Rappresentazione della conoscenza

Lezione 5

Sommario

- ♢ Logiche non monotone
- ♢ Ipotesi di mondo chiuso
- ♢ Negazione come fallimento
- ♢ Semantica dei modelli stabili
- ♢ SLDNF

Monotonicità

Un formalismo si dice **monotono** quando l'aggiunta di nuovi elementi descrittivi può solo aumentare l'insieme delle conclusioni derivabili.

Se $\Gamma \subseteq \Delta$ e $\Gamma \vdash A$ allora $\Delta \vdash A$, o anche $Cn(\Gamma) \subseteq Cn(\Delta)$

Ragionamento non-monotono

Al contrario, in un formalismo non monotono l'aggiunta di nuove informazioni può anche invalidare alcune delle conclusioni precentemente derivabili.

La non-monotonicità è una proprietà del sistema formale piuttosto che del ragionamento.

Rappresentazione incompleta (mancanza di informazioni, per semplicità di rappresentazione).

Ragionamento di senso comune (Common sense reasoning)

Informazione negativa

$sivola(c1, c2)$ indica un collegamento aereo tra due città

$\forall xyz.(sivola(x, y) \wedge sivola(y, z) \rightarrow sivola(x, z))$

Usando la logica classica non si può formalmente concludere che due città non sono collegate.

Ipotesi: quando non si deriva l'esistenza di un collegamento, tale collegamento non esiste.

Per esempio, se non si può derivare $sivola(Roma, Orte)$, si assume $\neg sivola(Roma, Orte)$

Non monotonicità: aggiungendo $sivola(Roma, Orte)$ non vale più $\neg sivola(Roma, Orte)$

Asserzioni universali e generali

I violini hanno 4 corde

- **Universali**: le proprietà asserite valgono per tutte le istanze
- **Generali**: le proprietà valgono in genere

I violini hanno 4 corde tranne ... quando non hanno passato qualche guaio

Non monotonicità: Se un violino "perde una corda", rimane un ... violino con 3 corde.

Eccezioni

Gli uccelli volano:
$\forall x.uccello(x) \rightarrow vola(x)$

In questa forma la regola non ammette eccezioni, ma gli struzzi non volano.

$\forall x.uccello(x) \wedge \neg struzzo(x) \rightarrow vola(x)$

Oltre al caso degli struzzi ci sono molte altre eccezioni:

$\forall x.uccello(x) \wedge \neg struzzo(x) \wedge ... \rightarrow vola(x)$

Le eccezioni, in genere, non si conoscono tutte.

Asserzioni universali deboli

- **Generali**:
  - **Normalità**:
  - **Prototipicità**:
  - **Statistica**:
- **Persistenza**:
  - **Inerzia**:
  - **Tempo**:

Mancanza di informazione

- **Mancanza di conoscenza (del contrario)**:
  - **Familiarità**:
  - **Conoscenza di gruppo**:
- **Convenzioni**:
  - **Conversazione**:
  - **Rappresentazione**:

Ragionamento con mondo chiuso

Idea base:
Ci sono molte più cose false che vere. Se qualcosa è vero e rilevante è stato messo nella KB. Quindi se non c'è si può ragionevolmente assumere falso.

Negazione come fallimento finito (P è una KB e A un atomo)

Se da P non si dimostra A allora da P si deduce $\neg A$

- ipotesi di mondo chiuso
- negazione come fallimento

Ipotesi di mondo chiuso

Close World Assumption: CWA (Reiter '78).

$CLOSURE(KB) = KB \bigcup \{\neg L | L$ è un letterale positivo e $KB \not\models L\}$

$CLOSURE(KB) = KB^{+}$

KB^{+} contiene tuttel le frasi del tipo $\neg sivola(Roma, Orte)$

La risposta alle interrogazioni si ottiene usando KB^{+} invece di KB

Conoscenza completa

CWA produce una KB **completa** (i.e. per ogni α $KB^+ \models \alpha$ oppure $KB^+ \models \neg\alpha$)

Vale sia per gli atomi che per le formule.

In particolare, se $KB^+ \models A \vee B$, allora $KB^+ \models A$ oppure $KB^+ \models B$

Sulla base di questa proprietà la risposta ad una interrogazione α si può ottenere verificando se i letterali in α sono implicati da KB^+.

Consistenza della CWA

CWA è consistente con KB definite.

Se nella KB c'è conoscenza **incompleta**:
$KB \models A \vee B$, ma $KB \not\models A$ e $KB \not\models B$

In KB^+ ci sono sia $\neg A$ che $\neg B$, ed si ha una inconsistenza.

CWA generalizzata

GCWA $KB^* = KB \bigcup \{\neg L | L$ è un letterale positivo
e non esiste una clausola positiva C
tale che $KB \models L \vee C$ e $KB \not\models C\}$

Esempio:$KB = \{p \vee q\}$
$GCWA(KB) \models \neg r$, ma $GCWA(KB) \not\models p$

Si possono introdurre delle ulterioni restrizioni (CCWA, ECWA).

Quantificatori

Si prendono tutte le istanze **ground** degli atomi nella KB.

Supponendo che $Orte$ non sia nella KB dei voli (non ha aeroporto), si può concludere che:
$\neg\exists x sivola(x, Orte)$

Non con la CWA, occorre anche la **chiusura del dominio**:

$$KB^{DC} = KB^{+} \bigcup \{\forall x(x = c_1 \vee \ldots \vee x = c_n)\}$$

Con la chiusura del dominio le formule con i quantificatori si trattano considerando solo l'insieme **finito** di individui indicati nella chiusura.

Per le formule con l'uguaglianza occorre anche la **Unique Name Assumption**.

Negazione come fallimento

Negazione come fallimento finito (negation as failure) (P è una KB e A un atomo)

Se da P non si dimostra A allora da P si deduce $\neg A$

- completamento dei predicati
- semantica dei modelli stabili

Completamento dei predicati

Clarke '78: Consideriamo il programma T:

$p \leftarrow a$
$p \leftarrow b$

in cui a e b sono condizioni sufficienti per p (è sufficiente che a sia vero affinché p sia vero).

Il completamento di T è la teoria:

$p \leftrightarrow a \vee b$

Semantica della negazione nei programmi logici

Semantica dei modelli stabili

Il modello operazionale della programmazione logica si estende per trattare la negazione come fallimento finito.

Answer Set Programming: tratta i programmi disgiuntivi e calcola la risposta in base alla semantica dei modelli stabili.

Modelli minimali

L'idea di fondo è di trovare un modello **canonico/preferito** sui cui verificare la verità delle formule.

Le clausole di Horn godono della proprietà che il modello **minimale** è unico (Manca la disgiunzione)

$p(1)$
$q(2)$
$q(x) \leftarrow p(x)$

Modello minimale: $M = \{p(1), q(1), q(2)\}$
Altro modello: $M' = \{p(1), q(1), q(2), p(2)\}$

Negazione nel corpo delle clausole

$A \wedge \neg B \rightarrow C \equiv \neg A \vee B \vee C$

La negazione **classica** nel corpo della clausola introduce la disgiunzione e si perde l'unicità del modello minimo.

La **negazione come fallimento** è più debole della negazione.

$not\ Q \rightarrow P$

non si considera equivalente a: $P \vee Q$
si preferisce il modello $M = \{P\}$

Eliminazione delle negazioni

Dato un programma logico Π, cioè un insieme di clausole:

$A \leftarrow L_1, \ldots, L_m$ con $m \geq 0$

Si sostituisce ciascuna clausola con le sue **istanze ground**.

Dato un insieme M di atomi di Π, il **ridotto** Π_M è ottenuto eliminando:

♢ tutte le clausole che nel corpo hanno un atomo di M negato;

♢ tutti i letterali negativi non in M.

Modelli stabili

Il ridotto è un programma che ha solo clausole di Horn ed un unico modello minimale. Se questo coincide con M, si dice **insieme stabile**.

$Teorema$:Ogni insieme stabile di un programma Π è un modello di Herbrand minimale di Π

La **semantica dei modelli stabili** stabilisce che:
se un programma ammette **un solo modello stabile** questo è il modello canonico/preferito del programma.

Modelli stabili: esempio

$P \leftarrow not\ Q$

ha un unico modello stabile $\{P\}$.

infatti se $M = \{P\}$ il ridotto diventa P;

mentre se $M = \{Q\}$ il ridotto diventa il programma vuoto, il cui modello minimale è $\{\} \neq \{Q\}$.

Modelli stabili: esempio

$p(1,2)$
$q(x) \leftarrow p(x,y), not\ q(y)$

ha un unico modello stabile $\{p(1,2), q(1)\}$.

Modelli stabili: esempio

$P \leftarrow not\ Q$
$Q \leftarrow not\ P$

ha due modelli stabili $\{P\}$ e $\{Q\}$.

mentre

$P \leftarrow not\ P$

non ha modelli stabili.

SLDNF = risoluzione-SLD + NF

NF (Negazione come Fallimento Finito):
un atomo $\neg A$ ha successo se l'albero di derivazione corrispondente al goal A è finito e le sue foglie sono tutte di fallimento.

Consistente (con le varie caratterizzazioni della negazione), ma incompleta.

Le condizioni per la completezza sono molto stringenti:

- variabili istanziate negli atomi negati
- vincoli sulla struttura del programma

L'answer set programming estende la SLDNF, calcolando la risposta in base alla semantica dei modelli stabili.

Negazione come fallimento in PROLOG

In prolog la negazione è realizzata come **fallimento** della ricerca.

Se `X` è una lista di atomi, `not(X)` è vero se l'interprete non riesce a trovare una dimostrazione vera per `X`.

```
student(bill).
student(joe).
married(joe).

unmarried_student(X) :- not married(X), student(X).
? unmarried_student(bill/joe).
```

Terminazione della negazione

Il meccanismo di negazione del prolog non è nè corretto nè completo:
dipende dall'ordine di esame delle clausole. La terminazione di `not(X)` dipende dalla terminazione di `X`:
- Se `X` termina, `not(X)` termina.
- Se `X` ha infinite soluzioni, `not(X)` termina in funzione del fatto che venga o meno trovato un nodo che porti al successo prima di un ramo infinito.

```
married(gino, remberta).
married(X,Y):-married(Y,X).

? not(married(remberta, gino)).
```

Esempio

```
p(s(X)):-p(X).
q(a).
```

L'interrogazione `not (p(X), q(X)).` non termina.

Terminerebbe `not (q(X), p(X))`.

Utilizzare la negazione con atomi completamente istanziati, non genera comportamenti scorretti, nel caso gli atomi terminino. Se si utilizza la negazione con variabili occorre tenere in considerazione il meccanismo di esecuzione del prolog.

Problemi con la negazione

```
unmarried_student(X):-not(married(X)),student(X).
student(bill).
married(joe).
?- unmarried_student(X).

unmarried_student1(X) :- student(X), not(married(X)).
student(bill).
student(joe).
married(joe).
?- unmarried_student1(X).
```

Condizione **sufficiente** per la correttezza:
le variabili di un atomo negato devono istanziate prima che questo venga eseguito.