



Progetto e sviluppo di un reasoner per alcune famiglie di logiche descrittive

Davide Eynard - mat. 624151

Relatore: Prof. Marco Colombetti

Correlatore: Ing. Mario Arrigoni Neri



Introduzione

- Le *logiche descrittive* (DL) sono una famiglia di formalismi utilizzati per la *rappresentazione della conoscenza*
- Negli ultimi anni l'interesse nei confronti delle DL è aumentato, insieme alle loro applicazioni



Introduzione

- Le *logiche descrittive* (DL) sono una famiglia di formalismi utilizzati per la *rappresentazione della conoscenza*
- Negli ultimi anni l'interesse nei confronti delle DL è aumentato, insieme alle loro applicazioni
 - Web semantico, Web Service, basi di dati, medicina...



Introduzione

- Le *logiche descrittive* (DL) sono una famiglia di formalismi utilizzati per la *rappresentazione della conoscenza*
- Negli ultimi anni l'interesse nei confronti delle DL è aumentato, insieme alle loro applicazioni
 - Web semantico, Web Service, basi di dati, medicina...
 - Servizi di *ragionamento*: possibilità di dedurre conoscenze *implicite* a partire da quelle *esplicitamente* contenute nella KB



Stato dell'arte

| | Renamed ABox and Concept Expression Reasoner (RACER) | cwm | DAMLJessKB | Euler proof mechanism | Java Theorem Prover (JTP) | OWL P | OpenCyc | Pellet | Surnia | Semantic Web Enabling Technologies for Jess (SweetJess) | TRIPLE | F-OWL |
|--------------------------------|--|---------------------|--------------------------------------|---------------------------------------|---|-----------------------|--------------------------------------|---------------------------------------|---------------------------------------|---|--------------------------------------|---------------------------------------|
| Language Support | OWL | RDF | DAML+OIL | RDF | DAML+OIL | OWL | DAML+OIL | OWL | OWL | DAML+OIL | DAML+OIL | OWL |
| Ontology Tool | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No | Yes | No | Yes |
| Data Tool | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No | Yes | No | Yes |
| Web Site Assessment | Medium | Medium | Medium | Medium | Medium | Low | Medium | Medium | Low | Medium | Medium | Medium |
| Installation Assessment | High | Medium | Low | Medium | High | Low | High | Low | Low | Low | Medium | Low |
| Documentation Assessment | Medium | Medium | Medium | Medium | Medium | Low | Medium | Medium | Low | Medium | Low | Medium |
| Open Source | No | Yes | Yes | Yes | Yes | Yes | Yes | No | Yes | No | No | Yes |
| DAML Funded | No | Yes | No | No | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes |
| Assessment Score / Skip Reason | 100 | 88 | Does not support OWL | Other | Does not support OWL | Other | Does not support OWL | Failed to install/run | Failed to install/run | Does not support OWL | Does not support OWL | Failed to install/run |



Stato dell'arte

- RACER
 - Progetto maturo, supporta OWL, DAML+OIL e DIG
 - Servizio di ragionamento da linea di comando
- FaCT
 - Supporta OWL Lite e DIG (tramite servlet Java)
 - FaCT ++ è scritto in C++ e ha licenza GPL
- Pellet
 - *Quasi* OWL DL, molto seguito
 - Scritto in Java, con licenza MIT

Obiettivi

+

- Supporto per OWL
 - OWL Lite: *SHIF*
 - OWL DL: *SHOIN*
- Supporto per DIG
- Algoritmi di tableaux
- GPL
- Linguaggio diffuso e multiplatforma

Obiettivi

+

- Supporto per OWL
 - OWL Lite: *SHIF*
 - OWL DL: *SHOIN*
- Supporto per DIG
- Algoritmi di tableaux
- GPL
- Linguaggio diffuso e multiplatforma
- **Java OWL DL Inference Engine =**



Obiettivi

+

- Supporto per OWL
 - OWL Lite: *SHIF*
 - OWL DL: *SHOIN*
- Supporto per DIG
- Algoritmi di tableaux
- GPL
- Linguaggio diffuso e multiplatforma
- **Java OWL DL Inference Engine = JODIE**

Obiettivi

+

- Supporto per OWL
 - OWL Lite: *SHIF*
 - OWL DL: *SHOIN*
- Supporto per DIG
- Algoritmi di tableaux
- GPL
- Linguaggio diffuso e multiplatforma
- **Java OWL DL Inference Engine = JOLIE**



Motivazioni

Perché creare *un altro* reasoner?

- Codice libero su cui sperimentare
- Reasoner come strumento didattico
- Arricchimento personale



Logiche Descrittive

- Concetti
 - *predicati unari*, ad es. PERSONA, DONNA, FEMMINA
- Ruoli
 - *relazioni binarie*, ad es. haFiglio, haGenitore
- Costruttori
 - utilizzati per creare termini complessi
 - $DONNA \equiv PERSONA \sqcap FEMMINA$
 - $MADRE \equiv DONNA \sqcap \exists \text{ haFiglio}$

Famiglie di DL

- \mathcal{AL}
 - A (concetto atomico), \top , \perp , $\neg A$, $C \sqcap D$, $\forall R.C$, $\exists R.\top$
 - la negazione può essere applicata solo ai concetti atomici
 - solo TOP (\top) può essere usato insieme al quantificatore esistenziale
- $+\mathcal{U}$
 - $C \sqcup D$
- $+\mathcal{E}$
 - $\exists R.C$
- $+\mathcal{C}$
 - $\neg C$

Famiglie di DL

- \mathcal{AL}
 - A (concetto atomico), \top , \perp , $\neg A$, $C \sqcap D$, $\forall R.C$, $\exists R.\top$
 - la negazione può essere applicata solo ai concetti atomici
 - solo TOP (\top) può essere usato insieme al quantificatore esistenziale
- $+\mathcal{U}$
 - $C \sqcup D$
- $+\mathcal{E}$
 - $\exists R.C$
- $+\mathcal{C}$
 - $\neg C$

$$\begin{aligned}C \sqcup D &\equiv \neg(\neg C \sqcap \neg D) \\ \exists R.C &\equiv \neg \forall R. \neg C \\ \Rightarrow \mathcal{ALUE} &\equiv \mathcal{ALC}\end{aligned}$$



Famiglie di DL

- $\mathcal{S} \equiv \mathcal{ALC}_{R^+}$
 - ruoli transitivi
- $+ \mathcal{H}$ (*role Hierarchy*)
 - relazioni di inclusione fra i ruoli
- $+ \mathcal{O}$ (*One of*)
 - definizione di termini per enumerazione
- $+ \mathcal{I}$ (*Inverse role*)
 - ruoli inversi
- $+ \mathcal{F}$ (*Functional role*)
 - ruoli funzionali
- $+ \mathcal{N}$ (*Number restrictions*)
 - cardinalità non qualificate
- $+ \mathcal{Q}$ (*Qualified number restrictions*)
 - cardinalità qualificate



Sussunzione e soddisfacibilità

+

- L'algoritmo di *sussunzione* consente di determinare le relazioni fra concetti e può essere usato per calcolare la *tassonomia* di una TBox
- L'algoritmo di *soddisfacibilità* verifica se un dato concetto potrà mai avere un'istanza
- In presenza dell'operatore di negazione, la sussunzione può essere ricondotta alla soddisfacibilità.
 - $C \sqsubseteq D$ se e solo se $C \sqcap \neg D$ è insoddisfacibile
 - C è soddisfacibile se e solo se non vale $C \sqsubseteq P \sqcap \neg P$, dove P è un nome di concetto



Gli algoritmi di Tableaux

+

- Specializzazione della tecnica dei tableaux per la logica predicativa del prim'ordine
- Nonostante la complessità (esponenziale!), all'atto pratico forniscono risultati in tempi accettabili
- È possibile applicarvi diverse ottimizzazioni



Gli algoritmi di Tableaux

- Ricevono in ingresso un concetto C e una gerarchia di ruoli \mathcal{R} e cercano di provare la soddisfacibilità di C rispetto a \mathcal{R} costruendone un modello
- Il modello è solitamente rappresentato da un *albero* (*completion tree*)
 - *Individuo* \rightarrow *Nodo* (label: Concetti)
 - *Ruolo* \rightarrow *Arco*
- L'albero viene popolato applicando delle *regole di espansione* ai nodi

Regole di espansione

+

regola AND: se 1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$ e

2. $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$

allora $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C_1, C_2\}$

regola OR: se 1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$ e

2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$

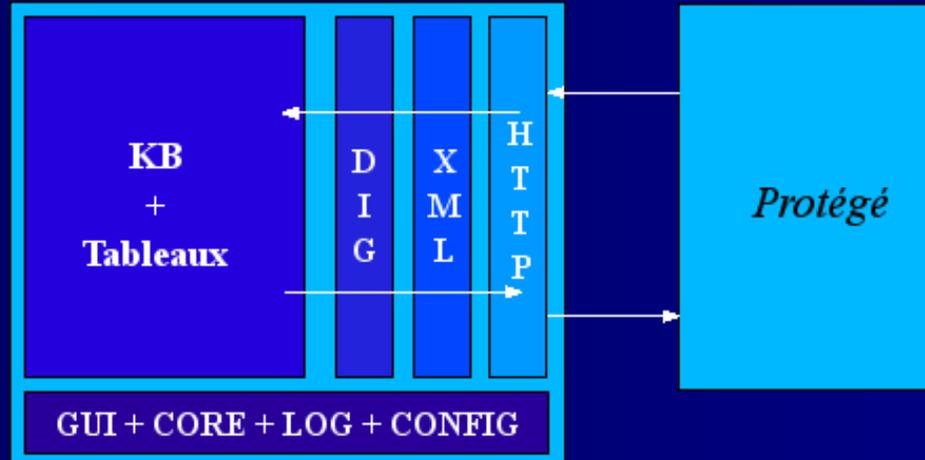
allora a. prova $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C_1\}$

se questo causa un clash ripristina il tableaux e

b. prova $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C_2\}$

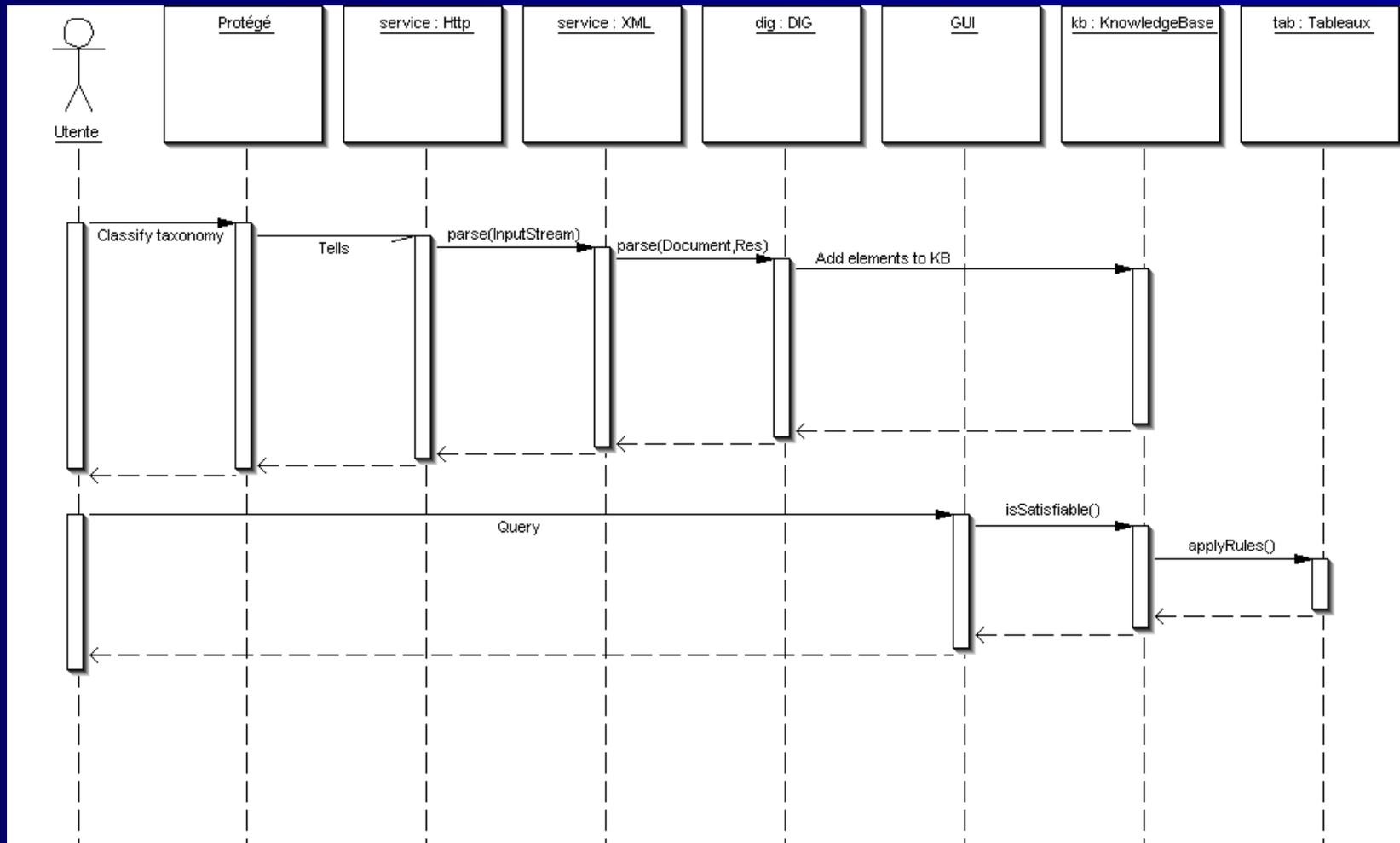
...

Struttura



- L'applicazione è composta da diversi moduli:
- HTTP
- XML + DIG
- Knowledgebase
- Tableaux
- GUI
- Core, Log, Config

HTTP, XML, DIG





KB, Tableaux

- KnowledgeBase
 - Interfaccia per il ragionamento
 - Calcolo dell'espressività e gestione delle strategie
 - TBox (Role, Concept), ABox (Individual)
 - Term
 - Negazione, semplificazione, normalizzazione, unfolding, NNF, equivalenza
- Tableaux
 - Completion tree (Node, Edge)
 - Assertion
 - toString(), toXML(), toDOT()



Core, Log, Config

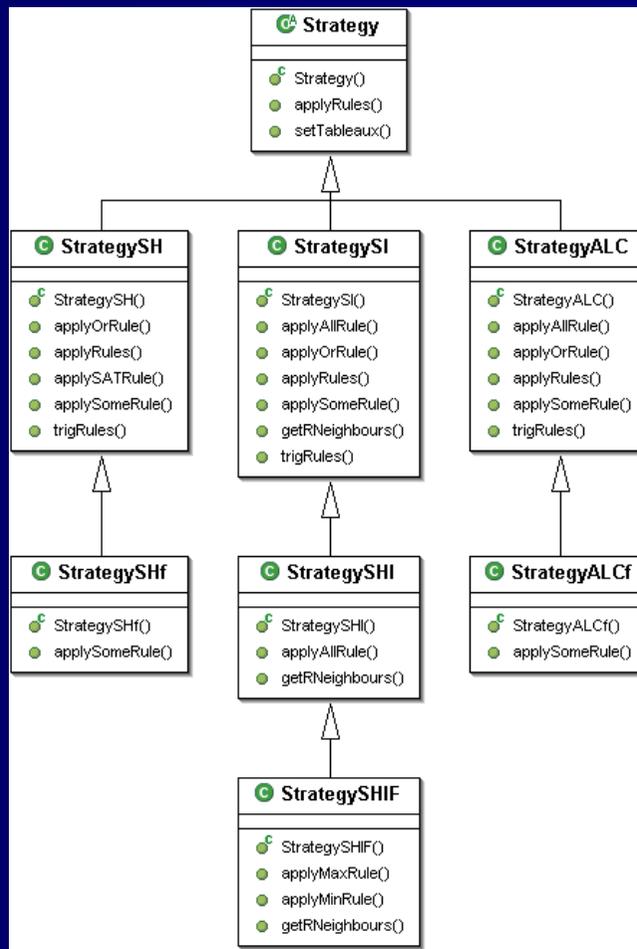
- Core
 - Punto di collegamento fra i moduli
 - HTTP, GUI, KB, LOG, Timer
- Log
 - Gestione *trasparente* dei messaggi
 - Hooking degli stream
 - Modalità GUI, CMD, OFF
- Config
 - Hardcoded, file xml, linea di comando
 - Profili dipendenti dal sistema operativo
 - Sezioni: base, log, output, strategies, path

Contenuti originali

```
<config>
  <profile name="common">
    <base>
      <debug>true</debug>
      <hasGUI>true</hasGUI>
      <hasGV>true</hasGV>
      <httpPort>8080</httpPort>
    </base>
    <log><!-- 0=OFF, 1=CMD, 2=GUI -->
      <main>2</main>
      <dig>2</dig>
      <res>2</res>
      <kb>0</kb>
      <tab>1</tab>
    </log>
    <output>
      <tableauxXML>tableaux.xml</tableauxXML>
      <dotfilename>mala.dot</dotfilename>
      <imageformat>png</imageformat>
      <imgfilename>mala.png</imgfilename>
    </output>
    <strategies>
      <ALL>SH</ALL>
      <ALC>ALC</ALC>
      <ALCF>ALCf</ALCF>
    </strategies>
  </profile>
```

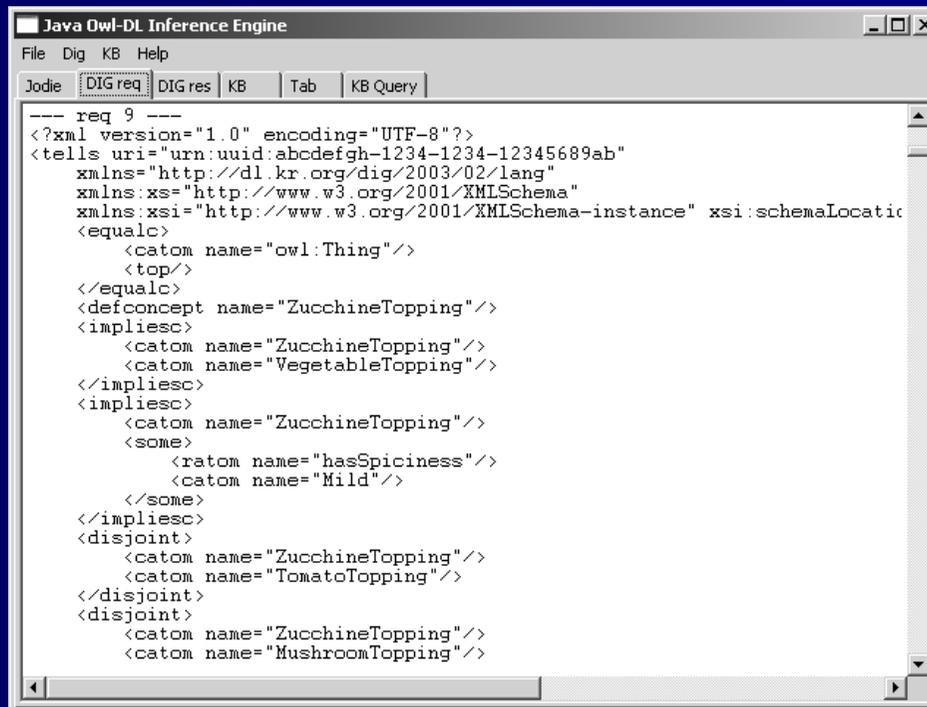
- Modulare e personalizzabile

Contenuti originali



- Modulare e personalizzabile
- Gestione strategie "a plugin" in base a configurazione ed espressività

Contenuti originali



```
--- req 9 ---
<?xml version="1.0" encoding="UTF-8"?>
<tells uri="urn:uuid:abcdefgh-1234-1234-12345689ab"
  xmlns="http://dl.kr.org/dig/2003/02/lang"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
  <equalc>
    <catom name="owl:Thing"/>
    <top/>
  </equalc>
  <defconcept name="ZucchineTopping"/>
  <impliesc>
    <catom name="ZucchineTopping"/>
    <catom name="VegetableTopping"/>
  </impliesc>
  <impliesc>
    <catom name="ZucchineTopping"/>
    <some>
      <ratom name="hasSpiciness"/>
      <catom name="Mild"/>
    </some>
  </impliesc>
  <disjoint>
    <catom name="ZucchineTopping"/>
    <catom name="TomatoTopping"/>
  </disjoint>
  <disjoint>
    <catom name="ZucchineTopping"/>
    <catom name="MushroomTopping"/>
  </disjoint>
```

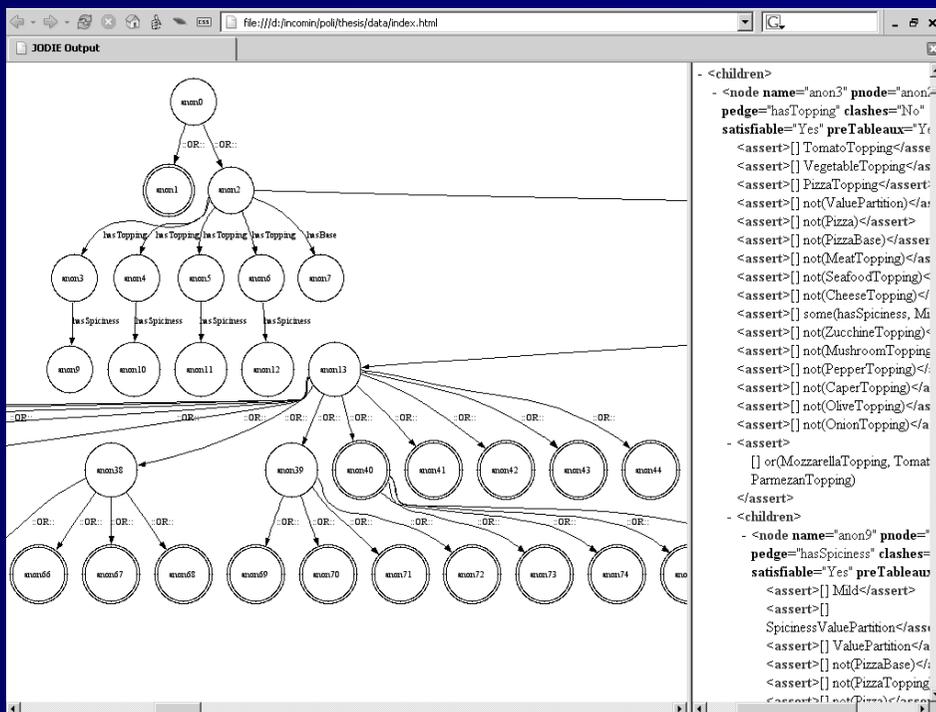
- Modulare e personalizzabile
- Gestione strategie "a plugin" in base a configurazione ed espressività
- GUI e Log



Contenuti originali

- Programmatori
 - Codice modulare e pulito
- Studiosi
 - Implementazione *alla lettera*
 - Log per seguire protocolli e algoritmi
- Docenti
 - Rappresentazione grafica
- Modulare e personalizzabile
- Gestione strategie "a plugin" in base a configurazione ed espressività
- GUI e Log
- Impostazione didattica

Contenuti originali



- Modulare e personalizzabile
- Gestione strategie "a plugin" in base a configurazione ed espressività
- GUI e Log
- Impostazione didattica
- Rappresentazione grafica

Test



- Strumenti
 - Esterni: Protégé, Oiled, RACER, GraphViz
 - Interni: timer, log, toDOT()
 - Basi di conoscenze: pizza.owl, OpenGALEN, ad hoc
- Tipologie di test
 - Funzionamento: comunicazioni, test di soddisfazione
 - Velocità: non comparativi, sulle singole query, scelte implementative
 - Ottimizzazione: singole funzioni, mutua interazione



Risultati

- **Funzionamento**
 - Comunicazione efficace con gli editor di ontologie
 - I test di soddisfacibilità hanno avuto l'esito previsto
 - Le immagini generate da GraphViz hanno confermato i risultati
- **Velocità**
 - Individuati colli di bottiglia e studiate alternative
 - Casi: parser DOM vs. JavaBean, Log
- **Ottimizzazione**
 - Simplify è onerosa per la CPU, ma vantaggiosa per la memoria
 - Lazy unfolding e BCP sono le migliori, rispettivamente, in termini di tempo e di spazio
 - Il caching sull'unfolding offre vantaggi minimi

Conclusioni

- **Traguardi raggiunti:**
 - Supporto per OWL Lite (*SHIF*)
 - Supporto per lo standard DIG
 - Implementazione degli algoritmi di Tableaux ottimizzati
 - JODIE è multiplatforma e distribuito con licenza GPL/MIT
- **Sviluppi futuri:**
 - Ottimizzazioni
 - Supporto per ABox
 - Logiche più espressive
 - Strumenti esterni al motore (GUI, DIG, Config)

Fine



Grazie per l'attenzione.
Domande?

Perché *reasoner*?

- *Ragionatore*: poco diffuso, traduzione letterale
- *Motore inferenziale*: più diffuso, ma impreciso
 - Inferenza = induzione + deduzione
 - Induzione: dal particolare al generale
 - Deduzione: dal generale al particolare
 - I *reasoner* fanno uso della sola deduzione
- *Motore deduttivo*: inutilizzato



Sintassi e semantica delle DL

- La *semantica* di ruoli e concetti complessi viene definita in base a una *interpretazione* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$
 - Il dominio $\Delta^{\mathcal{I}}$ di \mathcal{I} è un insieme non vuoto di individui
 - La funzione di interpretazione $\cdot^{\mathcal{I}}$ mette in corrispondenza ogni concetto con un sottoinsieme di $\Delta^{\mathcal{I}}$ e ogni ruolo con un sottoinsieme di $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

Sintassi e semantica delle DL

| Nome del costrutto | Sintassi | Semantica | |
|-----------------------|----------------------|---|---------------|
| concetto atomico | A | $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ | \mathcal{S} |
| ruolo atomico | R | $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ | |
| ruolo transitivo | $R \in \mathbf{R}_+$ | $R^{\mathcal{I}} = (R^{\mathcal{I}})^+$ | |
| congiunzione | $C \sqcap D$ | $C^{\mathcal{I}} \cap D^{\mathcal{I}}$ | |
| disgiunzione | $C \sqcup D$ | $C^{\mathcal{I}} \cup D^{\mathcal{I}}$ | |
| negazione | $\neg C$ | $\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ | |
| quantif. esistenziale | $\exists R.C$ | $\{x \mid \exists y. \langle x, y \rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$ | |
| quantif. universale | $\forall R.C$ | $\{x \mid \forall y. \langle x, y \rangle \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}$ | |

Sintassi e semantica delle DL

| Nome del costrutto | Sintassi | Semantica | |
|--------------------------------|----------------------------|--|---------------|
| gerarchia di ruoli | $R \sqsubseteq S$ | $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ | \mathcal{H} |
| ruolo inverso | R^{-} | $\{\langle x, y \rangle \mid \langle y, x \rangle \in R^{\mathcal{I}}\}$ | \mathcal{I} |
| cardinalità non qualificate | $\geq nR$ $\leq nR$ | $\{x \mid \#\{y.\langle x, y \rangle \in R^{\mathcal{I}}\} \geq n\}$ $\{x \mid \#\{y.\langle x, y \rangle \in R^{\mathcal{I}}\} \leq n\}$ | \mathcal{N} |
| cardinalità qualificate | $\geq nR.C$ $\leq nR.C$ | $\{x \mid \#\{y.\langle x, y \rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geq n\}$ $\{x \mid \#\{y.\langle x, y \rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leq n\}$ | \mathcal{Q} |

Ottimizzazioni

- Normalizzazione

| | | |
|---------------------|---|----------------------------------|
| concetto semplice C | → | C |
| or(A, B, C) | → | not(and(not(A), not(B), not(C))) |
| some(R, C) | → | not(all(R, not(C))) |
| atm(n, R, C) | → | not(atl(n+1, R, C)) |

Ottimizzazioni

- Normalizzazione e semplificazione

| | | |
|---------------------------|---|------------------------|
| concetto semplice C | → | C |
| not(not(C)) | → | C |
| and(A, B, and(C, D)) | → | and(A, B, C, D) |
| and(A, A, B, C, C, C, D) | → | and(A, B, C, D) |
| and(A, B, not(B), C) | → | BOTTOM |
| and(A, B, C, TOP, D) | → | and(A, B, C, D) |
| and(A, B, C, BOTTOM, D) | → | BOTTOM |
| and(all(R, C), all(R, D)) | → | and(all(R, and(C, D))) |
| and(C) | → | C |
| all(R, TOP) | → | TOP |
| atl(0, R, C) | → | TOP |



Ottimizzazioni



- Normalizzazione e semplificazione
- Lazy unfolding



Ottimizzazioni



- Normalizzazione e semplificazione
- Lazy unfolding
- Boolean Constraint Propagation



Ottimizzazioni



- Normalizzazione e semplificazione
- Lazy unfolding
- Boolean Constraint Propagation
- Caching

Ottimizzazioni



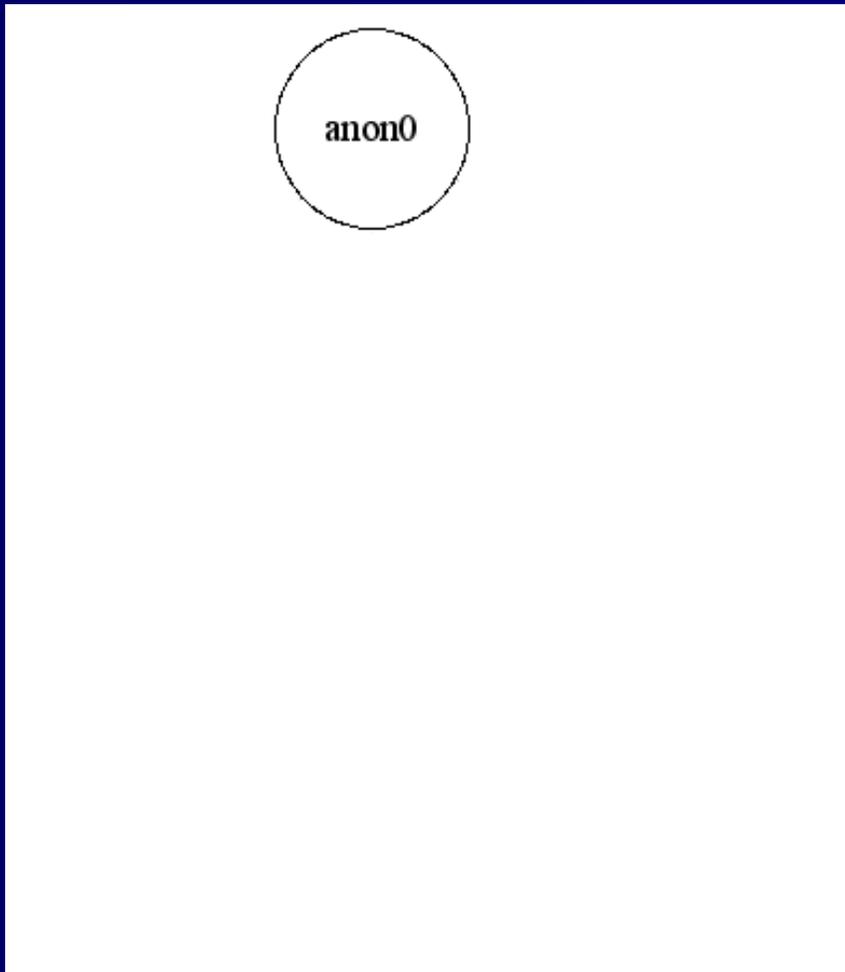
- Normalizzazione e semplificazione
- Lazy unfolding
- Boolean Constraint Propagation
- Caching
- Ricerca *depth first*

Ottimizzazioni



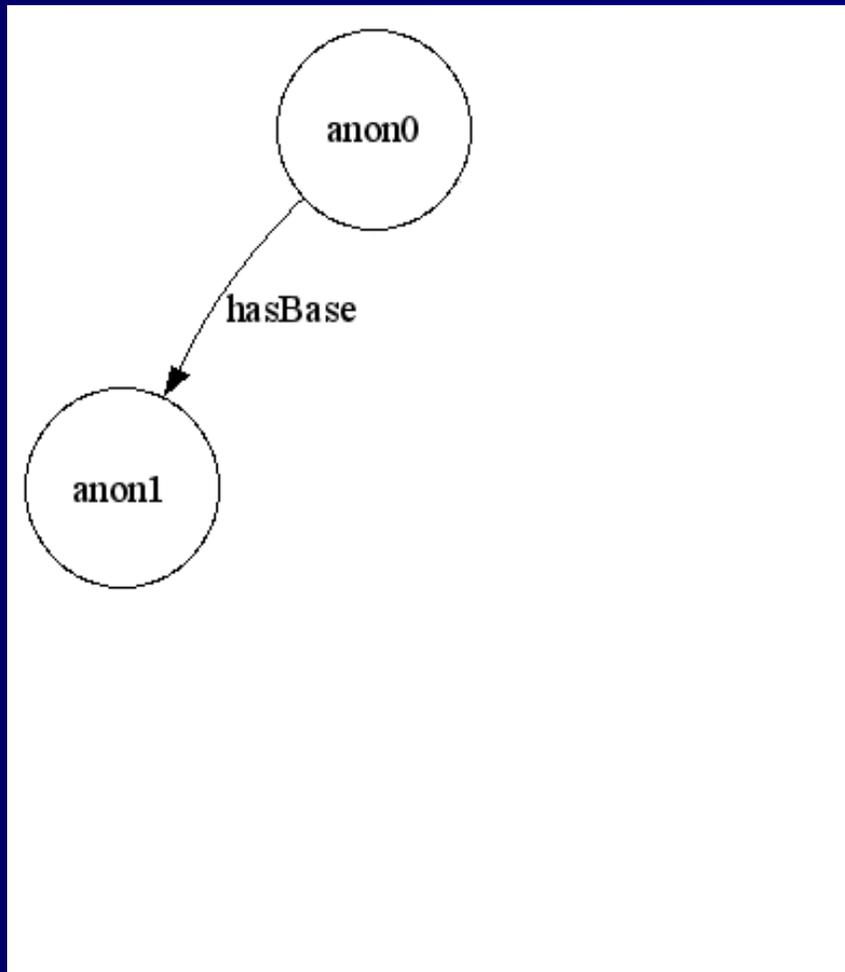
- Normalizzazione e semplificazione
- Lazy unfolding
- Boolean Constraint Propagation
- Caching
- Ricerca *depth first*
- Ordine di applicazione delle regole.

Tableaux - esempio



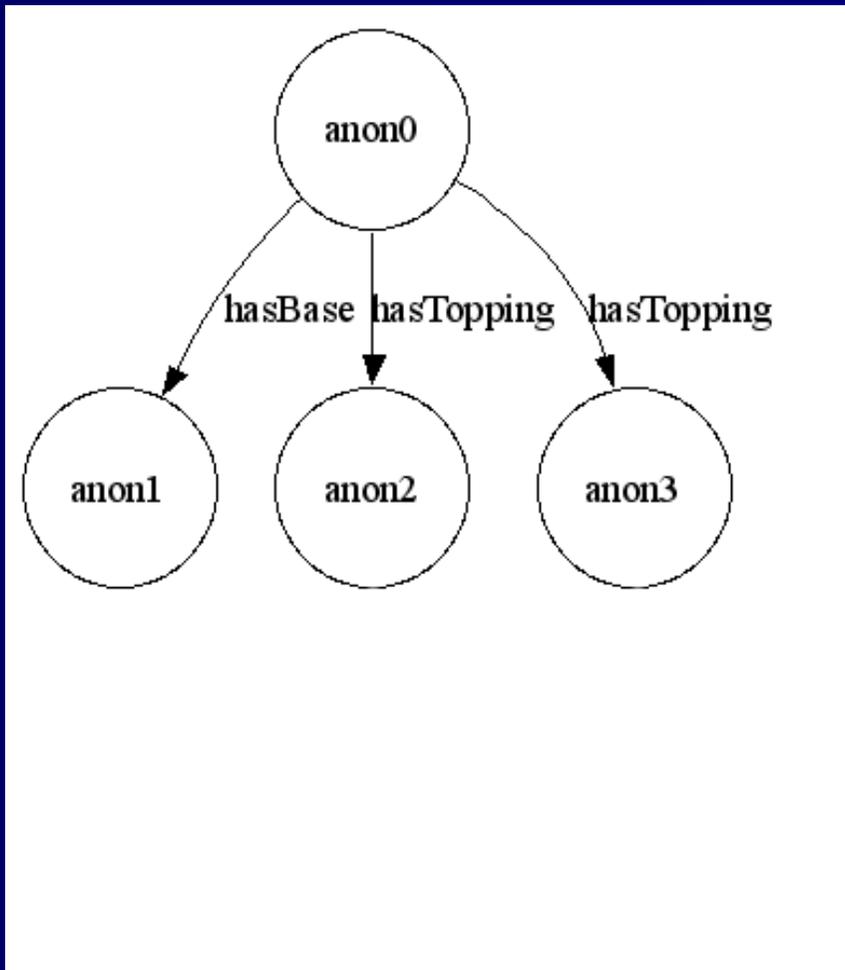
- anon0:
 - PizzaMargherita
 - Pizza
 - some(hasBase, PizzaBase)
 - some(hasTopping, Pomodoro)
 - some(hasTopping, Mozzarella)
 - all(hasTopping, or(Mozzarella, Pomodoro))

Tableaux - esempio



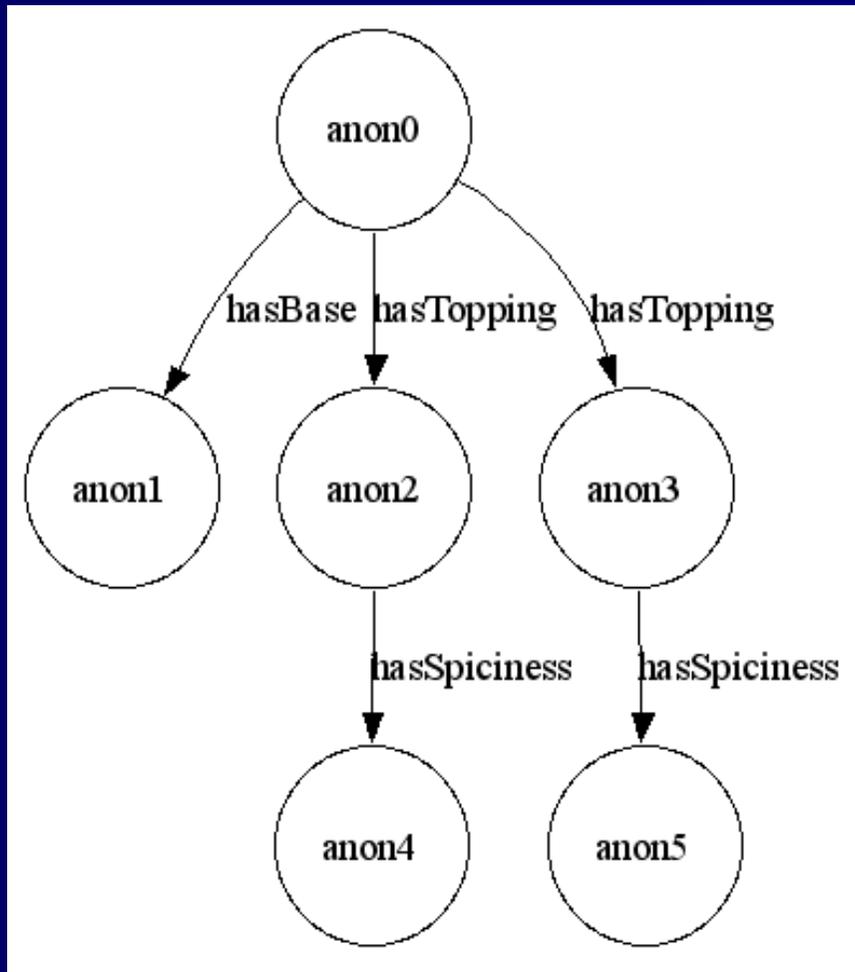
- anon0:
 - PizzaMargherita
 - Pizza
 - some(hasBase, PizzaBase)
 - some(hasTopping, Pomodoro)
 - some(hasTopping, Mozzarella)
 - all(hasTopping, or(Mozzarella, Pomodoro))
- anon1:
 - PizzaBase

Tableaux - esempio



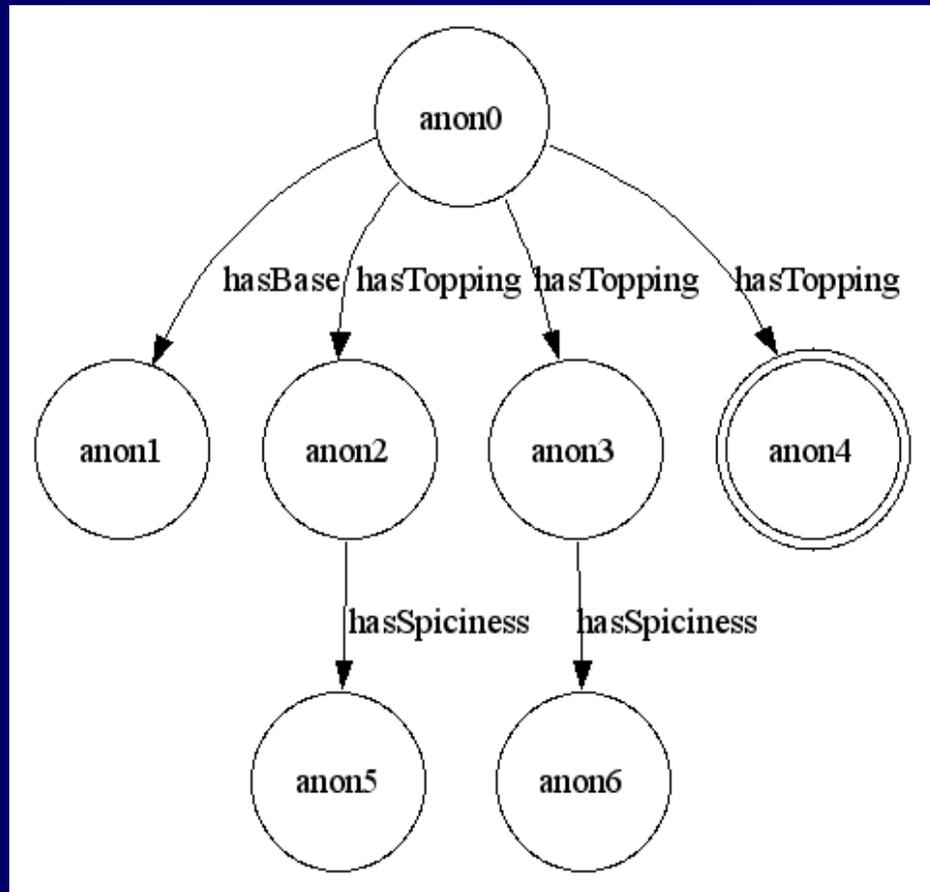
- anon2:
 - Pomodoro
 - some(hasSpiciness,Mild)
- anon3:
 - Mozzarella
 - some(hasSpiciness,Mild)

Tableaux - esempio



- anon2:
 - Pomodoro
 - some(hasSpiciness,Mild)
- anon3:
 - Mozzarella
 - some(hasSpiciness,Mild)
- anon4:
 - Mild
- anon5:
 - Mild

Tableaux - esempio



- $\text{and}(\text{PizzaMargherita}, \text{some}(\text{hasTopping}, \text{Zucchine}))$
- anon2:
 - Pomodoro
- anon3:
 - Mozzarella
- anon4:
 - Zucchine
 - $\text{or}(\text{Mozzarella}, \text{Pomodoro})$
- Infatti, in anon0:
 - $\text{all}(\text{hasTopping}, \text{or}(\text{Mozzarella}, \text{Pomodoro}))$
- \Rightarrow CLASH