ΤΕΧΝΟΛΟΓΙΕΣ ΓΝΩΣΕΩΝ ΕΡΓΑΣΙΑ 2

ΜΑΛΩΝΑΣ ΚΩΝΣΤΑΝΤΙΝΟΣ
ID: 7115112200020
EMAIL: cs22200020@di.uoa.gr

Exec_1

Query
P = (((?N, type ,Dwarf) AND (?N, hasFather, ?F)) OPT (?N, hasTitle, ?T))

Sparql query
SELECT ?N ?F ?T WHERE { ?N rdf:type ex:Dwarf . ?N ex:hasFather ?F . OPTIONAL (?N ex:hasTitle ?T)}

Result

mappings	?N	?F	?T
μ1	Thorin	Thrain	King under the Mountain
μ2	Thrain	Thor	

Algebraic evaluation
We can split the above **query** into 3 graph patterns with 1 triple each:
P1 = (?N, type ,Dwarf)
P2 = (?N, hasFather, ?F)
P3 = (?N, hasTitle, ?T)

For every graph pattern we replace its variables according to mapping μ.
For every mapping of P1, P2, P3 we have the following:

For P1:
$[[t1]]_D = \{\mu1 \mid dom(\mu1) = var(t1) \text{ and } \mu1(t1) \in D\}$

For P2:
$[[t2]]_D = \{\mu2 \mid dom(\mu2) = var(t1) \text{ and } \mu2(t1) \in D\}$

For P3:
$[[t3]]_D = \{\mu3 \mid dom(\mu3) = var(t3) \text{ and } \mu3(t3) \in D\}$

Following we have:
$[[(P1 \text{ AND } P2)]]_D = [[P1]]_D \bowtie [[P2]]_D$ ← We symbolize that with **P4**
$[[(P4 \text{ OPT } P3)]]_D = [[P4]]_D \mathbin{⟕} [[P3]]_D$

For the mappings μ1, μ2, μ3 we have:
$\Omega1 \bowtie \Omega2 = \{\mu1 \cup \mu2 \mid \mu1 \in \Omega1, \mu2 \in \Omega2 \text{ are compatible mappings}\}$ ← We symbolize that as **Ω4**
$\Omega4 \mathbin{⟕} \Omega3 = (\Omega4 \bowtie \Omega3) \cup (\Omega4 \setminus \Omega3)$

Exec_2

(a) All postgrad students are students.

$\forall x$ (PostgradStudent (x) $\rightarrow$ Student(x))

(b) Every postgrad student has bachelor degree.

ALCQO:
PostgradStudent $\equiv$ Student $\sqcap$ $\exists$ hasDegree.{BACHELOR}
FOL:
$\forall x$ (PostgradStudent (x) $\rightarrow$ hasBachelorDegree(x))

(c) There exists one postgrad student.

$\exists x$(PostgradStudent(x))

If we wanted to say <u>There exists **exactly** one postgrad student.</u> we would write:
$\exists x$ (PostgradStudent (x) $\wedge$ $\forall y$ (PostgradStudent (y) $\rightarrow$ y = x))

(d) Every student follows some course.

$\forall x$ (Student(x) $\rightarrow$ followsCourse(x))

(e) Every student follows Knowledge Technologies course and some other course.
In my opinion we cannot describe that sentence in ALCQO because it says at the end **"and some other course"** .

$\forall x$ (Student(x) $\rightarrow$ follows(Knowledge_Technologies, x) $\wedge$ $\exists$ y(isCourse(y) $\wedge$ follows(y, x) $\wedge$ y $\neq$ Knowledge_Technologies))

(f) There exists one student that all other students dislike him.

$\exists x$ (Student(x) $\wedge$ $\forall y$ ((Student(y) $\wedge$ y $\neq$ x) $\rightarrow$ dislikes(y,x)))

(g) Peter is a person.

Person $\equiv$ isPerson.{PETER}

(h) Peter does not like the Knowledge Technologies course.

Peter $\sqsubseteq$ Person $\sqcap$ $\exists$ doesNotLikeCourse.{KNOWLEDGE_TECHNOLOGIES}

(i) None of the students likes Peter.

Student ≡ Person ⊓ ∀ doesNotLike.{PETER}

(j) Peter follows at least one course.

∃ x (Course(x) ∧ follows(Peter,x))

(k) Paul does not follow any course.

∀ x (Course(x) → ¬follows(Paul, x))

(l) Every student follows at least one course.

∀ x (Student(x) → ∃ y(Course(y) ∧ follows(x, y)))

(m) Only one student failed the Knowledge Technologies course.

Student ⊑ Person ⊓ (=1 ∃ failedAt.{KNOWLEDGE_TECHNOLOGIES})

(n) None of the students failed the Knowledge Technologies course but at least one student failed at the Database course.

Student ⊑ Person ⊓ ¬(∀ failedAt.{KNOWLEDGE_TECHNOLOGIES}) ⊓
(∃ failedAt.{DATABASE})

(o) Every student that follows the Knowledge Technologies course follows also the Logic Programming course.

Student ⊑ Person ⊓ ∃ follows.{KNOWLEDGE_TECHNOLOGIES} ⊓
∃ follows.{LOGIC_PROGRAMMING}

(p) There is no student that can fool all the other students.

q) A biped is an animal that has exactly two legs

Biped ≡ Animal ⊓ ∀ (>=2 has.{LEGS}) ⊓ ∀ (<=2 has.{LEGS})

(r) A triangle is a polygon with exactly three edges and exactly three vertices which are line segments.

(s) A right-angled triangle is a triangle in which one angle is a right angle.

RightAngledTriangle ≡ Triangle ⊓ (>= 1 angles.{RIGHT}) ⊓ (<= 1 angles.{RIGHT})

Exec_3

(a) Person ⊓ **hasChild**: Incorrect

(b) ∃ **hasChild.** ≡ **Person**: Incorrect

(c) ∃ **hasChild.(≥ 1)**: Incorrect
≥ 1 should be in front of the role and after the (**.**) there should be a concept

(d) hasChild ⊑ **hasBaby**: Incorrect

(e) hasChild(ANNA): Incorrect

(f) **Person** ≡ ∃ **hasChild.** ⊥ : Correct

Exec_4

(a)

Let the TBox T be:
Frappe ⊑ Coffee # We could have not written that because it is implied
Greek ≡ Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe

Let the Abox A be:
Person(YANNIS),

(Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe)(YANNIS) # I didn't write only ∀ **drinks.Frappe** because that would mean Yannis drinks just coffee(Frappe)

drinks(YANNIS, FRAPPE)

φ: Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe(YANNIS)

(b)

<u>Description</u>
Concepts: Person, Greek, Frappe, Coffee
Roles: drinks
Individuals: YANNIS, FRAPPE

<u>Match individuals</u>

$YANNIS^I$ = yannis
$FRAPPE^I$ = frappe

$Person^I$ = {yannis}
$Greek^I$ = {yannis}
$Coffee^I$ = {frappe}
$(Person \sqcap \exists drinks.Coffee \sqcap \forall drinks.Frappe)^I$ = (yannis)

drinks(yannis, frappe)

(c)

Description
Concepts: Person, Greek, Frappe, Coffee
Roles: drinks
Individuals: ANNA, LATTE

Match individuals
$ANNA^I$ = anna
$LATTE^I$ = latte

$Person^I$ = {anna}
$Greek^I$ = {anna}
$(Person \sqcap \exists drinks.Coffee \sqcap \forall drinks.Frappe)^I$ = (anna)

drinks(anna, latte)

the above interpretation is **wrong**, latte concept does **not exist** in our KB

(d)

KB |= φ

T = { $Greek \equiv Person \sqcap \exists drinks.Coffee \sqcap \forall drinks.Frappe$ }

That =
{
$Greek \sqsubseteq (Person \sqcap \exists drinks.Coffee \sqcap \forall drinks.Frappe)$,
$(Person \sqcap \exists drinks.Coffee \sqcap \forall drinks.Frappe) \sqsubseteq Greek$
} →

That =
{
$\neg Greek \sqcup (Person \sqcap \exists drinks.Coffee \sqcap \forall drinks.Frappe)$,
$\neg(Person \sqcap \exists drinks.Coffee \sqcap \forall drinks.Frappe) \sqcup Greek$
}

nnf(That**) =**
{
¬Greek ⊔ (Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe),
¬Person ⊔ ∃ drinks.¬Coffee ⊔ ∀ drinks.¬Frappe ⊔ Greek
}

φ: Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe(YANNIS)

1. ¬Person ⊔ ∃ drinks.¬Coffee ⊔ ∀ drinks.¬Frappe ⊔ Greek (Y) , **(given)**
2. ¬Greek ⊔ (Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe)(Y), **(given)**
3. Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe(Y), **(given)**
4. ¬Person(Y) , **1 ⊔-Rule**
5. ∃ drinks.¬Coffee(Y) , **1 ⊔-Rule**
6. ∀ drinks.¬Frappe(Y) , **1 ⊔-Rule**
7. Greek(Y) , **1 ⊔-Rule**
8. ¬Greek(Y), **2 ⊔-Rule Clash 7-8**
9. Person(Y), **2 ⊓-Rule Clash 4-9**
10. ∃ drinks.Coffee(Y), **2 ⊓-Rule Clash 5-10**
11. ∀ drinks.Frappe(Y), **2 ⊓-Rule Clash 6-11**
12. Person(Y) , **3 ⊓-Rule Clash 4-12**
13. ∃ drinks.Coffee(Y) , **3 ⊓-Rule Clash 5-13**
14. ∀ drinks.Frappe(Y) , **3 ⊓-Rule Clash 6-14**

So **KB |= φ**

Exec_5

(a)
Tbox
Frappe ⊑ Coffee
Nescafe ⊑ Frappe
Greek ≡ Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe

Abox
Person(YANNIS)
Frappe(NESCAFE)
drinks.Nescafe(YANNIS)
Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Nescafe(YANNIS)

φ: Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe (YANNIS)

(b)

That =
{
Greek ⊑ Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe ,
Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe ⊑ Greek,
¬Frappe ⊔ Coffee,
¬Nescafe ⊔ Frappe
} →

That =
{
¬Greek ⊔ (Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe),
¬(Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe) ⊔ Greek,
¬Frappe ⊔ Coffee,
¬Nescafe ⊔ Frappe
}

According to **slide 58** of tableaux-techniques
nnf(That**) =**
{
¬Greek ⊔ (Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe),
¬Person ⊔ ∃ drinks.¬Coffee ⊔ ∀ drinks.¬Frappe ⊔ Greek,
¬Frappe ⊔ Coffee,
¬Nescafe ⊔ Frappe
}

1. ¬Person ⊔ ∃ drinks.¬Coffee ⊔ ∀ drinks.¬Frappe ⊔ Greek (Y) , **(given)**
2. ¬Greek ⊔ (Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe)(Y), **(given)**
3. Person ⊓ ∃ drinks.Coffee ⊓ ∀ drinks.Frappe(Y), **(given)**
4. ¬Person(Y) , **1** ⊔**-Rule**
5. ∃ drinks.¬Coffee(Y) , **1** ⊔**-Rule**
6. ∀ drinks.¬Frappe(Y) , **1** ⊔**-Rule**
7. Greek(Y) , **1** ⊔**-Rule**
8. ¬Greek(Y), **2** ⊔**-Rule Clash 6-7**
9. Person(Y), **2** ⊓**-Rule Clash 4-9**
10. ∃ drinks.Coffee(Y), **2** ⊓**-Rule Clash 5-10**
11. ∀ drinks.Frappe(Y), **2** ⊓**-Rule Clash 6-11**
12. Person(Y) , **3** ⊓**-Rule Clash 3-7 Clash 4-12**
13. ∃ drinks.Coffee(Y) , **3** ⊓**-Rule Clash 5-13**
14. ∀ drinks.Frappe(Y) , **3** ⊓**-Rule Clash 6-14**

So **KB |= φ**

Exec_6

(a)

Tbox
Pet ⊑ Animal
AnimalLover ≡ Person ⊓ ∃ hasAnimal.Pet ⊓ (>= 3 hasAnimal.Pet)

Abox
Person(WALT)
(Person ⊓ ∃ hasAnimal.Pet ⊓ (>= 3 hasAnimal.Pet) ⊓ (< 4 hasAnimal.Pet))(WALT)
Pet(HUEY), Pet(DEWEY), Pet(LOUIE)
hasAnimal(WALT, HUEY)
hasAnimal(WALT, DEWEY)
hasAnimal(WALT, LOUIE)

φ
Person ⊓ ∃ hasAnimal.Pet ⊓ (>= 3 hasAnimal.Pet)(WALT)

(b)

That = {
AnimalLover ⊑ Person ⊓ ∃ hasAnimal.Pet ⊓ (>= 3 hasAnimal.Pet),
Person ⊓ ∃ hasAnimal.Pet ⊓ (>= 3 hasAnimal.Pet) ⊑ AnimalLover, ¬Pet ⊔ Animal } →

That = {
¬AnimalLover ⊔ Person ⊓ ∃ hasAnimal.Pet ⊓ (>= 3 hasAnimal.Pet),
¬Person ⊔ ∃ hasAnimal.¬Pet ⊔ (<= 2 hasAnimal.Pet) ⊔ AnimalLover, ¬Pet ⊔ Animal
}

nnf(That) = {
¬AnimalLover ⊔ Person ⊓ ∃ hasAnimal.Pet ⊓ (>= 3 hasAnimal.Pet),
¬Person ⊔ ∃ hasAnimal.¬Pet ⊔ (<= 2 hasAnimal.Pet) ⊔ AnimalLover,
¬Pet ⊔ Animal
}

1. (¬Person ⊔ ∃ hasAnimal.¬Pet ⊔ (<= 2 hasAnimal.Pet) ⊔ AnimalLover)(W) **(given)**
2. ¬AnimalLover ⊔ Person ⊓ ∃ hasAnimal.Pet ⊓ (>= 3 hasAnimal.Pet)(W) **(given)**
3. (Person ⊓ ∃ hasAnimal.Pet ⊓ (>= 3 hasAnimal.Pet))(W) **(given)**
4. ¬Person(W) , **1** ⊔**-Rule**
5. ∃ hasAnimal.¬Pet(W), **1** ⊔**-Rule**
6. (<= 2 hasAnimal.Pet)(W) , **1** ⊔**-Rule**
7. AnimalLover(W) , **1** ⊔**-Rule**
8. ¬AnimalLover(W), **2** ⊔**-Rule Clash 7-8**
9. Person(W), **2** ⊔**-Rule Clash 4-9**
10. ∃ hasAnimal.Pet(W), **2** ⊓**-Rule Clash 5-10**

11. (>= 3 hasAnimal.Pet)(W), **2 ⊓-Rule Clash 6-11**
12. Person(W), **3 ⊔-Rule Clash 4-12**
13. ∃ hasAnimal.Pet(W) , **3 ⊔-Rule Clash 5-13**
14. (>= 3 hasAnimal.Pet)(W), **3 ⊔-Rule Clash 6-14**

So **KB |= φ**

Exec_7

1)
My ontology is at **my_ontology.owl** file

2)

Question	Mark
How much of the specific fragment of the ontology did you manage to recreate? (1 for none of it, 10 for all of it)	5
How helpful was the tool?	4
Would you reuse such a tool for future projects with ontologies?	1
What level of KT-knowledge do you think is required to use this tool? (1 for none of it, 10 for very good knowledge of KTs)	6
How familiar were you with the language used?	7
Do the explanations given by the tool suffice to complete your task?	3
To what level did the tool require from you some extra effort (e.g., to memorize information)	8
To what level were you aware of what the tool was doing at each step?	2
Did you often face error messages (how did you deal with them?)?	2. I did not face error messages, and maybe that would be a good addition to the program so the user will know if the ontology s/he creating is correct.

The tool Cha2O was very interesting to use. Although some of the main aspects that can be improved are:

1. More detailed documentation. I was not aware most of the time how to do what was necessary to create my ontology. The importance of documentation and instructions to the user becomes more clear even more, when a person with little knowledge of ontology construction tries to use this tool.
2. The user should be able to delete/undo an action
3. I have found it very difficult to define relations. E.g PostgraduateRecord in my ontology contains Postgraduate Courses and up to two Undergraduate. I could not find a way to declare that relationship in Cha2O.
4. The user must be able to declare data and object restrictions

I could not declare the object relationship undergraduate record contains undergraduate courses (for some reason I was able to do that with the postgraduate record to an extent). I was able to show that a postgraduate student follows postgraduate and undergraduate courses but I could not declare the constraint that s/he can take up to two undergraduate courses. I could not show that postgraduate students have postgraduate records. Additionally I was not able to show that up to two undergraduate courses can be contained in a postgraduate record of a postgraduate student. Also I could not show that marks in postgraduate and undergraduate courses should be of type double.

Overall with more documentation and instructions from the program to the user during the construction of the ontology, the delete action functionality and the capability to add more object and data restrictions Och2O could be a very useful tool for ontology construction.