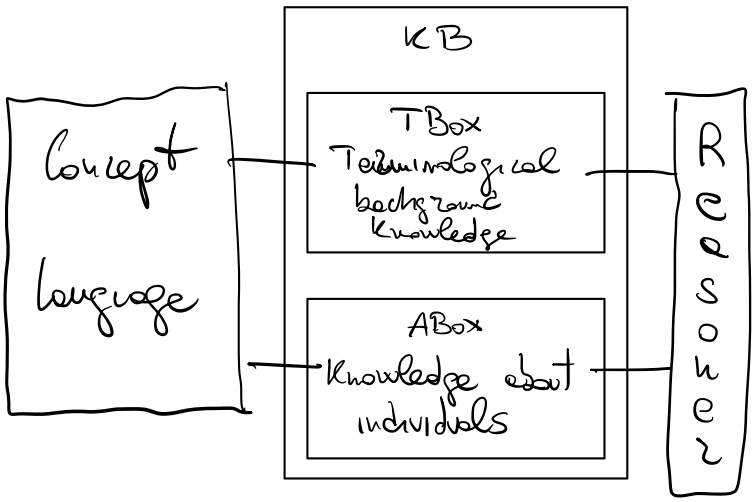
Description logic

KNOWLEDGE REPRESENTATION

Develop formalisms for providing high-level descriptions of the world that can be effectively used to build intelligent applications.



Concept language

- Concept names assign a name to group of objects
- role names assign a name to relations between objects
- constructors allows to related concept names and role names

$\mathcal{ALC}$

concept names $A, B, \ldots$

role names $r, s, \ldots$

$\neg C$ negation

$C \sqcap D$ conjunction

$C \sqcup D$ disjunction

$\exists r.C$ existential restriction

$\forall r.C$ value restriction

$\top$ top concept

$\bot$ bottom concept

Semantic of complex concepts:

$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$

$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$

$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$

$(\exists r.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \exists e : e \in \Delta^{\mathcal{I}} \text{ with } (d,e) \in r^{\mathcal{I}} \text{ and } e \in C^{\mathcal{I}}\}$

$(\forall r.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \forall e : e \in \Delta^{\mathcal{I}}, (d,e) \in r^{\mathcal{I}} \text{ implies } e \in C^{\mathcal{I}}\}$

model of C: interpretation $\mathcal{I}$ with $C^{\mathcal{I}} \neq 0$

C is <u>satisfiable</u> if there exists a model of C

Does $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ hold for all $\mathcal{I}$? <u>(subsumption)</u>
If $C \sqsubseteq D$, then D is <u>more general</u> than C

Does $C^{\mathcal{I}} = D^{\mathcal{I}}$ hold for all $\mathcal{I}$? <u>(equivalence)</u>
If $C \equiv D$, then D and C 'say the same'

<u>TBOX</u>

<u>concept axioms</u>: - primitive concept definition $A \sqsubseteq D$ $A \in N_C$
- concept definition $A \equiv D$ $A \in N_C$
- <u>General concept inclusion</u> $C \sqsubseteq D$
holds in an interpretation $\mathcal{I}$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
- General concept equivalence $C \equiv D$
holds in an interpretation $\mathcal{I}$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$

$\mathcal{I}$ is a model of a TBox $\mathcal{T}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all $C \sqsubseteq D \in \mathcal{T}$

Kinds of TBox

1- general TBox if:
- finite set of concept axioms
- cyclic definitions and GCIs are allowed

2- unfoldable TBox if:
- only (primitive) concept definitions
- concept names at most once on the left-hand side of definitions
- no cyclic definitions, no GCIs

→ can be conceived as macro definitions

Example

CONCEPT NAMES = $\{A, B\}$

ROLE NAMES = $\{r\}$

INTERPRETATION $\mathcal{I}_1 = (\Delta^{\mathcal{I}_1}, \cdot^{\mathcal{I}_1})$

$\Delta^{\mathcal{I}_1} = \{d_1, d_2, d_3\}$

$A^{\mathcal{I}_1} = \{d_1, d_2\}$

$B^{\mathcal{I}_1} = \{d_2\}$

$r^{\mathcal{I}_1} = \{(d_1, d_3), (d_3, d_2)\}$

CONCEPT EXPRESSIONS:

$(\neg A)^{\mathcal{I}_1} = \Delta^{\mathcal{I}_1} - A^{\mathcal{I}_1} = \{d_3\}$

$(A \sqcap \neg B)^{\mathcal{I}_1} = A^{\mathcal{I}_1} \cap (\neg B)^{\mathcal{I}_1} = A^{\mathcal{I}_1} \cap (\Delta^{\mathcal{I}_1} - B^{\mathcal{I}_1}) = \{d_1, d_2\} \cap \{d_1, d_3\} = \{d_1\}$

$(\neg A \sqcup B)^{\mathcal{I}_1} = (\neg A)^{\mathcal{I}_1} \cup B^{\mathcal{I}_1} = \{d_3\} \cup \{d_2\} = \{d_2, d_3\}$

$(\exists r.A)^{\mathcal{I}_1} = \{d \in \Delta^{\mathcal{I}_1} \mid \underline{\exists e \in \Delta^{\mathcal{I}_1}} \text{ s.t. } \underline{(d,e) \in r^{\mathcal{I}_1}} \text{ AND } \underline{e \in A^{\mathcal{I}_1}}\}$

- d_1 is not satisfying all conditions
- d_2 " "
- d_3 satisfy both conditions $\rightarrow$ $\{d_3\}$

$(\exists r.\neg A)^{\mathcal{I}_1} : \{d_1\}$

$(\forall r, A)^{\mathcal{I}_1} = \{d \in \Delta^{\mathcal{I}_1} \mid \underline{\forall e \in \Delta^{\mathcal{I}_1}}, \underline{\text{IF } (d,e) \in r_1^{\mathcal{I}_1}} \text{ THEN } \underline{e \in A^{\mathcal{I}_1}}\}$

$= \{d_3, d_2\}$

for d_2 we have to consider it also if it not appears in r
(IF-THEN IS FALSE ONLY IF THE THEN PART IS FALSE)

Reasoning tasks for TBoxes:

- Concept satisfiability w.r.t. TBoxes
  given C and T, there exist a common model of C and T?
- Concept subsumption w.r.t. TBoxes $(C \sqsubseteq_T D)$
  given C, D and T, Does $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ hold in all models of T?
- classification of the TBoxes
  computation of all subsumption relationship between all named concepts in T

ABox

Assertions in DL systems are:

- Concept assertions $C(a)$
- role assertions $r(a,b)$

Extend interpretations to individuals

$a \in N_I$, $a^I \in \Delta^I$

Semantics of assertions:

- Concept assertions: I satisfies $C(a) \longleftrightarrow a^I \in C^I$
- role assertions: I satisfies $r(a,b) \longleftrightarrow (a^I, b^I) \in r^I$

An ABox $\mathcal{A}$ is a finite set of assertions

I is a model for an ABox $\mathcal{A}$ if I satisfies all assertions in $\mathcal{A}$

Reasoning tasks for ABoxes:

- ABox consistency

  given A and T, do they have a common model?
- Instance checking

  given A, T, individual a and concept C, does $a^I \in C^I$ hold in all models of A and T?
- ABox realization

  given A and T. Compute for each individual a in A:

  the named concepts in T of which a is an instance of

Example

Person(Ann) Person(Joe) Employee(Mary) works-with(Ann, Mary)

Woman(Ann) → A

$I = (\Delta^I, \cdot^I)$ $\Delta^I = \{d_1, d_2, d_3, d_4, d_5\}$

$Ann^I = d_1$ $Joe^I = d_2$ $Mary^I = d_4$

$Person^I = \{d_1, d_2, d_3, d_4\}$ $Employee^I = \{d_3, d_4\}$ $Woman^I = \{d_1, d_3, d_4\}$

$works\text{-}with = \{(d_1, d_2), (d_2, d_4)\}$

Is $\mathcal{I}$ a model of ABox A ?

- $Ann^{\mathcal{I}} \in Person^{\mathcal{I}} \rightarrow d_1 \in \{d_1, d_2, d_3, d_4\}$ ✓
- $bc^{\mathcal{I}} \in Person^{\mathcal{I}} \rightarrow d_2 \in \{d_1, d_2, d_3, d_4\}$ ✓
- $Mary^{\mathcal{I}} \in Employee^{\mathcal{I}} \rightarrow d_4 \in \{d_3, d_4\}$ ✓
- $(Ann^{\mathcal{I}}, Mary^{\mathcal{I}}) \in works\text{-}with^{\mathcal{I}} \rightarrow (d_1, d_4) \in \{(d_1, d_2), (d_2, d_4)\}$ ✗ we can stop here

-let's change works-with $= \{(d_1, d_2), (d_1, d_4)\}$

- $(Ann^{\mathcal{I}}, Mary^{\mathcal{I}}) \in works\text{-}with^{\mathcal{I}} \rightarrow (d_1, d_4) \in \{(d_1, d_2), (d_2, d_4)\}$ ✓
- $Ann^{\mathcal{I}} \in Woman^{\mathcal{I}} \rightarrow d_1 \in \{d_1, d_3, d_4\}$ ✓

-let's add some TBox

$\mathcal{T}$ = Employee $\sqsubseteq$ Person

Woman $\sqsubseteq$ works-with. Woman

$Employee^{\mathcal{I}} \subseteq Person^{\mathcal{I}}$? → yes

$Woman^{\mathcal{I}} \subseteq (\forall \underbrace{works\text{-}with. Woman}_{\text{↳ } d_1 \text{ violate}})^{\mathcal{I}} \rightarrow \{d_1, d_3, d_4\} \subseteq \{d_2, d_3, d_4\}$? → NO

d_2 never participates, so yes

d_3, d_4 same as before

$\mathcal{I}$ is not a model of $\mathcal{T}$

Description Logic beyond $\mathcal{ALC}$

number restriction

$$(\leq n\, r)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}}\} \leq n\}$$

$$(\geq n\, r)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}}\} \geq n\}$$

qualified number restriction

$$(\leq n\, r\, C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq n\}$$

$$(\geq n\, r\, C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq n\}$$

Role declarations

r atomic role $\quad r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

f feature or attribute $\quad f^{\mathcal{I}} = \{(x,y) \mid (x,y) \in f^{\mathcal{I}} \wedge (x,z) \in f^{\mathcal{I}} \Rightarrow y = z\}$

$r \sqsubseteq s$ role inclusion $\quad r \sqsubseteq s$ holds in $\mathcal{I} \iff r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$

r^+ transitive role $\quad (r^+)^{\mathcal{I}} = \{(x,z) \mid (x,y) \in r^{\mathcal{I}}, (y,z) \in r^{\mathcal{I}} \Rightarrow (x,z) \in r^{\mathcal{I}}\}$

r^- inverse role $\quad (r^-)^{\mathcal{I}} = \{(y,x) \mid (x,y) \in r^{\mathcal{I}}\}$

good reasoning algorithm should be:

- terminating : you get always an answer
- sound : every positive answer is correct
- complete : every negative answer is correct

Reduction of inferences

Equivalence $\iff$ Subsumption $\quad C \equiv_{\mathcal{T}} D$ iff $C \sqsubseteq_{\mathcal{T}} D$ and $D \sqsubseteq_{\mathcal{T}} C$

Subsumption $\iff$ Satisfiability $\quad C \sqsubseteq_{\mathcal{T}} D$ iff $C \sqcap \neg D$ unsatisfiable w.r.t. $\mathcal{T}$

$C \not\sqsubseteq_{\mathcal{T}} \bot$ iff C is satisfiable w.r.t. $\mathcal{T}$ unsatisfiable w.r.t. $\mathcal{T}$

Instance checking $\iff$ ABox consistency $\quad a$ is instance of C w.r.t. $(\mathcal{T}, \mathcal{A})$ iff $(\mathcal{T}, \mathcal{A} \cup \{\neg C(a)\})$ is incosistent

Satisfiability $\iff$ ABox consistency $\quad C$ is satisfiable w.r.t. $\mathcal{T}$ iff $(\mathcal{T}, \{C(a)\})$ is consistent

Reasoning Method:

1. use the reduction to reformulate the reasoning problem
2. expand concepts w.r.t. TBox
3. normalize concept descriptions
4. apply tableux rules

Expansion of Concept descriptions

Let C be concept, $\mathcal{T}$ unfoldable TBox

1- replace every concept name of a defined concept with the right-hand side of its definitions $A \equiv C$

2- repeat until no more replacement can be made.

expansion process terminates due to acyclicity of the concept definitions

A concept C is in negation normal form (NNF) if negation occurs only in front of concept names

$$\neg\neg C \rightarrow C$$
$$\neg(C \sqcap D) \rightarrow \neg C \sqcup \neg D$$
$$\neg(C \sqcup D) \rightarrow \neg C \sqcap \neg D$$
$$\neg(\exists r.C) \rightarrow \forall r.\neg C$$
$$\neg(\forall r.C) \rightarrow \exists r.\neg C$$

Tableau Algorithm

- represent potential models by proof ABoxes
- to decide satisfiability of C_0, start with one initial proof ABox A_0
- repeatedly apply tableau rules and check for obvious contradictions
- return "satisfiable" iff a complete and contradiction-free proof ABox was found

Rules

	Precondition	Replace A by:
$\rightarrow_{\sqcap}$	$(C_1 \sqcap C_2)(x) \in A$ $C_1(x) \notin A$ or $C_2(x) \notin A$	$A' := A \cup \{C_1(x), C_2(x)\}$
$\rightarrow_{\sqcup}$	$(C_1 \sqcup C_2)(x) \in A$ $C_1(x) \notin A$ and $C_2(x) \notin A$	$A' := A \cup \{(C_1)(x)\}$ $A'' := A \cup \{(C_2)(x)\}$
$\rightarrow_{\exists}$	$(\exists r.C)(x) \in A$ but no z in A s.t. $\{r(x,z), C(z)\} \subseteq A$	$A' := A \cup \{r(x,z), C(z)\}$
$\rightarrow_{\forall}$	$\{(\forall r.C)(x), r(x,y)\} \subseteq A$ but $C(y) \notin A$	$A' := A \cup \{C(y)\}$

Initially, S contains proof ABox for concept C_0:

$S := \{A_0\}$, with $A_0 := \{C_0(x_0)\}$

Apply tableaux rules to set of proof ABoxes S until

- a proof ABox is complete (no more rules applicable)

or

- there exists an individual x in A s.t. $\{B(x), \neg B(x)\} \subseteq A$ for some concept name B (clash) or $\bot(x) \in A$

Role depth of concepts $d(C)$

$d(A) = 0 \qquad A \in N_C$

$d(\neg C) = d(C)$

$d(C \sqcap D) = d(C \sqcup D) = \max\{d(C), d(D)\}$

$d(\exists r.C) = d(\forall r.C) = d(C) + 1$

Example:

$C = A \sqcap \neg(B \sqcap \exists r.A)$ decide if C is sat?

- transform into NNF

$$A \sqcap (\neg B \sqcup \neg \exists r.A) = A \sqcap (\neg B \sqcup \forall r.\neg A)$$

ABox $A_0 = \{A \sqcap (\neg B \sqcup \forall r.\neg A)(x_0)\}$

AND-RULE: $A_1 = A_0 \cup \{A(x_0), \neg B \sqcup \forall r.\neg A(x_0)\}$

OR-RULE: $A_2 = A_1 \cup \{\neg B(x_0)\}$ $A_3 = A_1 \cup \{\forall r.\neg A(x_0)\}$

A_2: COMPLETE, CLASH-FREE

A_3: COMPLETE, CLASH-FREE

FOR ALL RULE NOT APPLICABLE

$\Rightarrow$ The initial ABox A_0 is satisfiable $\Rightarrow$ The initial concept C is satisfiable too.

Build a model for it: considering A_2

$\Delta^{\mathcal{I}} = \{d_1\}$ $\quad A^{\mathcal{I}} = \{d_1\}$ (because $A(x_0) \in A_2$)

$x_0^{\mathcal{I}} = d_1$ $\quad B^{\mathcal{I}} = \emptyset$

$r^{\mathcal{I}} = \emptyset$

Tableau rule for general TBoxes

1. Code all GCIs into one

For $\mathcal{T} = \{C_1 \sqsubseteq D_1, C_2 \sqsubseteq D_2, \dots, C_n \sqsubseteq D_n\}$

build the GCI $\top \sqsubseteq C_{GCI}$ with

$$C_{GCI} \equiv (\neg C_1 \sqcup D_1) \sqcap (\neg C_2 \sqcup D_2) \sqcap \dots \sqcap (\neg C_n \sqcup D_n)$$

2. Assert C_{GCI} for every individual: new tableau rule

$\longrightarrow_{\top \sqsubseteq C_{GCI}}$: If x in $\mathcal{A}$ and $C_{GCI}(x) \notin \mathcal{A}$,

then replace $\mathcal{A}$ with $\mathcal{A}' = \mathcal{A} \cup \{C_{GCI}(x)\}$

Ancestor Blocking

An individual x is <u>directly blocked</u> by an individual y iff:

- there is a path from y to x in $\mathcal{A}$
- x was generated by $\longrightarrow_\exists$ after y
- $\{C \mid C(x) \in \mathcal{A}\} \subseteq \{D \mid D(y) \in \mathcal{A}\}$

An individual x is <u>indirectly blocked</u> if:

- there is a path from y to x in $\mathcal{A}$
- y is directly blocked

Replace the exists rule $\longrightarrow_\exists$ by a exists rule with blocking $\longrightarrow_{\exists\square}$

$\longrightarrow_{\exists\square}$

Precondition	Replace $\mathcal{A}$ by
$(\exists r.C)(x) \in \mathcal{A}$ and x is not (indirectly) blocked but no z in $\mathcal{A}$ s.t. $\{r(x,z), C(z)\} \subseteq \mathcal{A}$	$\mathcal{A}' := \mathcal{A} \cup \{r(x,z), C(z)\}$

Build model w.r.t. blocking:

How to obtain a model for $\mathcal{T} = \{B \sqsubseteq \exists r.B\}$ Introduce 'back links'

Example $\mathcal{T} = \{B \sqsubseteq \exists r.B\}$ $\quad \mathcal{A} = \{B(a)\}$ $\quad C_{GCI} = \neg B \sqcup \exists r.B$

$A_0 = \{B(a)\}$

C_{GCI}-rule $A_1 = A_0 \cup \{(\neg B \sqcup \exists r.B)(a)\}$

$\sqcup$-rule $A_2 = A_1 \cup \{\neg B(a)\} \mid A_3 = A_1 \cup \{\exists r.B(a)\}$

closed

$\exists$-rule: $A_4 = A_3 \cup \{r(a,x_1), B(x_1)\}$

C_{GCI}-rule: $A_5 = A_4 \cup \{(\neg B \sqcup \exists r.B)(x_1)\}$

$\sqcup$-rule: $A_6 = A_5 \cup \{\neg B(x_1)\} \mid A_7 = A_5 \cup \{\exists r.B(x_1)\}$

CLOSED

$\exists$ rule is blocked because x_1 is blocked by a
- a is connected to x_1
- a is 'older' than x_1
- x_1 is generated by an $\exists$ rule
- $\{C \mid C(x_1) \in A_7\} \subseteq \{C \mid C(a) \in A_7\}$

$\{B, (\neg B \sqcup \exists r.B), \exists r.B\}$ $\quad =\!=$ $\quad \{B, (\neg B \sqcup \exists r.B), \exists r.B\}$

$\longrightarrow A_7$ is complete and open $\rightarrow$ alg return yes ($(\mathcal{T}, \mathcal{A})$ is satisfiable)

Can we derive a model?

Model for $(\mathcal{T}, \mathcal{A})$ derived by A_7:

$\Delta^{\mathcal{I}} = \{d_1, d_2\}$

$a^{\mathcal{I}} = d_1$

$x_1^{\mathcal{I}} = d_2$

$B^{\mathcal{I}} = \{d_1, d_2\}$

$r^{\mathcal{I}} = \{(d_1, d_2)\}$

$\mathcal{I}$ is a model of $\mathcal{A}$

Is $\mathcal{I}$ a model of $\mathcal{T}$? $(\exists r.B)^{\mathcal{I}} = \{d_1\}$

$B^{\mathcal{I}} \subseteq (\exists r.B)^{\mathcal{I}}$?

$\{d_1, d_2\} \not\subseteq \{d_1\}$ NO

to solve this problem, we must consider a and x_1 as the same domain element

So the real model is: $\Delta^{\mathcal{I}} = \{d_1\}$ $\quad x_1^{\mathcal{I}} = d_1$

$a^{\mathcal{I}} = d_1$ $\quad B^{\mathcal{I}} = \{d_1\}$

$r^{\mathcal{I}} = \{(d_1, d_1)\}$

$\Rightarrow$ $(\exists r.B)^{\mathcal{I}} = \{d_1\}$

$B^{\mathcal{I}} \subseteq (\exists r.B)^{\mathcal{I}}$? YES

Exercise

Exercise 1 Given the following TBox:

$$\begin{aligned} A &\sqsubseteq B \\ B &\sqsubseteq C \\ C &\sqsubseteq \exists r.D \\ D &\sqsubseteq \neg A \end{aligned}$$

1. tell whether the TBox $\mathcal{T}$ is satisfiable, and if so, show a model for $\mathcal{T}$;
2. tell whether the concept D is satisfiable with respect to $\mathcal{T}$, and if so, show a model for $\mathcal{T}$ where the interpretation of D is non-empty;
3. tell whether the concept expression $A \sqcap D$ is satisfiable with respect to $\mathcal{T}$, and if so, show a model for $\mathcal{T}$ where the interpretation of $A \sqcap D$ is non-empty.

1- we look for $\mathcal{I}$ s.t. $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$, $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, $C^{\mathcal{I}} \subseteq (\exists r.D)^{\mathcal{I}}$, $D^{\mathcal{I}} \subseteq (\neg A)^{\mathcal{I}}$

Let's define $A^{\mathcal{I}} = B^{\mathcal{I}} = C^{\mathcal{I}} = D^{\mathcal{I}} = \emptyset$

$\Delta^{\mathcal{I}} = \{d\}$

$r = \emptyset$

It works only if we have on the left side no negation!

2- we modify the previous $\mathcal{I}$

$$\mathcal{I}' = \begin{cases} \Delta^{\mathcal{I}'} = \{d\} \\ A^{\mathcal{I}'} = B^{\mathcal{I}'} = C^{\mathcal{I}'} = \emptyset \\ D^{\mathcal{I}'} = \{d\} \\ r = \emptyset \end{cases}$$

$\mathcal{I}'$ is a model of $\mathcal{T}$?

$D^{\mathcal{I}} \subseteq (\neg A)^{\mathcal{I}}$

$\{d\} \quad \{d\}$ YES

3- we should modify also the int. of A

$$\mathcal{I}'' = \begin{cases} \Delta^{\mathcal{I}''} = \{d\} \\ A^{\mathcal{I}'} = \{d\} \\ D^{\mathcal{I}'} = \{d\} \\ B^{\mathcal{I}'} = C^{\mathcal{I}'} = \emptyset \\ r = \emptyset \end{cases}$$

but we have to satisfy

$D^{\mathcal{I}} \subseteq (\neg A)^{\mathcal{I}}$

$D^{\mathcal{I}} \wedge A^{\mathcal{I}}$

} however is impossible to satisfy both conditions

$\Downarrow$

$A \sqcap B$ not satisfiable w.r.t. $\mathcal{T}$

The tableau method can be used to solve TBox satisfiability in this way:

Start from the ABox $\{T(x_0)\}$; if there is a model of this ABox, the TBox is satisfiable, otherwise unsat.

Exercise 3 Given the knowledge base (KB) $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is the following TBox:

$$\begin{aligned} A &\sqsubseteq B \sqcup C \\ B &\sqsubseteq \exists r.D \\ C &\sqsubseteq \exists r.E \\ A &\sqsubseteq \forall r.F \\ D \sqcap F &\sqsubseteq G \end{aligned}$$

and $\mathcal{A}$ is the following ABox:

$$A(a)$$

1. using the tableau method, tell whether the concept assertion $\exists r.F(a)$ is entailed by $\mathcal{K}$;
2. using the tableau method, tell whether the concept assertion $\exists r.G(a)$ is entailed by $\mathcal{K}$;
3. using the tableau method, tell whether the concept assertion $\exists r.(D \sqcup E)(a)$ is entailed by $\mathcal{K}$.

Tableau starts with

$A_0 = A \cup \{\neg \exists r.F(a)\} = \{A(a), \forall r.\neg F(a)\}$

$C_{GCI} = (\neg A \sqcup B \sqcup C) \sqcap (\neg B \sqcup \exists r.D) \sqcap (\neg C \sqcup \exists r.E) \sqcap (\neg A \sqcup \forall r.F) \sqcap (\neg D \sqcup \neg F \sqcup G)$

C_{GCI}-rule: $A_1 = A_0 \cup \{C_{GCI}(a)\}$

$\sqcap$-rule: $A_2 = A_1 \cup \{(\neg A \sqcup B \sqcup C)(a), (\neg B \sqcup \exists r.D)(a), (\neg C \sqcup \exists r.E)(a), (\neg A \sqcup \forall r.F)(a), (\neg D \sqcup \neg F \sqcup G)(a)\}$

$\sqcup$-rule: $A_3 = A_2 \cup \{\neg A(a)\} \mid A_4 = A_2 \cup \{B(a)\} \mid A_5 = A_2 \cup \{C(a)\}$

A_3: CLOSED

$\sqcup$-rule: $A_6 = A_5 \cup \{\neg A(a)\} \mid A_7 = A_5 \cup \{\forall r.F(a)\}$

A_6: CLOSED

$\sqcup$-rule $A_8 = A_7 \cup \{\neg C(a)\} \mid A_9 = A_7 \cup \{\exists r.E(a)\}$

A_8: CLOSED

$\exists$-rule: $A_{10} = A_9 \cup \{r(a, x_1), E(x_1)\}$

$\forall$-rule: $A_{11} = A_{10} \cup \{F(x_1)\}$

$\forall$-rule $= A_{12} = A_{11} \cup \{\neg F(x_1)\}$

A_{12}: CLOSED

$\sqcup$-rule: $A_{13} = A_4 \cup \{\neg B(a)\} \mid A_{14} = A_4 \cup \{\exists r.D\}$

A_{13}: CLOSED

$\exists$-rule: $A_{15} = A_{14} \cup \{r(a, x_1), D(x_1)\}$

$\forall$-rule: $A_{16} = A_{15} \cup \{\neg F(x_1)\}$

$\sqcup$-rule: $A_{17} = A_{16} \cup \{\neg A(a)\} \mid A_{18} = A_{16} \cup \{\forall r.F(a)\}$

A_{17}: CLOSED

$\forall$-rule. $A_{19} = A_{18} \cup \{F(x_1)\}$

A_{19}: CLOSED

Tableau returns **false**

$\Downarrow$

$\exists r.F(a)$ is entailed by $(\mathcal{T}, \mathcal{A})$

Datalog and Answer Set Programming

- Predicate Pred, associated with an arity
- Constant Const (use lowercase letters)
- Variable Var (use uppercase letters)
- Term = Const ∪ Var

Atom is an expression of the form $p(t_1, \ldots, t_n)$

Positive Rule is an expression of the form $\alpha :- \beta_1, \ldots, \beta_n$

A fact is a positive rule with an empty body. When representing facts we omit the ":-" symbol

Predicates in a Datalog program are actually partitioned into EDB (Extensional DB) predicates and IDB (Intensional DB) predicates.

A Datalog rule is recursive if the predicate occurring in its head also occurs in its body

A Datalog rule is ground if no variable occurs in it

Given a Datalog program P, the Herbrand Universe of P (HU(P)) is the set of constant symbols occurring in P

Given a ground Datalog program P, the Herbrand Base of P (HB(P)) is the set of ground atoms occurring in P

Given a ground Datalog program P, an interpretation for P is a subset of HB(P)

A ground positive rule r is satisfied in an interpretation I if:

- either some atom in the body of r does not belong to I
- or the atom in the head of r belongs to I

An interpretation I is a model for a ground Datalog Program P if all the rules in P are satisfied in I.

Herbrand Base of a non-ground program P = $HB(P)$ = set of all the ground atoms that can be built with the predicates and the constants occurring in P

Given a non-ground Datalog program P, an interpretation for P is a subset of $HB(P)$.

A variable instantiation for a rule r in a program P is a function $\mu : Vars(r) \to HU(P)$, i.e. a function mapping every variable symbol occurring in r to a constant symbol in $HU(P)$

Given a variable instantiation μ for r in P, we denote as $\mu(r)$ the ground rule obtained from r by replacing every variable X occurring in r with $\mu(X)$. We call $\mu(r)$ an instantiation of rule r in P.

The grounding of a rule r in P, denoted as $ground(r,P)$, is the set of all the possible instantiations $\mu(r)$ of r in P.

The grounding of a Datalog program P, $ground(P)$, is the ground Datalog program obtained by the union of all the sets $ground(r,P)$ s.t. $r \in P$.

An interpretation I of a non-ground Datalog program P is a model for P if I is a model for $ground(P)$.

An interpretation I is a minimal model for a (non-ground) Datalog program P, if I is a model for P and there exists no model I' for P s.t. I' is a strict subset of I.

Every positive Datalog program P has exactly one minimal model.

Exercise

Reasoning in Positive Datalog

Given a Datalog program P and a ground atom α, we say that P entails α if $\alpha \in MM(P)$

Given a ground positive Datalog program P, the <u>immediate consequence operator for P (T_P)</u>, is the function over the domain of interpretations for P defined as:

$T_P(I) = \{ \alpha \mid$ there exists a rule $\alpha :- \beta_1, \ldots, \beta_n$ in P s.t. $\{\beta_1, \ldots, \beta_n\} \subseteq I \}$

The least fixed point of the function T_P is the minimal interpretation I s.t. $T_P(I) = I$

For every ground positive Datalog program P, the immediate consequence operator T_P has a unique least fixed point that coincides with MM(P).

Alg: Naive Eval.

```
let I' = {}
repeat
    let I = I'
    compute I' = T_P(I)
until I' == I
return I
end
```

Example: on the ex. before

$I_0 = \{\}$

$T_P(I_0) = \{q(b),\ p(b,a)\} = I_1$ (they have empty body)

$T_P(I_1) = \{r(b,a),\ p(b,a),\ q(b)\} = I_2$

$T_P(I_2) = \{r(b,a),\ p(b,a),\ q(b)\} = I_2$ least fix point $\Rightarrow I_2$ MM of P

How to optimize Naive-Eval.?

↳ Reformulate a program P in a way s.t. the immediate consequence operator T_P can derive a ground atom only if its derivation depends on at least one ground atom derived in the previous application of T_P in the algorithm. This idea is realized by introducing delta versions of the IDB predicates of the program.

In the algorithm, the extension of a Δ-predicate Δp represents the ground atoms relative to predicate p derived by the previous application of the immediate consequence operator

Δ-transformation of a rule

rule r: $\alpha :- \underbrace{\beta_1, \ldots, \beta_k}_{\text{IDB Atom}}, \underbrace{\gamma_1, \ldots, \gamma_h}_{\text{EDB Atom}}$

Δr is the following set of k rules:

$\Delta'\alpha :- \Delta\beta_1, \beta_2, \ldots, \beta_k, \gamma_1, \ldots, \gamma_h$

$\Delta'\alpha :- \beta_1, \beta_2, \ldots, \Delta\beta_k, \gamma_1, \ldots, \gamma_h$

Therefore, if the body of r contains K atoms with IDB predicates, Δr contains K rules.

Given a positive Datalog program P, the Δ-transformation of $P(\Delta P)$, is the program obtained as the union of all the sets Δr of every rule r in P.

The algorithm then computes the immediate consequence operator $T_{\Delta P}$ of ΔP, treating both Δ-predicates and Δ'-predicates like all other standard predicates.

Alg Semi-Naive Eval.

```
let I = EDB(P)
compute I' = T_P(I)
if I = I' then return I
Δ'I = {Δ'α | α ∈ I' - I}
repeat
    let I = I ∪ {α | Δ'α ∈ Δ'I}
    let ΔI = {Δα | Δ'α ∈ Δ'I}
    let Δ'I = T_ΔP(I ∪ ΔI)
until Δ'I == {}
return I
end
```

Positive Datalog with constraints

A constraint is a new kind of rule of the form:

$$:- B_1, \ldots, B_n$$

A constraint is ground if it does not contain occurrences of variables.

A ground constraint $:- B_1, \ldots, B_n$ is satisfied in an interpretation I if at least one of its atom B_i does not belong to I.

An interpretation I is a model for a Datalog program with constraints P if every ground rule and every ground constraint in ground(P) is satisfied in I

Let P be a Datalog program with constraints. Then:

- Let P' be the program obtained from P eliminating all the constraints;
- Compute MM(P');
- Check whether every constraint in P is satisfied in MM(P'):
  if this is the case, then MM(P') is the minimal model of P; otherwise P has no models

-Exercise-

hasParent(x,y) :- hasMother(x,y). ←

hasParent(x,y) :- hasFather(x,y). ←

hasChild(x,y) :- hasParent(x,y).

hasSibling(x,z) :- hasParent(x,y), hasParent(z,y)

hasGrandParent(x,z) :- hasParent(x,y), hasParent(y,z)

hasMother(Paul, Ann). hasFather(Mary, Joe). hasMother(Bob, Mary).

hasMother(Jane, Mary). hasFather(Joe, Paul).

} P

EDB = {hasFather, hasMother}

IDB = {hasParent, hasChild, hasSibling, hasGrandFather}

$I_0 = EDB(P) = \{$hasMother(Paul, Ann), hasFather(Mary, Joe), hasMother(Bob, Mary), hasMother(Jane, Mary), hasFather(Joe, Paul)$\}$

$I_1 = T_P(I_0) = I_0 \cup \{$hasParent(Paul, Ann), hasParent(Bob, Mary), hasParent(Jane, Mary), hasParent(Mary, Joe), hasParent(Joe, Paul)$\}$

$I_2 = T_P(I_1) = I_1 \cup \{$ hasChild(Ann, Paul), hasChild(Mary, Bob), hasChild(Mary, Jane),
hasChild(Joe, Mary), hasChild(Paul, Joe) hasSibling(Bob, Jane)
hasSibling(Paul, Paul), hasSibling(Bob, Bob), hasSibling(Jane, Jane),
hasSibling(Mary, Mary), hasSibling(Joe, Joe), hasGrandParent(Bob, Joe),
hasGrandParent(Mary, Paul), hasGrandParent(Joe, Ann), hasGrandParent(Jane, Joe)$\}$

we need to modify the rule $x \neq z$ but I cannot insert this at the moment in Datalog

$I_3 = T_P(I_2) = I_2 \longrightarrow MM(P)$ is I_2

Datalog with negation

$\alpha :- \beta_1, \dots, \beta_n, \text{not } \gamma_1, \dots, \text{not } \gamma_m.$

Safeness condition: every variable symbol occurring in the rule must appear in at least one of the positive atoms of the rule body $\beta_1, \dots, \beta_n$

A ground rule $\alpha :- \beta_1, \dots, \beta_n, \text{not } \gamma_1, \dots, \text{not } \gamma_m.$ is satisfied in an interpretation I if at least one of the following condition holds:
- at least one of the atoms $\beta_1, \dots, \beta_n$ does not belongs to I
- at least one of the atoms $\gamma_1, \dots, \gamma_m$ belongs to I
- the atom α belongs to I

An interpretation I is a model for a ground Datalog program with negation P if all the rules of P are satisfied in I.

Differently from the positive case, multiple $MM(P)$ may exist.
Moreover, some minimal models are not "intended" ones.

Answer Set Semantics

Let P be a program with negation and let I be an interpretation.
The reduct of P with respect to I is the positive ground program obtained as follows:

Delete from ground(P) every rule R s.t. an atom not B occurs in the body of R and B belongs to I;
Delete from every rule R in ground(P) every atom not B occurring in the body of R s.t. B does not belong to I.

An interpretation I is an answer set of P if I is the minimal model of the positive program P/I

Properties:

- Every answer set is a minimal model P, but not viceversa.
- A program with negation may have 0, 1 or multiple answer sets.

Basic reasoning tasks:
- decide whether P has at least one answer sets
- compute all the answer sets of P

Derived Reasoning tasks:
- Skeptical entailment: given a program P and a ground atom α, establish whether α belongs to all the answer sets of P.
- Credulous entailment: given a program P and a ground atom α, establish whether α belongs to at least one answer set of P.

Alg: AS = {};
for every interpretation I over HB(P) do
if I == MM(P/I)
then add I to AS;
return AS;

comp. cost is exponential

Exercises

Given the following positive Datalog program P:

r(X,Y) :- s(X,Y).
r(X,Y) :- r(X,Z), s(Z,Y).
t(X) :- r(X,X).
q(Y) :- t(X), r(X,Y).
s(a,b).
s(b,c).
s(c,a).

1) compute the minimal model of P;
2) tell if atom q(a) is entailed by P.

To compute the minimal model, we use the semi-naive evaluation method. We first define the program P' with Δ-relations:

Δ'r(X,Y) :- Δr(X,Z), s(Z,Y). [rule R1]
Δ't(X) :- Δr(X,X). [rule R2]
Δ'q(Y) :- Δt(X), r(X,Y). [rule R3]
Δ'q(Y) :- t(X), Δr(X,Y). [rule R4]
s(a,b). s(b,c). s(c,a).

We then execute the iterative computation of the intensional predicates r, t, q through semi-naive evaluation on P':

Initialization:
I = { s(a,b), s(b,c), s(c,a) },
I' = T_P(I) = I U { r(a,b), r(b,c), r(c,a) } (using 1st rule of P),
Δ'I = { Δ'r(a,b), Δ'r(b,c), Δ'r(c,a) }

1st execution of the repeat-until loop:
I = I U { r(a,b), r(b,c), r(c,a) },
ΔI = { Δr(a,b), Δr(b,c), Δr(c,a) }
Δ'I = $T_{\Delta P}$(I) = { Δ'r(a,c), Δ'r(b,a), Δ'r(c,b) } (using rule R1)

2nd execution of the repeat-until loop:
I = I U { r(a,c), r(b,a), r(c,b) },
ΔI = { Δr(a,c), Δr(b,a), Δr(c,b) }
Δ'I = $T_{\Delta P}$(I) = { Δ'r(a,a), Δ'r(b,b), Δ'r(c,c) } (using rule R1)

3rd execution of the repeat-until loop:
I = I U { r(a,a), r(b,b), r(c,c) },
ΔI = { Δr(a,a), Δr(b,b), Δr(c,c) }
Δ'I = $T_{\Delta P}$(I) = { Δ't(a), Δ't(b), Δ't(c) } (using rule R2)

4th execution of the repeat-until loop:
I = I U { t(a), t(b), t(c) },
ΔI = { Δt(a), Δt(b), Δt(c) }
Δ'I = $T_{\Delta P}$(I) = { Δ'q(a), Δ'q(b), Δ'q(c) } (using rule R3)

5th execution of the repeat-until loop:
I = I U { q(a), q(b), q(c) },
ΔI = { Δq(a), Δq(b), Δq(c) }
Δ'I = $T_{\Delta P}$(I) = { }

The minimal model of P is thus the following:

MM(P) = { s(a,b), s(b,c), s(c,a), r(a,b), r(b,c), r(c,a), r(a,c), r(b,a), r(c,b), r(a,a), r(b,b), r(c,c), t(a), t(b), t(c), q(a), q(b), q(c) }

Since q(a) belongs to the minimal model of P, it is entailed by P

Given the following positive Datalog program with constraints P':

```
r(X,Y) :- s(X,Y).
r(X,Y) :- r(X,Z), s(Z,Y).
t(X) :- r(X,X).
q(Y) :- t(X), r(X,Y).
:- t(X), q(X).
s(a,b). s(b,c). s(c,a).
```

compute the minimal model of P'.

We notice that the program P' is the same as the positive program of Exercise 1, plus the constraint :- t(X), q(X). Namely, P'= P $\cup$ { :- t(X), q(X) }

So, to answer the question we only have to check whether the minimal model of P satisfies such a constraint.

The minimal model M of P (see Exercise 1) is:

MM(P) = { s(a,b), s(b,c), s(c,a), r(a,b), r(b,c), r(c,a), r(a,c), r(b,a), r(c,b), r(a,a), r(b,b), r(c,c), t(a), t(b), t(c), q(a), q(b), q(c) }

M does not satisfy the constraint :- t(X), q(X).
(e.g., both t(a) and q(a) belong to M).

So, we conclude that there exists no (minimal) model for P.

Datalog with stratified negation

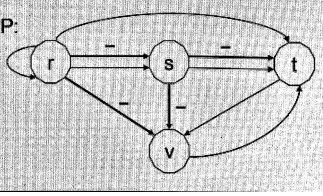
Let P be a Datalog program with negation. The labeled dependency graph of P is a graph $G = (V, E, L)$ where V is the set of vertices, E is the set of edges, and L is a labeling of the edges in E, defined as follows:

- one vertex v in V for every IDB predicate in P;
- there is an edge (s,t) in E (without label) if s,t are IDB predicates and P contains a rule R s.t. t appears in the head of R and s appears in a positive atom in the body of R;
- there is a negated edge (-) (s,t) in E if s,t are predicates and P contains a rule R s.t. t appears in the head of R and s appears in a negative atom in the body of R.

Example: let P be the following Datalog program with negation:

$r(X,Y) :- p(X,Y),\ not\ p(Y,X).$
$r(X,Y) :- p(X,Y),\ r(Y,Z).$
$s(X,Y) :- r(Z,X),\ r(Z,Y),\ not\ r(X,Y).$
$t(X) :- r(Y,X),\ s(X,X),\ not\ s(Y,Y).$
$t(X) :- v(X,Y).$
$v(X,Y) :- t(X),\ t(Y),\ not\ r(X,Y),\ not\ s(X,Y).$
$p(a,b).\ p(b,c).$

Labeled dependency graph G of P:



We say that P is stratified if no cycle of the labeled dependency graph of P contains a negated edge.

Property: every stratified program P has a unique answer set, which coincides with the unique minimal model of P, and is denoted by MM(P).

A vertex v of G has a negative dependency if there exists a vertex v' in G s.t. there is a path from v' to v passing through a negated edge.

The stratification of a stratified program P is a sequence $S_1, \dots, S_k$ of sets of IDB predicates of P (called strata) obtained as follows:

```
G = labeled dependency graph of P;
i = 0;
while G is not empty do begin
   i = i+1;
   S_i = the set of vertices of G that do not have negative dependencies;
   G = the labeled dependency graph of P obtained considering (in addition to the
      initial EDB predicates) the predicates in S_1, ..., S_i as EDB
end;
return S_1, ..., S_i;
```

Alg. for computing MM(P) in this case →

```
Let S_1, ..., S_k be the stratification of P;
MM_0 = EDB(P);
For i = 1 to k do begin
   P(S_i) = the program obtained from P by considering only the rules
      having a predicate from S_i in their head;
   MM_i = minimal model of P(S_i) ∪ MM_{i-1}
end;
return MM_i;
```

Adding disjunction to Datalog

A disjunctive rule is an expression of the form:

$$\alpha_1 \vee \dots \vee \alpha_k :- \beta_1, \dots, \beta_n, \text{not } \gamma_1, \dots, \text{not } \gamma_m.$$

Safeness condition: every variable symbol occurring in the rule must appear in at least one of the positive atoms of the rule body $\beta_1, \dots, \beta_n$

The rule is called positive if $m=0$, is called non-disjunctive if $k=1$, and is called constraint if $k=0$

A ground disjunctive rule $\alpha_1 \vee \dots \vee \alpha_k :- \beta_1, \dots, \beta_n, \text{not } \gamma_1, \dots, \text{not } \gamma_m.$ is satisfied in an interpretation I if at least one of the following conditions holds:

- at least one of the atoms $\beta_1, \dots, \beta_n$ does not belong to I
- at least one of the atoms $\gamma_1, \dots, \gamma_m$ belongs to I
- at least one of the atoms $\alpha_1, \dots, \alpha_k$ belongs to I

Let P be a disjunctive program and let I be an interpretation. The reduct of P with respect to I is the positive disjunctive ground program obtained as follows:
Delete from ground(P) every rule R s.t. an atom not B occurs in the body of R and B belongs to I;
Delete from every rule R in ground(P) every atom not B occurring in the body of R s.t. B does not belong to I.
An interpretation I is an answer set of P if I is a minimal model of the positive disjunctive program P/I.

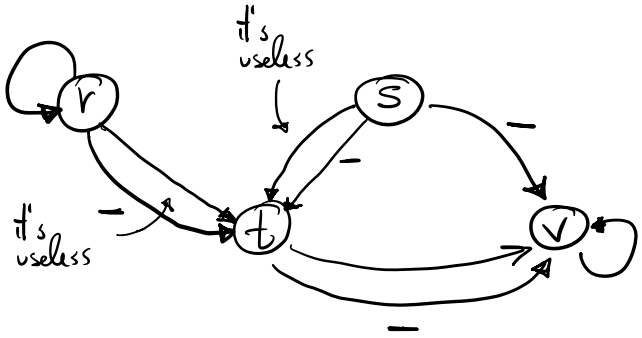
Exercise:

Given the following ASP program P:

```
r(X,Y) :- p(X,Y).
r(X,Y) :- p(X,Z), r(Z,Y).
s(X,Y) :- q(X,Y).
t(X,Y) :- r(X,Y), not s(X,Y).
t(X,Y) :- s(X,Y), not r(X,Y).
v(X,Y) :- t(X,Y), not s(Y,X).
v(X,Y) :- v(X,Z), v(Z,Y), not t(Z,X).
p(a,b).  p(b,c).  q(a,b).  q(c,a).
```

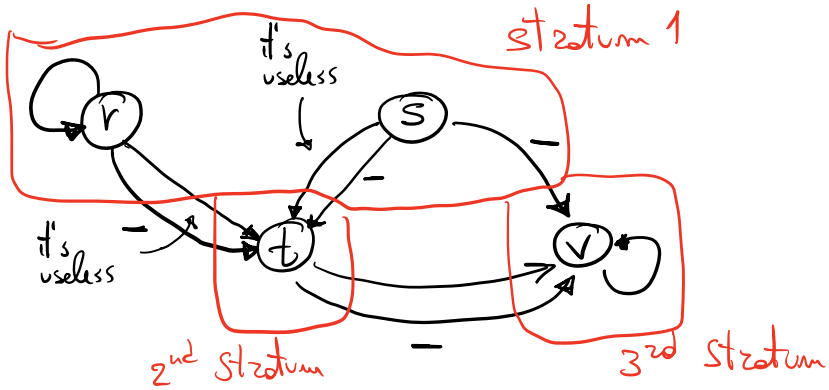
1) tell whether P is stratified;
2) compute the answer sets of P.

A- draw labeled dependency graph



no cycles with negative edges in the graph → P is stratified

B-



Given the following ASP program P:

P1
```
r(X,Y) :- p(X,Y).
r(X,Y) :- p(X,Z), r(Z,Y).
s(X,Y) :- q(X,Y).
```
P2
```
t(X,Y) :- r(X,Y), not s(X,Y).
t(X,Y) :- s(X,Y), not r(X,Y).
```
P3
```
v(X,Y) :- t(X,Y), not s(Y,X).
v(X,Y) :- v(X,Z), v(Z,Y), not t(Z,X).
p(a,b).  p(b,c).  q(a,b).  q(c,a).
```

1) tell whether P is stratified;
2) compute the answer sets of P.

$MM_0 = \{ p(a,b), p(b,c), q(a,b), q(c,a) \}$

we compute the MM of P_1, starting from the initial interpretation MM_0

S 1 : $T_{P_1}(MM_0) = \{ \underbrace{r(a,b), r(b,c)}_{R1}, \underbrace{s(a,b), s(c,a)}_{R3} \} = I_1$

$T_{P_1}(I_1 \cup MM_0) = I_1 \cup \{ \underbrace{r(a,c)}_{R2} \} = I_2$

$T_{P_1}(I_2 \cup MM_0) = I_2$ fix point → $MM_1 = MM_0 \cup I_2$

S2 : $T_{P_2}(MM_1) = \{t(b,c), t(a,c), t(c,a)\} = I_3$

R4 : $t(b,c), t(a,c)$ — R5 : $t(c,a)$

$T_{P_2}(I_3 \cup MM_1) = I_3$ fixpoint $\rightarrow MM_2 = MM_1 \cup I_3$

S3 : $T_{P_3}(MM_2) = \{v(b,c), v(c,a)\} = I_4$

R6 : $v(b,c), v(c,a)$

$T_{P_3}(MM_2 \cup I_4) = I_4 \cup \{v(b,a)\} = I_5$

$T_{P_3}(MM_2 \cup I_5) = I_5$ fixpoint $\rightarrow MM_3 = MM_2 \cup I_5$

MM_3 is the answer set (or minimal model) of P

Comparison on SQL, DLs, Datalog and ASP

CWA vs OWA

- CWA : Databases are based on a closed-world assumption, only one world is possible
- OWA : Knowledge expressed by the theory is not complete, many possible world

Datalog and ASP inherit CWA from databases, but ASP is able to deal with incomplete knowledge (disjunction)

Complete vs Incomplete Knowledge

- DL Knowledge bases are incomplete specifications of the domain of interest
KB has multiple models (possible worlds in which the KB is satisfied)

- In relational DBs, the database instance is a complete specification:
  - scheme constraints in databases are integrity constraints (they MUST be satisfied by the DB instances)

Recursion

- DLs are able to deal with recursive statements, do not allow for recursive queries
- SQL does not allow for recursive queries
- Datalog allows for expressing recursive queries over DB
- ASP allows for expressing recursive queries over DB

Negation

- DLs do not allow for recursive negation in queries
- SQL allows for negation in queries, but does not allow for recursive negation
- Datalog does not allow for expressing negation
- ASP allows for expressing negation, and even recursive negation

Skolemization

DLs allow for expressing so-called value invention ($Person \subseteq \exists hasMother . T$)
The above axiom cannot be represented in Datalog
Value invention is a form of incomplete Knowledge

To overcome the above limitation, an extension of Datalog called existential rules, has been proposed. ER do not have the range restriction on variables. In general reasoning with existential rules is undecidable

Introduction to the Semantic Web

Semantic Web is a Web of actionable information – information derived from data through a semantic theory for interpreting the symbols.

Problem 1: web documents do not distinguish between information content and presentation.

Problem 2: different web documents may represent in different ways semantically related pieces of information.

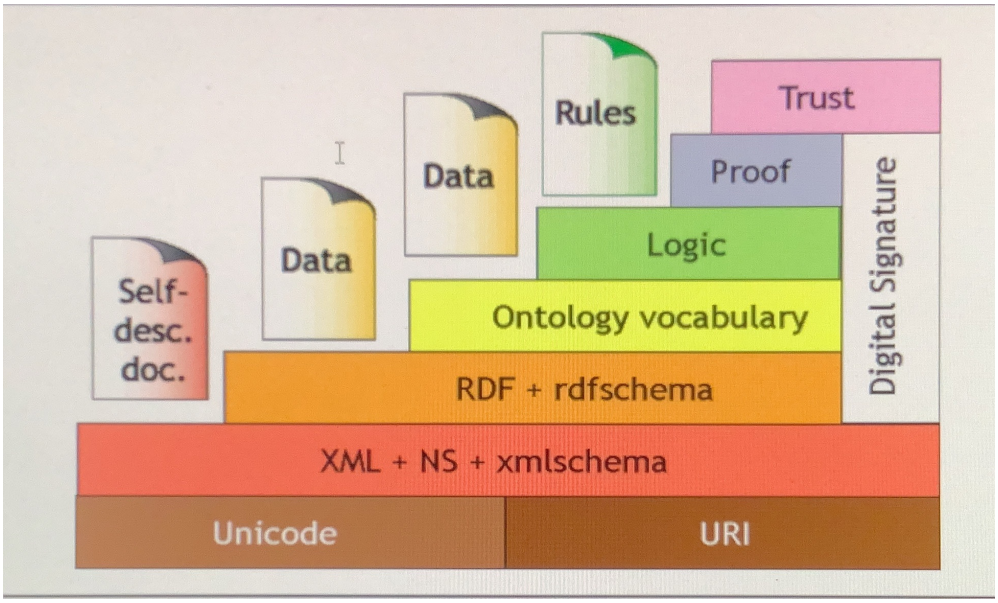
XML does not solve all the problems:

- legacy HTML documents
- different XML documents may express information with the same meaning using different tags

→ need of shared semantics

Goal: to provide a common framework to share data on the Web across application boundaries.

Semantic Web Tower



XML Layer

- XML
- URI: - universal naming for web resources
  - Same URI = same resource
  - URIs are the "ground terms" of the SW
- W3C standards

RDF + RDFS layer

*RDF model = set of RDF <u>triples</u> (expression (statement))

•an RDF model is a <u>graph</u>

(subject, predicate, object)

subject → resource
predicate → property
object → value

Ontology layer

ontology = shared conceptualization of a domain of interest

Conceptual model, expressed in a true Knowledge representation language.

OWL (Web Ontology Language) = standard language for ontologies

Proof / Rule Layer

Rules (informations notion) are used to perform inference over ontologies, can be seen as a tool for capturing further knowledge

Trust Layer

Support for provenance / trust: - where does the information come from?
- how this information has been obtained?
- can I trust this information?

- no standardization effort

Ontologies

logical theory → declarative
→ formal semantics
→ reasoning

well established correspondence between conceptual modeling formalisms and logic

OWL is based on a fragment of FOL
Description logics (DLs) are a subclasses of FOL
- only unary and binary predicates
- function-free
- quantification allowed only in restricted form
- variable-free syntax
- decidable reasoning

RDF

RDF is a data model (domain-neutral, application-neutral, ready for internationalization)
The model can be viewed as directed, labeled graph or as an object-oriented model

node and edge labels :- URI
- literal
- blank node

but: - a literal can only appear in object positions
- a blank node can only appear in subject or object positions

Blank nodes

bnode = RDF graph node with "anonymous label"

ex: Jean has a friend born the 21st of April

ex : Jean foaf : knows _:p1 → blank nodes
_ :p1 foaf : birthDate 04-21

Containers

Containers are collections, they allow grouping of resources.
Different types of containers exist:
- bag : unordered collection
- seq : ordered collection
- alt : represents alternatives

Duplicate values are permitted

Higher-order statements

They allow us to express beliefs; are represented by modeling RDF in RDF itself.

⇓

Reification

Reification in RDF is using an RDF statement as the subject (or object) of another RDF statement.

RDF provides a built-in predicate vocabulary for reification:

- rdf : subject
- rdf : predicate
- rdf : object
- rdf : statement

Using this vocabulary it is possible to represents a triple through a blank nodes

RDF syntaxes

- N3 notation
- Turtle notation
- concrete (serialized) syntax

RDFa : RDF in XHTML document

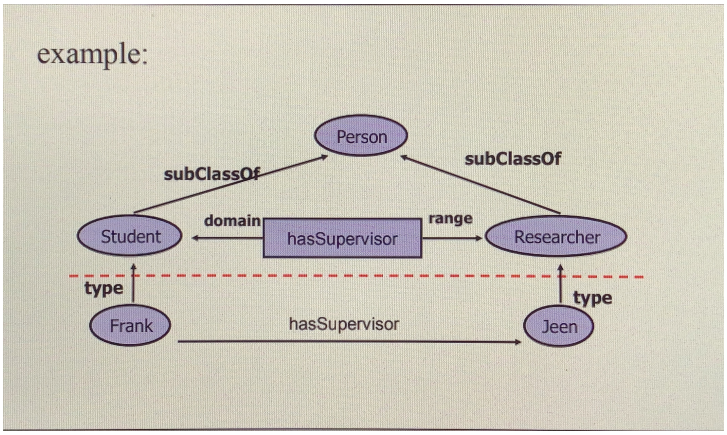
RDFa specifies a set of attributes that can be used in XHTML elements. Such attributes allow for expressing RDF triples inside the document

RDFS : RDF Schema

Defines small vocabulary for RDF

- Class, subClassOf, type
- Property, subPropertyOf
- domain, range

example:



bnodes = existential values (null values) → introduce incomplete information in RDF graphs. RDF graph can be seen as an incomplete DB represented in the form of a naive table or as a boolean conjunctive query over the unique relation T

RDFS statements = constraints over the RDF graph

entailment in RDF + RDFS = reasoning over an incomplete database with constraints

Querying RDF : SPARQL

example:

```
PREFIX
   abc: <http://mynamespace.com/exampleOntology#>
SELECT ?capital ?country
WHERE { ?x abc:cityname ?capital.
        ?y abc:countryname ?country.
        ?x abc:isCapitalOf ?y.
        ?y abc:isInContinent abc:africa. }
```

variables are outlined through the "?" prefix

?capital and ?country will be returned

basic graph pattern (BGP) query = RDF graph with possibly variables as labels

SPARQL defines on top of a BGP additional operators (AND, FILTER, UNION, OPTIONAL)

Current main application of RDF: Linked Data

Linked Data: set of best practices for publishing and connecting structured data on the Web using URIs and RDF

Principles:

- Use URIs as names for things
- Use HTTP URIs, so that people can look up those names
- When someone looks up a URI, provide useful information, using the standards
- Include links to other URIs, so that they can discover more things.

Linked Data uses hyperlinks to connect disparate data into a single global data space.

Popular Vocabularies

- **Friend-of-a-Friend (FOAF)**, vocabulary for describing people
- **Dublin Core (DC)** defines general metadata attributes
- **Semantically-Interlinked Online Communities (SIOC)**, vocabulary for representing online communities
- **Description of a Project (DOAP)**, vocabulary for describing projects
- **Simple Knowledge Organization System (SKOS)**, vocabulary for representing taxonomies and loosely structured knowledge
- **Music Ontology** provides terms for describing artists, albums and tracks
- **Review Vocabulary**, vocabulary for representing reviews
- **Creative Commons (CC)**, vocabulary for describing license terms

Exercise 1

Write an RDF model representing the following statements:

- Document 1 has been created by Paul
- Document 2 and document 3 have been created by the same author (who is unknown)
- Document 3 says that document 1 has been published by W3C

Use predicates `dc:Creator` and `dc:Publisher`, and assume that the three documents are represented by URIs `doc1`, `doc2`, `doc3`, respectively.

Exercise 1: Solution

```
@prefix rdf:
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix myns: <http://example.org/myVocabulary/> .

myns:doc1 dc:Creator "Paul".
myns:doc2 dc:Creator _:x.
myns:doc3 dc:Creator _:x.
myns:doc3 myns:says _:y.
_:y rdf:subject myns:doc1.
_:y rdf:predicate dc:Publisher.
_:y rdf:object "W3C".
_:y rdf:type rdf:statement.
```

Exercise 2

Write an RDF/RDFS model representing the following statements:

- URI1 and URI2 are classes
- URI3 and URI7 are properties
- URI1 is a subclass of URI2
- URI3 is a subproperty of URI7
- URI3 has domain URI1 and range URI2
- URI4 is an instance of class URI1
- URI5 and URI6 are instances of class URI2
- (URI6,URI4) is an instance of property URI3

Exercise 2: Solution

```
@prefix rdf:
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:
  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix myns: <http://example.org/myVocabulary/> .

myns:URI1 rdf:type rdfs:Class .
myns:URI2 rdf:type rdfs:Class .
myns:URI3 rdf:type rdf:Property .
myns:URI7 rdf:type rdf:Property .
myns:URI1 rdfs:subClassOf myns:URI2 .
myns:URI3 rdfs:subPropertyOf myns:URI7 .
myns:URI3 rdfs:domain myns:URI1 .
myns:URI3 rdfs:range myns:URI2 .
```

Exercise 2: Solution (cont'd)

```
myns:URI4 rdf:type myns:URI1 .
myns:URI5 rdf:type myns:URI2 .
myns:URI6 rdf:type myns:URI2 .
myns:URI6 myns:URI3 myns:URI4 .
```

Exercise 3

Write a SPARQL query corresponding to the following requests:

1. “return all URIs having an author and a date of creation”
2. “return all predicates having as subject both URI1 and URI2”
3. “return all predicates having as subject either URI1 or URI2”
4. “return the name of the authors of any document having a date of creation”

Exercise 3: Solution

1)
```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?x
WHERE { ?x dc:creator ?y .
        ?x dc:date ?z . }
```

2)
```
PREFIX myns: <http://example.org/myVocabulary>
SELECT ?x
WHERE { myns:uri1 ?x ?y .
        myns:uri2 ?x ?z . }
```

Exercise 3: Solution (cont.)

```
3) PREFIX myns: <http://example.org/myVocabulary>
   SELECT ?x
   WHERE { { myns:uri1 ?x ?y . } UNION
           { myns:uri2 ?x ?z . } }

4) PREFIX dc: <http://purl.org/dc/elements/1.1/>
   SELECT ?z
   WHERE { ?x dc:creator ?y .
           ?y dc:name ?z .
           ?x dc:date ?w . }
```

Exercise 4

Write a SPARQL query corresponding to the following request: “return the name of authors and date of creation of any document having an author and, optionally, a date of creation”.

Exercise 4: Solution

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?z ?w
WHERE { ?x dc:creator ?y .
        ?y dc:name ?z .
        OPTIONAL { ?x dc:date ?w . } }
```

Exercise 5

Write a SPARQL query corresponding to the following request: “return all URIs corresponding to documents created by Paul”. Assume that the triples expressing creators of documents are reified.

Exercise 5: Solution

```
PREFIX rdf:
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?x
WHERE { ?b rdf:subject ?x .
        ?b rdf:predicate dc:creator .
        ?b rdf:object "Paul" .
        ?b rdf:type rdf:statement . }
```

Ontologies and Owl

ontology = shared conceptualization of a domain of interest

Terms = names for important concept in the domain

Relationships between terms = background knowledge / constraints on the domain

XMLS is not an ontology language

RDFS is recognizable as an ontology language, but it is too weak to describe resources in sufficient detail.

OWL

OWL family is constituted by 3 different languages:

- OWL Full
- OWL - DL
- OWL - Lite

OWL class constructors

Constructor	DL Syntax	Example	Modal Syntax
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human ⊓ Male	$C_1 \wedge \ldots \wedge C_n$
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor ⊔ Lawyer	$C_1 \vee \ldots \vee C_n$
complementOf	$\neg C$	¬Male	$\neg C$
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{john} ⊔ {mary}	$x_1 \vee \ldots \vee x_n$
allValuesFrom	$\forall P.C$	∀hasChild.Doctor	$[P]C$
someValuesFrom	$\exists P.C$	∃hasChild.Lawyer	$\langle P \rangle C$
maxCardinality	$\leqslant nP$	⩽1hasChild	$[P]_{n+1}$
minCardinality	$\geqslant nP$	⩾2hasChild	$\langle P \rangle_n$

- XMLS datatypes as well as classes in ∀P.C and ∃P.C
  - E.g., ∃hasAge.nonNegativeInteger
- Arbitrarily complex nesting of constructors
  - E.g., Person ⊓ ∀hasChild.Doctor ⊔∃hasChild.Doctor

OWL axioms

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human ⊑ Animal ⊓ Biped
equivalentClass	$C_1 \equiv C_2$	Man ≡ Human ⊓ Male
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male ⊑ ¬Female
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{President_Bush} ≡ {G_W_Bush}
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{john} ⊑ ¬{peter}
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter ⊑ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost ≡ price
inverseOf	$P_1 \equiv P_2^-$	hasChild ≡ hasParent⁻
transitiveProperty	$P^+ \sqsubseteq P$	ancestor⁺ ⊑ ancestor
functionalProperty	$\top \sqsubseteq \leqslant 1P$	⊤ ⊑ ⩽1hasMother
inverseFunctionalProperty	$\top \sqsubseteq \leqslant 1P^-$	⊤ ⊑ ⩽1hasSSN⁻

Axioms (mostly) reducible to inclusion (⊑)

$C \equiv D$ iff both $C \sqsubseteq D$ and $D \sqsubseteq C$

Abbreviations:

Letters	Meaning	Letters	Meaning	Letters	Meaning	Letters	Meaning
C	class expression	CN	class name	D	data range	DN	datatype name
P	object property expression	PN	object property name	R	data property	A	annotation property
a	individual	aN	individual name	_:a	anonymous individual (a blank node label)	v	literal
n	non-negative integer	f	facet	ON	ontology name	U	IRI
s	IRI or anonymous individual	t	IRI, anonymous individual, or literal	p	prefix name	_:x	blank node
$(a_1 \ldots a_n)$	RDF list						

Declarations:

Language Feature	Functional Syntax	RDF Syntax
class	Declaration(Class(CN))	CN rdf:type owl:Class.
datatype	Declaration(Datatype(DN))	DN rdf:type rdfs:Datatype.
object property	Declaration(ObjectProperty(PN))	PN rdf:type owl:ObjectProperty.
data property	Declaration(DataProperty(R))	R rdf:type owl:DatatypeProperty.
annotation property	Declaration(AnnotationProperty(A))	A rdf:type owl:AnnotationProperty.
named individual	Declaration(NamedIndividual(aN))	aN rdf:type owl:NamedIndividual.

Boolean operators and enumeration:

Language Feature	Functional Syntax	RDF Syntax
intersection	ObjectIntersectionOf(C_1 … C_n)	_:x rdf:type owl:Class. _:x owl:intersectionOf (C_1 … C_n).
union	ObjectUnionOf(C_1 … C_n)	_:x rdf:type owl:Class. _:x owl:unionOf (C_1 … C_n).
complement	ObjectComplementOf(C)	_:x rdf:type owl:Class. _:x owl:complementOf C.
enumeration	ObjectOneOf(a_1 … a_n)	_:x rdf:type owl:Class. _:x owl:oneOf (a_1 … a_n).

Object Property restrictions:

Language Feature	Functional Syntax	RDF Syntax
universal	ObjectAllValuesFrom(P C)	_:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:allValuesFrom C
existential	ObjectSomeValuesFrom(P C)	_:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:someValuesFrom C
individual value	ObjectHasValue(P a)	_:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:hasValue a.
local reflexivity	ObjectHasSelf(P)	_:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:hasSelf "true"^^xsd:boolean.
exact cardinality	ObjectExactCardinality(n P)	_:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:cardinality n.
qualified exact cardinality	ObjectExactCardinality(n P C)	_:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:qualifiedCardinality n. _:x owl:onClass C.

Language Feature	Functional Syntax	RDF Syntax
maximum cardinality	ObjectMaxCardinality(n P)	_:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:maxCardinality n.
qualified maximum cardinality	ObjectMaxCardinality(n P C)	_:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:maxQualifiedCardinality n. _:x owl:onClass C.
minimum cardinality	ObjectMinCardinality(n P)	_:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:minCardinality n.
qualified minimum cardinality	ObjectMinCardinality(n P C)	_:x rdf:type owl:Restriction. _:x owl:onProperty P. _:x owl:minQualifiedCardinality n. _:x owl:onClass C.

Data Property restrictions:

Language Feature	Functional Syntax	RDF Syntax
universal	DataAllValuesFrom(R D)	_:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:allValuesFrom D.
existential	DataSomeValuesFrom(R D)	_:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:someValuesFrom D.
literal value	DataHasValue(R v)	_:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:hasValue v.
exact cardinality	DataExactCardinality(n R)	_:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:cardinality n.
qualified exact cardinality	DataExactCardinality(n R D)	_:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:qualifiedCardinality n. _:x owl:onClass D.

Language Feature	Functional Syntax	RDF Syntax
maximum cardinality	DataMaxCardinality(n R)	_:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:maxCardinality n.
qualified maximum cardinality	DataMaxCardinality(n R D)	_:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:maxQualifiedCardinality n. _:x owl:onClass D.
minimum cardinality	DataMinCardinality(n R)	_:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:minCardinality n.
qualified minimum cardinality	DataMinCardinality(n R D)	_:x rdf:type owl:Restriction. _:x owl:onProperty R. _:x owl:minQualifiedCardinality n. _:x owl:onClass D.

Object and data property expressions:

Language Feature	Functional Syntax	RDF Syntax
named object property	PN	PN
universal object property	owl:topObjectProperty	owl:topObjectProperty
empty object property	owl:bottomObjectProperty	owl:bottomObjectProperty
inverse property	ObjectInverseOf(PN)	_:x owl:inverseOf PN

Language Feature	Functional Syntax	RDF Syntax
named data property	R	R
universal data property	owl:topDataProperty	owl:topDataProperty
empty data property	owl:bottomDataProperty	owl:bottomDataProperty

Class Axioms:

Language Feature	Functional Syntax	RDF Syntax
subclass	SubClassOf(C_1 C_2)	C_1 rdfs:subClassOf C_2.
equivalent classes	EquivalentClasses(C_1 … C_n)	C_j owl:equivalentClass C_{j+1}. j=1…n-1
disjoint classes	DisjointClasses(C_1 C_2)	C_1 owl:disjointWith C_2.
pairwise disjoint classes	DisjointClasses(C_1 … C_n)	_:x rdf:type owl:AllDisjointClasses. _:x owl:members (C_1 … C_n).
disjoint union	DisjointUnionOf(CN C_1 … C_n)	CN owl:disjointUnionOf (C_1 … C_n).

Object Property Axioms:

Language Feature	Functional Syntax	RDF Syntax
subproperty	SubObjectPropertyOf(P_1 P_2)	P_1 rdfs:subPropertyOf P_2.
property chain inclusion	SubObjectPropertyOf(ObjectPropertyChain(P_1 … P_n) P)	P owl:propertyChainAxiom (P_1 … P_n).
property domain	ObjectPropertyDomain(P C)	P rdfs:domain C.
property range	ObjectPropertyRange(P C)	P rdfs:range C.
equivalent properties	EquivalentObjectProperties(P_1 … P_n)	P_j owl:equivalentProperty P_{j+1}. j=1…n-1
disjoint properties	DisjointObjectProperties(P_1 P_2)	P_1 owl:propertyDisjointWith P_2.
pairwise disjoint properties	DisjointObjectProperties(P_1 … P_n)	_:x rdf:type owl:AllDisjointProperties. _:x owl:members (P_1 … P_n).
inverse properties	InverseObjectProperties(P_1 P_2)	P_1 owl:inverseOf P_2.
functional property	FunctionalObjectProperty(P)	P rdf:type owl:FunctionalProperty.
inverse functional property	InverseFunctionalObjectProperty(P)	P rdf:type owl:InverseFunctionalProperty.

Language Feature	Functional Syntax	RDF Syntax
reflexive property	ReflexiveObjectProperty(P)	P rdf:type owl:ReflexiveProperty.
irreflexive property	IrreflexiveObjectProperty(P)	P rdf:type owl:IrreflexiveProperty.
symmetric property	SymmetricObjectProperty(P)	P rdf:type owl:SymmetricProperty.
asymmetric property	AsymmetricObjectProperty(P)	P rdf:type owl:AsymmetricProperty.
transitive property	TransitiveObjectProperty(P)	P rdf:type owl:TransitiveProperty.

Data Property Axioms:

Language Feature	Functional Syntax	RDF Syntax
subproperty	SubDataPropertyOf(R_1 R_2)	R_1 rdfs:subPropertyOf R_2.
property domain	DataPropertyDomain(R C)	R rdfs:domain C.
property range	DataPropertyRange(R D)	R rdfs:range D.
equivalent properties	EquivalentDataProperties(R_1 … R_n)	R_j owl:equivalentProperty R_{j+1}. j=1…n-1
disjoint properties	DisjointDataProperties(R_1 R_2)	R_1 owl:propertyDisjointWith R_2.
pairwise disjoint properties	DisjointDataProperties(R_1 … R_n)	_:x rdf:type owl:AllDisjointProperties. _:x owl:members (R_1 … R_n).
functional property	FunctionalDataProperty(R)	R rdf:type owl:FunctionalProperty.

Assertions (ABox statements):

Language Feature	Functional Syntax	RDF Syntax
individual equality	SameIndividual(a_1 … a_n)	a_j owl:sameAs a_{j+1}. j=1…n-1
individual inequality	DifferentIndividuals(a_1 a_2)	a_1 owl:differentFrom a_2.
pairwise individual inequality	DifferentIndividuals(a_1 … a_n)	_:x rdf:type owl:AllDifferent. _:x owl:members (a_1 … a_n).
class assertion	ClassAssertion(C a)	a rdf:type C.
positive object property assertion	ObjectPropertyAssertion(PN a_1 a_2)	a_1 PN a_2.
positive data property assertion	DataPropertyAssertion(R a v)	a R v.
negative object property assertion	NegativeObjectPropertyAssertion(P a_1 a_2)	_:x rdf:type owl:NegativePropertyAssertion. _:x owl:sourceIndividual a_1. _:x owl:assertionProperty P. _:x owl:targetIndividual a_2.
negative data property assertion	NegativeDataPropertyAssertion(R a v)	_:x rdf:type owl:NegativePropertyAssertion. _:x owl:sourceIndividual a. _:x owl:assertionProperty R. _:x owl:targetValue v.

OWL DL semantics

Mapping OWL to equivalent DL ($SHOIN(D_n)$)

DL semantics defined by interpretations: $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where

- $\Delta^{\mathcal{I}}$ is the domain
- $\cdot^{\mathcal{I}}$ is an interpretation function that maps:
  - Concept name A → subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$
  - Role name R → binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$
  - Individual name i → $i^{\mathcal{I}}$ element of $\Delta^{\mathcal{I}}$

OWL 2

OWL 2 Profiles:

- OWL 2 QL based on the DL DL-Lite
- OWL 2 RL based on the DL RL
- OWL 2 EL based on the DL EL

Each of the 3 profile has tractable complexity of reasoning. In particular for query answering:

- OWL 2 QL/DL-Lite: algorithm based on query rewriting
- OWL 2 RL: algorithm based on ABox materialization
- OWL 2 EL: algorithm based on a combination of query rewriting and ABox materialization

DL-Lite

DL-Lite is a family of Description logics

$DL\text{-}Lite_{core}$ = basic DL-Lite language

- $\mathcal{F}$: core + role functionality
- $\mathcal{R}$: core + role hierarchies
- $\mathcal{A}$: $\mathcal{F}+\mathcal{R}$ + attributes + domains

$DL\text{-}Lite_{\mathcal{F}}$ syntax

concept expressions:

- atomic concept A
- role domain $\exists R$
- role range $\exists R^-$

role expressions:

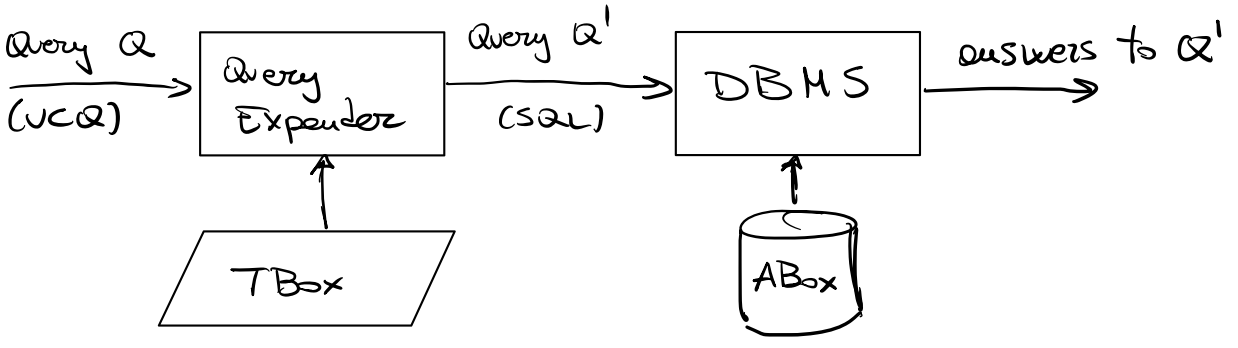
- atomic role R
- inverse atomic role R^-

TBox is a set of:
- concept inclusions
- concept disjointness assertions
- functional assertions

ABox is a set of ground atoms
- $A(a)$ with A concept name
- $R(a,b)$ with R role name

All TBox reasoning tasks in DL-Lite are solvable in polytime.
Instance checking and instance retrieval in DL-Lite are solvable in polytime.

Query answering in DL-Lite can be reduced to evaluation of an SQL query over a relational DB



Chase of the ABox with respect to the TBox is adding to the ABox all instance assertions that are logical consequences of the TBox

The rewriting algorithm iteratively applies 2 rewriting rules:
- atom rewrite: it takes an atom of the Conjunctive Query and rewrites it applying a TBox inclusion (rewriting rule, right-to-left)
- reduce: it takes 2 unifiable atoms of the Conjunctive Query and merges them

```
Algorithm PerfectRef (q, T)
Input: conjunctive query q, DL-Lite TBox T
Output: union of conjunctive queries PR
PR := {q};
repeat
    PR0 := PR;
    for each q ∈PR0 do
    (a) for each g in q do
            for each positive inclusion I  in T do
              if I is applicable to g
              then PR := PR ∪{atom-rewrite(q,g,I)};
    (b) for each g1, g2 in q  do
          if g1 and g2 unify then PR := PR ∪{reduce(q,g1,g2)}
until PR0 = PR;
return PR
```

QuOnto is a reasoner for DL-Lite, it is able to deal with very large instances

RL

syntax:

concept expressions:

- atomic concept A
- concept conjunction $C_1 \sqcap C_2$
- qualified existential $\exists R.C$
- qualified existential $\exists R.\bot$

role expressions:

- atomic role R
- inverse role R^-

TBox is a set of:
- concept inclusions of the form $C \sqsubseteq A$ or $C \sqsubseteq \bot$
- role inclusions $R_1 \sqsubseteq R_2$

ABox is a set of ground atoms
- $A(a)$ with A concept name
- $R(a,b)$ with R role name

Reasoning in RL

ABox reasoning and query answering in RL can be done through forward chaining (materialization), which corresponds to the chase procedure:

- chase of the ABox wrt the TBox
- In the case of RL no new individual is introduced by the chase
- After this materialization step, the TBox can be discarded and conjunctive queries can be answered by evaluating them on the materialized ABox

EL

syntax:

concept expressions:

- atomic concept A
- concept conjunction $C_1 \sqcap C_2$
- qualified existential $\exists R.C$

role expressions:

- atomic role R

TBOX is a set of: - concept inclusions

ABOX is a set of ground atoms
- $A(a)$ with A concept name
- $R(a,b)$ with R role name

Reasoning in EL

The chase for an EL ontology may be infinite, so ABox materialization is not applicable to EL. Also query rewriting approach does not work too.

↳ Use an hybrid approach: ABox partially expanded wrt the TBox
query partially rewritten wrt the TBox

Knowledge Representation and Semantic Technologies

Exercises on OWL

Riccardo Rosati

Corso di Laurea Magistrale in Ingegneria Informatica
Sapienza Università di Roma
2019/2020

Exercise 1

Write an OWL ontology representing the following statements:

- URI1 and URI2 are classes
- URI3 is a property
- URI4 is an instance of class URI1, and URI5 and URI6 are instances of class URI2
- URI3 has domain URI1 and range URI2
- (URI6,URI4) is an instance of property URI3

Exercise 1: Solution

```
Declaration(Class(myns:URI1))
Declaration(Class(myns:URI2))

Declaration(ObjectProperty(myns:URI3))

ClassAssertion(myns:URI1 myns:URI4)
ClassAssertion(myns:URI2 myns:URI5)
ClassAssertion(myns:URI2 myns:URI6)

SubClassOf(
  ObjectSomeValuesFrom(myns:URI3 owl:Thing)
  myns:URI1)
```

Exercise 1: Solution (continued)

```
SubClassOf(
  ObjectSomeValuesFrom(
    ObjectInverseOf(myns:URI3)
    owl:Thing)
  myns:URI2)

ObjectPropertyAssertion(myns:URI3 myns:URI6
  myns:URI4)
```

Exercise 2

Write an OWL ontology that formalizes knowledge about the domain of people, in particular the classes `person`, `man`, `woman`, and the properties `hasParent`, `hasMother`, `hasFather`.

Try to express all the knowledge you have about such classes and properties (e.g.: every man is a person, every woman is a person, every mother is a woman, etc.).

Exercise 2: Solution

```
SubClassOf(myns:man myns:person)
```
(every man is a person)

```
SubClassOf(myns:woman myns:person)
```
(every woman is a person)

```
SubObjectPropertyOf(myns:hasMother myns:hasParent)
```
(hasMother is a subproperty of hasParent)

```
SubObjectPropertyOf(myns:hasFather myns:hasParent)
```
(hasFather is a subproperty of hasParent)

```
SubClassOf(
  ObjectSomeValuesFrom(
    ObjectInverseOf(myns:hasMother)
    owl:Thing)
  myns:woman)
```
(every mother is a woman)

Exercise 2: Solution (continued)

```
SubClassOf(
  ObjectSomeValuesFrom(
    ObjectInverseOf(myns:hasFather)
    owl:Thing)
  myns:man)
```
(every father is a man)

```
ClassAssertion(myns:man myns:Joe)
```
(Joe is a man)

```
ObjectPropertyAssertion(myns:hasMother myns:Joe
  myns:Ann)
```
(Ann is the mother of Joe)

Exercise 3

Add to the ontology of Exercise 2 the following information:

- Man and woman are disjoint classes
- Every person has a mother
- Every person has a father
- Every person has exactly two parents
- Every person has a father, who is a man
- Every person has a mother, who is a woman
- Every person has a father and a mother

Exercise 3: Solution

1) `DisjointClasses(myns:man myns:woman)` (man and woman are disjoint classes)

2)
```
SubClassOf(
  myns:person
  ObjectSomeValuesFrom(myns:hasMother owl:Thing))
```
(every person has a mother)

3)
```
SubClassOf(
  myns:person
  ObjectSomeValuesFrom(myns:hasFather owl:Thing))
```
(every person has a father)

Exercise 3: Solution (continued)

4)
```
SubClassOf(
  myns:person
  ObjectExactCardinality(2 myns:hasParent))
```
(every person has exactly two parents)

5)
```
SubClassOf(
  myns:person
  ObjectSomeValuesFrom(myns:hasFather myns:man))
```
(every person has a father who is a man)

6)
```
SubClassOf(
  myns:person
  ObjectSomeValuesFrom(myns:hasMother myns:woman))
```
(every person has a mother who is a woman)

Exercise 3: Solution (continued)

7) **SubClassOf(**

myns:person

ObjectIntersectionOf(

ObjectSomeValuesFrom(myns:hasMother owl:Thing)

ObjectSomeValuesFrom(myns:hasFather owl:Thing)))

(every person has a mother and a father)

Notice that axiom 7) is equivalent to the above pair of axioms 2) and 3)

Knowledge Representation and Semantic Technologies

Exercises on OWL 2 profiles

Riccardo Rosati

Corso di Laurea Magistrale in Ingegneria Informatica
Sapienza Università di Roma
2019/2020

Exercise 1

Given the following TBox:

(1) MALE ⊑ PERSON
(2) FEMALE ⊑ PERSON
(3) hasMother ⊑ hasParent
(4) hasFather ⊑ hasParent
(5) hasChild ⊑ hasParent⁻
(6) MALE ⊓ FEMALE ⊑ ⊥ in DL-Lite$_R$ with negation
(7) ∃hasParent ⊑ IS-CHILD
(8) IS-CHILD ⊑ ∃hasParent
(9) ∃hasParent.HAPPY-PARENT ⊑ HAPPY-CHILD
(10) ∃hasChild.HAPPY-CHILD ⊑ HAPPY-PARENT
(11) HAPPY-CHILD ⊑ ∃hasParent
(12) HAPPY-PARENT ⊑ ∃hasChild
(13) HAPPY-PARENT ⊓ HAPPY-CHILD ⊑ HAPPY
(14) HAPPY ⊑ HAPPY-PARENT
(15) HAPPY ⊑ HAPPY-CHILD

	DL-Lite$_R$	EL	RL
1	×	×	×
2	×	×	×
3	×	—	×
4	×	—	×
5	×	—	×
6	×	×	×
7	×	×	×
8	×	×	—
9	—	×	×
10	—	×	×
11	×	×	—
12	×	×	—
13	—	×	×
14	×	×	×
15	×	×	×

Exercise 1

(a) Tell which of these axioms can be expressed in DL-Lite$_R$, EL, and RL, respectively;

Exercise 1

(b) given the following ABox:

(A1) MALE(Bob)
(A2) MALE(Paul)
(A3) FEMALE(Ann)
(A4) hasMother(Paul,Ann)
(A5) hasFather(Mary,Paul)
(A6) hasChild(Paul,Jane)
(A7) hasChild(Jane,Bob)
(A8) HAPPY(Ann)
(A9) HAPPY-CHILD(Jane)

and the TBox obtained from the previous one by discarding the axioms not expressible in RL, determine the instances of the concept HAPPY by applying forward chaining;

Exercise 1

(c) Given the above ABox and the TBox obtained from the previous one by discarding the axioms not expressible in DL_Lite$_R$:

(c1) determine the instances of the concept HAPPY by applying query rewriting;

(c2) determine the instances of the query q(x) :- hasParent(x,y) by applying query rewriting.

Exercise 1(a): Solution

The axioms expressible in DL-Lite$_R$ are:

(1), (2), (3), (4), (5), (6), (7), (8), (11), (12), (14), (15)

Notice that axiom (6) can be expressed in DL-Lite$_R$ by the equivalent axiom MALE $\sqsubseteq \neg$ FEMALE

The axioms expressible in EL are:

(1), (2), (7), (8), (9), (10), (11), (12), (13), (14), (15)

The axioms expressible in RL are:

(1), (2), (3), (4), (5), (6), (7), (9), (10), (13), (14), (15)

Exercise 1(b): Solution

The ABox obtained by chasing the initial ABox with the RL axioms of the TBox is obtained by adding to the initial ABox the following assertions:

(A10) PERSON(Bob) (follows from (A1) and TBox axiom (1))
(A11) PERSON(Paul) (follows from (A2) and TBox axiom (1))
(A12) PERSON(Ann) (follows from (A3) and TBox axiom (2))
(A13) hasParent(Paul,Ann) (follows from (A4) and TBox axiom (3))
(A14) hasParent(Mary,Paul) (follows from (A5) and TBox axiom (4))
(A15) hasParent(Jane,Paul) (follows from (A6) and TBox axiom (5))
(A16) hasParent(Bob,Jane) (follows from (A7) and TBox axiom (5))
(A17) IS-CHILD(Paul) (follows from (A13) and TBox axiom (7))
(A18) IS-CHILD(Mary) (follows from (A14) and TBox axiom (7))
(A19) IS-CHILD(Jane) (follows from (A15) and TBox axiom (7))
(A20) IS-CHILD(Bob) (follows from (A16) and TBox axiom (7))

Exercise 1(b): Solution (cont'd)

(A21) HAPPY-PARENT(Ann) (follows from (A8) and TBox axiom (14))
(A22) HAPPY-CHILD(Ann) (follows from (A8) and TBox axiom (15))
(A23) HAPPY-CHILD(Paul) (follows from (A13), (A21) and TBox axiom (9))
(A24) HAPPY-PARENT(Paul) (follows from (A6), (A9) and TBox axiom (10))
(A25) HAPPY(Paul) (follows from (A21), (A22) and TBox axiom (13))
(A26) HAPPY-CHILD(Mary) (follows from (A14), (A24) and TBox axiom (9))

Therefore, the instances of the concept HAPPY are: {Ann, Paul}

Exercise 1(c1): Solution

The rewriting of the query

q(x) :- HAPPY(x)

w.r.t. the DL-LiteR axioms of the TBox is simply:

q(x) :- HAPPY(x)

since there are no subconcepts of HAPPY (notice that axiom (13) is not a DL-LiteR axiom, hence it is ignored).

By evaluating such a query on the initial ABox, we obtain the answers {Ann}.

Exercise 1(c2): Solution

The rewriting of the query

q(x) :- hasParent(x,y)

w.r.t. the DL-LiteR axioms of the TBox is the following:

(Q1) q(x) :- hasParent(x,y) (initial query)
(Q2) q(x) :- hasMother(x,y) (obtained from (Q1) and TBox axiom (3))
(Q3) q(x) :- hasFather(x,y) (obtained from (Q1) and TBox axiom (4))
(Q4) q(x) :- hasChild(y,x) (obtained from (Q1) and TBox axiom (5))
(Q5) q(x) :- IS-CHILD(x) (obtained from (Q1) and TBox axiom (8))
(Q6) q(x) :- HAPPY-CHILD(x) (obtained (Q1) and TBox axiom (11))
(Q7) q(x) :- HAPPY(x) (obtained from (Q6) and TBox axiom (15))

By evaluating such a query on the initial ABox, we obtain the answers {Paul, Mary, Jane, Bob, Ann}.

Knowledge Representation and Semantic Technologies

Exercises on Knowledge Representation

Riccardo Rosati

Corso di Laurea Magistrale in Ingegneria Informatica
Sapienza Università di Roma
2019/2020

Exercise 1

We want to formalize knowledge about the domain of students and professors.

In particular, we want to formalize the following statements:

Exercise 1 (contd.)

Student, Person, Professor : Classes

1. Every student is a person
2. Every professor is a person
3. Every student has an ID (matricola) existential statement
4. Every student attends at least one course range
5. Every course is taught by a professor
6. Every student is not a professor disjoint Class
7. Every exam is constituted of a course, a student, a date and a grade

3. existential in the head part of the rule is not allowed in Datalog, ASP
existential in superclass position not allowed in RL
no concept constructor in RDFS

4,5 no qualified existential restriction in DL-Lite$_R$ (only unqualified exist. restriction)

6 EL doesn't have empty class, negation

Exercise 1 (contd.)

1) Choose the most appropriate knowledge representation language for expressing the above knowledge among the following:

following:	1	2	3	4	5	6	7	
• ALC	×	×	×	×	×	×	×	
• Datalog	×	×	—	—	—	—	×	
• ASP	×	×	—	—	—	×	×	
• OWL-DL	×	×	×	×	×	×	×	
• $DL\text{-}Lite_R$	×	×	×	—	—	×		
• EL	×	×	×	×	×	—		
• RL	×	×	—	×	×	×		
• RDFS	×	×	—	×	×	×		

2) Express the above knowledge in the formalism chosen at the previous point.

Exercise 1 - Solution

1) The formalism chosen is ALC, because:
   - According to statements 3, 4, 5, we need to refer to «existential» individuals (and Datalog, ASP, RL and RDFS are not suited for this, so we can discard them)
   - According to statements 4,5, we need qualified existential restriction (DL-Lite$_R$ is not suited for this)
   - According to statement 6, we need to express disjointness between concepts (EL is not able to express this)

   So, only ALC and OWL-DL are able to express all the above statements. Therefore, we choose ALC (we do not need the extra abilities of OWL-DL).

Exercise 1 - Solution

2) We choose the following vocabulary (set of predicates and individuals):

Concepts: STUDENT, PERSON, PROFESSOR, COURSE, GRADE, DATE

Roles: hasID, attends, isTaughtBy

Moreover, we have ro reify the notion of exam at point 7 (that would be naturally encoded by a relation of arity 4) using a concept EXAM and four auxiliary roles examStudent, examCourse, examDate, examGrade

Exercise 1 - Solution

2) Here is the formalization of the statements in ALC:

$$\text{STUDENT} \sqsubseteq \text{PERSON}$$
$$\text{PROFESSOR} \sqsubseteq \text{PERSON}$$
$$\text{STUDENT} \sqsubseteq \exists \text{hasID}.\text{T}$$
$$\text{STUDENT} \sqsubseteq \exists \text{attends}.\text{COURSE}$$
$$\text{COURSE} \sqsubseteq \exists \text{isTaughtBy}.\text{PROFESSOR}$$
$$\text{STUDENT} \sqsubseteq \neg\, \text{PROFESSOR}$$
$$\text{EXAM} \sqsubseteq \exists \text{examCourse}.\text{COURSE}$$
$$\text{EXAM} \sqsubseteq \exists \text{examStudent}.\text{STUDENT}$$
$$\text{EXAM} \sqsubseteq \exists \text{examGrade}.\text{GRADE}$$
$$\text{EXAM} \sqsubseteq \exists \text{examDate}.\text{DATE}$$

Exercise 1 - Comment

- Notice that statement 7 is problematic for ALC (and Description Logics in general), since DLs do not allow for expressing relations of arity greater than 2.
- However, the reification technique allows for representing (although unnaturally) these kinds of n-ary relations.
- Datalog and ASP allow for directly define a 4-ary exam relation, so, with respect to statement 7 alone, they should be preferred: however, they have more important limitations with respect to the other statements (in particular, both Datalog and ASP are not able to express statements 3,4,5; in addition, Datalog is not able to express statement 6)

Exercise 2

We want to formalize knowledge about the domain of students and professors.

In particular, we want to formalize the following statements:

Exercise 2 (contd.)

1. Every student is a person
2. Every professor is a person
3. Every student has an ID (matricola)
4. Every course is taught by a professor
5. Every student who attends a course is an active student
6. Every student is not a professor

new class

	DL-Lite$_R$	EL	RL
1	X	X	X
2	X	X	X
3	—	X	—
4	—	X	—
5	—	X	X
6	X	—	X

STUDENT $\sqcap \exists$ attends. COURSE $\sqsubseteq$ ACTIVE STUDENT

No language can express all statement

Exercise 2 (contd.)

For each of the following description logics:

- DL-Lite$_R$
- EL
- RL

identify which of the above statements can be formalized, and write the corresponding axioms.