

Un Algoritmo per l'Apprendimento di Concetti Basato su Controfattuali

Floriana Esposito, Nicola Fanizzi, Luigi Iannone, Ignazio Palmisano, Domenico Redavid, e Giovanni Semeraro

Dipartimento di Informatica
Università degli Studi di Bari
Via Orabona 4, 70125 Bari, Italy

{*esposito, fanizzi, iannone, palmisano, redavid, semeraro*}@di.uniba.it

Sommario La realizzazione della visione del Semantic Web necessita di algoritmi e strumenti per la costruzione automatica di ontologie, poiché la costruzione manuale di ontologie non può reggere il passo, sia per questioni di costo che di tempo, con l'enorme quantità di conoscenza disponibile sul Web attualmente. In questo articolo presentiamo un algoritmo, basato su metodi di Machine Learning, che permette di automatizzare parte del processo di costruzione di ontologie da parte degli ingegneri della conoscenza.

1 Introduzione

Dalla creazione del nome Semantic Web (SW), da parte di Tim Berners-Lee, per identificare la sua visione del nuovo web [4], molti ricercatori, soprattutto dell'area Knowledge Representation & Reasoning (KRR), si sono impegnati per trovare il formalismo migliore per rappresentare la conoscenza nel SW. Il nome Semantic Web identifica infatti un insieme di specifiche che hanno l'obiettivo di rendere l'informazione (o meglio la conoscenza) direttamente elaborabile dalle macchine. Questo insieme di specifiche si basa su tecnologie già esistenti e di larghissima diffusione, come XML¹ e URI². Su questi standard, poggia un framework per la rappresentazione di metadati, che hanno lo scopo di esplicitare la conoscenza contenuta in risorse web esistenti: RDF³. Per garantire l'interoperabilità, i metadati dovrebbero essere espressi facendo riferimento a *vocabolari* condivisi, in cui sono definite classi e relazioni tra classi. L'evoluzione delle specifiche per l'espressione di questi *vocabolari* è iniziata da RDFSchema [10] ed è giunta ad OWL (Web Ontology Language) [8], rendendo chiaro che il formalismo scelto nel SW per la rappresentazione di concetti e relazioni nei metadati è una

¹ eXtensible mark-up language <http://www.w3.org/XML>

² Uniform Resource Identifiers <http://www.w3.org/Addressing/>

³ Resource Description Framework - <http://www.w3.org/RDF>

particolare logica descrittiva [2]. Il termine *ontologia* è preso in prestito dalla filosofia, ma con un significato differente: un'ontologia per il SW è “la specificazione di una concettualizzazione” [7]. Ciò rappresenta la base per l'evoluzione del Semantic Web: attualmente, sono disponibili specifiche per la scrittura di documenti portabili (XML), specifiche per la scrittura di metadati (RDF) per questi o per altri documenti (per esempio pagine HTML), e per la costruzione di ontologie per i metadati, che sono insiemi di relazioni tassonomiche e non tassonomiche tra le classi definite per i metadati. Poiché queste ontologie sono basate sulle Description Logics, esse hanno una semantica formale, e pertanto offrono la possibilità di implementare algoritmi di inferenza su rappresentazioni basate su ontologie. Illustreremo ora le problematiche emergenti, le quali necessitano di un approccio basato su tecniche di Machine Learning (Sezione 2). Di seguito, presenteremo la nostra soluzione sia dal punto di vista teorico (Sezione 3) che dal punto di vista pratico (Sezione 4). Infine, nella Sezione 5, trarremo alcune conclusioni e presenteremo gli sviluppi futuri di questo lavoro.

2 Motivazione

Il Semantic Web assicura l'interoperabilità semantica grazie ad ontologie che specificano il significato dei metadati in termini delle loro relazioni con le entità che compongono il dominio. Il problema è che, attualmente, il Web non ha ancora abbastanza ontologie, e quelle disponibili, oltre ad essere poche, si riferiscono a domini ristretti. Inoltre, costruire un'ontologia da zero è un compito molto pesante e difficile [9], tenendo anche conto del fatto che due esperti del dominio progetteranno ontologie diverse per lo stesso dominio. Anche se le discrepanze tra le ontologie possono apparire banali, esse possono dipendere da diversi fattori (livello di granularità, punti di vista differenti), e non possono essere armonizzate e integrate facilmente dalle macchine. Di conseguenza, è necessario un approccio alla costruzione di ontologie che sia:

- Semi-automatico (minimalmente).
- Prevedibile e controllabile, nel senso che, fissati i parametri di input e il dominio, i risultati non variano tra un'esecuzione e l'altra.

In questo contesto, un approccio basato su Machine Learning è appropriato, poiché fornisce sia la necessaria flessibilità che il supporto formale per l'acquisizione di ontologie. In particolare, il problema che affrontiamo è la costruzione della definizione di un concetto a partire da esempi positivi e negativi per il concetto in questione. Nel seguito (Sezione 5) vedremo come questo approccio può essere utilizzato per la costruzione di intere ontologie. Tuttavia, una situazione pratica in cui questo algoritmo si rivela utile è quella in cui un'ontologia preesistente deve evolversi per includere una nuova definizione. L'ingegnere della conoscenza può seguire due strade:

- Esprimere la definizione intensionale del nuovo concetto nel linguaggio scelto (per esempio OWL).

- Usare un algoritmo di apprendimento automatico supervisionato per indurre la definizione del nuovo concetto a partire da esempi positivi e negativi per tale concetto.

Anche se la prima soluzione può apparire più semplice, essa ha alcuni svantaggi. Per esempio, nessuno può garantire che la nuova definizione sia consistente con gli esempi (istanze/individui) già presenti nella base di conoscenza. Inoltre, nella scrittura della definizione l'ingegnere potrebbe non considerare caratteristiche importanti ma non evidenti se non si guarda agli esempi. Un esempio pratico consiste nell'estensione di un interessante lavoro di I. Astrova [1], in cui l'autrice propone una metodologia per la costruzione di ontologie da database relazionali. Supponiamo che, dopo il processo, ci si accorga che l'ontologia risultante manca di alcune classi (concetti) che possono essere costruiti a partire dall'ontologia di base costruita in precedenza. Invece di scrivere la definizione mancante da zero, l'ingegnere della conoscenza può selezionare adeguati esempi positivi e negativi (in questo caso un esempio corrisponde a una o più tuple nel database) ed eseguire l'algoritmo per l'apprendimento di concetti che noi proponiamo.

3 Algoritmo per l'Apprendimento di Concetti

In questa sezione illustriamo il nostro algoritmo dal punto di vista teorico, cioè l'apprendimento di una definizione espressa in uno dei linguaggi della summenzionata famiglia di formalismi per la rappresentazione della conoscenza chiamata Description Logics (DL). In particolare, mostriamo i risultati per una particolare DL chiamata \mathcal{ALC} . I formalismi DL differiscono tra loro per i costrutti consentiti. Per facilità di lettura, riportiamo sintassi e semantica per \mathcal{ALC} ; per una descrizione più approfondita si rimanda a [11]. In ogni DL, i concetti primitivi $N_C = \{C, D, \dots\}$ sono interpretati come un sottinsieme di un dominio di oggetti (risorse) mentre i ruoli primitivi $N_R = \{R, S, \dots\}$ sono interpretati come relazioni binarie su questo dominio (proprietà). Descrizioni di concetti più complesse sono costruite usando concetti atomici e ruoli primitivi attraverso l'uso dei costruttori in Tabella 1. Il loro significato è definito attraverso un'interpretazione $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, dove $\Delta^{\mathcal{I}}$ è il dominio dell'interpretazione e il funtore $\cdot^{\mathcal{I}}$ sta per funzione di interpretazione. Una base di conoscenza $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ contiene due componenti: una T-Box \mathcal{T} e una A-box \mathcal{A} . \mathcal{T} è un insieme di definizioni di concetti $C \equiv D$, che vuol dire $C^{\mathcal{I}} = D^{\mathcal{I}}$, dove C è il nome del concetto e D la descrizione in termini dei costruttori del linguaggio. In realtà, esistono T-Box generiche che permettono anche assiomi come $C \sqsubseteq D$ o $C \sqsupseteq D$ e cicli nelle definizioni, ma in questo articolo ci limitiamo a quelle che in letteratura sono chiamate T-Box *acicliche*, in cui vi sono solo definizioni di concetti. Queste definizioni sono nella forma $ConceptName \equiv D$ (si vede facilmente che esse sono equivalenti a T-Box acicliche con concetti complessi su entrambi i lati del segno di equivalenza). \mathcal{A} contiene asserzioni estensionali su concetti e ruoli, per esempio $C(a)$ e $R(a, b)$, intendendo, rispettivamente, che $a^{\mathcal{I}} \in C^{\mathcal{I}}$ e $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. La nozione semantica di *sussunzione* tra concetti (o ruoli) può essere data in termini delle interpretazioni:

NOME	SINTASSI	SEMANTICA
concetto top	\top	$\Delta^{\mathcal{I}}$
concetto bottom	\perp	\emptyset
negazione di un concetto	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
congiunzione di concetti	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disgiunzione di concetti	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
restrizione esistenziale	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
restrizione universale	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y (x, y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$

Tabella 1. I costruttori per le descrizioni \mathcal{ALC} e la loro interpretazione.

Definizione 3.1 (sussunzione) *Date due descrizioni di concetti C e D in \mathcal{T} , C sussume D , denotato con $C \sqsupseteq D$, se e solo se per ogni interpretazione \mathcal{I} di \mathcal{T} vale $C^{\mathcal{I}} \supseteq D^{\mathcal{I}}$. Pertanto, $C \equiv D$ equivale a $C \sqsupseteq D$ e $D \sqsupseteq C$.*

Esempio 3.1 *Una possibile definizione di concetto nel linguaggio proposto è:*
 $Father \equiv Human \sqcap Male \sqcap \exists hasChild.Human$
che traduce la frase: “a father is a male human that has some humans as his children”. Le asserzioni della A-Box sono del tipo:
 $Father(Tom), Father(Bill), hasChild.Human(Bill, Joe)$ *e così via.*
Ora, se definiamo due esempi:
 $FatherWithoutSons \equiv Human \sqcap Male \sqcap \exists hasChild.Human \sqcap \forall hasChild.(\neg Male)$
 $Parent \equiv Human \sqcap (Male \sqcup Female) \sqcap \exists hasChild.Human$
è facile vedere che $Father \sqsupseteq FatherWithoutSons$ e $Parent \sqsupseteq Father$, mentre $Father \not\sqsupseteq Parent$ e $FatherWithoutSons \not\sqsupseteq Father$

Sottolineiamo che la sussunzione impone una relazione d'ordine parziale su qualunque insieme di concetti DL. Nel seguito, infatti, considereremo un insieme di definizioni di concetti ordinati con la sussunzione come spazio di ricerca (\mathcal{S}, \succeq) in cui l'algoritmo deve trovare una definizione consistente per il concetto da definire. Il nostro problema di induzione, nella sua forma più semplice, può essere ora definito formalmente come un problema di apprendimento supervisionato:

Definizione 3.2 (problema di apprendimento induttivo) *In uno spazio di ricerca (\mathcal{S}, \succeq) , data una base di conoscenza $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ e un insieme di asserzioni positive e negative $\mathcal{A}_C = \mathcal{A}_C^+ \cup \mathcal{A}_C^-$ riguardanti l'appartenenza (o non appartenenza) di alcuni individui ad un generico concetto C tali che: $\mathcal{A} \not\models_{\mathcal{T}} \mathcal{A}_C$
Trovare una nuova T-Box $\mathcal{T}' = (\mathcal{T} \setminus \{C \equiv D\}) \cup \{C \equiv D'\}$ tale che: $\mathcal{A} \models_{\mathcal{T}'} \mathcal{A}_C$*

Quindi, se un concetto C non è definito nella terminologia \mathcal{T} , abbiamo un caso di *problema di induzione* che richiede di trovare le definizioni $C \equiv D$ di

⁴ $\not\models_{\mathcal{T}}$ può essere definita informalmente come “ \neq secondo la T-Box \mathcal{T} ”; per una definizione più formale, consultare [2]

cui sono conseguenza logica le (nuove) asserzioni \mathcal{A}_C . D'altra parte, quando una definizione esistente in \mathcal{T} si rivela scorretta, cioè ci sono asserzioni positive in \mathcal{A}_C non coperte (definizione *incompleta*) oppure ci sono asserzioni negative coperte (definizione *inconsistente*), questo crea un *problema di raffinamento* dove una nuova definizione corretta $C \equiv D'$ deve essere trovata, sulla base degli esempi precedenti e di quelli nuovi. Entrambi i problemi possono essere visti come problemi di ricerca nello spazio di tutte le possibili definizioni di concetti in \mathcal{ALC} . Per muoversi in questo spazio, sono necessari operatori per spostarsi tra concetti in due direzioni (poiché lo spazio è parzialmente ordinato con la sussunzione). Infatti, data una definizione di concetto in questo spazio, si può:

- Ottenere una definizione più generale (operatore di raffinamento verso l'alto).
- Ottenere una definizione più specifica (operatore di raffinamento verso il basso).

Dipendentemente dalla DL scelta, si possono immaginare molti operatori di raffinamento. Tipicamente, si tratta di operatori che manipolano la struttura sintattica del concetto, che preventivamente è stato posto in una particolare *forma normale*. Questo tipo di riscrittura sintattica, di solito, ha il vantaggio di semplificare gli operatori di raffinamento e di esistere per ogni descrizione di concetto nella DL scelta. Alcuni esempi di operatori di raffinamento si possono trovare in [3] e in [6] per due differenti DL. Dal punto di vista teorico, si possono studiare alcune proprietà degli operatori di raffinamento che garantiscono che l'implementazione sarà efficace nell'attraversamento dello spazio di ricerca. Queste proprietà sono:

- Finitzza locale, che vuol dire che i possibili concetti ottenibili attraverso il raffinamento sono in numero finito.
- Correttezza, che assicura che ogni passo di raffinamento produce un concetto strettamente più generale (o più specifico) del concetto di partenza.
- Completezza, che garantisce che ogni concetto che sussume (o che è sussunto) il concetto di partenza è raggiungibile con una catena di raffinamenti (cioè n applicazioni di raffinamento a partire dal concetto iniziale).
- Minimalità, cioè ogni possibile raffinamento di un concetto non può essere raggiunto da due catene di raffinamenti diversi (raffinamento non ridondante).

Un operatore di raffinamento *ideale* ha la proprietà di essere localmente finito, completo e corretto. Per \mathcal{ALC} , nessuno ha ancora trovato un operatore con queste caratteristiche; l'implementazione di un operatore completo e ridondante, peraltro, non sarebbe sensata in termini di efficienza. Risulterebbe, infatti, essere una strategia di tipo *generate and test*, consistente nella generazione di tutti i possibili raffinamenti dei concetti in input e nel test di correttezza e consistenza rispetto agli esempi positivi e negativi nel problema di learning. Questo approccio, oltre a essere scarsamente performante, non sfrutta l'informazione disponibile negli esempi utilizzabile per la costruzione del raffinamento del concetto. In questo lavoro, illustreremo in particolare un operatore di raffinamento verso il basso

(specializzazione) basato su esempi, che specializza definizioni troppo generali (definizioni che coprono esempi negativi, che invece non dovrebbero essere coperti), mentre la generalizzazione viene di proposito lasciata all'implementatore, poiché ci sono scelte particolarmente efficienti (soprattutto in \mathcal{ALC}), come vedremo nel seguito. L'idea di base per la specializzazione è che, esaminando una definizione troppo generale, è necessario *rimuovere* la porzione della definizione che è responsabile per l'inclusione delle istanze negative. Un'idea per scegliere la parte di definizione da rimuovere consiste nel trovare il residuo [12] tra la definizione sbagliata e gli esempi negativi come spiegato nel seguito. Quindi, una volta che il concetto responsabile è stato individuato, esso può essere negato (poiché in \mathcal{ALC} la negazione è permessa davanti a concetti complessi) e si può calcolare l'intersezione tra la definizione troppo generale e il residuo negato. Ovviamente, si tratta di una specializzazione, poiché nella teoria degli insiemi si ha che se A, B sono due insiemi, allora $A \cap B \subseteq A$; prendendo $A = C^I$ e $B = (\neg D)^I$, dove C è la definizione originale sbagliata e D il residuo calcolato, allora abbiamo $C \sqcap C \sqcap \neg D$. Il residuo negato è chiamato *controfattuale* [13] e può essere generalizzato per eliminare il maggior numero di negativi possibile dalla definizione inconsistente di partenza, come specificato nella prossima sezione.

3.1 L'Algoritmo

Il processo di apprendimento può iniziare quando ci sono esempi e controesempi nella A-Box di un concetto per cui una nuova definizione è richiesta. Si assume che la classificazione degli esempi sia data dall'ingegnere della conoscenza. Tuttavia, la metodologia si può applicare anche in un contesto analogo, dove esista già una definizione per il concetto da apprendere, ma essa risulta errata (troppo generale) perché da essa derivano concetti che sono stati classificati come negativi per il concetto da apprendere. Per applicare l'algoritmo alla seconda situazione, è sufficiente chiamare la routine *counterfactuals* dell'algoritmo descritto nel seguito. Le asserzioni non sono elaborate direttamente; un rappresentante nel linguaggio per la descrizione dei concetti (*single representation trick*) viene derivato preliminarmente, nella forma di concetto più specifico (*most specific concept, msc*). Il *msc* richiesto per l'algoritmo è la descrizione di un concetto DL massimamente specifica da cui deriva l'asserzione in questione. Poiché in alcune DL tale definizione non esiste, consideriamo la sua approssimazione fino a una certa profondità [5]. Di conseguenza, nell'algoritmo gli esempi positivi e negativi saranno descrizioni congiuntive molto specifiche. L'algoritmo si basa su due coroutine (vedere Figura 1) che eseguono, rispettivamente, generalizzazione e controfattuali, e si chiamano l'un l'altra per convergere a una definizione corretta. L'algoritmo di generalizzazione cerca di spiegare gli esempi positivi costruendo una definizione disgiuntiva. Ad ogni iterazione più esterna, una definizione molto specializzata (l'*msc* di un esempio) viene selezionata come punto di partenza per una nuova generalizzazione parziale; quindi, iterativamente, l'ipotesi è generalizzata per mezzo dell'operatore di generalizzazione δ (non definito qui, ma l'implementatore dovrebbe scegliere un operatore con euristiche per massimizzare il numero di positivi coperti con la generalizzazione) finché

```

generalizzazione(Positivi, Negativi, Generalizzazione)
input Positivi, Negativi: istanze positive e negative (definizioni);
output Generalizzazione: definizione generalizzata
begin
  RisPositivi  $\leftarrow$  Positivi
  Generalizzazione  $\leftarrow$   $\perp$ 
  while RisPositivi  $\neq$   $\emptyset$  do
    ParGen  $\leftarrow$  select_seed(RisPositivi)
    PosCoperti  $\leftarrow$  {Pos  $\in$  RisPositivi | ParGen  $\sqsupseteq$  Pos}
    NegCoperti  $\leftarrow$  {Neg  $\in$  Negativi | ParGen  $\sqsupseteq$  Neg}
    while PosCoperti  $\neq$  RisPositivi and NegCoperti  $=$   $\emptyset$  do
      ParGen  $\leftarrow$  select( $\delta$ (ParGen), RisPositivi)
      PosCoperti  $\leftarrow$  {Pos  $\in$  RisPositivi | ParGen  $\sqsupseteq$  Pos}
      NegCoperti  $\leftarrow$  {Neg  $\in$  Negativi | ParGen  $\sqsupseteq$  Neg}
    if NegCoperti  $\neq$   $\emptyset$  then
      K  $\leftarrow$  counterfactuals(ParGen, PosCoperti, NegCoperti)
      ParGen  $\leftarrow$  ParGen  $\sqcap$   $\neg K$ 
    Generalizzazione  $\leftarrow$  Generalizzazione  $\sqcup$  ParGen
    RisPositivi  $\leftarrow$  RisPositivi  $\setminus$  PosCoperti
  return Generalizzazione
end

counterfactuals(ParGen, PosCoperti, NegCoperti, K)
input ParGen: definizione inconsistente
       PosCoperti, NegCoperti: descrizioni positive e negative coperte
output K: counterfactual
begin
  NuoviPositivi  $\leftarrow$   $\emptyset$ 
  NuoviNegativi  $\leftarrow$   $\emptyset$ 
  for each Ni  $\in$  NegCoperti do
    NewPi  $\leftarrow$  residual(Ni, ParGen)
    NuoviPositivi  $\leftarrow$  NuoviPositivi  $\cup$  {NewPi}
  for each Pj  $\in$  PosCoperti do
    NewNj  $\leftarrow$  residual(Pj, ParGen)
    NuoviNegativi  $\leftarrow$  NuoviNegativi  $\cup$  {NewNj}
  K  $\leftarrow$  Generalizzazione(NuoviPositivi, NuoviNegativi)
return K
end

```

Figura 1. Le coroutine usate nel metodo.

tutti i rappresentanti dei concetti positivi sono coperti o tutti i rappresentanti dei negativi sono spiegati. In questo caso, la definizione corrente *ParGen* deve essere specializzata attraverso i controfattuali. La coroutine, che riceve gli esempi coperti come input, trova una sottodescrizione K che sia capace di eliminare gli esempi negativi presentati. Nella routine per la costruzione dei controfattuali, data un'ipotesi precomputata *ParGen*, che si suppone completa (che copre le asserzioni positive) ma inconsistente rispetto a nuove asserzioni negative, si mira a trovare i controfattuali da congiungere all'ipotesi iniziale per tornare a una definizione corretta che permetta di escludere le istanze negative. L'algoritmo è basato sulla costruzione di problemi di apprendimento residui basati su sottodescrizioni che causano la sussunzione di esempi negativi, rappresentati dai rispettivi *msc*. In questo caso, per ogni modello il residuo è derivato considerando la parte della definizione scorretta *ParGen* che non ha avuto parte nella sussunzione. Il residuo viene poi usato come un'istanza positiva di quella parte della descrizione che dovrebbe essere esclusa dalla definizione (attraverso la negazione). Analogamente, l'*msc* derivato dalle asserzioni positive assumerà il ruolo opposto di istanza negativa per il problema di apprendimento residuo in costruzione. Infine, il problema viene risolto chiamando la coroutine che generalizza queste descrizioni di esempi e quindi congiungendo la negazione dei risultati ritornati.

4 Un Esempio

In questa sezione, presentiamo un esempio per illustrare l'algoritmo attraverso una esecuzione completa.

Esempio 4.1 (Apprendimento Supervisionato) *Supponendo che la A-Box di partenza sia*

$\mathcal{A} = \{M(d), r(d, l), r(j, s), \neg M(m), r(m, l), \neg M(a), w(a, j), r(a, s), F(d), F(j), \neg F(m), \neg F(a)\}$

(assumendo $F \equiv \text{Father}$, $M \equiv \text{Man}$ $r \equiv \text{parentOf (role)}$, $w \equiv \text{wifeOf}$ per rendere l'esempio comprensibile)

F è il concetto da apprendere, pertanto gli esempi e i controesempi sono, rispettivamente: Positivi = $\{d, j\}$ and Negativi = $\{m, a\}$

Gli msc approssimati sono:

$$\begin{aligned} msc(j) &= \exists r. \top \\ msc(d) &= M \sqcap \exists r. \top \\ msc(m) &= \neg M \sqcap \exists r. \top \\ msc(a) &= \neg M \sqcap \exists r. \top \sqcap \exists w. \top \end{aligned}$$

L'esecuzione dell'algoritmo è:

generalize:

ResiduoPositivi $\leftarrow \{msc(d), msc(j)\}$

Generalizzazione $\leftarrow \perp$

/ while esterno */*

$ParGen \leftarrow msc(d) = M \sqcap \exists r. \top$
 $PosCoperti \leftarrow \{msc(d)\}$
 $NegCoperti \leftarrow \{\}$
 $ParGen \leftarrow \exists r. \top$ /* M rimosso nel loop interno */
 $PosCoperti \leftarrow \{msc(d), msc(j)\}$
 $NegCoperti \leftarrow \{msc(m), msc(a)\}$
 Call **counterfactuals**($\exists r. \top, \{msc(d), msc(j)\}, \{msc(m), msc(a)\}$)

counterfactuals:

$NewP_1 \leftarrow \neg M \sqcap \exists r. \top \sqcup \neg \exists r. \top = \neg M$
 $NuoviPositivi \leftarrow \{\neg M\}$
 $NewP_2 \leftarrow \neg M \sqcap \exists r. \top \sqcap \exists w. \top \sqcup \neg(\exists r. \top) = \neg M \sqcap \exists w. \top$
 $NuoviPositivi \leftarrow \{\neg M, \neg M \sqcap \exists w. \top\}$
 $NewN_1 \leftarrow M \sqcap \exists r. \top \sqcup \neg \exists r. \top = M$
 $NuoviNegativi \leftarrow \{M\}$
 $NewN_2 \leftarrow \top$
 $NuoviNegativi \leftarrow \{M, \top\}$
 Call **generalize**($\{\neg M, \neg M \sqcap \exists w. \top\}, \{M, \top\}$)
 ...

Il risultato è: $F = M \sqcap \exists r. \top$

5 Conclusioni e Sviluppi futuri

In questo lavoro abbiamo affrontato il problema della costruzione di ontologie in maniera semiautomatica. In particolare, abbiamo presentato un algoritmo capace di inferire descrizioni di concetti nella Description Logic \mathcal{ALC} da istanze di concetti disponibili in una A-Box. L'algoritmo rappresenta la base per un potente strumento per l'ingegnere della conoscenza. Esso è stato implementato in un sistema chiamato *YINYANG* (Yet another INduction Yields to ANother Generalization), e, al momento attuale, è sottoposto a sperimentazione empirica per valutare estensivamente l'applicabilità pratica dell'approccio. Inoltre, nei problemi reali è possibile che le A-Box siano inconsistenti, cosa che risulterebbe nel fallimento dell'algoritmo. Pertanto, un'altra linea di ricerca futura è l'indagine sul modo migliore di affrontare questo problema.

5.1 Riconoscimenti

Lo sviluppo di questa ricerca è stato finanziato in parte dalla Comunità Europea sotto il Progetto Integrato IST VIKEF - Virtual Information and Knowledge Environment Framework (Contratto n. 507173 - altre informazioni a <http://www.vikef.net>) e dal Ministero dell'Istruzione, dell'Università e della Ricerca, progetto COFIN 2003 "Tecniche di intelligenza artificiale per il reperimento di informazione di qualità sul Web".

Riferimenti bibliografici

- [1] Irina Astrova. Reverse engineering of relational databases to ontologies. In *ESWS*, pages 327–341, 2004.
- [2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
- [3] L. Badea and S.-H. Nienhuys-Cheng. A refinement operator for description logics. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *LNAI*, pages 40–59. Springer, 2000.
- [4] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.
- [5] S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in description logics. In D. Fensel, F. Giunchiglia, D. McGuinness, and M.-A. Williams, editors, *Proceedings of the International Conference on Knowledge Representation*, pages 203–214. Morgan Kaufmann, 2002.
- [6] Floriana Esposito, Nicola Fanizzi, Luigi Iannone, Ignazio Palmisano, and Giovanni Semeraro. Knowledge-intensive induction of terminologies from metadata. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proceedings of the 3rd International Semantic Web Conference, ISWC2004*, volume 3298 of *LNCS*, pages 411–426. Springer, 2004.
- [7] Thomas R. Gruber. A translation approach to portable ontology specifications, 1993.
- [8] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From *SHIQ* and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
- [9] Alexander Maedche and Steffen Staab. Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16(2), March/April 2001.
- [10] RDF-Schema. RDF Vocabulary Description Language 1.0: RDF Schema, 2003. Available at: <http://www.w3c.org/TR/rdf-schema>.
- [11] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
- [12] G. Teege. A subtraction operation for description logics. In P. Torasso, J. Doyle, and E. Sandewall, editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 540–550. Morgan Kaufmann, 1994.
- [13] S.A. Vere. Multilevel counterfactuals for generalizations of relational concepts and productions. *Artificial Intelligence*, 14:139–164, 1980.