Tesina di Seminari di Ingegneria del Software

Testing QuOnto su ABox di dimensione crescente

Claudio Corona

A.A. 2006-2007

Sommario

1. Introduzione

Lo scopo del seguente lavoro è mettere in luce, attraverso un esempio concreto, le due caratteristiche che esaltano QuOnto (Quering Ontologies): a) query answering con complessità computazionale LOGSPACE, rispetto alla dimensione dei dati b) scalabilità.
In particolare, prenderemo in esame un dataset messo a disposizione dall'IBM sul proprio sito internet e su cui l'azienda ha effettuato i test del proprio OWL reasoner, SHER.
SHER (Scalable Highly Expressive Reasoner) è una tecnologia innovativa che permette di effettuare analisi su ontologie molto espressive. La logica supportata corrisponde a *SHIN(D)*, che cattura tutto OWL-DL

$\mathcal{SHIN}^{(\mathcal{D})}$	
Simbolo	Spiegazione
$\mathcal{S}$	Abbreviation for ALC with transitive roles. ALC allows concept intersection, full negation, full universal quantification, full existential
$\mathcal{H}$	Role (property) hierarchy
$\mathcal{I}$	Inverse roles (properties which have inverses specified, or properties that are symmetric)
$\mathcal{N}$	Number restrictions (cardinality restrictions, also includes functional properties
$(\mathcal{D})$	Datatypes

Tabella 1: SHIN(D)

ad eccezione dei *nominals*, o classi enumerate (esempio di classe enumerata: *dayOfTheWeek* = { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday }). Può ragionare fino a 7 milioni di triple al secondo, e gestisce datasets contenenti fino a 60 milioni di triple. Inoltre, supporta il controllo della consistenza logica dei dati, e aiuta l'utente ad eliminare tali inconsistenze prima di avviare la fase di analisi, ovvero le queries semantiche. Infine, nel restituire un certo risultato ad una query, SHER fornisce una spiegazione (o giustificazione) del motivo che lo ha portato a fornire quel risultato.
Internamente SHER fa uso di Pellet, un OWL-DL reasoner open-source e molto popolare scritto in Java, di cui parleremo meglio nel paragrafo riguardante i test.

2. L'ontologia dell'IBM

L'ontologia in questione ha come dominio di interesse l'ambiente universitario ed è stata costruita sulla base dell'ontologia *Lehigh University BenchMark* (LUBM), sviluppata a sua volta all'interno di una suite di benchmarking che potesse permettere di valutare i Semantic Web repositories (o Knowledge Base Systems) in modo standard e sistematico, ponendo l'accento su queries estensionali poste su grandi datasets aventi a riferimento una singola e realistica ontologia (dapprima la suite è stata sviluppata per valutare solo repositories in DAMN+OIL e successivamente è stata estesa per poter essere usata anche con ontologie e dataset scritti in OWL). Tale ontologia originaria è scritta in OWL-Lite, di conseguenza quella dell'IBM sarà costituita da un livello intensionale più ricco dal punto di vista espressivo.

La struttura dei file e delle directories del dataset messo a disposizione dall'IBM è la seguente:

dataset
| univ-bench-dl.nonominals.owl
| *Rappresenta il livello intensionale dell'ontologia, ovvero la TBox*
|
+ - - -1 (23,1 MB)
| 1-ub-dl-univ0.owl
| *Rappresenta alcune informazioni di interesse per l'unica università presente*
| {1-ub-dl-univ0-dept?.owl }
| *? è un numero che va da 0 a 19; ogni file corrisponde ad un dipartimento di quell'università*
+ - - -5 (100 MB)
| {5-ub-dl-univ?.owl }
| *? è un numero che va da 0 a 4; ogni file descrive alcune informazioni d'interesse per*
| *quell'università*
| {5-ub-dl-univ?-dept??.owl}
| *? è un numero che va da 0 a 4; ?? è un numero che va da 0 fino ad un massimo di 21*
| *(abbiamo 5 università da 15~22 dipartimenti ciascuna)*
|
+ - - -10 (196 MB)
| {10-ub-dl-univ?.owl}
| *? è un numero che va da 0 a 9; ogni file descrive alcune informazioni d'interesse per*
| *quell'università*
| {10-ub-dl-univ?-dept??.owl}
| *? è un numero che va da 0 a 9; ?? è un numero che va da 0 fino ad un massimo di 21*
| *(abbiamo 10 università da 15~22 dipartimenti ciascuna)*
|
\ - - -30 (700 MB)
{30-ub-dl-univ?.owl}
? è un numero che va da 0 a 29; ogni file descrive alcune informazioni d'interesse per quell'università
{30-ub-dl-univ?-dept??.owl}
? è un numero che va da 0 a 29; ?? è un numero che va da 0 fino ad un massimo di 24 (abbiamo 30 università da 15~25 dipartimenti ciascuna)

Le 4 cartelle presenti corrispondono a 4 possibili Aboxes, o livelli estensionali, contenenti, in ordine, 1,5,10 e 30 università e, di conseguenza, aventi dimensione via via crescente. Ad ogni università sono associati n+1 files, dove n è il numero di dipartimenti che costituiscono quell'università, mentre l'altro file è relativo ad informazioni generiche riguardanti l'intera università.
Il nostro obiettivo è quello di trasformare questo dataset in un input per QuOnto e vedere come questo si comporta al crescere dei dati. I passi saranno quindi i seguenti:

1. Trasformeremo la TBox scritta in formato OWL in una TBox scritta in formato XML,secondo la DTD di QuOnto, ed eliminando tutto ciò che non può essere espresso in *DL-Lite*$_A$;
2. Svilupperemo un tool scritto in Java in grado di tradurre automaticamente le Aboxes scritte in formato OWL in Aboxes scritte in formato XML, secondo la DTD definita in QuOnto;
3. Effettueremo il testing su QuOnto, comparandolo con altri due reasoners.

2.1. LUBM TBox vs. IBM TBox

Come detto poco sopra, la TBox dell'IBM è più ricca della TBox della LUBM. Vediamo ora in cosa consistono le novità introdotte a livello intensionale.

CONCETTI

1. Gerarchia del concetto Organization:
    a. viene aggiunto il concetto WomanCollege come figlio di College
2. Gerarchia/caratterizzazione del concetto Person:
    a. Il concetto Person viene reso equivalente a Man ⊔ Woman
    b. Vengono aggiunti i concetti Man, Woman e la disgiunzione Man ⊑ ¬ Woman
    c. Cambia la definizione del concetto di Student e (parzialmente)la sua gerarchia: nella TBox originaria, Student ≡ Person ⊓ ∃takesCourse.Course, mentre nella TBox dell'IBM Student ≡ Person ⊓ ∃isStudentOf.Organization
    Nella TBox originaria i figli di Student sono UndergraduateStudent e ResearchAssistant (quest'ultimo con il vincolo di dover lavorare per qualche gruppo di ricerca), nella TBox dell'IBM i figli sono UndergraduateStudent, ScienceStudent e NonScienceStudent, mentre ResearchAssistant ⊑ ∃worksFor.ResearchGroup e ResearchAssistant ⊑ Person, quindi un ResearchAssistant può anche non essere uno studente
    d. Vengono aggiunti tutti i concetti del tipo ..Fan e ..Lover (vedi punto 3)
    e. Viene aggiunto il concetto PeopleWithManyHobbies (collegato al concetto di cui al punto 3)
    f. Viene aggiunto il concetto PeopleWithHobby (collegato al concetto di cui al punto 3)
3. Viene aggiunto il concetto Interest (con Music e Sports come figli)
4. Viene aggiunto il concetto AcademicSubject con la sua gerarchia
5. Viene aggiunto il concetto LeisureStudent

RUOLI

6. Il punto 3 dà vita al ruolo like-love, che lega, per lo più, persone con interessi
7. Il punto 4 dà vita al ruolo hasMajor, che lega studenti con materie accademiche
8. Viene aggiunto il ruolo isStudentOf ⊑ worksFor ed il ruolo enrollIn ⊑ isStudentOf
9. Viene aggiunto isTaughtBy come ruolo inverso di teacherOf
10. Viene aggiunto il ruolo (simmetrico) isFriendOf
11. Viene aggiunto il ruolo (simmetrico e transitivo) hasSameHomeTownWith.

Possiamo notare come tutte le novità introdotte nella TBox dell'IBM riguardano concetti che nella loro caratterizzazione fanno uso di ∀, ¬ e partecipazione a ruoli con cardinalità minima o massima maggiore di 1. Inoltre, nella TBox originale non vengono mai dichiarati ruoli transitivi, simmetrici o funzionali. Ciò, chiaramente, è determinato dal fatto che il reasoner dell'IBM, SHER, si basa su una logica descrittiva più

espressiva di OWL Lite. La maggior parte dei concetti che sono stati introdotti nella TBox della IBM non sono esprimibili neanche in *DL-Lite*$_A$: solo l'operatore unario $\neg$, talvolta, può essere espresso. In tutti gli altri casi, i concetti coinvolti saranno rilassati nella misura tale da poter rientrare nella sfera di espressività di *DL-Lite*$_A$ (rilassamento minimo). Qualora tale misura sia tale da non poter fare alcuna asserzione su un certo concetto, questo sarà soltanto presente nell'alfabeto dell'ontologia, senza darne una qualche caratterizzazione a livello di asserzioni (rilassamento minimo coincidente con il rilassamento massimo).

2.2. La TBox dell'IBM e il suo adattamento a *DL-Lite*$_A$

La figura sottostante esprime alcuni dati numerici sulla TBox dell'IBM.

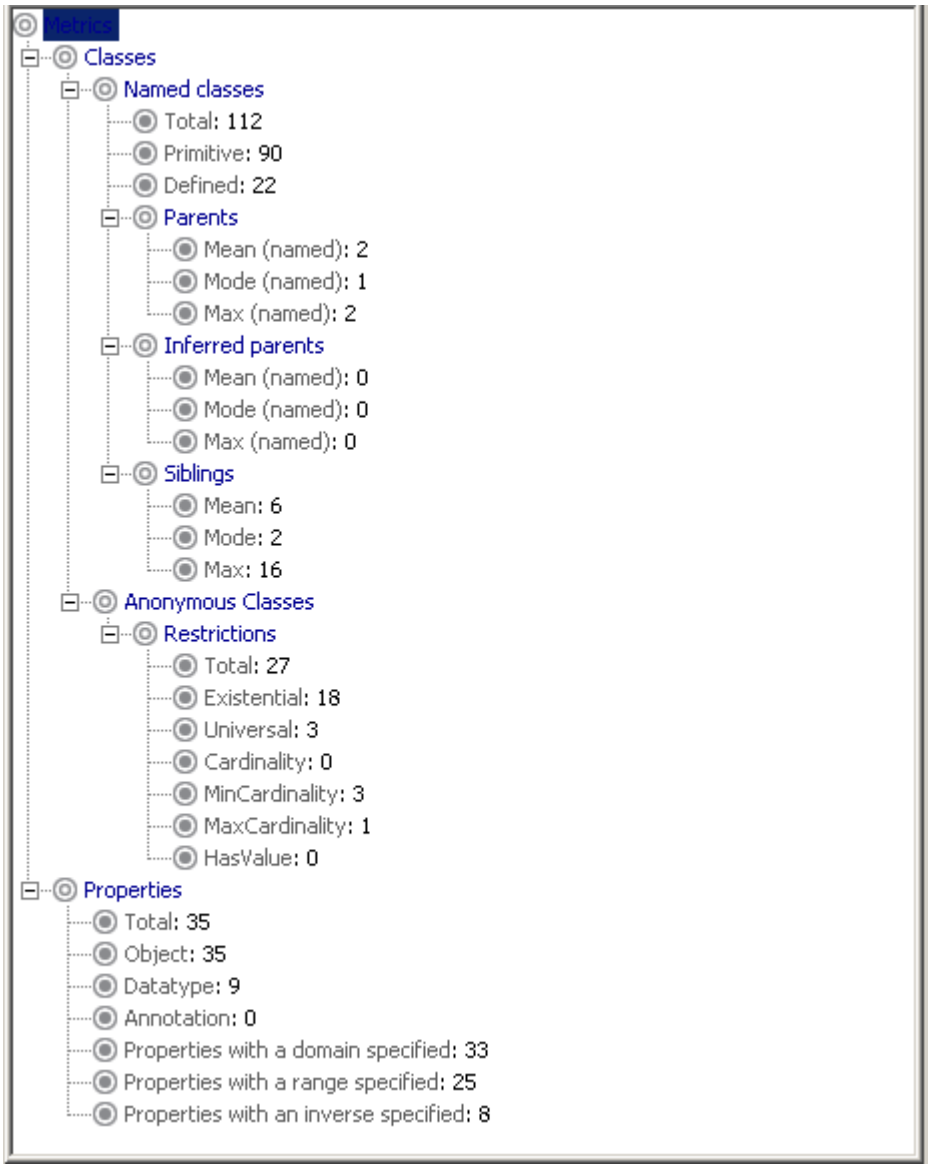

Figura 1: i numeri della TBox

I dati più importanti sono i seguenti:

a) I ruoli totali, trascurando quelli inversi (che di fatto sono degli "alias" di quelli diretti) sono 35-8=27;
b) I concetti totali, considerando solo quelli primitivi, sono 90 (91 con *Thing,* il concetto Top);
c) I concetti totali, considerando anche quelli derivati (ossia quei concetti che fanno uso del costrutto di equivalenza), sono 112.

Ora ci concentriamo sulla fase di adattamento della TBox nel linguaggio *DL-Lite*$_A$.

Di seguito si fornisce una rappresentazione gerarchico-formale in OWL-DL della TBox, con annesso adattamento a *DL-Lite*$_A$, qualora necessario, indicato con un colore più chiaro.

Concetti e attributi di concetto

1. Publication ⊑ Thing
   - 1.1. Book ⊑ Publication
   - 1.2. Manual ⊑ Publication
   - 1.3. Specification ⊑ Publication
   - 1.4. Software ⊑ Publication
   - 1.5. UnofficialPubblication ⊑ Publication
   - 1.6. Article ⊑ Publication
     - 1.6.1. TechnicalReport ⊑ Article
     - 1.6.2. JournalArticle ⊑ Article
     - 1.6.3. ConferencePaper ⊑ Article

2. Person ⊑ Thing
   Person ≡ Woman ⊔ Man
   In DL-Lite: Woman ⊑ Person ; Man ⊑ Person
   δ(title) ⊑ Person
   ρ(title) ⊑ xsd:string
   δ(age) ⊑ Person
   ρ(age) ⊑ xsd:string
   δ(telephone) ⊑ Person
   ρ(telephone) ⊑ xsd:string
   δ(firstname) ⊑ Person
   ρ(firstname) ⊑ xsd:string
   δ(lastname) ⊑ Person
   ρ(lastname) ⊑ xsd:string
   δ(email) ⊑ Person
   ρ(email) ⊑ xsd:string
   - 2.1. ResearchAssistant ⊑ Person
     ResearchAssistant ⊑ ∃worksFor.ResearchGroup
   - 2.2. TennisFan ≡ Person ⊓ ∃isCrazyAbout.TennisClass
     In DL-Lite: TennisFan ⊑ Person; TennisFan ⊑ ∃isCrazyAbout.TennisClass
   - 2.3. BasketBallLover ≡ Person ⊓ ∃like.BasketBallClass
     In DL-Lite: BasketBallLover ⊑ Person ; BasketBallLover ⊑ ∃like.BasketBallClass
   - 2.4. PeopleWithHobby ≡ Person ⊓ ≥ 1 like
     In DL-Lite: PeopleWithHobby ⊑ Person ; PeopleWithHobby ⊑ ∃like
   - 2.5. SwimmingFan ≡ Person ⊓ ∃isCrazyAbout.SwimmingClass
     In DL-Lite: come per 2.2
   - 2.6. Employee ≡ Person ⊓ ∃worksFor.Organization
     In DL-Lite: come per 2.2
     - 2.6.1. Faculty ⊑ Employee
       - 2.6.1.1. Professor ⊑ Faculty

2.6.1.1.1. Dean ≡ ∃isHeadOf.College

In DL-Lite: Dean ⊑∃isHeadOf.College (il contrario no!)

Dean ⊑ Professor

2.6.1.1.2. VisitingProfessor ⊑ Professor

2.6.1.1.3. AssistantProfessor ⊑ Professor

2.6.1.1.4. AssociateProfessor ⊑ Professor

2.6.1.1.5. FullProfessor ⊑ Professor

2.6.1.2. Lecturer ⊑ Faculty

2.6.1.3. PostDoc ⊑ Faculty

2.6.2. SupportingStaff ⊑ Employee

2.6.2.1. SystemStaff ⊑ SupportingStaff

2.6.2.2. ClericalStaff ⊑ SupportingStaff

2.7. SportsLover ≡ Person ⊓ ∃like.Sports

In DL-Lite: come per 2.2

2.8. BasketBallFan ≡ Person ⊓ ∃isCrazyAbout.BasketBallClass

In DL-Lite: come per 2.2 (i prossimi casi equivalenti come traduzione a questo verranno ignorati)

2.9. TeachingAssistant ≡ Person ⊓ ∃teachingAssistantOf.Course

2.10. SwimmingLover ≡ Person ⊓ ∃like.SwimmingClass

2.11. Student ≡ Person ⊓ ∃isStudentOf.Organization

2.11.1. ScienceStudent ≡ Student ⊓ ∃hasMajor.Science

2.11.2. UndergraduateStudent ⊑ Student

2.11.3. NonScienceStudent ≡ Student ⊓ ¬∃hasMajor.Science

In DL-Lite: NonScienceStudent ⊑ Student ; NonScienceStudent ⊑ ¬ScienceStudent

2.12. BaseballLover ≡ Person ⊓ ∃like.BaseballClass

2.13. SportsFan ≡ Person ⊓ ∃isCrazyAbout.Sports

2.14. Director ≡ Person ⊓ ∃isHeadOf.Program

2.15. BaseballFan ≡ Person ⊓ ∃isCrazyAbout.BaseballClass

2.16. Chair ≡ Person ⊓ ∃isHeadOf.Department

Chair ⊑ Professor

3. Interest ⊑ Thing

3.1. Music ⊑ Interest

3.2. Sports ⊑ Interest

3.2.1. SwimmingClass ⊑ Sports

3.2.2. BaseballClass ⊑ Sports

3.2.3. BasketBallClass ⊑ Sports

3.2.4. TennisClass ⊑ Sports

4. Man ⊑ Thing

Man ⊑ ¬Woman

5. Woman ⊑ Thing

6. Schedule ⊑ Thing

7. PeopleWithManyHobbies ⊑ Thing
   PeopleWithManyHobbies ≡ ≥ 3 like
   In DL-Lite: possiamo ignorare tale vincolo (il rilassamento minimo corrisponde a PeopleWithHobby)

8. Work ⊑ Thing
   8.1. Course ⊑ Work
      8.1.1. GraduateCourse ⊑ Course
   8.2. Research ⊑ Work

9. Organization ⊑ Thing
   9.1. Program ⊑ Organization
   9.2. Department ⊑ Organization
   9.3. Institute ⊑ Organization
   9.4. University ⊑ Organization
   9.5. College ⊑ Organization
      9.5.1. WomanCollege ≡ College ⊓ ∀hasStudent.UndergraduateStudent ⊓ ∀hasStudent. ¬Man
         In DL-Lite: il rilassamento minimo corrisponde a WomanCollege ⊑ College
   9.6. ResearchGroup ⊑ Organization

10. AcademicSubject ⊑ Thing
   10.1. FineArts ⊑ AcademicSubject
      10.1.1. Latin_ArtsClass ⊑ FineArts
      10.1.2. Performing_ArtsClass ⊑ FineArts
      10.1.3. Asian_ArtsClass ⊑ FineArts
      10.1.4. Media_Arts_And_Sciences_ArtsClass ⊑ FineArts
      10.1.5. Theatre_and_DanceClass ⊑ FineArts
      10.1.6. MusicClass ⊑ FineArts
      10.1.7. Modern_ArtsClass ⊑ FineArts
      10.1.8. Medieval_ArtsClass ⊑ FineArts
      10.1.9. Drama_ArtsClass ⊑ FineArts
      10.1.10. ArchitectureClass ⊑ FineArts
   10.2. Science ⊑ AcademicSubject
      10.2.1. ChemistryClass ⊑ Science
      10.2.2. StatisticsClass ⊑ Science
      10.2.3. Material_ScienceClass ⊑ Science
      10.2.4. MathematicsClass ⊑ Science
      10.2.5. BiologyClass ⊑ Science
      10.2.6. AstronomyClass ⊑ Science
      10.2.7. GeosciencesClass ⊑ Science
      10.2.8. Marine_ScienceClass ⊑ Science
      10.2.9. PhysicsClass ⊑ Science
      10.2.10. Computer_ScienceClass ⊑ Science
   10.3. HumanitiesAndSocial ⊑ AcademicSubject
      10.3.1. EnglishClass ⊑ HumanitiesAndSocial
      10.3.2. AnthropologyClass ⊑ HumanitiesAndSocial
      10.3.3. PhilosophyClass ⊑ HumanitiesAndSocial

10.3.4. EconomicsClass ⊑ HumanitiesAndSocial
10.3.5. PsycologyClass ⊑ HumanitiesAndSocial
10.3.6. Modern_LanguagesClass ⊑ HumanitiesAndSocial
10.3.7. HistoryClass ⊑ HumanitiesAndSocial
10.3.8. HumanitiesClass ⊑ HumanitiesAndSocial
10.3.9. ReligionsClass ⊑ HumanitiesAndSocial
10.3.10. LinguisticsClass ⊑ HumanitiesAndSocial

10.4. Engineering ⊑ AcademicSubject
10.4.1. Civil_Engineering ⊑ Engineering
10.4.2. Computer_Engineering ⊑ Engineering
10.4.3. Material_Science_Engineering ⊑ Engineering
10.4.4. Mechanical_Engineering ⊑ Engineering
10.4.5. Electrical_Engineering ⊑ Engineering
10.4.6. Aeronautical_Engineering ⊑ Engineering
10.4.7. Petroleuml_Engineering ⊑ Engineering
10.4.8. Chemical_Engineering ⊑ Engineering
10.4.9. Biomedical_Engineering ⊑ Engineering
10.4.10. Industry_Engineering ⊑ Engineering

11. LeisureStudent ⊑ Thing
    LeisureStudent ≡ ≤3 takesCourse
    In DL-Lite: il rilassamento minimo corrisponde a LeisureStudent ⊑ Student

12. GraduateStudent ⊑ Thing
    GraduateStudent ≡ ≥1 takesCourse ⊓ ∀takesCourse.GraduateCourse
    In Dl-Lite: il rilassamento minimo coincide con l'asserzione GraduateStudent ⊑ takesCourse

Ruoli

Alcune considerazioni sull'adattamento dei ruoli:

1. La transitività e la simmetria presenti talvolta nella definizione di un ruolo verranno ignorate: sarà cura di chi effettua la query porla in modo da cogliere dette proprietà;
2. Nella TBox sono anche presenti ruoli che sono definiti come inversi di un altro ruolo: un esempio è isMemberOf – hasMember. Per ogni coppia di ruoli inversi <ruolo1> - <ruolo2> verranno prodotte due inclusion assertions: <ruolo1> ⊑ <ruolo2>$^-$ e <ruolo2>$^-$ ⊑ <ruolo1>. Una conseguenza di questa traduzione è che va eliminato ogni eventuale vincolo di funzionalità (diretto o inverso) presente su tali ruoli.

13. ∃teachingAssistantOf ⊑ TeachingAssistant
    ∃ teachingAssistantOf$^-$ ⊑ Course
14. isCrazyAbout ⊑ like
15. ∃takesCourse ⊑ Student
16. ∃softwareDocumentation ⊑ Software
    ∃softwareDocumentation$^-$ ⊑ Publication
17. ∃researchProject ⊑ ResearchGroup
    ∃researchProject$^-$ ⊑ Research

18. $\exists isAdvisedBy \sqsubseteq Person$
    $\exists isAdvisedBy^{-} \sqsubseteq Professor$
19. $\exists publicationDate \sqsubseteq Publication$
20. $\exists tenured \sqsubseteq Professor$
21. $\exists isAffiliateOf \sqsubseteq Organization$
    $\exists isAffiliateOf^{-} \sqsubseteq Person$
22. $\exists isAffiliatedOrganizationOf \sqsubseteq Organization$
    $\exists isAffiliatedOrganizationOf^{-} \sqsubseteq Organization$
23. $\exists publicationAuthor \sqsubseteq Publication$
    $\exists publicationAuthor^{-} \sqsubseteq Person$
24. $love \sqsubseteq like$
    $like \sqsubseteq love$
25. $\exists softwareVersion \sqsubseteq Software$
26. $\exists orgPublication \sqsubseteq Organization$
    $\exists orgPublication^{-} \sqsubseteq Publication$
27. $\exists listedCourse \sqsubseteq Schedule$
    $\exists listedCourse^{-} \sqsubseteq Course$
28. $\exists publicationResearch \sqsubseteq Publication$
    $\exists publicationResearch^{-} \sqsubseteq Research$
29. $\exists subOrganizationOf \sqsubseteq Organization$
    $\exists subOrganizationOf^{-} \sqsubseteq Organization$
30. $\exists hasSameHomeTownWith \sqsubseteq Person$
    $\exists hasSameHomeTownWith^{-} \sqsubseteq Person$
31. $\exists isFriendOf \sqsubseteq Person$
    $\exists isFriendOf^{-} \sqsubseteq Person$
32. $\exists hasDegreeFrom \sqsubseteq Person$
    $\exists hasDegreeFrom^{-} \sqsubseteq University$
    $hasUndergraduateDegreeFrom \sqsubseteq hasDegreeFrom$
    $hasMasterDegreeFrom \sqsubseteq hasDegreeFrom$
    $hasDoctoralDegreeFrom \sqsubseteq hasDegreeFrom$
33. $hasAlumnus \sqsubseteq hasDegreeFrom^{-}$
    $hasDegreeFrom^{-} \sqsubseteq hasAlumnus$
    $\exists hasAlumnus \sqsubseteq University$
    $\exists hasAlumnus^{-} \sqsubseteq Person$
34. $\exists hasMajor^{-} \sqsubseteq AcademicSubject$
35. $\exists hasMember \sqsubseteq Organization$
    $\exists hasMember^{-} \sqsubseteq Person$
36. $hasStudent \sqsubseteq hasMember$
    $\exists hasStudent^{-} \sqsubseteq Student$
37. $isMemberOf \sqsubseteq hasMember^{-}$
    $hasMember^{-} \sqsubseteq isMemberOf$
38. $isStudentOf \sqsubseteq hasStudent^{-}$
    $hasStudent^{-} \sqsubseteq isStudentOf$
    $\exists isStudentOf \sqsubseteq Student$
    $\exists isStudentOf^{-} \sqsubseteq Organization$
    $isStudentOf \sqsubseteq isMemberOf$

39. enrolIn ⊑ isStudentOf
    ∃enrolIn ⁻ ⊑ Department
40. worksFor ⊑ isMemberOf
41. isHeadOf ⊑ worksFor
42. ∃isTaughtBy ⊑ Course
    ∃isTaughtBy ⁻ ⊑ Faculty
43. teacherOf ⊑ isTaughtBy ⁻
    isTaughtBy ⁻ ⊑ teacherOf
    ∃teacherOf ⊑ Faculty
    ∃teacherOf ⁻ ⊑ Course

Ruoli funzionali

44. (func isHeadOf ⁻) (da eliminare: vedere considerazione 2)
45. (func isTaughtBy) (da eliminare: vedere considerazione 2)

2.3. Le ABoxes

Per le ABox non è necessario alcun adattamento da OWL-DL a *DL-Lite*. Pertanto, occorre soltanto un processo di traduzione automatica che trasformi le ABox in formato OWL in ABox in formato XML, senza alcuna perdita di informazione, e coerenti con le DTD definite in QuOnto.
Vediamo un esempio di traduzione. Consideriamo il seguente frammento, preso da uno dei file .owl delle abox:

```
<benchmark-dl:Department rdf:about="http://www.Department5.University0.edu">
        <benchmark-dl:name>Department5</benchmark-dl:name>
        <benchmark-dl:subOrganizationOf rdf:resource="http://www.College2.University0.edu" />
</benchmark-dl:Department>
```

Tutte le asserzioni presenti nelle ABox sono strutturate come sopra: ogni file è composto da vari blocchi, che chiameremo nodi, ed in ogni nodo sono presenti tutte le proprietà/caratteristiche di un certo individuo. In particolare, il valore dell'attributo di tali nodi (ovvero il valore di `rdf:about`) sarà il soggetto di tutte le proprietà che saranno presenti come figli di tale nodo.

Dal frammento si evincono le seguenti asserzioni:

1. *http://www.Department5.University0.edu* è un'istanza del concetto *Department* (concept membership assertion)
2. *http://www.Department5.University0.edu* ha nome *Department5* (attribute concept membership assertion)
3. *http://www.Department5.University0.edu* è una sotto-organizzazione di *http://www.College2.University0.edu* (role membership assertion)

che verrà tradotto, in XML, in questo modo:

1. <CMembership atomicC="Department" object="http://www.Department5.University0.edu"/>
2. <CAMembership atomicCA="name" object="object="http://www.Department5.University0.edu" value="Department5"/>
3. <RMembership atomicR="subOrganizationOf" rightObject="http://www.College2.University0.edu" leftObject="http://www.Department5.University0.edu" />

Da notare come nel file xml il soggetto del nodo relativo al frammento in esame viene ripetuto per tutte le proprietà che lo coinvolgono, determinando, in tal modo, un aumento di spazio delle abox risultanti. Più precisamente, per ogni nodo contenente *n* asserzioni, vengono prodotte *1+n* membership assertions, dove le *n* membership assertions hanno tutte lo stesso soggetto della prima (concept) membership assertion.

Il tool che effettua la trasformazione di tali abox funziona in questo modo:

1. Con il programma "Directory Lister" (disponibile gratuitamente all'indirizzo http://freeware.prv.pl/) viene prodotto un file txt contenente tutti i file delle directory che contengono i file owl di interesse per la produzione della abox, con il vincolo che il primo file presente nel listato sia la TBox. Ad esempio selezionando la cartella "1", verrà prodotto un file txt contenente riga per riga tutti i file contenuti nella directory "1", con il chiaro intento di creare l'abox relativa alla singola università;
2. Il tool sviluppato chiede all'utente di indicare il file txt suddetto come input e di indicare dove voler salvare il file xml che rappresenterà l'abox per QuOnto;
3. Il tool scandisce il file txt filtrando solo i file con estensione .owl, mandandoli uno per volta al convertitore;
4. Il convertitore esamina il file owl attraverso un algoritmo di visita ricorsivo (che sfrutta DOM) e lancia le opportune membership assertions on-the-fly;
5. L'effettivo processo di scrittura su file xml delle membership assertions viene effettuato, sempre on-the-fly, attraverso SAX (contrariamente a DOM, SAX processa la scrittura immediatamente, e non aspetta la fine del documento, circostanza che porterebbe inevitabilmente al lancio di un eccezione per memoria heap insufficiente).

In media, per convertire un file .owl della dimensione di 1,2 MB occorrono circa 0,7 secondi (Environment: Intel Centrino @ 1,8 Ghz , Java SDK con 1000MB di memoria heap)

3. Le queries

Nonostante la TBox dell'IBM sia più ricca dalla versione originale, sono state prese in considerazione tutte le 14 queries fornite all'interno della suite di benchmarking della LUBM, in quanto rappresentano uno standard di confronto con gli altri reasoners.

Di seguito si forniscono le suddette queries nel linguaggio del calcolo relazionale:

1. {x | GraduateStudent(x) AND takesCourse(x,"Dep0.Univ0/GraduateCourse0")}
2. {x,y,z | GraduateStudent(x) AND University(y) AND Department(z) AND subOrganizationOf(z,y) AND memberOf(x,z) AND undergraduateDegreeFrom(x,y)}
   (query con pattern triangolare)
3. {x | Publication(x) AND publicationAuthor(x,"Dep0.Univ0/AssistantProfessor0")}
   (query simile alla 1 ma con un'ampia gerarchia relativa a Publication)
4. {x,y1,y2,y3 | Professor(x) AND worksFor(x,"Dep0.Univ0") AND name(x,y1) AND emailAddress(x,y2) AND telephone(x,y3)}
   (query con 3 attributi di concetto, ampia gerarchia per Professor)
5. {x | Person(x) AND memberOf(x,"Dep0.Univ0")}
   (Person e memberOf hanno una gerarchia molto ampia)
6. {x | Student(x)}
   (query con grande quantità in input)

7. {x,y | Student(x) AND Course(y) AND takesCourse(x,y) AND teacherOf("Dep0.Univ0/AssociateProfessor0",y)}
   (query simile alla precedente ma con selettività più alta)
8. {x,y,z | Student(x) AND Department(y) AND memberOf(x,y) AND subOrganizationOf(y,"Univ0") AND emailAddress(x,z)}
   (query ancora più complessa della 7)
9. {x,y,z | Student(x) AND Faculty(y) AND Course(z) AND advisor(x,y) AND teacherOf(y,z) AND takesCourse(x,z)}
   (query con pattern triangolare)
10. {x | Student(x) AND takesCourse(x,"Dep0.Univ0/GraduateCourse0")}
11. {x | ResearchGroup(x) AND subOrganizationOf(x,"Univ0")}
12. {x,y | Chair(x) AND Department(y) AND worksFor(x,y) AND subOrganizationOf(y,"Univ0")}
13. {x | Person(x) AND hasAlumnus("Univ0",x)}
    (query di verifica per relazioni inverse)
14. {x | UndergraduateStudent(x)}
    (è la query più semplice: grande quantità in input, bassa selettività e assenza di gerarchia)

Occorre precisare che le precedenti queries non tengono conto del fatto che nella TBox di QuOnto non vengono tradotte le proprietà di simmetria (presente in isFriendOf e hasSameHomeTownWith) e di transitività (presente in hasSameHomeTownWith e subOrganizationOf). Pertanto, le queries che fanno uso di ruoli simmetrici/transitivi andranno leggermente modificate. Ad esempio, nelle ABoxes i dipartimenti non vengono mai asseriti come sottoorganizzazioni di un'università in modo esplicito: si dice che un dipartimento è sottoorganizzazione di un college e che un college è sottoorganizzazione di un'università.

4. Risultati dei test[1]

Prima di esporre e commentare i risultati dei test, occorre dare una breve panoramica delle fasi seguite da QuOnto prima di poter rispondere ad una query.

1. Il sistema effettua dapprima il parsing della TBox e della ABox, in modo separato;
2. La Abox viene memorizzata in un DBMS relazionale, nella fattispecie MySql, mappando le relazioni unarie in tabelle aventi lo stesso nome della relazione unaria e con un singolo attributo, "term", e le relazioni binarie in tabelle aventi anch'esse lo stesso nome della relazione binaria e con due attributi, "term1" e "term2";
3. Fase opzionale ma consigliabile, e a cura dell'utente: vengono creati gli indici per le tabelle appena create, in modo da ottenere il massimo dal query answering;
4. A questo punto è possibile operare sul sistema (chiedendo ad esempio un controllo di consistenza dell'abox con la tbox oppure delle queries).

Le prime 3 fasi vengono eseguite *off-line*. Di queste, la più onerosa, in termini di tempo, è la 2; a livello di scalabilità, invece, la fase determinante è la 1: a seconda della tecnologia usata per effettuare il parsing dei file xml, la quantità di memoria centrale utilizzata può variare di molto.

Sono stati inoltre presi in considerazione anche altri due software che effettuano reasoning su ontologie: RacerPro 1.9 e Pellet 1.5. Entrambi, a differenza di QuOnto, non poggiano su un DBMS, ma lavorano caricando i dati direttamente ed unicamente in memoria centrale. Secondo quanto indicato nella documentazione ufficiale, Pellet combina due algoritmi che sono uno sound and complete per OWL-DL

[1] Test Environment: Intel Centrino @ 1,8 Ghz, OS: Windows XP Pro, Java SDK 1.5 con 1000 MB di max heap memory

senza nominals (ovvero SHIN(D), lo stesso linguaggio su cui si basa SHER), l'altro sound and complete per OWL-DL senza proprietà inverse (ovvero SHON(D)), ottenendo una tecnica di ragionamento che è sound ma incomplete rispetto a tutto OWL-DL. RacerPro è un sistema di rappresentazione della conoscenza che implementa il calcolo basato su tableau altamente ottimizzato per una logica descrittiva molto espressiva, chiamata *ACLQHIR+*, anche nota come *SHIQ.* Oltre ad essere un sistema di logica descrittiva, è anche un motore di ragionamento del web semantico. E' in grado di processare sia documenti OWL-Lite sia documenti OWL-DL con approssimazione per i nominali.
A parità di TBox e di queries, è stato possibile analizzare il comportamento dei tre sistemi, in termini di scalabilità e tempi di risposta.

4.1. Fase di start-up

Nella figura 2 possiamo vedere a confronto i 3 sistemi nella fase di preparazione, o *start-up*, al query answering. Alcune considerazioni:

1. Come già indicato nell'articolo (1), Pellet e soprattutto RacerPro, hanno un tempo di caricamento dei dati che cresce esponenzialmente con la dimensione degli stessi;
2. RacerPro e Pellet riescono a caricare le prime due aboxes (ma solo per una di esse RacerPro riesce ad effettuare il controllo di consistenza), mentre QuOnto riesce a caricarle e a controllarle tutte;
3. QuOnto richiede un certo tempo di creazione degli indici di cui gli altri programmi non hanno bisogno, lavorando questi in memoria centrale (in particolare Pellet non prevede alcun tempo di ottimizzazione aggiuntivo alla fase di caricamento dei dati; per RacerPro, invece, le fasi di ottimizzazione corrispondo alle operazioni di REALIZE-ABOX e PREPARE-ABOX).

QuOnto

Tempo di caricamento su MySql

# Università	Tempo
1	5' 22''
5	20'
10	40'
30	3 h 26'

Tempo di creazione indici

# Università	Tempo
1	2' 34''
5	7' 28''
10	15' 29''
30	1 h 17'

Controllo di consistenza

# Università	Tempo 1	Tempo 2
1	2,2''	0,26''
5	2,9''	0,4''
10	7,5''	0,6''
30	33,2''	32,1''

RacerPro

Tempo di caricamento

# Università	Tempo
1	60''
5	16'
10	ERR
30	ERR

Tempo di ottimizzazione

# Università	Tempo
1	4-5''
5	4'
10	ERR
30	ERR

Controllo di consistenza

# Università	Tempo 1
1	2'
5	ERR
10	ERR
30	ERR

Pellet 1.5

# Università	Tempo
1	Time: 95734 ms (Loading: 29156 Species Validation: 63547 Consistency: 3031)
5	Time: 873656 ms (Loading: 153328 Species Validation: 394375 Consistency: 325953)
10	Out of memory exception (dati dell'IBM ci dicono che non bastano anche 4 GB di heap)
30	Out of memory exception (dati dell'IBM ci dicono che non bastano anche 4 GB di heap)

Figura 2: Fase di preparazione dei sistemi

Una nota è doverosa sul tempo di memorizzazione della abox su MySql. Un lettore che rimanga sorpreso dall'eccessiva quantità di tempo richiesta a tale scopo da QuOnto, cambierà sicuramente parere dopo aver osservato la figura 3, tratta dall'articolo (2). Il reasoner più performante (dal punto di vista del caricamento dei dati) basato su database, ovvero DLDB-OWL, per memorizzare il dataset più grande, che ha dimensione minore dell' abox dell'IBM con 30 università, impiega più di 12 ore!

Table 1. Load time and repository sizes

	Dataset	File #	Total Size (MB)	Triple #	Load Time (hh:mm:ss)	Repository Size (KB)
DLDB-OWL	LUBM (1, 0)	15	8.6	103,397	00:05:43	16,318
Sesame-DB					00:09:02	48,333
Sesame-Memory					00:00:13	-
OWLJessKB-P					03:16:12	-
OWLJessKB-NP					02:19:18	-
DLDB-OWL	LUBM (5, 0)	93	54.2	646,128	00:51:57	91,292
Sesame-DB					03:00:11	283,967
Sesame-Memory					00:01:53	-
OWLJessKB					-	-
DLDB-OWL	LUBM (10, 0)	189	110.6	1,316,993	01:54:41	184,680
Sesame-DB					12:27:50	574,554
Sesame-Memory					00:05:40	-
OWLJessKB					-	-
DLDB-OWL	LUBM (20, 0)	402	234.8	2,782,419	04:22:53	388,202
Sesame-DB					46:35:53	1,209,827
Sesame-Memory					-	-
OWLJessKB					-	-
DLDB-OWL	LUBM (50, 0)	999	583.6	6,890,933	12:37:57	958,956
Sesame-DB					-	-
Sesame-Memory					-	-
OWLJessKB					-	-

Figura 3: Tempi di caricamento di altri reasoners basati su database come QuOnto

4.2. Query answering

Nella figura 4 possiamo vedere come i risultati dei test effettuati su QuOnto confermano le aspettative: a) il sistema è in grado di gestire tutti i datasets (1,5,10 e 30 università) b) il query answering è tempo-polinomiale rispetto alla dimensione dei dati.

In QuOnto, i fattori più importanti che determinano la complessità di una query sono 2: il numero di join e il numero di union presenti nella query espansa. Prendiamo le queries 6 e 14, che sembrano abbastanza simili come struttura, eppure hanno dei tempi di risposta molto diversi fra loro. La differenza sta nel fatto che la 6, parlando di studenti, porta con se una gerarchia abbastanza ampia che determina la presenza di un certo numero di union nella query espansa che invece sono del tutto assenti nell'altra query espansa, che di fatto corrisponde ad un semplice instances retrieval senza gerarchia. Le altre 2 queries che

emergono, come tempi di risposta, sono la 2 e la 9, entrambe con pattern triangolare, molti join e presenza di gerarchie.
Per quanto riguarda gli altri due programmi, occorre sottolineare come nonostante RacerPro riesca a caricare le prime due ABoxes, esso riesce in realtà a rispondere soltanto a queries poste sulla prima Abox, mentre sulla seconda l'operazione termina negativamente anche forzando il sistema a non effettuare il controllo di consistenza. C'è da dire, d'altra parte, che quando Racer riesce a rispondere,lo fa più velocemente rispetto a QuOnto, dando lo stesso numero di risposte ad ogni query.

	Query 1	Query 2	Query 3	Query 4	Query 5	Query 6	Query 7
Racer	5167/32	359	31	31	156	2250	31
Quonto	4047/312	4156	297	2090	109	1969	531

	Query 8	Query 9	Query 10	Query 11	Query 12	Query 13	Query 14
Racer	4016	641	16	47	63	46	1782
Quonto	20859	1657	31	469	500	797	313

Tabella 2: confronto RacerPro-Quonto, singola run, tempi in ms

Solo la query 8, inizialmente, sembrava causare un errore in RacerPro (l'eccezione riguardava l'impossibilità nel creare un array di dimensione troppo grande). Provando la stessa query senza considerare l'attributo di concetto *emailAddress*, venivano fornite esattamente 13320 risposte (il numero esatto) in 420ms. Tuttavia, il problema non poteva essere nell'attributo di concetto, in quanto RacerPro riesce a rispondere correttamente alla query 4, che contempla ben 3 attributi di concetto. Il motivo sta nel fatto che anche se tale query non è quella con il maggior numero di risposte (la query 2 ne ha infatti 13570), è quella con il più alto numero di elementi nella risposta, intendendo con essi il numero delle risposte moltiplicato per il numero di variabili presenti nella target list della query. Di conseguenza, si è pensato che potesse essere soltanto un limite di RacerPorter, il client/interfaccia di RacerPro. L'ipotesi è stata confermata utilizzando come client JRacer (cui sono relativi i tempi indicati in tabella 2), una API per comuncare attraverso Java direttamente con il server di RacerPro. Attraverso l'uso di JRacer, è stato possibile osservare un comportamento piuttosto curioso quanto strano di RacerPro.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Racer 1° run 10	964	699	753	46	946	2476	231	4128	497	206	659	1040	265	10196
Racer 2° run 10	215	4331	212	40	165	3204	34	4621	498	205	51	1033	457	2184
Racer 3° run 10	221	1048	218	446	340	2615	32	6984	1078	217	51	456	43	1945
Quonto 1°run 10	378	1868	37	337	153	901	112	20026	1698	29	187	362	445	314
Quonto 2°run 10	396	103	35	400	39	445	62	178	354	6	120	282	98	81

Tabella 3: confronto QuOnto-RacerPro, run di 10, tempi medi in ms

Le prime 2 righe rappresentano 2 esecuzioni consecutive (a distanza di 10 secondi circa) del programma Java che effettua il test di RacerPro sulla run da 10,con il seguente pattern: q1,q2,…,q14,q1,q2,…,q14,etc.. .
La terza riga rappresenta lo stesso test ma con diverso pattern:q1,q1,….,q1,q2,q2,…,q2,…..,q14,q14,…,q14.
La quarta riga è una run di 10 effettuata su QuOnto secondo il primo pattern, mentre la quinta riga rappresenta quella effettuata seguendo il secondo pattern. Dalla tabella emergono i seguenti aspetti:

- rispetto alla run singola (presente nella tabella 2), le performance di RacerPro degradano quando si eseguono run multiple di queries sia con il primo sia con il secondo pattern: fissando una query, possiamo notare come dei 3 valori, quasi sempre uno sia nettamente peggiore degli altri due;

- dalle prime 2 righe, si nota come la stessa esecuzione con lo stesso pattern da parte di RacerPro porti a tempi medi di risposta talvolta piuttosto diversi, cosa che con QuOnto non avviene;
- la riga 4 ci dice che 1) QuOnto ha performance migliori su run multiple (si veda la run singola della tabella 2) 2) per 9 queries su 14 si comporta (talvolta molto) meglio delle run da 10 eseguite da RecerPro e tabulate nelle prime 2 righe;
- confrontando la 3° e la 5° riga, si nota come con il secondo pattern le queries a favore di QuOnto diventano 10.

Un'ultima considerazione sulla tabella 2, relativa al confronto dei 2 reasoner rispetto alla run singola. Nonostante siano state effettuate tutte le fasi di ottimizzazione prima di effettuare il query answering, RacerPro impiega un po' di tempo per rispondere alla prima query che gli viene fornita: ad esempio, se invece della query 1 si pone come prima query in input la query 2, impiega 63 secondi per rispondere, un tempo piuttosto alto, indice del fatto che evidentemente ci sono altre operazioni che svolge internamente. Similmente, QuOnto impiega sempre un certo tempo per effettuare la prima connessione a MySql e accedere allo schema desiderato di quel database: anche in questo caso, la prima query risente di questa inizializzazione. Nella tabella 2 possiamo notare che per la query 1 viene indicato anche un secondo valore, pari al tempo impiegato dal reasoner per rispondere a tale query, quando un'altra query viene posta per prima.

Nella tabella 4 possiamo invece confrontare Pellet, RacerPro e QuOnto relativamente alle aboxes con 1 e 5 università (1 università sola per RacerPro). I dati si riferiscono a una singola run di queries che, per quanto riguarda Pellet, sono state poste attraverso riga di comando. Alla fine dell'operazione, Pellet restituisce una descrizione dettagliata sui tempi impiegati per rispondere alle queries: è da questo resoconto che sono stati prelevati i valori riportati in tabella.

	Query 1	Query 2	Query 3	Query 4	Query 5	Query 6	Query 7
Pellet 1	16	616265	16	16	31	ERR	0
QuOnto 1	4047	4156	297	2090	109	1969	531
Racer 1	5157	359	31	31	156	2250	31
Pellet 5	16	ERR	16	15	16	ERR	15
Quonto 5	1297	8453	600	1250	219	8078	688

	Query 8	Query 9	Query 10	Query 11	Query 12	Query 13	Query 14
Pellet 1	33469	297	16	31	94	0	ERR
QuOnto 1	20859	1657	31	469	500	797	313
Racer 1	4016	641	16	47	63	46	1782
Pellet 5	38344	1078	0	31	94	0	ERR
Quonto 5	26750	11359	188	641	1453	1906	1234

Tabella 4: confronto QuOnto-RacerPro-Pellet, singola run, tempi in ms

La prima cosa da notare è che Pellet non riesce a rispondere alle queries 6 e 14 per la prima ABox, a cui si aggiunge la query 2 quando si considera la seconda ABox (errori tutti dovuti a memoria heap insufficiente). Per quanto riguarda la query 2, il comportamento può essere prevedibile, visto che il reasoner si trova in difficoltà a rispondere già alla stessa query sulla prima ABox. Le perplessità sono tutte rivolte alle altre 2 queries: la 14 è la più semplice di tutte (si tratta di instances retrieval senza gerarchia), e la 6 è molto simile (instances retreival con gerarchia). Il parametro che le accomuna è la bassa selettività. Occorre a questo

punto citare l'articolo (3): secondo quanto indicato, il test di Pellet[2] effettuato sulla LUBM (la versione originale dell'ontologia in esame in questo lavoro, basata su OWL-Lite) ha pessimi risultati, non riuscendo a rispondere ad alcuna delle 14 queries (anche in questo caso per memoria heap insufficiente). Riportiamo alcune righe dell'articolo, che sono sembrate particolarmente interessanti.

> ***"We were surprised by this result: the ontology is still Horn, so an ABox completion can be computed in advance and used as a cache for query answering. By analyzing a run of Pellet on lubm 1 in a debugger, we observed that the system performs disjunctive reasoning (i.e., it performs branch splits). Further investigation showed that this is due to absorption [9]—a well-known optimization technique used by all tableau reasoners. Namely, an axiom of the form C ⊑ D, where C is a complex concept,increases the amount of don't-know nondeterminism in a tableau because it yields a disjunction ¬C ⊔D in the label of each node. If possible, such an axiom is transformed into an equivalent definition axiom A ⊑ C′ (where A is an atomic concept), which can be handled in a deterministic way. The LUBM ontology contains several axioms that are equivalent to A ⊑ B⊓∃R.C and B⊓∃R.C ⊑ A. Now the latter axiom contains a complex concept on the left-hand side of ⊑, so it is absorbed into an equivalent axiom B ⊑ A⊔∀R.¬C.Whereas this is a definition axiom, it contains a disjunction on the right-hand side, and thus causes branch splits. This could perhaps be improved by extending the tableau calculus with an inference rule similar to hyperresolution. Namely, an axiom B ⊓ ∃R.C ⊑ A is equivalent to the clause B(x) ∧R(x, y) ∧ C(y) → A(x). In resolution, one can select the literals on the left-hand side of the implication, which allows the clause to "fire" only if all three literals can be resolved simultaneously. It remains to see whether this is possible in a tableau setting without affecting the correctness and the termination of the calculus.***

Per quanto riguarda i tempi, Pellet risponde sempre più rapidamente degli altri 2 reasoners (è leggermente più rapido di RacerPro) ad eccezione delle queries 2 e 8, con le quali si comporta molto peggio.

Come abbiamo visto finora, tutti e 3 i reasoners restituiscono le stesse identiche risposte ad ogni query. Si è pensato allora di formularne qualcuna particolarmente interessante in quanto, come indicato in (4), è noto che esistono contesti nei quali RacerPro non risponde correttamente a determinate queries, ignorando alcune risposte. Possibili query sono quelle con pattern triangolare e cardinalità della target list pari ad uno. Poiché nei test sono presenti 2 queries con pattern triangolare e cardinalità della target list pari a 3, queste sono state riproposte ognuna nelle 3 versioni con cardinalità della target list pari ad 1, ma senza successo: tutti e 3 i reasoners restituiscono ancora lo stesse risposte.
Una possibile query (che ha un pattern a pentagono) in cui si nota questa incompletezza nelle risposte è la seguente:

q15: {x | hasSameHomeTownWith(x,y) AND isMemberOf(y,z) AND hasMember(z,t) AND isCrazyAbout(t,w) AND isCrazyAbout(x,w)}

[2] Test effettuato su Intel processor @ 2 Ghz, OS: Windows Xp Pro, Java SDK 1.5 con 800 MB di memoria heap

Considerando la ABox con 1 università,

- QuOnto risponde in circa 37 secondi fornendo 94 risposte;
- RacerPro risponde in 29 secondi fornendo 89 risposte;
- Pellet risponde in 9,5 secondi fornendo 89 risposte.

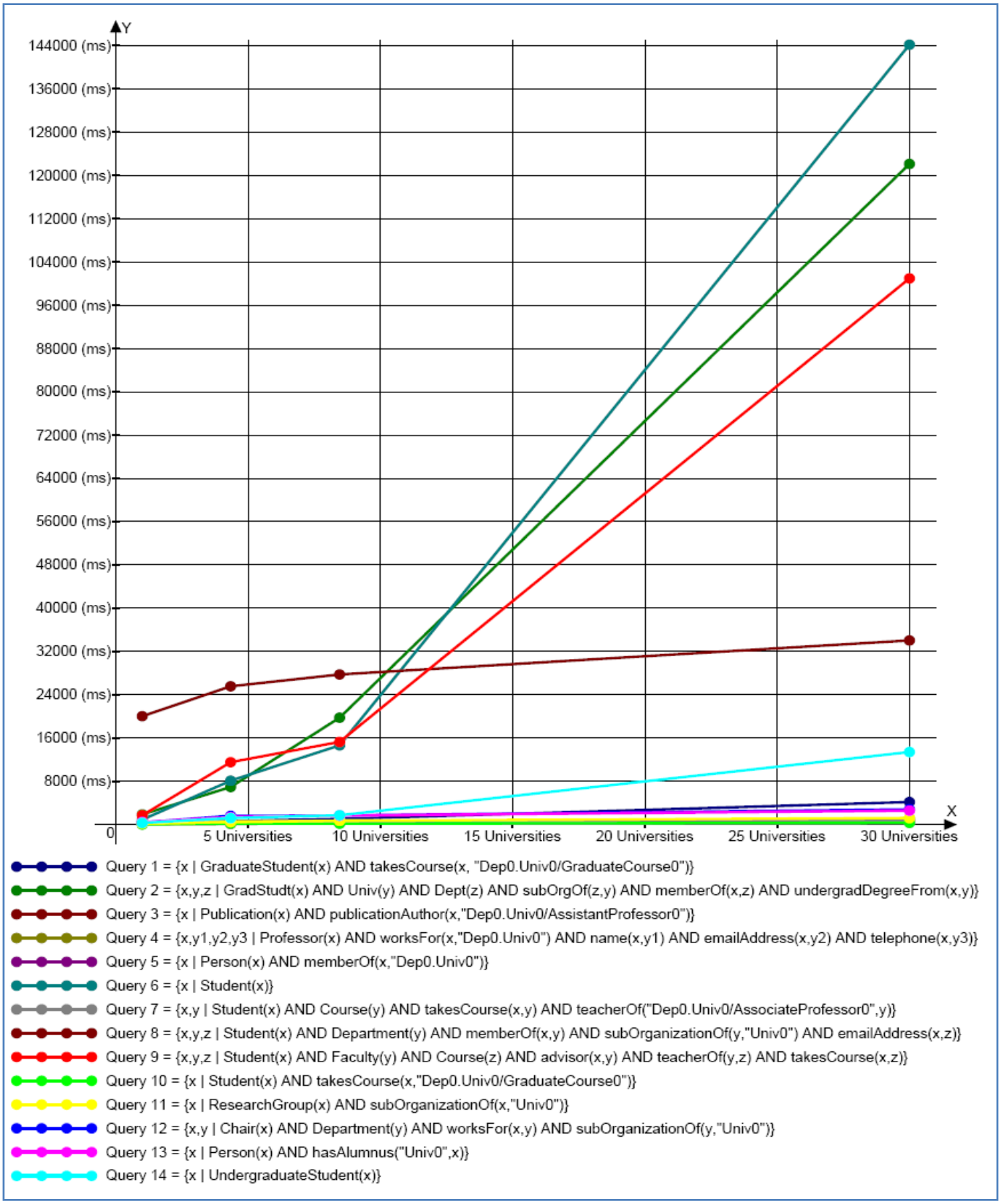

Figura 4: Il test su QuOnto

	1 università	5 università	10 università	30 università
# asserzioni	260790	1133194	2218773	7771143
rapporto	1	4.34	8.46	29,62

Numero risposte query, valide per QuOnto, Pellet e RacerPro

# Università	Query 1	Query 2	Query 3	Query 4	Query 5	Query 6	Query 7
1	12	103	7	22	739	13570	45
5	6	492	9	19	736	57017	40
10	6	879	9	14	732	110768	43
30	22	2761	10	15	739	354125	95

Query 8	Query 9	Query 10	Query 11	Query 12	Query 13	Query 14
13320	184	12	200	20	161	10280
13320	851	6	200	20	757	43367
13320	1640	6	200	20	1643	82158
13320	15623	22	200	20	4792	269191

Numero risposte query non espanse, valide per QuOnto

# Università	Query 1	Query 2	Query 3	Query 4	Query 5	Query 6	Query 7
1	12	0	0	0	0	0	0
5	6	0	0	0	0	0	0
10	6	0	1	0	0	0	0
30	22	0	1	0	0	0	0

Query 8	Query 9	Query 10	Query 11	Query 12	Query 13	Query 14
0	0	0	200	0	0	10280
0	0	0	200	0	0	43367
0	0	0	200	0	0	82158
0	0	0	200	0	0	269191

Tabella 5: numero di risposte alle queries

Bibliografia

1. **Timo Weithoner, Thorsten Liebig, Marko Luther, Sebastian Bohm, Friedrich von Henke, Olaf Noppens.** *Real-World Reasoning with OWL.* s.l. : Inst. of AI, Ulm University, Ulm, Germany, 2007.

2. **Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin.** *LUBM: A Benchmark for OWL Knowledge Base Systems.* s.l. : Computer Science & Engineering Department Lehigh University, 2005.

3. **Boris Motik, Ulrike Sattler.** *A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes.* University of Manchester : s.n., 2006.

4. **Alessio, De Gaetanis.** *Seminari di Ingegneria del Software: "Studio di RacerPro per fare delle interrogazioni in nRQL".* 2007.