↔←→¬∨∧⊆≠∪∈∉

Lezione 6

Le prove Datalog.
Costruzione di una KB
ed esempi

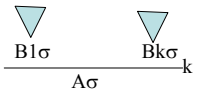
1. Le prove Datalog

- Basate su
  - ***sostituzione*** $\sigma = \{X_1/t_1,..,X_n/t_n\}$
  - ***applicazione*** di σ ad una clausola C:
    - sostituisce uniformemente X_1 con t_1, .., X_n con t_n
    - si indica con $C\sigma$
- ESEMPIO:
  vado(**X**,**Y**) ← aperta(**X**,**Y**) **{X/a,Y/W}** =
  vado(**a**,**W**) ← aperta(**a**,**W**)

1.1. Le prove Datalog in una KB

Base $\frac{}{A\sigma}\,k$ è una prova sse k. A è in KB

Passo



è una prova sse k. A ← B1 ∧ .. ∧ Bk è in KB e gli alberi sopra la riga sono, ricorsivamente, prove

Esempio

1. vado1(X,Y) ← aperta(X,Y).
2. vado1(X,Y) ← aperta(Y,X).
3. vado2(X,Y) ← vado1(X,Z) ∧ vado1(Z,Y).
4. vado3(X,Y) ← vado1(X,Z) ∧ vado2(Z,Y).
5. aperta(c,b).
6. aperta(c,d).

$$\dfrac{\dfrac{\dfrac{}{\text{aperta(c,b)}}\,5}{\text{vado1(b,c)}}\,2 \quad \dfrac{\dfrac{}{\text{aperta(c,d)}}\,6}{\text{vado1(c,d)}}\,1}{\text{vado2(b,d)}}\,3$$

1.2. La procedura Top Down

- ***Si basa sull'unificazione:***
  A unifica con B con unificatore σ sse Aσ coincide con Bσ
- ESEMPIO
  vado(a,X) unifica con vado(Y,Z) via {Y/a, X/Z}:
  vado(a,X){Y/a, X/Z} = vado(a,Z) coincide con
  vado(Y,Z){Y/a, X/Z} = vado(a,Z)
- Per non perdere soluzioni usa ***l'unificatore più generale*** (**mgu**, most general unifier):

- ***A unifica con B con mgu σ sse***
  - $A\sigma = B\sigma = U$
  - Per ogni unificatore δ con $A\delta = B\delta = V$,
    - V è un *caso particolare di* U, cioè $V = U\eta$ per qualche η

Esempio

- mgu(vado(X,a) , vado(I,J)) = {X/I, J/a}:
  vado(X,a){X/I, J/a} = vado(I,J){X/I, J/a} = **vado(I,a)**
- prendiamo l'unificatore {X/b,I/b,J/a}: si ha
  vado(X,a){X/b, I/b, J/a} = vado(I,J){X/b, I/b, J/a} = **vado(b,a)**

vado(b,a) è un caso particolare di **vado(I,a):**
vado(b,a) = **vado(I,a)**{I/b}

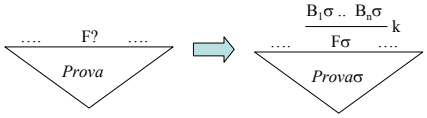
L'algoritmo con unificazione

- Ingresso: KB e atomo A? (anche aperto)
- Albero Prova := A?
- **Fino a che** vi è un F? e nessun F?FAIL
  - ***scegli*** un F?
  - **se** esiste una clausola la cui testa unifica con F
    - ***prendi*** k: H ← Body
      tale che H, F ***unifichino*** ;
      applica k alla foglia F? selezionata
  - **altrimenti**
    poni F?FAIL

Vediamo **applica** ….

•Siano *Prova* l'albero di prova
F? la foglia scelta
k. $H \leftarrow B_1 \wedge .. \wedge B_n$ la clausola presa

- ***rinomina*** k con ***variabili fresche***
  - ***standardizzazione,*** evita conflitti fra variabili
- calcola mgu(H,F) = σ
- calcola $H\sigma \leftarrow B_1\sigma \wedge .. \wedge B_n\sigma$ e *Prova* σ
  e procedi come nel caso proposizionale



Esempio

1. vado1(X,Y) ← aperta(X,Y).
2. vado1(X,Y) ← aperta(Y,X).
3. vado2(X,Y) ← vado1(X,Z) ∧ vado1(Z,Y).
4. vado3(X,Y) ← vado1(X,Z) ∧ vado2(Z,Y).
5. aperta(c,b).
6. aperta(c,d).

vado2(X1,Y1) ← vado1(X1,Z1) ∧ vado1(Z1,Y1)
mgu(vado2(X1,Y1), vado2(b,X)) = {X1/b,X/Y1}

vado2(b,X)?

Esempio

1. vado1(X,Y) ← aperta(X,Y).
2. vado1(X,Y) ← aperta(Y,X).
3. vado2(X,Y) ← vado1(X,Z) ∧ vado1(Z,Y).
4. vado3(X,Y) ← vado1(X,Z) ∧ vado2(Z,Y).
5. aperta(c,b).
6. aperta(c,d).

1. vado1(X2,Y2) ← aperta(X2,Y2)
mgu(vado1(X2,Y2), vado1(Z1,X)) = {X2/Z1,Y2/X}

vado1(b,Z1)? **vado1(Z1,X)?** 3
vado2(b,X)

Esempio

1. vado1(X,Y) ← aperta(X,Y).
2. vado1(X,Y) ← aperta(Y,X).
3. vado2(X,Y) ← vado1(X,Z) ∧ vado1(Z,Y).
4. vado3(X,Y) ← vado1(X,Z) ∧ vado2(Z,Y).
5. aperta(c,b).
6. aperta(c,d).

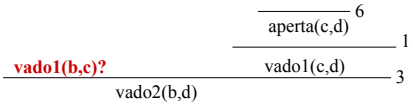
6. aperta(c,d)
mgu(aperta(c,d), aperta(Z1,X)) = {Z1/c,X/d}
Binding X=d

aperta(Z1,X)? 2
vado1(b,Z1)? vado1(Z1,X) 3
vado2(b,X)

Esempio

1. vado1(X,Y) ← aperta(X,Y).
2. vado1(X,Y) ← aperta(Y,X).
3. vado2(X,Y) ← vado1(X,Z) ∧ vado1(Z,Y).
4. vado3(X,Y) ← vado1(X,Z) ∧ vado2(Z,Y).
5. aperta(c,b).
6. aperta(c,d).

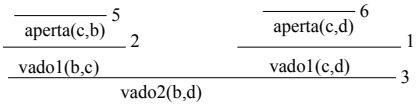
Binding X=d
ESERCIZIO: andate avanti voi



Esempio

1. vado1(X,Y) ← aperta(X,Y).
2. vado1(X,Y) ← aperta(Y,X).
3. vado2(X,Y) ← vado1(X,Z) ∧ vado1(Z,Y).
4. vado3(X,Y) ← vado1(X,Z) ∧ vado2(Z,Y).
5. aperta(c,b).
6. aperta(c,d).

QUERY: vado(b,X)?
RISPOSTA X=d



1.3. Query/Answer in DataLog

- Una ***query*** è una congiunzione Q di atomi, ad es.
  - vado2(a,X) ∧ aperta(X,Y) ?
- Una ***risposta*** (answer) a Q è un sostituzione σ delle variabili presenti nella query, tale che
  KB |= Qσ
- Una ***risposta calcolata*** è una sostituzione δ calcolata dall'interprete (top down o bottom up):
  KB |- Qδ
- **Theo: La procedura è completa:** per ogni query Q e ogni risposta σ, esiste una risposta calcolata δ tale
  Qσ = (Qδ)η
  cioè Qσ è un caso particolare di Qδ.

Esempio

KB: a(X,3).

Query: a(U,V)

Una risposta: U=2, V=3
Risposta calcolata: U = X_100, V=3
a(2,3) = a(X_100,3){X_100/2}

Esercizio

scappa(X) ← gatto(X) ∧ vede(X,Y) ∧ cane(Y).
gatto(felix).
cane(fido).
vede(fido,felix).

- ? gatto(X)
- ? cane(X)
- ? scappa(fido)
- ? vede(X,felix)
- ? vede(X,fido)
- ? scappa(X)

1.4. T_{KB}, la procedura Bottom Up e il modello minimo

- **Si procede come in DLP.**
- Cambia solo la nozione di conseguenza immediata, che considera le istanze ground.
- Rivediamo il tutto, anche come ripasso

- Bottom up: genera dal basso tutti i fatti deducibili per ***Modus Ponens*** e ***istanze ground***

$$\frac{A\sigma \leftarrow B_1\sigma \wedge .. \wedge B_n\sigma \qquad B_1\sigma , \dots , B_n\sigma}{A\,\sigma}$$

$A\sigma$ ***conseguenza immediata*** di $B_1\sigma , \dots , B_n\sigma$
e della la regola $A \leftarrow B1 \wedge .. \wedge Bn$

Data $\mathbf{H} \subseteq$ Atomi_Ground(KB):

T_{KB} (**H**) = **H** $\cup$ {C | C è un'istanza ground di un fatto di KB o una conseguenza immediata di **H** e di almeno una regola di KB}

Solita procedura

PROCEDURA

H := {};

while $T_{KB}(H) \neq H$ do I := T_{KB} (H).

Il risultato finale si indica ancora con KB*

Soliti risultati

Theo. Validità: $A \in KB^* \rightarrow KB \models A$

Theo. Completezza: $KB \models A \rightarrow A \in KB^*$

Theo. Esiste una prova di A, cioè $KB \vdash A$, sse $A \in KB^*$

Theo. KB* è un punto fisso di T: $T_{KB}(KB^*) = KB^*$

Theo. KB* è il minimo punto fisso.

Theo. KB* è il modello minimo di KB.

Complessità: lineare nella dimensione delle istanze ground di KB

- ogni istanza di clausola è usata al più una volta per aggiungere un atomo

Esempio

1. vado1(X,Y) ← aperta(X,Y).
2. vado1(X,Y) ← aperta(Y,X).
3. vado2(X,Y) ← vado1(X,Z) ∧ vado1(Z,Y).
4. vado3(X,Y) ← vado1(X,Z) ∧ vado2(Z,Y).
5. aperta(c,b).
6. aperta(c,d).

H0 = {}
H1 = T(H0) = {aperta(c,b), aperta(c,d)}
H2 = T(H1) = {aperta(c,b), aperta(c,d), vado1(c,b), vado1(b,c), vado1(c,d), vado1(d,c) }
Esercizio: proseguire fino ad ottenere KB*

Esercizio

scappa(X) ← gatto(X) ∧ vede(X,Y) ∧ cane(Y).
gatto(felix).
cane(fido).
vede(fido,felix).

Applicare la procedura bottom-up

2. Come si costruisce una KB

- Analisi del problema
- Individuazione di Individui e Relazioni
- Individuazione delle proprietà
  - generali: sotto forma di regole (e fatti) aperte
  - particolari o contingenti: sotto forma di fatti
- Uso del sistema in termini di
  - query / answer

Esempio: casa di Barbablù rivista

- ***Il problema:***
  - stabilire se si possa andare da una stanza ad un'altra senza passare per quella proibita, in k passi (eventualmente ripetuti)
- **Individui:**
  - gli individui sono le stanze; per introdurre, fra gli infiniti nomi di costante disponibili in Datalog, quelli corrispondenti al dominio delle stanze, usiamo il predicato:

    stanza(X)

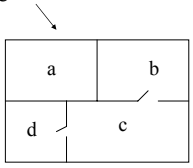
- **Relazioni:**
  - due stanze possono essere messe in comunicazione da una porta aperta:
    - aperta(X,Y) : c'è una porta aperta che si apre spingendo, se si procede da X ad Y
  - Il problema è stabilire se posso andare da una stanza X ad una stanza Y:
    - vadok(X,Y) vero se posso andare da X a Y in k passi
- Come rappresentare le nostre conoscenze?
  - **Regole generali**, valide per tutte le case
  - **Fatti,** relativi ad una casa ben precisa

REGOLE GENERALI (per case con al più 5 stanze):

vado1(X,Y) ← aperta(X,Y).
vado1(X,Y) ← aperta(Y,X).
vado2(X,Y) ← vado1(X,Z) ∧ vado1(Z,Y).
vado3(X,Y) ← vado1(X,Z) ∧ vado2(Z,Y).
vado4(X,Y) ← vado1(X,Z) ∧ vado3(Z,Y).
vado5(X,Y) ← vado1(X,Z) ∧ vado4(Z,Y).

FATTI PARTICOLARI: la casa in figura

stanza(a).
stanza(b).
stanza(c).
stanza(d).
aperta(c,b).
aperta(c,d).



Esercizio

- Codificare in DataLog quanto segue, distinguendo fra fatti e regole.
  Dato che in Datalog non si può usare la negazione, si usi non_allergico(X) per indicare che X non è allergico
  - *Se una persona golosa vede una mela, la mangia, a meno che non sia allergica alle mele.*
  - *Se una persona vede una mela ed ha molta fame, la mangia (anche se è allergica)*
  - *Se una persona allergica mangia una mela, poi si sente male.*
  - *Mario è goloso.*
  - *Luigi ha molta fame.*
- Dare il modello minimo e dare un modello non minimo.
- Dare un non modello.
- Aggiungere il fatto che Luigi è allergico alle mele. Cosa cambia?

3.1. DCL come linguaggio per DB (cap. 3, par. 3)

- Possiamo rappresentare le tabelle di una base dati come elenchi di fatti e le clausole come regole di interrogazione; possiamo fare:
  - select
  - unione
  - join
  - proiezione

Esempio

```
/* tabella corsi */
corso(ai).
corso(logi).
corso(alg).

/* tabelle aule allocate */
in(ai, comelico, 5)
in(logi, comelico, 4).
in(alg, venezian, 5).
in(linux, comelico, delta).

/* tabella fse */
fse(robot).
fse(linux).

/* tabella studenti */
stud(567100, alda).
stud(542301, gigi).
stud(590011, lea).
stud(340055, ugo).

/* tabella frequenze */
segue(567100, ai).
segue(567100, logi).
segue(542301, ai).
segue(542301, alg).
segue(590011, logi).
segue(340055, alg).
segue(340055, linux).
```

Le operazioni sulle basi dati:

- **Select:** Trovare i corsi tenuti in Via Comelico
- acomelico(Corso,Aula) :- in(Corso, **comelico**, Aula).

acomelico(ai, 5)
acomelico(logi, 4)
acomelico(linux, delta)

in(ai, **comelico**, 5).
in(logi, **comelico**, 4).
in(alg, venezian, 5).
in(linux, **comelico**, delta).

Vediamo come avviene top-down

1. acomelico(Corso,Aula) :- in(Corso, comelico, Aula).
2. in(ai, comelico, 5).
3. in(alg, venezian, 5).
4. in(logi, comelico, 4).

acomelico(Corso,Aula)?

NOTA.Per brevità usiamo una base dati più piccola

Vediamo come avviene top-down

1. acomelico(Corso,Aula) :- in(Corso, comelico, Aula).
2. in(ai, comelico, 5).
3. in(alg, venezian, 5).
4. in(logi, comelico, 4).

Binding: Corso=ai, Aula=5

in(Corso, comelico, Aula)?
acomelico(Corso,Aula) 1

Vediamo come avviene top-down

1. acomelico(Corso,Aula) :- in(Corso, **comelico**, Aula).
2. in(ai, comelico, 5).
3. in(alg, venezian, 5).
4. in(logi, comelico, 4).

Binding: Corso=ai, Aula=5

OK.
Answer1: Corso=ai, Aula=5
More? Backtrack

in(ai, **comelico**, 5). 2
acomelico(ai,5) 1

Vediamo come avviene top-down

1. acomelico(Corso,Aula) :- in(Corso, comelico, Aula).
2. in(ai, comelico, 5).
3. in(alg, venezian, 5).
4. in(logi, comelico, 4).

Prendo 3, fallisce
Prendo 4.

Binding: Corso=logi, Aula=4

in(Corso, comelico, Aula)?
acomelico(Corso,Aula) 1

Vediamo come avviene top-down

1. acomelico(Corso,Aula) :- in(Corso, comelico, Aula).
2. in(ai, comelico, 5).
3. in(alg, venezian, 5).
4. in(logi, comelico, 4).

Binding: Corso=logi, Aula=4

OK.
Answer2: Corso=logi, Aula=4
More? Backtrack

in(logi, **comelico**, 4). 4
acomelico(logi,4) 1

Vediamo come avviene top-down

1. acomelico(Corso,Aula) :- in(Corso, comelico, Aula).
2. in(ai, comelico, 5).
3. in(alg, venezian, 5).
4. in(logi, comelico, 4).

Nessun'altra clausola per il predicato **in:**
FAIL, Backtrack

in(Corso, comelico, Aula)?
acomelico(Corso,Aula) 1

Vediamo come avviene top-down

1. acomelico(Corso,Aula) :- in(Corso, comelico, Aula).
2. in(ai, comelico, 5).
3. in(alg, venezian, 5).
4. in(logi, comelico, 4).

Nessun'altra clausola per il predicato **acomelico**
Fail, stop
Nessun'altra risposta

acomelico(Corso,Aula)?

Tutte le risposte?

- Siccome il programma termina, siccome la procedura è completa, tutte le possibili prove sono state trovate.
- Si noti che la, procedura è completa non-deterministicamente (se prendo le clausole giuste); ciò comporta dei problemi se il programma non termina con la strategia di ricerca implementata (Prolog usa depth first, che è incompleta; vedremo)
- Se però il programma termina, siamo sicuri che tutte le risposte sono state calcolate (per la completezza del calcolo).

Le operazioni sulle basi dati:

- **Unione:** gli insegnamenti riconosciuti comprendono corsi e fse:

riconosciuto(Ins) :- corso(Ins).
riconosciuto(Ins) :- fse(Ins).

corso(ai).
corso(logi).
corso(alg).

riconosciuto(ai).
riconosciuto(logi).
riconosciuto(alg).
riconosciuto(robot).
riconosciuto(linux).

fse(robot).
fse(linux).

Le operazioni sulle basi dati:

- Join: elencare gli studenti frequentanti, prendendo i corsi frequentati dalla tabella segue e i nomi dalla tabella stud:

frequenta(Matr,Nome,Corso) :- segue(Matr,Corso), stud(Matr,Nome).

frequenta(567100, alda, ai).
frequenta(567100, alda, logi).
frequenta(542301, gigi, ai).
frequenta(542301, gigi, alg).
frequenta(590011, lea, logi).
frequenta(340055, ugo, alg).
frequenta(340055, ugo, linux).

stud(567100, alda).
stud(542301, gigi).
stud(590011, lea).
stud(340055, ugo).

segue(567100, ai).
segue(567100, logi).
segue(542301, ai).
segue(542301, alg).
segue(590011, logi).
segue(340055, alg).
segue(340055, linux).

Le operazioni sulle basi dati:

- **Proiezione:** elencare i corsi ai quali è stata allocata un'aula:

allocato(Corso) :-in(Corso,Via, Aula).

allocato(ai)
allocato(logi).
allocato(alg).
allocato(linux).

in(ai, comelico, 5)
in(logi, comelico, 4).
in(alg, venezian, 5).
in(linux, comelico, delta).

Combinazione di operazioni

- Trovare gli studenti che seguono ai, con matricola e nome:

  segueai(Nome,Matr) :- segue(Matr,ai), stud(Matr,Nome).

ESERCIZIO. Proseguire, trovando :

a) gli studenti che seguono logi oppure ai;

b) quelli che seguono sia logi, sia ai.

ESERCIZIO

- Si supponga che due studenti diventano amici se frequentano lo stesso corso. Aggiungere tale conoscenza e usarla per trovare gli amici.
- L'assunzione del mondo chiuso vale in modo ragionevole?
  - ovvero: due studenti sono amici solo se si trovano nella tabella amici così calcolata?
  - Come dev'essere interpretata la risposta NO?
- Assiomatizzare: due studenti si incontrano se frequentano corsi in aule vicine e due aule sono vicine se si trovano entrambe in comelico o entrambe in venezian.

ESERCIZIO.

Programmare in Prolog quanto visto sin qui.