

Logica dei predicati - Risoluzione

Logica Matematica

Brunella Gerla

Università dell'Insubria, Varese
`brunella.gerla@uninsubria.it`

Ricapitoliamo la risoluzione nella logica proposizionale

Definizione

Una **clausola** è una disgiunzione di letterali.

Per comodità possiamo vedere una clausola come un insieme di letterali, e una formula come un insieme di clausole. Infatti:

- La disgiunzione è commutativa e associativa, quindi non importa in che ordine scriviamo i letterali;
- la disgiunzione è idempotente e quindi non importa ripetere due volte lo stesso letterale.

Definizione

La **clausola vuota** (denotata con \square) è l'insieme vuoto di letterali.

Non bisogna confondere la clausola vuota \square con l'insieme vuoto di clausole che rappresenteremo con l'usuale simbolo \emptyset .

Risoluzione nella logica proposizionale

Ogni formula della logica proposizionale è equivalente ad una formula in forma normale congiuntiva, cioè una congiunzione di disgiunzioni di letterali.

Quindi ogni formula è equivalente ad una congiunzione di clausole,

Quindi ogni formula φ può essere rappresentata come insieme di clausole che denotiamo con φ^C .

Risoluzione nella logica proposizionale

Definizione

Siano C_1 e C_2 due clausole tali che esista un letterale $L \in C_1$ e $\bar{L} \in C_2$. Allora il **risolvente** R di C_1 e C_2 (rispetto al letterale L) è la clausola

$$R = (C_1 \setminus \{L\}) \cup (C_2 \setminus \{\bar{L}\}).$$

Diciamo anche che R si ottiene per **risoluzione** da C_1 e C_2 .

Proposizione: correttezza della risoluzione

Il risolvente R è conseguenza logica della congiunzione $\{C_1, C_2\}$.

Quindi si ha che

$$\{C_1, C_2\} \equiv \{C_1, C_2, R\}.$$

Nota che se $R = \square$ allora si ha $\{C_1, C_2\} \equiv \{C_1, C_2, \square\}$ che è insoddisfacibile.

Questo procedimento si può ripetere, applicando la risoluzione più volte.

Risoluzione nella logica proposizionale

Definizione

Una clausola C è derivabile **per risoluzione** da un insieme di clausole S se esiste una sequenza C_1, \dots, C_n di clausole tale che $C_n = C$ e per ogni $i = 1, \dots, n - 1$ si ha che $C_i \in S$ oppure C_i si ottiene per risoluzione da clausole di S e da qualche C_j con $j < i$.

In questo caso scriviamo

$$S \vdash_R C.$$

Definizione

Una **refutazione** di S è una derivazione della clausola vuota \square da S .
 S è refutabile se $S \vdash_R \square$.

Risoluzione nella logica proposizionale

Teorema

Una formula φ è insoddisfacibile se e solo se $\varphi^C \vdash_R \square$.

Analogamente vale che

S è un insieme di clausole insoddisfacibile se e solo se $S \vdash_R \square$.

e

φ è una tautologia se e solo se $(\neg\varphi)^C \vdash_R \square$.

Risoluzione nel calcolo dei predicati

Consideriamo una formula in forma di Skolem (abbiamo detto che questa non è una restrizione quando si studia la soddisfacibilità di una formula):

$$\varphi = \forall x_1 \cdots \forall x_n \psi.$$

Dato che ψ non ha quantificatori, la si può scrivere come congiunzione di disgiunzioni di letterali (cioè di formule atomiche o negazioni di formule atomiche).

Considero quindi anche le formule in forma di Skolem come insiemi di clause.

Esempio

Le formule $\forall x(A(x) \vee B(x))$ e $\forall x(A(f(c)) \rightarrow C(z))$ sono le clause $\{A(x), B(x)\}$ e $\{\neg A(f(c)), C(z)\}$. Queste due clause non sono risolvibili, però si intravede un modo per renderle risolvibili...

$\{A(x), B(x)\}$ e $\{\neg A(f(c)), C(z)\}$ diventano risolvibili se si istanzia la variabile x con il termine $f(c)$.

In questo caso la prima clausola diventa $\{A(f(c)), B(f(c))\}$ e la clausola risolvente sarebbe $\{B(f(c)), C(z)\}$.

Proviamo a generalizzare questo ragionamento. Per far questo abbiamo bisogno di parlare di **sostituzioni**.

Sostituzioni

In questo contesto ci interessano sostituzioni all'interno di formule non quantificate.

Rappresentiamo una sostituzione come

$$\sigma = \{t_1/x_1, \dots, t_n/x_n\}$$

con il significato di sostituire la variabile x_i con il termine t_i per ogni $i = 1, \dots, n$.

Se E è un insieme di formule (o termini), scriviamo $E\sigma$ per indicare l'insieme ottenuto applicando la sostituzione σ a tutte le formule (o termini) di E .

Esempio

Sia $E = \{P(f(x), y), A(x) \rightarrow B(f(y))\}$ e $\sigma = \{g(x)/y\}$. Allora

$$E\sigma = \{P(f(x), g(x)), A(x) \rightarrow B(f(g(x)))\}.$$

Indichiamo con ϵ la sostituzione vuota: $E\epsilon = E$.

La **composizione** di due sostituzioni σ e τ è la sostituzione (indicata con $\sigma \circ \tau$ oppure con $\sigma\tau$) che si ottiene applicando prima σ e poi τ .

Esempio

Sia $E = \{P(f(x), y), A(x) \rightarrow B(f(y))\}$, $\sigma = \{g(x)/y\}$ e $\tau = \{c/x\}$.

Allora

$$\begin{aligned} E\sigma\tau &= \{P(f(x), g(x)), A(x) \rightarrow B(f(g(x)))\}\tau = \\ &= \{P(f(c), g(c)), A(c) \rightarrow B(f(g(c)))\}. \end{aligned}$$

Si ha $\sigma \circ \tau = \{f(c)/y, c/x\}$.

In genere se $\sigma = \{t_1/x_1, \dots, t_n/x_n\}$ e $\tau = \{s_1/y_1, \dots, s_m/y_m\}$ allora $\sigma \circ \tau$ si ottiene togliendo dall'insieme

$$\{t_1\tau/x_1, \dots, t_n\tau/x_n, s_1/y_1, \dots, s_m/y_m\}$$

tutti gli elementi s_i/y_i per cui $y_i \in \{x_1, \dots, x_n\}$.

Nota che la composizione di sostituzioni non è commutativa.

Definizione

Una sostituzione σ è un **unificatore** per un insieme di espressioni $E = \{\varphi_1, \dots, \varphi_n\}$ se $E\sigma$ ha un solo elemento, cioè se σ rende uguali tutti gli elementi di E .

Esempio

Sia $E = \{A(x), A(f(y)), A(f(g(z)))\}$. La sostituzione $\sigma = \{f(g(z))/x, g(z)/y\}$ è un unificatore per E , infatti

$$E\sigma = \{A(f(g(z)))\}.$$

Anche $\tau = \{f(g(a))/x, g(a)/y, a/z\}$ è un unificatore per E ma si cerca l'unificatore più *piccolo*.

Definizione

Un unificatore σ per un insieme E è un **unificatore più generale** (o mgu) se per ogni altro unificatore τ di E esiste una sostituzione θ tale che

$$\tau = \sigma \circ \theta.$$

Nell'esempio precedente, σ è un mgu per E e $\tau = \sigma \circ \{a/z\}$.

Costruiamo un algoritmo che permetta di trovare il mgu di un insieme di formule, se esiste.

Algoritmo di Robinson

Definizione

L'**insieme di disaccordo** di un insieme di formule E è un insieme di espressioni (formule o termini) $D(E)$ ottenute nel seguente modo: si leggono contemporaneamente tutte le formule di E e si raccolgono in $D(E)$ le prime formule o i primi termini per cui le formule differiscono.

Esempio

Se $E = \{A(x, f(y)), A(x, f(g(y))), A(x, h(z))\}$ allora

$$D(E) = \{f(y), f(g(y)), h(z)\}.$$

Esempio

Sia $E = \{A(x) \rightarrow B(f(y)), A(z) \rightarrow B(f(g(x)))\}$, allora

$$D(E) = \{x, z\}.$$

Se voglio unificare E devo iniziare a sostituire x con z (o, in questo caso in cui ci sono due variabili, z con x): se $\sigma_1 = \{z/x\}$

$$E\sigma_1 = \{A(z) \rightarrow B(f(y)), A(z) \rightarrow B(f(g(z)))\}.$$

Si ha

$$D(E\sigma_1) = \{y, g(z)\}$$

e ponendo $\sigma_2 = \{g(z)/y\}$ otteniamo

$$E\sigma_1\sigma_2 = \{A(z) \rightarrow B(f(g(z)))\}.$$

Algoritmo di Robinson

- ① $k = 0$ e $\sigma_0 = \epsilon$;
- ② Se $|E\sigma_k| = 1$ allora σ_k è mgu.
- ③ Altrimenti calcola $D(E\sigma_k)$:
 - ▶ Se esistono x e t in $D(E\sigma_k)$ tali che x non occorre in t allora poni

$$\sigma_{k+1} = \sigma_k \circ \{t/x\}$$

- incrementa k e ripeti da 2;
- ▶ altrimenti E non è unificabile.

Teorema

L'algoritmo di Robinson restituisce il mgu di un insieme E se questo insieme è unificabile, altrimenti dice che l'insieme non è unificabile.

Esempio

Verificare se i seguenti insiemi sono unificabili:

$$\{B(a, y, f(y)), B(z, z, u)\}$$

$$\{A(f(y, g(v)), h(b)), A(f(h(w), g(a)), t), A(f(h(b), g(v)), t)\}$$

$$\{A(g(x), y), A(y, y), A(y, f(u))\}$$

Dimostrazione.

Notiamo che l'algoritmo termina sempre. Se termina restituendo σ dobbiamo dimostrare che σ è un mgu.

Sia τ un unificatore per E e proviamo per induzione su k esiste una sostituzione γ_k tale che

$$\tau = \sigma_k \circ \gamma_k.$$

Se $k = 0$ allora $\sigma_0 = \epsilon$ e quindi basta porre $\gamma_0 = \tau$.

Supponiamo sia vero per k e proviamo il risultato per $k + 1$.

Se $|E\sigma_k| = 1$ ok.

Dimostrazione.

Se $|E\sigma_k| > 1$ si ha che $\sigma_{k+1} = \sigma_k \circ \{t/x\}$ dove x e t sono in $D(E\sigma_k)$ e x non occorre in t . Poniamo

$$\gamma_{k+1} = \gamma_k - \{t\gamma_k/x\}.$$

Per ipotesi di induzione $\tau = \sigma_k \circ \gamma_k$. Dato che x non occorre in t si ha

$$t\gamma_{k+1} = t\gamma_k.$$

Quindi

$$\gamma_k = \{t\gamma_k/x\} \cup \gamma_{k+1} = \{t\gamma_{k+1}/x\} \cup \gamma_{k+1} = \{t/x\} \circ \gamma_{k+1}$$

e

$$\tau = \sigma_k \circ \gamma_k = \sigma_k \circ \{t/x\} \circ \gamma_{k+1} = \sigma_{k+1} \circ \gamma_{k+1}.$$

In particolare, se l'algoritmo termina dopo n passi si ha che σ_n è un unificatore e per ogni altro unificatore τ si ha $\tau = \sigma_n \circ \gamma_n$.



Occur check

La verifica che t non contenga la variabile x si chiama *occur check* e serve ad evitare situazioni di questo tipo:

$$E = \{A(x), A(f(x))\}$$

$$D(E) = \{x, f(x)\}.$$

Se fosse possibile porre $\sigma_1 = \{f(x)/x\}$ allora sarebbe

$$E\sigma_1 = \{A(f(x)), A(f(f(x)))\}$$

con

$$D(E\sigma_1) = \{x, f(x)\}$$

e così via.

Ma l'occur check può peggiorare di molto le prestazioni dell'algoritmo:

$$E = \{A(x_1, \dots, x_n), A(f(x_0, x_0), f(x_1, x_1), \dots, f(x_{n-1}, x_{n-1}))\}$$

$$D(E) = \{x_1, f(x_0, x_0)\}.$$

$$E\sigma_1 = \{A(f(x_0, x_0), x_2, \dots, x_n), A(f(x_0, x_0), f(f(x_0, x_0), f(x_0, x_0)), \dots, f(x_{n-1}, x_{n-1}))\}$$

$$D(E\sigma_1) = \{x_2, f(f(x_0, x_0), f(x_0, x_0))\}.$$

$$E\sigma_2 = \{A(f(x_0, x_0), f(f(x_0, x_0), f(x_0, x_0)), \dots, x_n), \\ A(f(x_0, x_0), f(f(x_0, x_0), f(x_0, x_0)), f(f(f(x_0, x_0), f(x_0, x_0)), f(f(x_0, x_0), f(x_0, x_0))), \dots, x_n)\}$$

e così via...

Un letterale è una formula atomica o la negazione di una formula atomica. Una clausola è una disgiunzione di letterali. Se ℓ è un letterale denoto con $\bar{\ell}$ il letterale che si ottiene negando ℓ ed eventualmente eliminando la doppia negazione.

Definizione

Siano C_1, C_2 e R clausole della logica dei predicati e supponiamo che C_1 e C_2 non contengano variabili in comune (tramite una sostituzione è sempre possibile riportarsi a questo caso). R è **risolvente** di C_1 e C_2 se

- esistono dei letterali $\ell_1, \dots, \ell_n \in C_1$ e $\ell'_1, \dots, \ell'_m \in C_2$ tali che

$$\{\ell_1, \dots, \ell_n, \bar{\ell}'_1, \dots, \bar{\ell}'_m\}$$

è unificabile e ha σ come mgu.

-

$$R = ((C_1 \setminus \{\ell_1, \dots, \ell_n\}) \cup (C_2 \setminus \{\ell'_1, \dots, \ell'_m\}))\sigma.$$

Esempio

$C_1 = \{A(x, f(y)), A(x, z), C(z)\}$ e $C_2 = \{\neg A(f(x_1), z_1), B(x_1)\}$. Provare che

$$R = \{C(f(y)), B(x_1)\}$$

è una risolvente di C_1 e C_2 .

Esempio

Trovare i risolventi delle seguenti clausole:

$$\{A(x, y), A(y, z)\}, \{\neg A(u, f(u))\}$$

$$\{B(x, x), \neg C(x, f(x))\}, \{C(x, y), D(y, z)\}$$

Teorema (Correttezza)

Se R è una risolvente di C_1 e C_2 allora R è conseguenza logica di C_1 e C_2 .

Dimostrazione.

Supponiamo che $(\mathcal{A}, e^{\mathcal{A}})$ sia un modello di C_1 e C_2 e sia

$$R = ((C_1 \setminus \{\ell_1, \dots, \ell_n\} \cup (C_2 \setminus \{\ell'_1, \dots, \ell'_m\})))\sigma.$$

Allora se $\{\ell_1, \dots, \ell_n\}\sigma = \ell$ e $\{\ell'_1, \dots, \ell'_m\}\sigma = \bar{\ell}$, si ha

$$R = ((C_1\sigma \setminus \{\ell\}) \cup (C_2\sigma \setminus \{\bar{\ell}\})).$$

Dobbiamo provare che $(\mathcal{A}, e^{\mathcal{A}})$ è anche un modello di R .

Si noti che se $(\mathcal{A}, e^{\mathcal{A}}) \models C_1$ allora anche $(\mathcal{A}, e^{\mathcal{A}}) \models C_1\sigma$, e analogamente per C_2 .

Dimostrazione.

- Se $v^{\mathcal{A},e}(\ell) = 1$ allora $v^{\mathcal{A},e}(\bar{\ell}) = 0$. Dato che $(\mathcal{A}, e^{\mathcal{A}}) \models C_2\sigma$ e $C_2\sigma$ è una disgiunzione di letterali, se $v^{\mathcal{A},e}(\bar{\ell}) = 0$ allora sarà ancora $v^{\mathcal{A},e}(C_2\sigma \setminus \{\bar{\ell}\}) = 1$ e quindi $(\mathcal{A}, e) \models R$.
- Se $v^{\mathcal{A},e}(\ell) = 0$ allora $(\mathcal{A}, e^{\mathcal{A}}) \models C_1\sigma$ e $C_1\sigma$ è una disgiunzione di letterali. Se $v^{\mathcal{A},e}(\ell) = 0$ allora sarà ancora $v^{\mathcal{A},e}(C_1\sigma \setminus \{\ell\}) = 1$ e quindi $(\mathcal{A}, e) \models R$.



Teorema (Completezza)

Un insieme di clausole è insoddisfacibile se e solo se $S \vdash_R \square$.

La dimostrazione di questo teorema si basa (tramite il teorema di Herbrand) sulla dimostrazione del caso proposizionale. Ma abbiamo bisogno di un lemma di passaggio:

Lemma (Lifting lemma)

Siano C_1 e C_2 clausole della logica dei predicati (con variabili diverse) e C'_1, C'_2 due istanze ground di C_1 e C_2 risolvibili, con risolvente R' . Allora esiste una risolvente di C_1 e C_2 tale che R' è una istanza ground di R .

Dimostrazione.

[Dimostrazione del lifting lemma]

Sia $C'_1 = C_1\sigma_1$ e $C'_2 = C_2\sigma_2$. Poiché C_1 e C_2 non hanno variabili in comune, allora se $\sigma = \sigma_1\sigma_2$ si ha $C'_1 = C_1\sigma$ e $C'_2 = C_2\sigma$.

$$R' = (C'_1 \setminus \{\ell\}) \cup (C'_2 \setminus \{\bar{\ell}\})$$

e esistono $\ell_1, \dots, \ell_n \in C_1$ e $\ell'_1, \dots, \ell'_m \in C_2$ tali che

$$\{\ell_1, \dots, \ell_n\}\sigma = \ell \qquad \{\ell'_1, \dots, \ell'_m\}\sigma = \bar{\ell}$$

e quindi σ è un unificatore dell'insieme $\{\ell_1, \dots, \ell_n, \bar{\ell}'_1, \dots, \bar{\ell}'_m\}$. Dato che tale insieme è unificabile, esisterà un unificatore più generale τ di $\{\ell_1, \dots, \ell_n, \bar{\ell}'_1, \dots, \bar{\ell}'_m\}$ e si avrà (per la definizione di mgu) $\sigma = \tau \circ s$ dove s è una sostituzione.

Continua.

Quindi

$$\begin{aligned} R' &= (C_1' \setminus \{\ell\}) \cup (C_2' \setminus \{\bar{\ell}\}) \\ &= (C_1\sigma \setminus \{\ell\}) \cup (C_2\sigma \setminus \{\bar{\ell}\}) \\ &= ((C_1 \setminus \{\ell_1, \dots, \ell_n\} \cup (C_2 \setminus \{\ell'_1, \dots, \ell'_m\})))\sigma \\ &= ((C_1 \setminus \{\ell_1, \dots, \ell_n\} \cup (C_2 \setminus \{\ell'_1, \dots, \ell'_m\})))\tau \circ s \end{aligned}$$

e quindi

$$R = ((C_1 \setminus \{\ell_1, \dots, \ell_n\} \cup (C_2 \setminus \{\ell'_1, \dots, \ell'_m\})))\tau$$

è una risolvente di C_1 e C_2 , e inoltre R' è una istanza ground di R ($R' = Rs$).



Esempio del Lifting Lemma

Esempio

Sia $C_1 = \{A(x), A(f(z)), B(x)\}$ e $C_2 = \{C(y), \neg A(y)\}$. Si considerino inoltre le istanze ground

$$C'_1 = \{A(f(c)), B(f(c))\} \quad \text{e} \quad C'_2 = \{C(f(c)), \neg A(f(c))\}$$

ottenute tramite le sostituzioni $\sigma_1 = \{f(c)/x, c/z\}$ e $\sigma_2 = \{f(c)/y\}$. Poniamo $\sigma = \sigma_1\sigma_2$, si ha $C_1 = C'_1\sigma$, $C_2 = C'_2\sigma$.

Possiamo considerare le clausole C'_1 e C'_2 come clausole proposizionali:

$$C'_1 = \{A, B\} \quad \text{e} \quad C'_2 = \{C, \neg A\}$$

che sono risolvibili e hanno come risolvente la clausola

$$R' = \{B, C\} = \{B(f(c)), C(f(c))\}.$$

Esempio (continua...)

Osserviamo che l'insieme $E = \{A(x), A(f(z)), A(f(y))\}$ è unificato da σ , ma σ non è l'unificatore più generale, che è invece

$$\tau = \{f(z)/x, f(z)/y\}.$$

Si ha $\sigma = \tau \circ \{c/z\}$ e

$$\begin{aligned} R' &= (C'_1 \setminus \{A(f(c))\}) \cup (C'_2 \setminus \{\neg A(f(c))\}) \\ &= (C_1\sigma \setminus \{A(x), A(f(z))\}\sigma) \cup (C_2\sigma \setminus \{\neg A(y)\}\sigma) \\ &= ((C_1 \setminus \{A(x), A(f(z))\}) \cup (C_2 \setminus \{\neg A(y)\}))\sigma \\ &= (((C_1 \setminus \{A(x), A(f(z))\}) \cup (C_2 \setminus \{\neg A(y)\}))[\tau \circ \{c/z\}] \\ &= [((C_1 \setminus \{A(x), A(f(z))\}) \cup (C_2 \setminus \{\neg A(y)\}))\tau] \{c/z\} \end{aligned}$$

Quindi la clausola

$$R = ((C_1 \setminus \{A(x), A(f(z))\}) \cup (C_2 \setminus \{\neg A(y)\}))\tau = \{B(f(z)), C(f(z))\}$$

è la risolvente di C_1 e C_2 e inoltre R' è una istanza ground di R .

Teorema

Un insieme di clausole è insoddisfacibile se e solo se $S \vdash_R \square$.

Dimostrazione.

La correttezza ($S \vdash_R \square$ implica S insoddisfacibile) è come nel proposizionale una conseguenza del fatto che il risolvete è conseguenza logica delle clausole risolvibili.

Per l'altra implicazione, dal teorema di Herbrand abbiamo che se S è un insieme di clausole insoddisfacibile, allora esiste un insieme S' di istanze ground (quindi, essendo clausole, possono essere viste come formule proposizionali) di S che è insoddisfacibile. Per la completezza della risoluzione della logica proposizionale, esiste una sequenza di clausole $C'_1, \dots, C'_k = \square$ tale che ogni C'_i o appartiene a S o si ottiene per risoluzione dalle clausole precedenti.

Per il lifting lemma, questa sequenza di clausole proposizionali può essere trasformata in una sequenza di clausole della logica dei predicati e quindi si ha $S \vdash \square$.



Esempio

Riprendiamo l'esempio del barbiere con le formule

$$\forall x \forall y (B(x) \wedge \neg R(y, y) \rightarrow R(x, y))$$

$$\forall x \forall y (B(x) \wedge R(y, y) \rightarrow \neg R(x, y))$$

Trasformarle in clausole e provare che se si aggiunge la clausola $\{B(a)\}$ si ottiene \square .

Clausole di Horn

Definizione

Una **clausola di Horn** è una clausola nella quale compare al più un letterale non negato.

Nota che applicando la risoluzione a clausole di Horn si ottiene una clausola di Horn.

Esempio

Consideriamo le clausole $\{A, \neg B, \neg C\}$ e $\{C, \neg D\}$. Applicando la risoluzione alla variabile C otteniamo la clausola $\{A, \neg B, \neg D\}$ che è ancora una clausola di Horn.

Proviamo a scrivere le formule corrispondenti:

$$B \wedge C \rightarrow A \qquad D \rightarrow C.$$

Applicando la risoluzione si ottiene

$$B \wedge D \rightarrow A.$$

Definizione

Una prova per **risoluzione lineare** di una clausola C a partire da un insieme di clausole S è una sequenza di clausole C_1, \dots, C_n tali che $C_n = C$, e ogni C_i si ottiene per risoluzione da C_{i-1} e da una clausola B che o appartiene a S oppure è stata ottenuta precedentemente per risoluzione.

Nella risoluzione lineare abbiamo quindi un vincolo sull'ordine da utilizzare per applicare la risoluzione.

Proposizione

La risoluzione lineare è completa (per refutazione), cioè se un insieme di clausole S è insoddisfacibile allora esiste una derivazione tramite risoluzione lineare della clausola vuota da S .

Definizione

Una prova per **risoluzione da input** di una clausola C da un insieme di clausole S è una sequenza C_1, \dots, C_n tale che $C_n = C$ e ad ogni passo una delle clausole risolventi è un elemento di S .

La risoluzione da input non è completa per refutazione. Ad esempio dall'insieme insoddisfacibile

$$S = \{\{A, B\}, \{A, \neg B\}, \{\neg A, B\}, \{\neg A, \neg B\}\}$$

non è possibile derivare la clausola vuota utilizzando la risoluzione da input.

Proposizione

La risoluzione da input è completa rispetto ad insiemi di clausole di Horn.

Esempio

Sia $S = \{\{A, \neg B, \neg C\}, \{B, \neg C\}, \{C, \neg D\}, \{D\}, \{\neg A\}\}$. L'insieme S è insoddisfacibile ed esiste una derivazione di \square da S usando la risoluzione da input.

Consideriamo le clausole di Horn.

Definizione

Una clausola di Horn in cui c'è il letterale non negato si chiama **clausola definita**. Le clausole di Horn che non contengono il letterale non negato si chiamano **clausole GOAL**.

Tra le clausole definite distinguiamo le **regole**:

$$A_1 \wedge \dots \wedge A_n \rightarrow B \equiv \neg A_1, \dots, \neg A_n, B$$

e i **fatti**:

$$\rightarrow B \equiv B.$$

Le clausole goal hanno la forma

$$A_1 \wedge \dots \wedge A_n \rightarrow \equiv \neg A_1, \dots, \neg A_n.$$

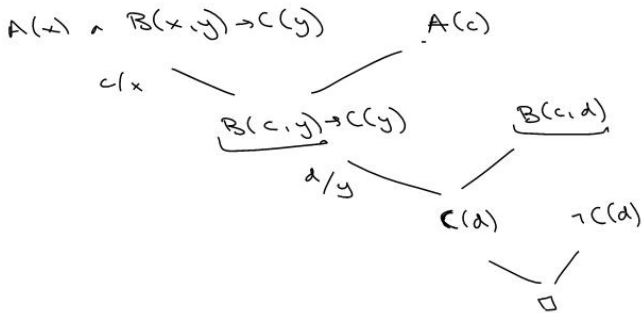
Esempio

$S = \{A(x) \wedge B(x, y) \rightarrow C(y), A(c), B(c, d)\}$ è un insieme di clausole di Horn.

$A(x) \wedge B(x, y) \rightarrow C(y) \equiv \neg A(x) \vee \neg B(x, y) \vee C(y)$ è una regola.

$A(c), B(c, d)$ sono fatti.

Proviamo che $S \cup \{\neg C(d)\} \vdash_R \square$ (nota che $\neg C(d)$ è una clausola goal):



Programmazione logica

La programmazione logica è un tipo programmazione *dichiarativo* che si contrappone al paradigma *imperativo* dei classici linguaggi come Fortran, Cobol, C, Java etc.

Nella programmazione di tipo imperativo ci si concentra a descrivere come risolvere un dato problema, e un programma è formato da un insieme di istruzioni.

Nei linguaggi di tipo dichiarativo invece, un programma serve per descrivere un problema e non si danno indicazioni su come risolverlo. Oltre alla programmazione logica, un altro esempio di programmazione dichiarativa è la programmazione funzionale.

Definizione

Un **programma** è un insieme di clausole di Horn definite, cioè un insieme di regole e di fatti.

Un programma serve per verificare clausole goal, come vedremo di seguito. La formula B nella regola $A_1, \dots, A_n \rightarrow B$ si chiama **testa** della regola. L'insieme delle regole che hanno lo stesso predicato P in testa si chiama **procedura** per P .

L'insieme dei fatti di un programma si chiama **base di dati**.

Esempio

Si consideri il programma contenente le seguenti clausole:

$$\begin{array}{l} p(x,y) \rightarrow q(x,y) \\ p(x,t), p(z,y) \rightarrow q(x,y) \end{array} \left. \vphantom{\begin{array}{l} p(x,y) \rightarrow q(x,y) \\ p(x,t), p(z,y) \rightarrow q(x,y) \end{array}} \right\} \text{procedura per } q(x,y)$$

$$\left. \begin{array}{l} p(a,b) \\ p(b,c) \\ p(d,b) \\ p(e,f) \end{array} \right\} \text{base di dati del programma}$$

Definizione (Risoluzione SLD)

Sia Π un programma e G una clausola goal. Allora una derivazione da Π tramite risoluzione SLD è una sequenza di clausole goal

$$G = G_0, G_1, \dots, G_n$$

tale che se

$$G_i = A_1, \dots, A_n \rightarrow$$

e nel programma c'è una regola

$$B_1, \dots, B_s \rightarrow A$$

dove A e A_1 sono unificabili tramite un mgu θ , allora la clausola G_{i+1} è

$$G_{i+1} = (B_1, \dots, B_s, A_2, \dots, A_n)\theta \rightarrow$$

Se $G_n = \square$ allora $\neg G$ è una conseguenza logica di Π .

Nota che abbiamo dato una versione semplificata della risoluzione SLD, in genere si può anche specificare una *regola di computazione* che dice quale letterale scegliere nella clausola goal.

Esempio

Consideriamo il programma $\Pi = \{(A(x), B(x, y) \rightarrow C(y)), A(c), B(c, d)\}$ e la clausola goal $C(d) \rightarrow$.

Ci stiamo chiedendo se $C(d)$ è una conseguenza delle formule del programma. Ragioniamo con la risoluzione SLD e cerchiamo una clausola la cui testa possa essere unificata con $C(d)$.

La sostituzione $\sigma = \{c/y\}$ permette di fare questa unificazione e quindi dalla clausola goal $C(d)$ e la regola $A(x), B(x, y) \rightarrow C(y)$ ottengo la clausola goal $A(x), B(x, d) \rightarrow$.

Adesso cerco una clausola la cui testa possa essere unificata con $A(x)$ e trovo $\rightarrow A(c)$, quindi dalla clausola goal precedente mi resta $B(c, d) \rightarrow$. E ancora unificando con $\rightarrow B(c, d)$ ottengo la clausola vuota.

Esempio

Si consideri il programma Π formato dalle seguenti clausole:

$$P(x, z), P(z, y) \rightarrow P(x, y)$$

$$P(x, y) \rightarrow P(y, x)$$

$$P(b, a)$$

$$P(c, b)$$

e sia $G = P(a, c) \rightarrow$.

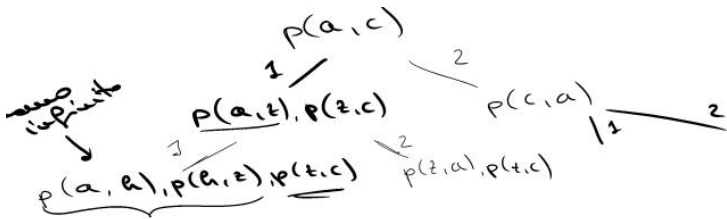
Dobbiamo cercare quale regola del programma ha la testa che unifica con il primo letterale della clausola goal, cioè con $P(a, c)$. La prima regola unifica tramite $\theta_1 = \{a/x, c/y\}$. Quindi la prima clausola goal della derivazione è

$$G_1 = P(a, z), P(z, c)$$

Continuare la derivazione.

Ci sono chiaramente più derivazioni possibili, quando per esempio ci sono più regole la cui testa unifica con un letterale della clausola goal in considerazione.

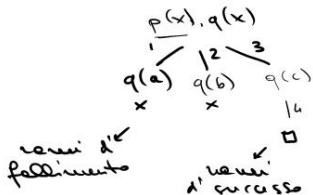
Tutte le derivazioni posso essere rappresentate su un albero:



Dato che ragioniamo sempre per refutazione, cerchiamo di ottenere la clausola vuota \square . I rami che permettono di ottenere la clausola vuota sono i *rami di successo*, gli altri sono rami di *fallimento*.

Esempio

$p(a) \leftarrow$
 $p(b) \leftarrow$
 $p(c) \leftarrow$
 $q(c) \leftarrow$
 $G \leftarrow p(x), q(x)$



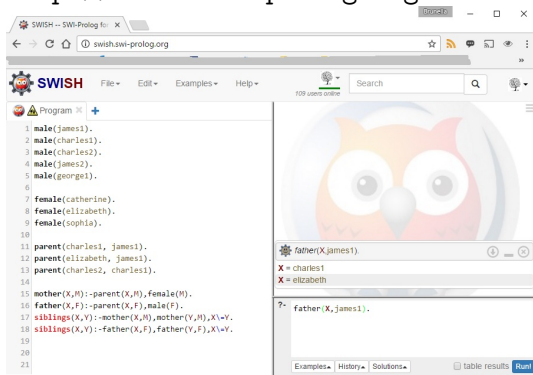
Nella ricerca di un ramo di successo si esplorano tutti i rami dell'albero con una tecnica di *backtracking*.

SWI-PROLOG

SWI-Prolog (<http://www.swi-prolog.org>) è una implementazione open-source del prolog sviluppata a partire dagli anni '80 dall'università di Amsterdam. Offre in realtà molto più del prolog puro, in quanto permette di interfacciarsi con altri linguaggi di programmazione, e con strumenti di sviluppo e controllo.

Per una versione semplificata online si può provare

<http://swish.swi-prolog.org>:



Vediamo la sintassi del PROLOG (SWI-PROLOG):

- Le variabili sono sequenze di simboli che iniziano con la lettera maiuscola;
- predicati e costanti hanno l'iniziale minuscola;
- ogni clausola termina con un punto;
- le clausole di Horn hanno la notazione:

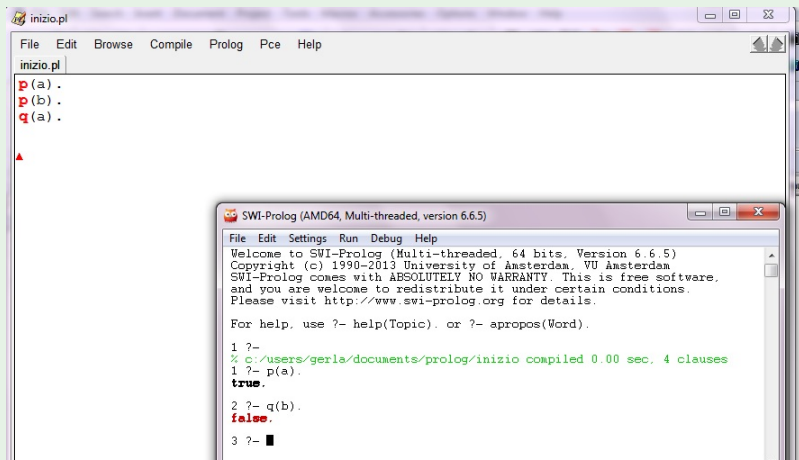
$$B:-A_1, \dots, A_n$$

dove $:-$ è da leggersi \leftarrow (si legge B vale se valgono $A_1 \dots, A_n$).

SWI-prolog - Esempio 1

Nell'esempio in figura, nel programma ci sono solo fatti. Quando viene richiesta la clausola `goal :-p(a)` si ha la risposta `true`. Quando viene richiesta la clausola `goal :-q(b)` si ha la risposta `false`.

Esempio



The screenshot displays two windows from the SWI-Prolog environment. The top window, titled 'inizio.pl', contains the following Prolog facts:

```
inizio.pl
p(a).
p(b).
q(a).
```

The bottom window, titled 'SWI-Prolog (AMD64, Multi-threaded, version 6.6.5)', shows the Prolog prompt and the execution of the facts:

```
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.6.5)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?-
% c:/users/gerla/documents/prolog/inizio compiled 0.00 sec. 4 clauses
1 ?- p(a).
true.

2 ?- q(b).
false.

3 ?- █
```


Esempio 1

Infatti consideriamo l'insieme S formato dalle clausole

$$\{p(a)\}, \{p(b)\}, \{q(a)\}.$$

Se aggiungiamo la clausola goal $\{\neg p(a)\}$ allora con un passaggio di risoluzione (lineare da input) otteniamo la clausola vuota.

Se invece aggiungiamo $\{\neg q(b)\}$ la clausola vuota non si ottiene.

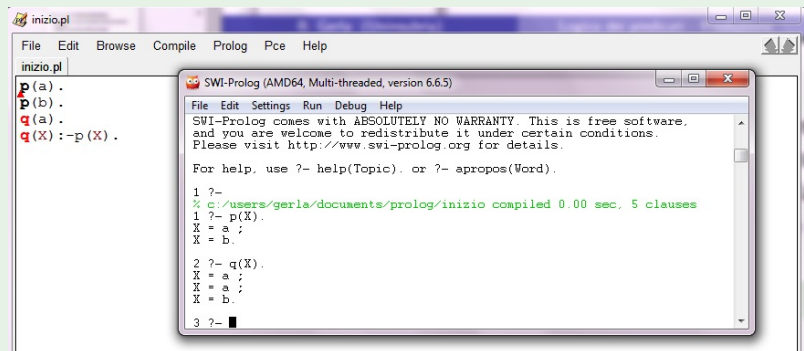
Se aggiungiamo la clausola $\{\neg q(X)\}$ dove X è una variabile, allora tramite la sostituzione $\{a/X\}$ si ottiene la clausola vuota, quindi si avrà a risposta $X=a$.

E se si aggiunge la clausola $\{\neg p(X), \neg q(X)\}$ (che è ancora una clausola goal) cosa succede? La risposta sarà ancora $X=a$.

SWI-prolog - Esempio 2

In questo esempio invece, nel programma c'è anche una regola. Quando viene richiesta la clausola goal $:-p(X)$ si ha la risposta true e viene descritta la sostituzione che permette di ottenere la clausola vuota. Analogamente quando viene richiesta la clausola goal $:-q(X)$ si ha la risposta true e la sostituzione. Provare a considerare la clausola goal $:-p(X), q(X)$ (che corrisponde a $\{\neg p(X), \neg q(X)\}$).

Esempio



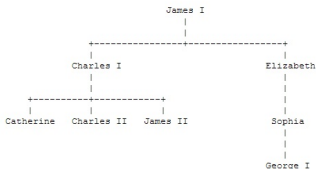
The screenshot shows the SWI-Prolog environment. The main window displays the Prolog program 'inizio.pl' with the following code:

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

An output window titled 'SWI-Prolog (AMD64, Multi-threaded, version 6.6.5)' shows the execution results:

```
File Edit Settings Run Debug Help  
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,  
and you are welcome to redistribute it under certain conditions.  
Please visit http://www.swi-prolog.org for details.  
  
For help, use ?- help(Topic). or ?- apropos(Word).  
  
1 ?-  
% c:/users/gerla/documents/prolog/inizio compiled 0.00 sec, 5 clauses  
1 ?- p(X).  
X = a ;  
X = b ;  
  
2 ?- q(X).  
X = a ;  
X = a ;  
X = b ;  
  
3 ?- 
```

Le relazioni in questo albero genealogico:



si possono rappresentare tramite questo programma:

```
genealogico.pl
File Edit Browse Compile Prolog Pce Help
genealogico.pl
male(james1).
male(charles1).
male(charles2).
male(james2).
male(george1).

female(catherine).
female(elizabeth).
female(sophia).

parent(charles1, james1).
parent(elizabeth, james1).
parent(charles2, charles1).
parent(catherine, charles1).
parent(james2, charles1).
parent(sophia, elizabeth).
parent(george1, sophia).

mother(X,M):-parent(X,M),female(M).
father(X,F):-parent(X,F),male(F).
siblings(X,Y):-mother(X,M),mother(Y,M),X\=Y.
siblings(X,Y):-father(X,F),father(Y,F),X\=Y.
```

Provare ad aggiungere la relazione di essere nonno.