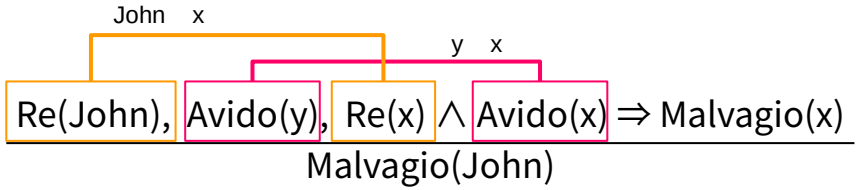
Esempio



Questa conclusione si appoggia alla sostituzione {x/John, y/John}

Le clausole focalizzano la ricerca della sostituzione e permettono di ragionare direttamente in FOL

Nella proposizionalizzazione, di contro, si costruiscono sostituzioni usando in modo esaustivo l'intero vocabolario di costanti

Sulla generalità del MPG: Lifting

- Il modus ponens non pone restrizioni sulla forma della formula antecedente dell' implicazione
- Il modus ponens generalizzato richiede invece che sia una congiunzione
- L' appellativo "generalizzato" deriva dall' avere "sollevato" la regola dalla logica proposizionale a quella del prim' ordine. Questo processo si chiama **lifting**
- Inoltre consente di avere considerare un numero qualsiasi di formule da cui trarre la conclusione (invece di due solamente)

Unificazione

- Unificazione: algoritmo chiave di tutte le tecniche di inferenza sul prim' ordine:
  - Date due formule F1 e F2
  - UNIFY(F1, F2) = θ tale che F1 θ = F2 θ
- Il risultato è una sostituzione che, applicata a entrambe le formule, le rende identiche
- Tale sostituzione, se esiste, è detta UNIFICATORE
- Se vi sono più unificatori si calcola e si usa il Most General Unifier (**MGU, unificatore più generale**)

Esempi

- UNIFY(knows(John, x), knows(John, Richard))
  - {x/Richard}
- UNIFY(knows(John, x), knows(y, Richard))
  - {x/Richard, y/John}
- UNIFY(knows(John, x), knows(y, MotherOf(y)))
  - { x/MotherOf(John), y/John }
- UNIFY(knows(John, x), knows(x, Richard))
  - **Fallisce** perché x non può assumere due valori contemporaneamente
  - **Standardizzazione separata (standardizing apart)**: rinoma delle variabili di una formula per evitare collisioni con un' altra formula.

Abbiamo visto

- Per rispondere alle query occorre fare **inferenza**
- L'inferenza del prim'ordine è diversa da quella proposizionale: abbiamo **quantificatori** e **variabili**
- È possibile ridurre una KB del prim'ordine in una proposizionale e poi applicare algoritmi di inferenza proposizionali:
  - Se la KB non contiene quantificatori esistenziali, quella trasformata sarà equivalente a quella FOL
  - Se la KB contiene quantificatori esistenziali, non sarà equivalente ma sarà inferenzialmente equivalente (se derivo una formula dalla seconda tale formula sarà derivabile anche da quella originaria)
- Rispondere alle query richiede di identificare opportune sostituzioni
- Tali sostituzioni si trovano più efficientemente se se si usano regole di inferenza di cui è stato fatto il lifting al prim' ordine
- È necessario tradurre la KB in clausole di Horn del prim' ordine

Clausole di Horn del prim'ordine

- La definizione è analoga a quelle delle clausole di Horn (e clausole definite) proposizionali, sono disgiunzioni di letterali di cui al più uno è positivo:
  - **Atomiche**:
    - Avido(x) la variabile è intesa come universalmente quantificata
    - Avido(John)
  - **Implicazione il cui antecedente è costituito da letterali positivi**
    $Re(x) \wedge Avido(x) \Rightarrow Malvagio(x)$

- Non tutte le KB possono essere tradotte in clausole di Horn ma molte possono. A queste è possibile applicare il forward chaining (e il modus ponens generalizzato) per fare inferenze

Una KB di esempio

Sapendo: "La legge dice che è un reato per un negozio vendere alcolici a un minorenne. Marco, minorenne, possiede della birra. Tale birra gli è stata venduta del minimarket Sotto Casa"

Obiettivo: "Dimostrare la reità di Sotto Casa"

Useremo questo esempio per imparare a scrivere una KB in clausole di Horn del prim' ordine e come funzionano le inferenze nel prim' ordine

Una KB di esempio: simboli

Costanti: Marco, SottoCasa

Predicati: Vende, Negozio, Supermarket, Birra, Alcolico, Minorenne, Possiede, Reo

Funzioni: nessuna

Interpretazione:

- Marco: il protagonista della storia
- Sotto Casa: minimarket presso il quale Marco fa acquisti abitualmente
- Vende: relazione ternaria che lega chi ha venduto qualcosa a chi
- Negozio: relazione unaria che indica la proprietà di essere un negozio
- ...

Una KB di esempio

La legge dice che è un reato per un negozio vendere alcolici a un minorenne.

Negozio(x) ∧ Vende(x, y, z) ∧ Alcolico(y) ∧ Minorenne(z) ⇒ Reo(x)

Una KB di esempio

La legge dice che è un reato per un negozio vendere alcolici a un minorenne.

Negozio(x) ∧ Vende(x, y, z) ∧ Alcolico(y) ∧ Minorenne(z) ⇒ Reo(x)

Marco possiede della birra.

∃x Possiede(Marco, x) ∧ Birra(x)

Applicando la regola di istanziazione dell' esistenziale (EI) questa formula si traduce nelle due formule atomiche:

Possiede(Marco, B), Birra(B)

B è una nuova costante, non utilizzata altrove. È la birra comperata da Marco. Le diamo il nome "B". Le costanti come B sono dette costanti di Skolem.

Una KB di esempio

La legge dice che è un reato per un negozio vendere alcolici a un minorenne.

Negozio(x) ∧ Vende(x, y, z) ∧ Alcolico(y) ∧ Minorenne(z) ⇒ Reo(x)

Marco possiede della birra.

Possiede(Marco, B), Birra(B)

Tale birra gli è stata venduta del minimarket Sotto Casa. Esprimiamo che se Marco possiede della birra l' ha comperata al minimarket

Possiede(Marco, x) ∧ Birra(x) ⇒ Vende(SottoCasa, x, Marco)

La birra è un alcolico

Birra(x) ⇒ Alcolico(x)

Una KB di esempio

Negozio(x) ∧ Vende(x, y, z) ∧ Alcolico(y) ∧ Minorenne(z) ⇒ Reo(x)

Possiede(Marco, B), Birra(B)

Possiede(Marco, x) ∧ Birra(x) ⇒ Vende(SottoCasa, x, Marco)

Birra(x) ⇒ Alcolico(x)

Poi ancora

Minimarket(SottoCasa)

Minimarket(x) ⇒ Negozio(x)

Minorenne(Marco)

La KB in clausole di Horn

C1) Negozio(x) ∧ Vende(x, y, z) ∧ Alcolico(y) ∧ Minorenne(z) ⇒ Reo(x)

C2) Possiede(Marco, B)

C3) Birra(B)

C4) Possiede(Marco, x) ∧ Birra(x) ⇒ Vende(SottoCasa, x, Marco)

C5) Birra(x) ⇒ Alcolico(x)

C6) Minimarket(SottoCasa)

C7) Minimarket(x) ⇒ Negozio(x)

C8) Minorenne(Marco)

NOTA: questa KB è costituita da **clausole di Horn del prim'ordine**. In particolare si tratta di clausole che non fanno uso di funzioni. Questa particolare classe di KB è detta **DATALOG**

La KB in clausole di Horn

- È l' unica modellazione possibile?
- **Probabilmente no**
- In generale dato un problema di rappresentazione esisteranno diverse alternative
- George Box (statistico):
  "Now it would be very remarkable if any system existing in the real world could be exactly represented by any simple model. However, cunningly chosen parsimonious models often do provide remarkably useful approximations. For example, the law PV = RT relating pressure P, volume V and temperature T of an "ideal" gas via a constant R is not exactly true for any real gas, but it frequently provides a useful approximation and furthermore its structure is informative since it springs from a physical view of the behavior of gas molecules. For such a model there is no need to ask the question "Is the model true?". If "truth" is to be the "whole truth" the answer must be "No". The only question of interest is "Is the model illuminating and useful?"

Riassunto talvolta in: **tutti i modelli sono "sbagliati", alcuni sono utili**

Forward Chaining

- L' algoritmo è simile a quello usato nel caso proposizionale
- Le variabili richiedono qualche cautela: un fatto è la rinomina di un altro se sono identici tranne nei nomi delle variabili

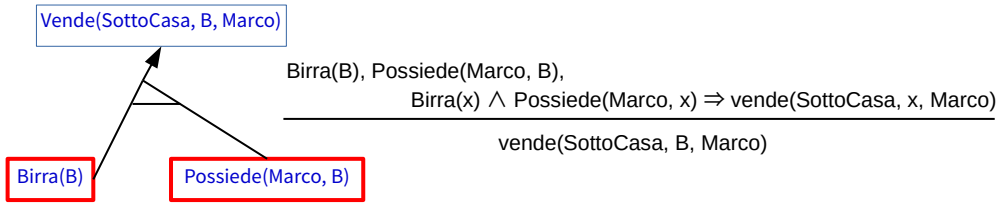
Esempio di rinomina:
Fratello(John, x) e Fratello(John, y)

- **Nel calcolo proposizionale un fatto inferito viene aggiunto alla KB solo se non vi compare già**
- **In FOL bisogna fare un test più complesso: un fatto è aggiunto alla KB solo se non è una rinomina di uno già presente**

Forward chaining

C2) Possiede(Marco, B)
C3) Birra(B)
C4) **Possiede(Marco, x) ∧ Birra(x)** ⇒ Vende(SottoCasa, x, Marco)

Si applica il modus ponens generalizzato

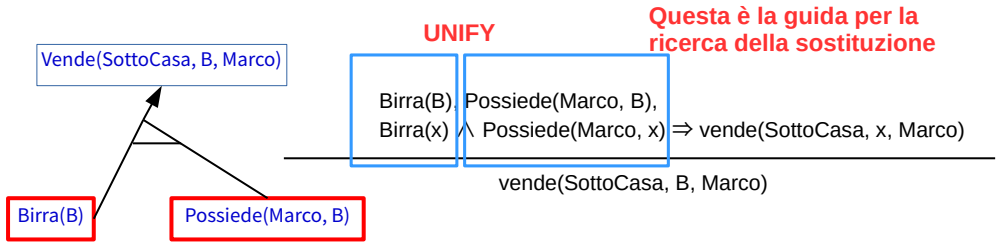


Parto da fatti che rendono vero l'antecedente (la premessa) di una implicazione

Forward chaining

C2) Possiede(Marco, B)
C3) Birra(B)
C4) **Possiede(Marco, x) ∧ Birra(x)** ⇒ Vende(SottoCasa, x, Marco)

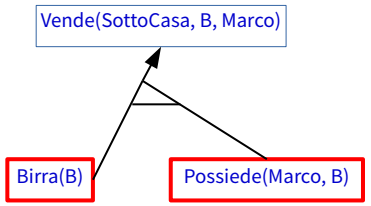
Si applica il modus ponens generalizzato



Parto da fatti che rendono vero l'antecedente (la premessa) di una implicazione e costruisco un grafo AND-OR

Forward chaining

C2) Possiede(Marco, B)
C3) Birra(B)
C4) **Possiede(Marco, x) ∧ Birra(x)** ⇒ Vende(SottoCasa, x, Marco)



Questa inferenza è possibile legando la variabile x a B.

L'inferenza procede creando delle **sostituzioni** per unificazione delle formule:
θ = { x/B }

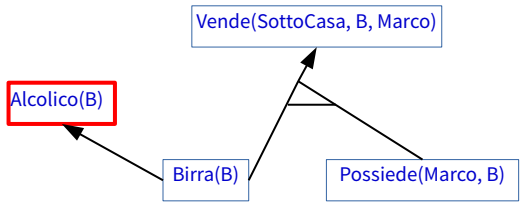
Birra(B)/B = Birra(x)/B
Possiede(Marco,x)/B =
Possiede(Marco, B)/B

Parto da fatti che rendono vero l'antecedente (la premessa) di una implicazione e costruisco un grafo AND-OR

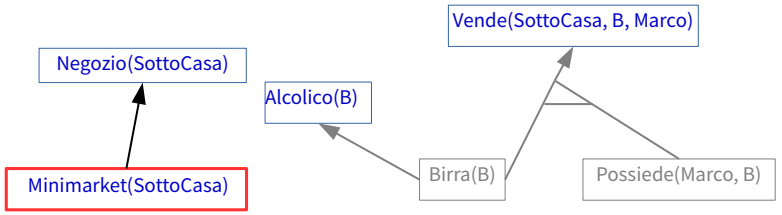
Forward Chaining

C2) Possiede(Marco, B)
C3) Birra(B)
C4) Possiede(Marco, x) ∧ Birra(x) ⇒ Vende(SottoCasa, x, Marco)
C5) Birra(x) ⇒ Alcolico(x)

Non si propagano solo i valori di verità. Si propagano anche le sostituzioni, che fanno da collante fra le diverse applicazioni del MPG

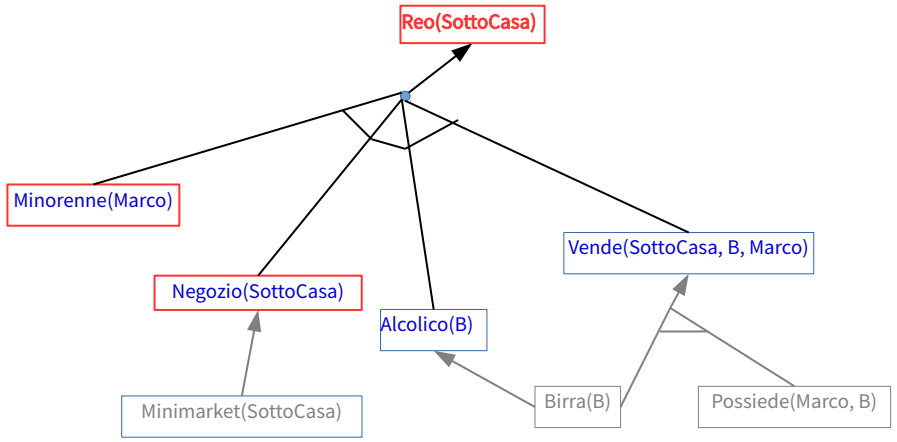


Uso del forward chaining



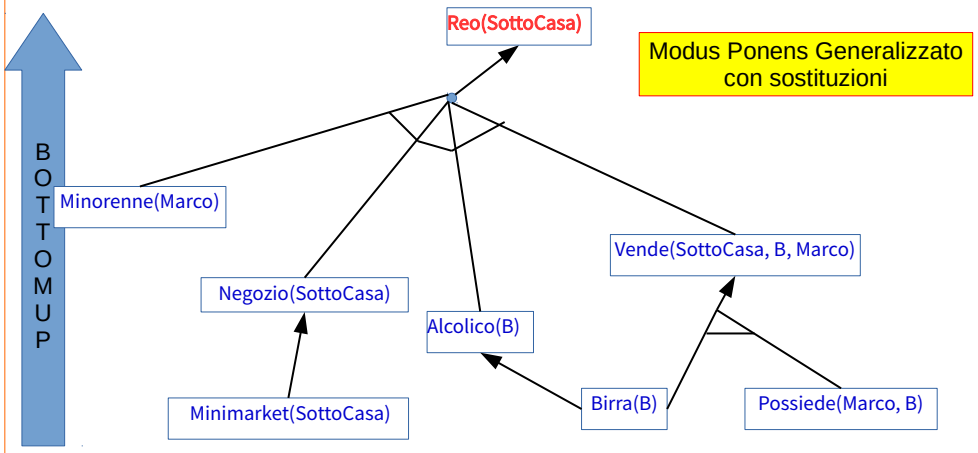
Uso del forward chaining

Inferisco che Sotto Casa è reo della vendita



Uso del forward chaining

Grafo AND-OR completo



Proprietà del forward chaining

- **Correttezza**
  - l'algoritmo e **corretto** perché si basa sulla regola di inferenza corretta GMP
- **Completezza**
  - È **completo** per KB costituite da **clausole di Horn**:
    - Se la KB è **DATALOG**, è **completo e termina**
    - Se la KB **prevede funzioni** – teorema di Herbrand – l'algoritmo **può non terminare** se la risposta non è implicata

Backward Chaining (BC)

- Come nel caso proposizionale il **ragionamento è guidato dall'obiettivo**
- L'obiettivo viene inserito in uno **stack**
- Poi iterativamente si estrae un obiettivo dallo stack e si cercano **le clausole la cui testa è unificabile** con l'obiettivo:
  - Quando un obiettivo unifica con un fatto (clausola con corpo vuoto), viene semplicemente rimosso (è risolto)
  - Altrimenti per ogni clausola che soddisfa quanto sopra si avvia un procedimento ricorsivo che inserisce nella pila le premesse della clausola, in cui le variabili saranno state debitamente sostituite
- Quando la pila è vuota si termina con successo
- Altrimenti se non è possibile applicare altre inferenze si termina con fallimento

Esempio

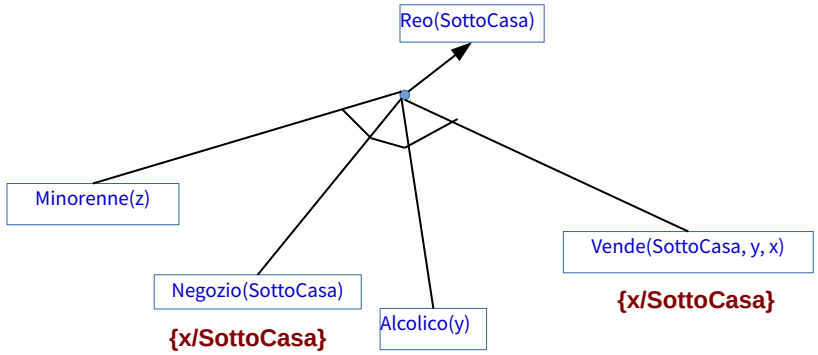
Stack: Reo(SottoCasa)

Unifica con la testa della clausola tramite **θ1 = {x/SottoCasa}**:
C1) Negozio(x) ∧ Vende(x, y, z) ∧ Alcolico(y) ∧ Minorenne(z) ⇒ Reo(x)

Si applica la sostituzione al corpo di C1, tutti i congiunti diventano obiettivi da dimostrare e vengono inseriti nello stack:
Stack: Negozio(SottoCasa), Vende(SottoCasa, y, z), Alcolico(y), Minorenne(z)

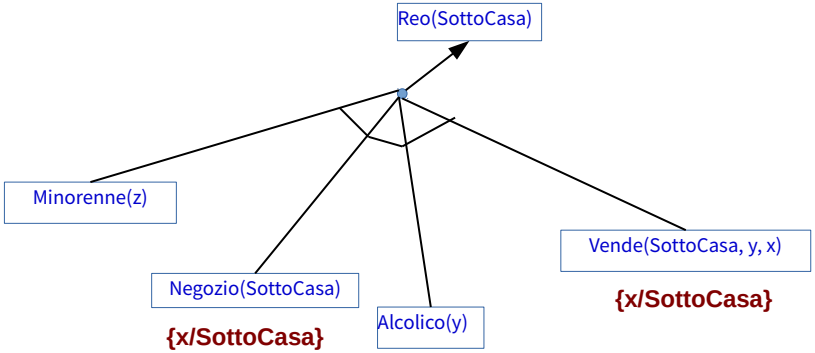
Esempio

C1) Negozio(x) ∧ Vende(x, y, z) ∧ Alcolico(y) ∧ Minorenne(z) ⇒ Reo(x)
Stack: Negozio(SottoCasa), Vende(SottoCasa, y, z), Alcolico(y), Minorenne(z)



Esempio

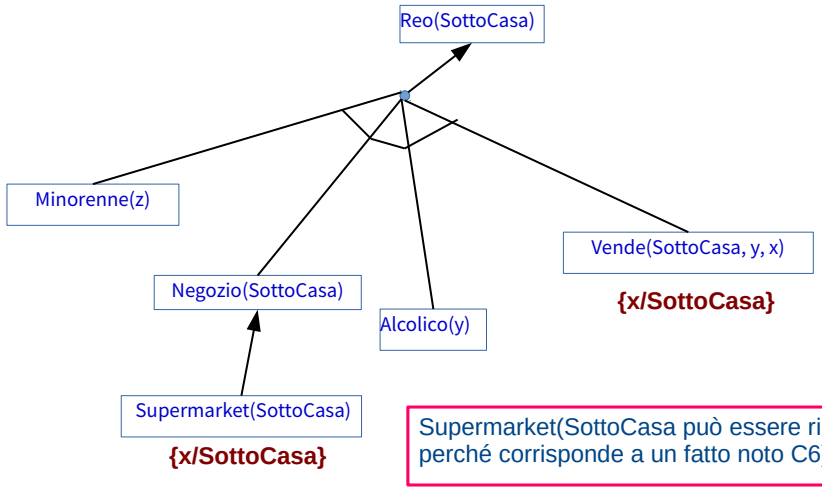
C1) Negozio(x) ∧ Vende(x, y, z) ∧ Alcolico(y) ∧ Minorenne(z) ⇒ Reo(x)
Stack: Vende(SottoCasa, y, z), Alcolico(y), Minorenne(z)
Risolviamo ricorsivamente Negozio(SottoCasa)



unifica con la testa della clausola C7) Supermarket(x) ⇒ Negozio(x)

Esempio

C1) Negozio(x) ∧ Vende(x, y, z) ∧ Alcolico(y) ∧ Minorenne(z) ⇒ Reo(x)
Stack: ~~**Supermarket(SottoCasa)**~~, Vende(SottoCasa, y, z), Alcolico(y), Minorenne(z)

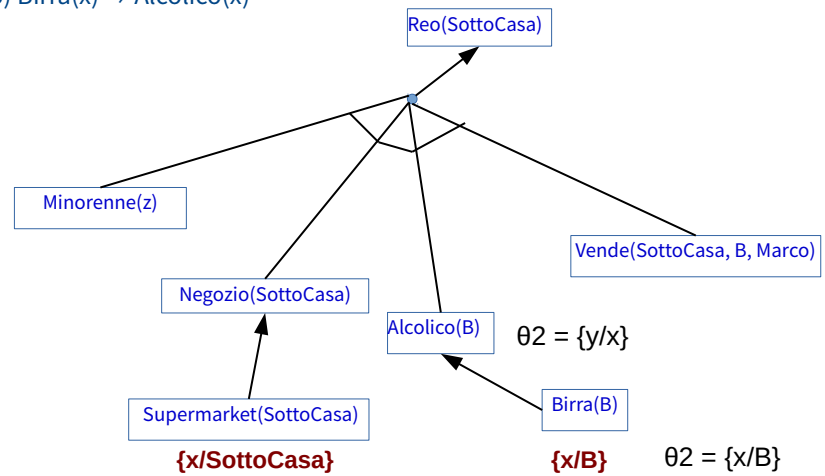


Supermarket(SottoCasa può essere rimosso perché corrisponde a un fatto noto C6)

Esempio

C3) Birra(B)

C5) Birra(x) ⇒ Alcolico(x)



Composizione delle sostituzioni

- l' algoritmo BC applica la composizione delle sostituzioni:

  COMPOSE(θ1, θ2) = θ3

- Vale la proprietà:

  SUBST(**COMPOSE(θ1, θ2)**, F) = SUBST(θ2, SUBST(θ1, F))

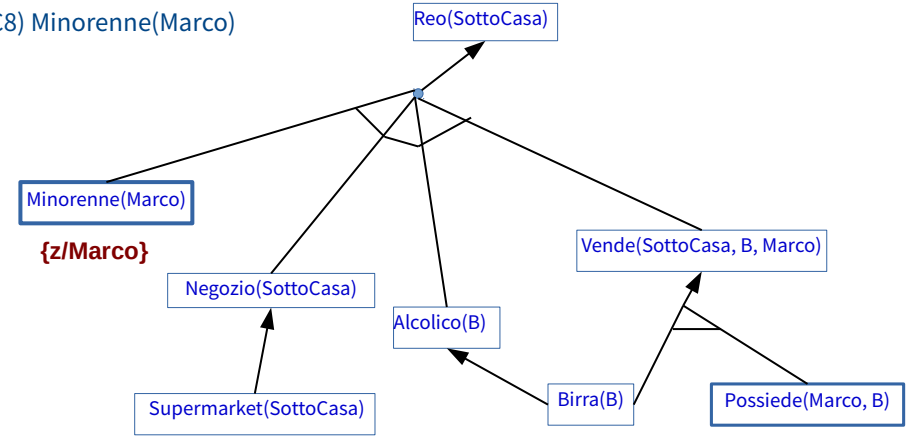
- Nell' esempio abbiamo:
  Negozio(x) ∧ Vende(x, y, z) ∧ Alcolico(y) ∧ Minorenne(z) ⇒ Reo(x)
  Birra(x) ⇒ Alcolico(x)
  Birra(B)
- Occorre concatenare le sostituzioni per effettuare l' inferenza corretta

Esempio

C2) Possiede(Marco, B)

C4) Possiede(Marco, x) ∧ Birra(x) ⇒ Vende(SottoCasa, x, Marco)

C8) Minorenne(Marco)



Valutazionde del backward chaining

- **Corretto**
- **Incompleto** in quanto implementa una *strategia depth-first*, può incorrere in loop infiniti e generare stati ripetuti
- Come la versione per il calcolo proposizionale è più efficiente del forward chaining

Uso della concatenazione all'indietro nella programmazione logica

- **Prolog**:
  un programma Prolog è un insieme di clausole di Horn del prim' ordine
- L' inferenza fa uso di backward chaining
- **Sintassi**:
  - Testa :- Corpo.
- Corpo: congiunzione di letterali dove ∧ è rappresentato da ','
- Le variabili iniziano per maiuscola, le costanti per minuscola
- Predicati **assert** e **retract** per modificare la KB

SWI-Prolog

- http://www.swi-prolog.org/
- Esempi e tutorial: http://www.learnprolognow.org/
- Programmi scritti in file con estensione .pl
  (Esempio: concatena.pl)
- Avvio da linea di comando: swipl
- Terminazione esecuzione: halt. (nota il '.' è necessario)
- Caricare un programma, vari modi ad es: [concatena]. (nota il '.' è necessario)

Liste in Prolog

- Viene usata la struttura dati **lista**
- Una lista è rappresentata come un elenco separato da virgole e incluso fra parentesi quadre, esempio: **[a, b, c, d]**
- Lista vuota: **[]**
- In una lista si possono identificare inizio e seguito usando '|':
  [A | B] = lista strutturata in parte unificata alla variabile A seguita dalla parte unificata alla variabile B

Esempio di programma Prolog

- Scriviamo il programma che concatena due liste

Esempio di programma Prolog

- Scriviamo il programma che concatena due liste
- La **lista vuota** concatenata a **una lista** dà quella lista:
  append([], X, X).

Esempio di programma Prolog

- Scriviamo il programma che concatena due liste
- La lista vuota concatenata a una lista dà quella lista:
  append([], X, X).
- Se concateno due liste che **non sono vuote** occorre usare una **clausola ricorsiva**:
  append([A|X], Y, [A|Z]) :- append(X, Y, Z).

Fine del programma

Esempi d'uso:

```
?- append([], [a], X).
X = [a].

?- append([c], [a,b], X).
X = [c, a, b].

?- append([c], [], X).
X = [c].

?- append([[a, b]], [c], X).
X = [[a, b], c].

?- append(X, Y, [[a, b]]).
X = [],
```


Altro esempio

Date le seguenti clausole per tradurre numeri in tedesco in numeri in inglese, scrivere un programma Prolog per tradurre una lista di numeri in tedesco in una lista delle rispettive traduzioni in inglese.

```
tran(eins,one).
tran(zwei,two).
tran(drei,three).
tran(vier,four).
tran(fuenf, ve).
tran(sechs,six).
tran(sieben,seven).
tran(acht,eight).
tran(neun,nine).
```

Altro esempio

```
tran(eins,one).
tran(zwei,two).
...
tran(acht,eight).
tran(neun,nine).

listtran([],[]).
listtran([X|Y], [X|Z]) :- tran(_, X), listtran(Y, Z).
listtran([X|Y], [W|Z]) :- tran(X, W), listtran(Y, Z).
```

Il programma riconosce liste miste di termini in inglese in tedesco e traduce solo i secondi

Altro esempio

```
tran(eins,one).
tran(zwei,two).
...
tran(acht,eight).
tran(neun,nine).

listtran([],[]).
listtran([X|Y], [X|Z]) :- tran(_, X), listtran(Y, Z).
listtran([X|Y], [W|Z]) :- tran(X, W), listtran(Y, Z).
```

Il programma riconosce liste miste di termini in inglese in tedesco e traduce solo i secondi

Esempi d'uso

```
?- listtran([eins, zwei],X).
X = [one, two].

?- listtran([eins, zwei, three],X).
X = [one, two, three].
```


Esempi d'uso

```
?- listtran([eins, zwei],X).
X = [one, two].

?- listtran([eins, zwei, three],X).
X = [one, two, three].
```

Però se la query è quale lista venga tradotta in [eight, nine], non tradurrà all'indietro:

```
?- listtran(X, [eight, nine]).
X = [eight, nine].
```

L'ordine delle clausole conta

```
tran(eins,one).
tran(zwei,two).
...
tran(acht,eight).
tran(neun,nine).

listtran([],[]).
listtran([X|Y], [W|Z]) :- tran(X, W), listtran(Y, Z).
listtran([X|Y], [X|Z]) :- tran(_, X), listtran(Y, Z).
```

Se invertiamo le due clausole ricorsive …

Si traduce anche all'indietro!

```
?- listtran(X, [eight, nine]).
X = [acht, neun].
```

Ma non le liste miste

```
?- listtran(X, [acht, eight, nine]).
false.
```


Negazione per fallimento

- Prolog permette l'uso della **negazione**, esempio:

```
vivo(X) :- not morto(X).
```

- Un letterale come "`not F(X)`" è vero quando non si può dimostrare che sia falso, cioè se "`F(X)`" non è inferibile in qualche modo dalla KB
- La clausola va quindi letta come: X è vivo se non si può dimostrare che è morto
- In swi-prolog si usa la "negazione per fallimento":

```
vivo(X) :- not morto(X).
```