $\mathcal{S}$ for d consists of S's s.t. whenever $I \models d$, some $S \in \mathcal{S}$ with $I \supseteq S$ exists
- Nonground Support Set for $d(\mathbf{X})$ is $S = \langle \overbrace{\mathit{projfile}(Y)}^{N}, \gamma \rangle$, where
  N: set of nonground literals over input predicates of $d(\mathbf{X})$

  $\gamma : \mathcal{C} \times \mathit{grnd}_{\mathcal{C}}(\mathit{projfile}(Y)) \rightarrow \{0, 1\}$ where $\gamma(c, \mathit{projfile}(c')) = 1$ if $\mathit{hasTarg}(c, c') \in \mathcal{A}$

Nonground Support Set Computation

$\mathcal{T} = \{StaffRequest \equiv \exists hasSubj.Staff \sqcap \exists hasTarg.Proj\}$
$d = \text{DL}[Proj \uplus projfile; StaffRequest](X)$

Nonground Support Set Computation

$\mathcal{T} = \{StaffRequest \equiv \exists hasSubj.Staff \sqcap \exists hasTarg.Proj\}$
$d = \mathrm{DL}[Proj \uplus projfile; StaffRequest](X)$

- Construct $\mathcal{T}_d$ by compiling info about input predicates of d into $\mathcal{T}$:
  $\mathcal{T}_d = \mathcal{T} \cup \{Proj_{projfile} \sqsubseteq Proj\}$

Nonground Support Set Computation

$\mathcal{T} = \{StaffRequest \equiv \exists hasSubj.Staff \sqcap \exists hasTarg.Proj\}$
$d = \mathrm{DL}[Proj \uplus projfile; StaffRequest](X)$

- Construct $\mathcal{T}_d$ by compiling info about input predicates of d into $\mathcal{T}$:
  $\mathcal{T}_d = \mathcal{T} \cup \{Proj_{projfile} \sqsubseteq Proj\}$

- Rewrite DL-query over normalized $\mathcal{T}_d$ into a datalog program:

$$\mathcal{T}_{d_{norm}} = \left\{ \begin{array}{ll} (1)\ StaffRequest \sqsubseteq \exists hasSubj.Staff & (2)\ Proj_{projfile} \sqsubseteq Proj \\ (3)\ StaffRequest \sqsubseteq hasTarg.Proj & (4)\ \exists hasSubj.Staff \sqsubseteq C_1 \\ (5)\ \exists hasTarg.Proj \sqsubseteq C_2 & (6)\ C_1 \sqcap C_2 \sqsubseteq StaffRequest \end{array} \right\}$$

Nonground Support Set Computation

$\mathcal{T} = \{StaffRequest \equiv \exists hasSubj.Staff \sqcap \exists hasTarg.Proj\}$
$d = \mathrm{DL}[Proj \uplus projfile; StaffRequest](X)$

- Construct $\mathcal{T}_d$ by compiling info about input predicates of d into $\mathcal{T}$:
  $\mathcal{T}_d = \mathcal{T} \cup \{Proj_{projfile} \sqsubseteq Proj\}$

- Rewrite DL-query over normalized $\mathcal{T}_d$ into a datalog program:

$$\mathcal{P}_{\mathcal{T}_{d_{norm}}} = \left\{ \begin{array}{l} (1^*)\ StaffRequest(X) \leftarrow C_1(X), C_2(X) \\ (2^*)\ C_1(X) \leftarrow hasSubj(X,Y), Staff(Y) \\ (3^*)\ C_2(X) \leftarrow hasTarg(X,Y), Proj(Y) \\ (4^*)\ Proj(X) \leftarrow Proj_{projfile}(X) \end{array} \right\}$$

Nonground Support Set Computation

$\mathcal{T} = \{StaffRequest \equiv \exists hasSubj.Staff \sqcap \exists hasTarg.Proj\}$
$d = \text{DL}[Proj \uplus projfile; StaffRequest](X)$

- Construct $\mathcal{T}_d$ by compiling info about input predicates of d into $\mathcal{T}$:
  $\mathcal{T}_d = \mathcal{T} \cup \{Proj_{projfile} \sqsubseteq Proj\}$
- Rewrite DL-query over normalized $\mathcal{T}_d$ into a datalog program:

$$\mathcal{P}_{\mathcal{T}_{d_{norm}}} = \left\{ \begin{array}{l} (1^*)\ StaffRequest(X) \leftarrow C_1(X), C_2(X) \\ (2^*)\ C_1(X) \leftarrow hasSubj(X,Y), Staff(Y) \\ (3^*)\ C_2(X) \leftarrow hasTarg(X,Y), Proj(Y) \\ (4^*)\ Proj(X) \leftarrow Proj_{projfile}(X) \end{array} \right\}$$

- Unfold the DL-query and extract support sets:

$StaffRequest(X) \leftarrow hasSubj(X,Y), Staff(Y), hasTarg(X,Z), Proj(Z)$
$StaffRequest(X) \leftarrow hasSubj(X,Y), Staff(Y), hasTarg(X,Z), Proj_{projfile}(Z)$

Nonground Support Set Computation

$\mathcal{T} = \{StaffRequest \equiv \exists hasSubj.Staff \sqcap \exists hasTarg.Proj\}$
$d = \text{DL}[Proj \uplus projfile; StaffRequest](X)$

- Construct $\mathcal{T}_d$ by compiling info about input predicates of d into $\mathcal{T}$:
  $\mathcal{T}_d = \mathcal{T} \cup \{Proj_{projfile} \sqsubseteq Proj\}$

- Rewrite DL-query over normalized $\mathcal{T}_d$ into a datalog program:

$$\mathcal{P}_{\mathcal{T}_{d_{norm}}} = \left\{ \begin{array}{l} (1^*)\ StaffRequest \leftarrow C_1(X), C_2(X) \\ (2^*)\ C_1(X) \leftarrow hasSubj(X, Y), Staff(Y) \\ (3^*)\ C_2(X) \leftarrow hasTarg(X, Y), Proj(Y) \\ (4^*)\ Proj(X) \leftarrow Proj_{projfile}(X) \end{array} \right\}$$

- Unfold the DL-query and extract support sets:

  $S_1 = \langle \emptyset, \gamma \rangle$ (i.e., $N = \emptyset$) and $\gamma(c, \emptyset) = 1$ if for some c', c'', we have $hasSubj(c, c'), Staff(c'), hasTarg(c, c''), Proj(c'') \in \mathcal{A}$

  $S_2 = \langle projfile(X), \gamma \rangle$ (i.e., $N = \{projfile(X)\}$), and $\gamma(c, projfile(c')) = 1$ if for some c'', we have $hasSubj(c, c''), Staff(c''), hasTarg(c, c') \in \mathcal{A}$

Nonground Support Set Computation

$\mathcal{T} = \{StaffRequest \equiv \exists hasSubj.Staff \sqcap \exists hasTarg.Proj\}$
$d = \text{DL}[Proj \uplus projfile; StaffRequest](X)$

- Construct $\mathcal{T}_d$ by compiling info about input predicates of d into $\mathcal{T}$:
  $\mathcal{T}_d = \mathcal{T} \cup \{Proj_{projfile} \sqsubseteq Proj\}$
- Rewrite DL-query over normalized $\mathcal{T}_d$ into a datalog program:

$$\mathcal{P}_{\mathcal{T}_{d_{norm}}} = \left\{ \begin{array}{l} (1^*)\ StaffRequest \leftarrow C_1(X), C_2(X) \\ (2^*)\ C_1(X) \leftarrow hasSubj(X, Y), Staff(Y) \\ (3^*)\ C_2(X) \leftarrow hasTarg(X, Y), Proj(Y) \\ (4^*)\ Proj(X) \leftarrow Proj_{projfile}(X) \end{array} \right\}$$

- Unfold the DL-query and extract support sets:

$S_1 = \{hasSubj(X, Y), Staff(X), hasTarg(X, Z), Proj(Z)\}$
$S_2 = \{hasSubj(X, Y), Staff(X), hasTarg(X, Z), Proj_{projfile}(Z)\}$

Nonground Support Set Computation

$\mathcal{T} = \{StaffRequest \equiv \exists hasSubj.Staff \sqcap \exists hasTarg.Proj\}$
$d = \text{DL}[Proj \uplus projfile; StaffRequest](X)$

- Construct $\mathcal{T}_d$ by compiling info about input predicates of d into $\mathcal{T}$:
  $\mathcal{T}_d = \mathcal{T} \cup \{Proj_{projfile} \sqsubseteq Proj\}$

- Rewrite DL-query over normalized $\mathcal{T}_d$ into a datalog program:

$$\mathcal{P}_{\mathcal{T}_{d_{norm}}} = \left\{ \begin{array}{l} (1^*)\ StaffRequest \leftarrow C_1(X), C_2(X) \\ (2^*)\ C_1(X) \leftarrow hasSubj(X,Y), Staff(Y) \\ (3^*)\ C_2(X) \leftarrow hasTarg(X,Y), Proj(Y) \\ (4^*)\ Proj(X) \leftarrow Proj_{projfile}(X) \end{array} \right\}$$

- Unfold the DL-query and extract support sets:

  – infinitely many support sets (axioms $\exists R.A \sqsubseteq A$)
  – exponentially many for acyclic $\mathcal{T}$

- Completeness is costly!
- Compute partial support families: bound size/number

Repair Answer Set Computation

✓ Compute partial support families **S** for all DL-atoms of Π

Repair Answer Set Computation

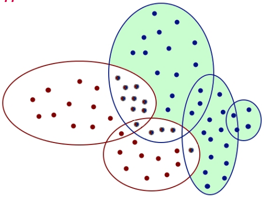
- ✓ Compute partial support families **S** for all DL-atoms of Π
- Construct $\hat{\Pi}$ from $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$:
  - Replace all DL-atoms a with normal atoms e_a
  - Add guessing rules on values of a: $e_a \vee ne_a$

Repair Answer Set Computation

- ✓ Compute partial support families **S** for all DL-atoms of Π
- Construct $\hat{\Pi}$ from $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$:
  - Replace all DL-atoms a with normal atoms e_a
  - Add guessing rules on values of a: $e_a \vee ne_a$
- For all $\hat{I} \in AS(\hat{\Pi})$: $D_p = \{a \mid e_a \in \hat{I}\}$; $D_n = \{a \mid ne_a \in \hat{I}\}$
- ✓ Ground support sets in **S** wrt. $\hat{I}$ and $\mathcal{A}$: $S_{gr}^{\hat{I}} \leftarrow Gr(\mathbf{S}, \hat{I}, \mathcal{A})$

Repair Answer Set Computation

- ✓ Compute partial support families **S** for all DL-atoms of Π
- Construct $\hat{\Pi}$ from $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$:
  - Replace all DL-atoms a with normal atoms e_a
  - Add guessing rules on values of a: $e_a \vee ne_a$
- For all $\hat{I} \in AS(\hat{\Pi})$: $D_p = \{a \mid e_a \in \hat{I}\}$; $D_n = \{a \mid ne_a \in \hat{I}\}$
- ✓ Ground support sets in **S** wrt. $\hat{I}$ and $\mathcal{A}$: $S^{\hat{I}}_{gr} \leftarrow Gr(\mathbf{S}, \hat{I}, \mathcal{A})$
- ✓ For all HS $H \subseteq \mathcal{A}$ of support families for all $a \in D_n$:
  - ✓ If all $a \in D_p$ have at least one $S \in S^{\hat{I}}_{gr}$, s.t. $S \cap H = \emptyset$, then do eval. postcheck on D_n (evaluate atoms from D_n over I and $\mathcal{A} \backslash H$)
  - ✓ Else do eval. postcheck on D_n and D_p
- ✓ Check minimality of $\hat{I}|_{\Pi}$ wrt. $\Pi' = \langle \mathcal{T}, \mathcal{A} \backslash H, \mathcal{P} \rangle$

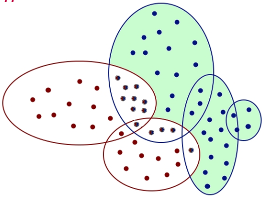


Repair Answer Set Computation

- ✓ Compute partial support families **S** for all DL-atoms of Π
- • Construct $\hat{\Pi}$ from $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$:
  - • Replace all DL-atoms a with normal atoms e_a
  - • Add guessing rules on values of a: $e_a \vee ne_a$

Sound wrt. deletion repair answer sets,
complete if all support families are complete!

- ✓ For all HS $H \subseteq \mathcal{A}$ of support families for all $a \in D_n$:
  - ✓ If all $a \in D_p$ have at least one $S \in S^{\hat{I}}_{gr}$, s.t. $S \cap H = \emptyset$, then do eval. postcheck on D_n (evaluate atoms from D_n over I and $\mathcal{A} \backslash H$)
  - ✓ Else do eval. postcheck on D_n and D_p
- ✓ Check minimality of $\hat{I}|_{\Pi}$ wrt. $\Pi' = \langle \mathcal{T}, \mathcal{A} \backslash H, \mathcal{P} \rangle$

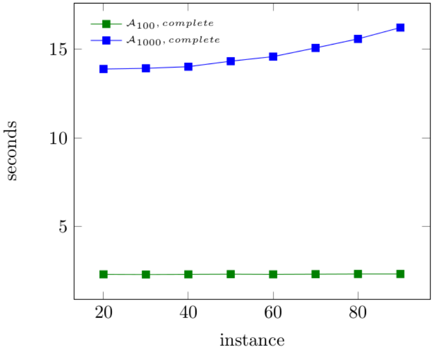


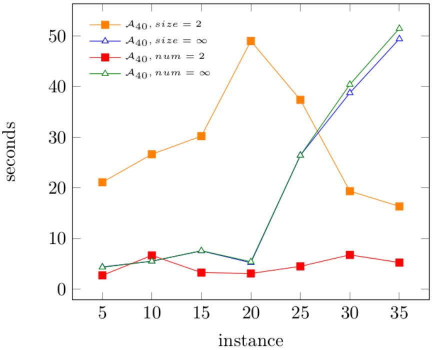
Declarative Implementation

- Implementation within DLVHEX
- Requiem tool is used for support set computation
- Repair is computed in a declarative manner using ASP techniques:

$$
\begin{array}{ll}
(1) & S_a(\mathbf{X}) \leftarrow S_a^{\mathcal{P}}(\mathbf{Y}) \\
(2) & S_a(\mathbf{X}) \leftarrow S_a^{\mathcal{A},\mathcal{P}}(\mathbf{Y}) \\
(3) & S_a^{\mathcal{P}}(\mathbf{Y}) \leftarrow rb(S_a^{p}(\mathbf{Y})) \\
(4) & S_a^{\mathcal{A},\mathcal{P}}(\mathbf{Y}) \leftarrow rb(S_a^{\mathcal{A},\mathcal{P}}(\mathbf{Y})), nd(S^{\mathcal{A},\mathcal{P}}(\mathbf{Y})) \\
(5) & \perp \leftarrow ne_a(\mathbf{X}), S_a^{\mathcal{P}}(\mathbf{Y}) \\
(6) & \bar{P}_{1a}(\mathbf{Y}) \vee \ldots \vee \bar{P}_{na}(\mathbf{Y}) \leftarrow ne_a(\mathbf{X}), S_a^{\mathcal{A},\mathcal{P}}(\mathbf{Y}) \\
(7) & eval_a(\mathbf{X}) \leftarrow e_a(\mathbf{X}), not\ C_a(\mathbf{X}), not\ S_a(\mathbf{X}) \\
(8) & eval_a(\mathbf{X}) \leftarrow ne_a(\mathbf{X}), not\ C_a(\mathbf{X})
\end{array}
$$

Benchmarks-Policy





- Add axiom *Blacklisted* $\sqsubseteq$ *Unauthorized*
- ABoxes $\mathcal{A}$: staff size n, 30% unauthorized 20% blacklisted
- $hasowner(p_i, s_i)$ with probability p (x-axis)
- Complete vs partial support families, with bounded/unbounded number/size of supports (2, ∞)
- Few support sets, but size > 2

Benchmarks-Open Street Map

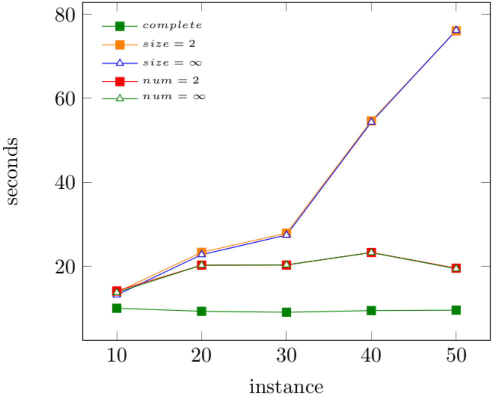
$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{BuildingFeature} \sqcap \exists \mathit{isLocatedInside}.\mathit{Private} \sqsubseteq \mathit{NoPublicAccess} \\ (2)\ \mathit{BusStop} \sqcap \mathit{Roofed} \sqsubseteq \mathit{CoveredBusStop} \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (9)\ \mathit{publicstation}(X) \leftarrow \mathrm{DL}[\mathit{BusStop} \uplus \mathit{busstop}; \mathit{CoveredBusStop}](X); \\ \qquad\qquad\qquad\qquad \mathit{not}\ \mathrm{DL}[; \mathit{Private}](X); \\ (10)\ \bot \leftarrow \mathrm{DL}[\mathit{BuildingFeature} \uplus \mathit{publicstation}; \mathit{NoPublicAccess}](X), \\ \qquad\qquad \mathit{publicstation}(X). \end{array} \right\}$$

- Rules on top of the MyITS ontology:[2]
  - personalized route planning with semantic information
  - TBox with 406 axioms
- $\mathcal{O}$ (part): building features located inside private areas are not publicly accessible, covered bus stops are those with roof.
- *P* checks that public stations don't lack public access, using CWA on private areas.

[2] http://www.kr.tuwien.ac.at/research/projects/myits/

Benchmarks-Open Street Map



- ABox $\mathcal{A}$ with bus stops (285) and leisure areas (682) of Cork, plus role *isLocatedInside* on them (9)
- Randomly made 80% bus stops roofed, 60% leisure areas private
- For *isLocatedInside*(*bs*, *la*) make *bs* a bus stop with *p* chance (*x*-axis)
- Many support sets have size $\leq$ 2

Conclusion and Future Work

Conclusions:

- Generalization of repair answer set computation for $\mathcal{EL}$
  - Partial support families: restricting support sets size/number
- Formal definition of support sets for $\mathcal{EL}$ and their computation
- Declarative realization within DLVHEX
- Evaluation on a set of benchmarks

Further and future work:

- Computing preferred repairs
- Syntactic conditions ensuring support families completeness
- Repair by bounded addition..