Esercitazione n. 3 - Inferenza IN PROP e FOL

* Da svolgere durante l'esercitazione.

Formalizzazione e inferenza nel caso proposizionale

Unicorno

"*Se l'unicorno è mitico allora è immortale*. *Se non è mitico allora è un mammifero mortale. Se l'unicorno è o immortale o un mammifero allora ha le corna. L'unicorno è magico se ha le corna*."
Il fatto che *l'unicorno è mitico*, che *l'unicorno è magico* e che *l'unicorno ha le corna*, sono conseguenza logica dei seguenti fatti?

Vogliamo stabilire se:
KB ⊨ Mitico
KB ⊨ Magico
KB ⊨ Corna

Formalizzazione:

1. Mitico ⇒ ¬Mortale
2. ¬Mitico ⇒ Mortale ∧ Mammifero
3. ¬Mortale ∨ Mammifero ⇒ Corna
4. Corna ⇒ Magico

Algoritmo TT-Entails?

KB ⊨ Mitico

TT-CHECK-ALL(KB, Mitico, [Mitico, Mortale, Mammifero, Corna], [])
  TT-CHECK-ALL(KB, Mitico, [Mortale, Mammifero, Corna], [Mitico=T])
    TT-CHECK-ALL(KB, Mitico, [Mammifero, Corna], [Mitico=T, Mortale=T])

‘Mitico’ non è conseguenza logica, ‘Corna’ e ‘Magico’ lo sono. Infatti costruendo la tabella di verità e limitandoci alle interpretazioni che sono modelli per la base di conoscenza, dalle colonne 1, 4 e 5 trovo che ‘Mitico’ non è sempre vero, mentre ‘Corna’ e ‘Magico’ lo sono.

mitico	mortale	mammifero	corna	magico
f	t	t	t	t
t	f	t	t	t
t	f	f	t	t

Usiamo adesso gli algoritmi per SAT.

Trasformazione in forma a clausole. Abbreviazioni usate nel seguito.

Mitico	Mi
Mortale	Mo
Mammifero	Mm
Magico	Ma
Corna	Co

1. Mitico ⇒ ¬Mortale	1. {¬Mi, ¬Mo}
2. ¬Mitico ⇒ Mortale ∧ Mammifero Mitico ∨ (Mortale ∧ Mammifero)	2.1 { Mi, Mo} 2.2 { Mi, Mm}
3. ¬Mortale ∨ Mammifero ⇒ Corna ¬(¬Mortale ∨ Mammifero) ∨ Corna (Mortale ∧ ¬Mammifero) ∨ Corna	3.1 { Mo, Co} 3.2 {¬Mm, Co}
4. Corna ⇒ Magico	4. {¬Co, Ma}

KB ⊨ Mitico sse KB ∪ {¬Mitico) è insoddisfacibile

DPLL

1. {¬Mi, ¬Mo} 2. { Mi, Mo} 3 { Mi, Mm} 4 { Mo, Co} 5 {¬Mm, Co} 6 {¬Co, Ma} 7 {¬Mi}

La tabella sotto fornisce un metodo ordinato di svolgere questo tipo di dimostrazioni. Ad ogni passo si evidenzia: l’ultimo assegnamento fatto, le clausole soddisfatte (ok) le clausole semplificate sulla base degli assegnamenti fatti (se A=t una clausola che contiene ¬A può essere vera solo in virtù degli altri letterali e quindi può essere semplificata, stessa cosa se A=f si semplificano le clausole che contengono A).

	Assign	{¬Mi, ¬Mo}	{ Mi, Mo}	{ Mi, Mm}	{ Mo, Co}	{¬Mm, Co}	{¬Co, Ma}	{¬Mi}
Passo 1	Ma=t (puro)	{¬Mi, ¬Mo}	{ Mi, Mo}	{ Mi, Mm}	{ Mo, Co}	{¬Mm, Co}	ok	{¬Mi}
Passo 2	Co=t (puro)	{¬Mi, ¬Mo}	{ Mi, Mo}	{ Mi, Mm}	ok	ok	ok	{¬Mi}
Passo 3	Mm=t (puro)	{¬Mi, ¬Mo}	{ Mi, Mo}	ok	ok	ok	ok	{¬Mi}
Passo 4	Mi=f (unit)	ok	{Mo}	ok	ok	ok	ok	ok
Passo 5	Mo=t (puro)	ok	ok	ok	ok	ok	ok	ok

Abbiamo trovato un modello, la KB di partenza (che include il goal negato) è soddisfacibile, quindi Mi **non** è conseguenza logica. Altra traccia: questa volta cerchiamo di vedere se Ma è conseguenza logica.

	Assign	{¬Mi, ¬Mo}	{ Mi, Mo}	{ Mi, Mm}	{ Mo, Co}	{¬Mm, Co}	{¬Co, Ma}	{¬Ma}
Passo 1	Ma=f (unit.)	{¬Mi, ¬Mo}	{ Mi, Mo}	{ Mi, Mm}	{ Mo, Co}	{¬Mm, Co}	{¬Co}	ok
Passo 2	Co=f (unit.)	{¬Mi, ¬Mo}	{ Mi, Mo}	{ Mi, Mm}	{ Mo}	{¬Mm}	ok	ok
Passo 3	Mo=t (unit.)	{¬Mi}	ok	{ Mi, Mm}	ok	{¬Mm}	ok	ok
Passo 4	Mi=f (unit.)	ok	ok	{ Mm}	ok	{¬Mm}	ok	ok

In questo caso non riusciamo a trovare un modello. Gli assegnamenti sono tutti obbligati.
Prima di concludere che non esistono modelli in genere però dobbiamo provare tutti i casi.

WALKSAT (una possibile soluzione, altre generabili dal codice)

Sappiamo che Mi non è conseguenza logica quindi presumibilmente aggiungendo il negato dovremmo essere in grado di trovare un modello. Se lo troviamo ne abbiamo una conferma.

1. {¬Mi, ¬Mo} 2. { Mi, Mo} 3 { Mi, Mm} 4 { Mo, Co} 5 {¬Mm, Co} 6 {¬Co, Ma} 7 {¬Mi}

Partiamo con un assegnamento a caso.
[Ma=F; Mi=F; Mo=F; Mm=F; Co=F]

1.{¬Mi, ¬Mo}	2.{Mi, Mo}	3.{Mi, Mm}	4.{ Mo, Co}	5.{¬Mm, Co}	6.{¬Co, Ma}	7. {¬Mi}
T	F	F	F	T	T	T

Prendiamo la 2. Passo di ottimizzazione.
[Ma=F; Mi=T; Mo=F; Mm=F; Co=F] 2 clausole insoddisfatte flippando Mi;
[Ma=F; Mi=F; Mo=T; Mm=F; Co=F] 1 clausole insoddisfatte flippando Mo. Scegliamo Mo=T.

1.{¬Mi, ¬Mo}	2.{Mi, Mo}	3.{Mi, Mm}	4.{Mo, Co}	5.{¬Mm, Co}	6.{¬Co, Ma}	7.{¬Mi}
T	T	F	T	T	T	T

Consideriamo la 3. Flippiamo Mm:
[Ma=F; Mi=F; Mo=T; Mm=T; Co=F]

1.{¬Mi, ¬Mo}	2.{Mi, Mo}	3.{Mi, Mm}	4.{Mo, Co}	5.{¬Mm, Co}	6.{¬Co, Ma}	7.{¬Mi}
T	T	T	T	F	T	T

[Ma=F; Mi=F; Mo=T; Mm=T; Co=T]

1.{¬Mi, ¬Mo}	2.{Mi, Mo}	3.{Mi, Mm}	4.{Mo, Co}	5.{¬Mm, Co}	6.{¬Co, Ma}	7.{¬Mi}
T	T	T	T	T	F	T

[Ma=T; Mi=F; Mo=T; Mm=T; Co=T]

1.{¬Mi, ¬Mo}	2.{Mi, Mo}	3.{Mi, Mm}	4.{Mo, Co}	5.{¬Mm, Co}	6.{¬Co, Ma}	7.{¬Mi}
T	T	T	T	T	T	T

Anche in questo caso abbiamo trovato un modello.

Risoluzione con PROP

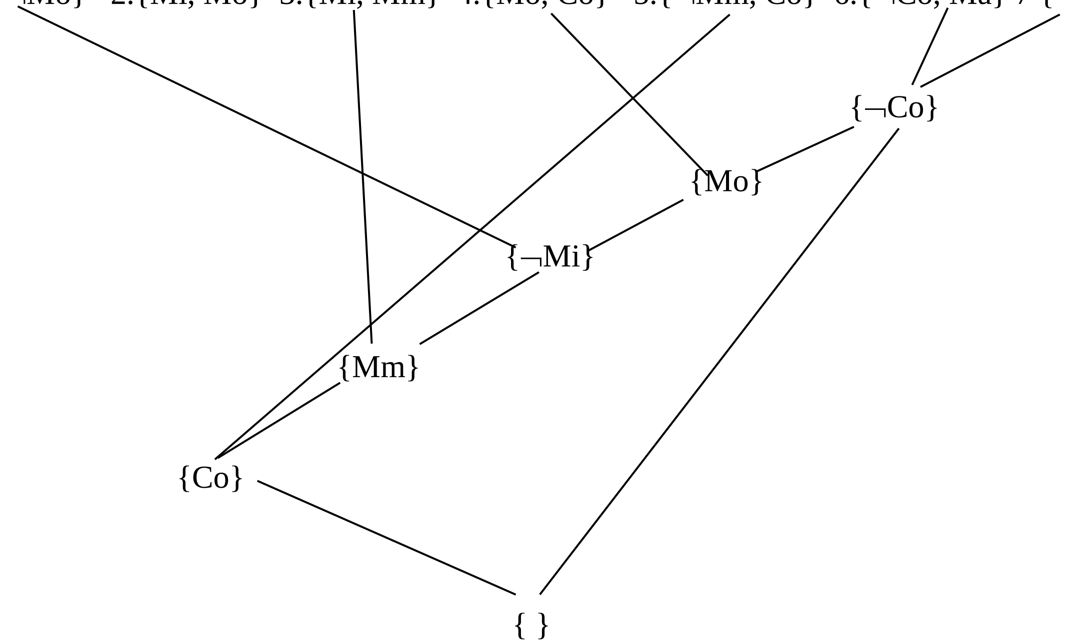
1. {¬Mi, ¬Mo} 2. { Mi, Mo} 3 { Mi, Mm} 4 { Mo, Co} 5 {¬Mm, Co} 6 {¬Co, Ma} 7 {¬Mi}

{¬Mo, Mo} Taut.

Non si arriva a dimostrare {}. Infatti Mi non è conseguenza logica della KB. In particolare, si noti che le prime due clausole non danno la clausola vuota.

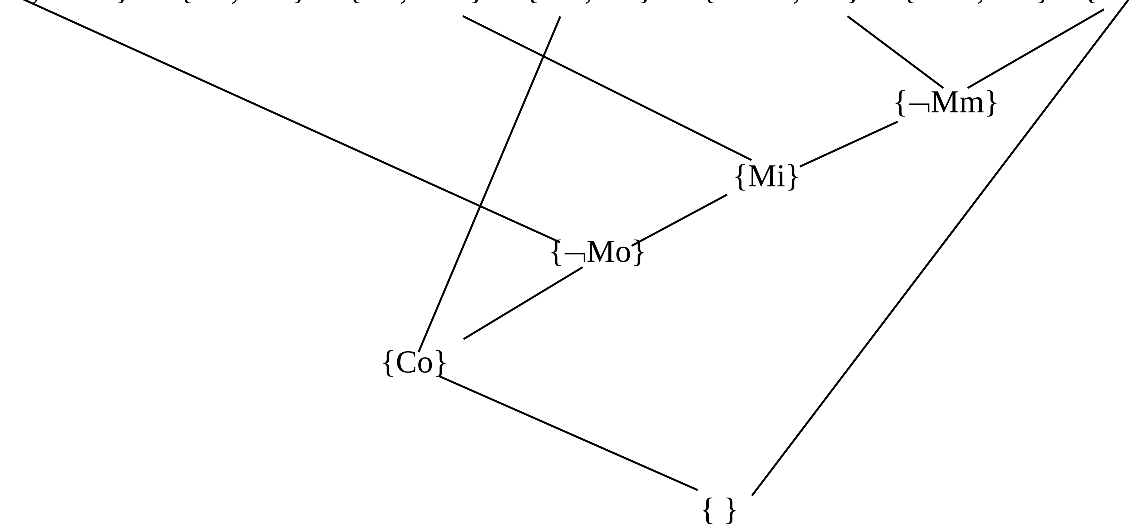
2. Proviamo con Ma.

1.{¬Mi, ¬Mo} 2.{Mi, Mo} 3.{Mi, Mm} 4.{Mo, Co} 5.{¬Mm, Co} 6.{¬Co, Ma} 7 {¬Ma}



3. Proviamo con Co usando una strategia lineare>

1.{¬Mi, ¬Mo} 2.{Mi, Mo} 3.{Mi, Mm} 4.{Mo, Co} 5.{¬Mm, Co} 6.{¬Co, Ma} 7 {¬Co}



È anche lineare da input.

Alcuni esempi di formalizzazioni in FOL

Da usare per autovalutazione

I soli cani gialli sono simpatici

A: Tra tutte le cose, solo i cani gialli sono simpatici
B: Tra i cani, solo quelli gialli sono simpatici.
Sono tutti simpatici? Direi di no.

A $\forall x\ Simpatico(x) \Rightarrow Cane(x) \wedge Giallo(x)$
B $\forall x\ Cane(x) \Rightarrow (Simpatico(x) \Rightarrow Giallo(x))$ equivalente a
$\forall x\ Cane(x) \wedge Simpatico(x) \Rightarrow Giallo(x)$

Non va invece bene:
$\forall x\ Cane(x) \wedge Giallo(x) \Rightarrow Simpatico(x)$
Perché non esclude che ci siano altre cose simpatiche ad esempio i gatti rossi, quindi non rende "*I soli cani gialli …*" ma semmai significa "*Tutti i cani gialli …*".

Potrebbe essere accettabile:
$\forall x\ Cane(x) \wedge Giallo(x) \Leftrightarrow Simpatico(x)$
Ma allora direbbe anche che **tutti** i cani gialli sono simpatici. Ma non mi sembra che sia implicito nella frase.

Tutte le scimmie sono fuggite su un albero

Tradurre i seguenti enunciati nella logica dei predicati; in caso di frase ambigua, e la prima lo è, dare le due possibile trascrizioni.

a. Tutte le scimmie sono fuggite su un albero.
A1. $\forall x\ (scimmia(x) \Rightarrow (\exists y\ albero(y) \wedge fuggita\_su(x, y)))$
Per ogni scimmia esiste un albero su cui questa è fuggita

A2. $\exists y\ (albero(y) \wedge (\forall x\ scimmia(x) \Rightarrow fuggita\_su(x, y)))$
Esiste un albero su cui tutte le scimmie sono fuggite

Altre soluzioni più fantasiose non sono da considerarsi rappresentative.

Nessuno ama un professore a meno che questi non sia intelligente

Le letture che se ne possono dare sono:

"Nessuno ama un professore non intelligente"
"Se uno ama un professore allora questi è necessariamente intelligente"
"I professori non intelligenti non sono amati da nessuno"

e tanti altri modi ...

Queste letture danno origine rispettivamente a queste formalizzazioni:

$\neg\exists x\ .\ (\exists y\ Professore(y) \wedge Ama(x, y) \wedge \neg\ Intelligente(y))$

$\forall x\ \forall y\ (Ama(x, y) \wedge Professore(y) \Rightarrow Intelligente\ (y))$
$\forall y\ Professore(y) \wedge\ \neg\ Intelligente(y) \Rightarrow \forall x\ \ \neg Ama\ (x, y)$

che sono tutte equivalenti. Infatti, se provate a portarle in forma a clausole si ottiene in tutti e tre i casi

A. $\{\neg Professore(y), \neg Ama(x, y), Intelligente(y)\}$

Perché invece questa non va bene?

$\forall x\ \exists y\ (Professore(y) \wedge Ama(x, y) => Intelligente(y))$

la semantica ci dice che per ogni individuo x ce n'è un altro (che può essere scelto in dipendenza da x) che ha la proprietà di "non essere professore, o non essere amato da x, o essere intelligente".

La forma a clausole è infatti:

B. $\{\neg Professore(f(x)), \neg Ama(x, f(x)), Intelligente(f(x)\}$

La prima era più forte perché ci diceva che per ogni individuo x, comunque scelgo l'altro individuo y questo ha la proprietà di "non essere professore, o non essere amato da x, o essere intelligente". Quindi sono diverse e la prima direi che è da preferire alla seconda poiché quando si dice "Nessuno ama un professore a meno che questo non sia intelligente" per "un professore" si intende un qualsiasi professore (il generico professore, o tutti) non uno particolare.

Chi va con lo zoppo impara a zoppicare

Quale zoppo? Uno particolare? uno generico? Stessa forma a clausole?

$\forall x\ (\exists y\ (Zoppo(y) \wedge VaCon(x, y))) => IZ(x)$	$\forall x\ \forall y\ ((Zoppo(y) \wedge VaCon(x, y))) => IZ(x)$
$\forall x\ (\neg\exists y\ (Zoppo(y) \wedge VaCon(x, y)) \vee IZ(x))$	$\forall x\ \forall y\ \neg(Zoppo(y) \wedge VaCon(x, y)) \vee IZ(x)$
$\forall x\ \forall y\ \neg(Zoppo(y) \wedge VaCon(x, y)) \vee IZ(x)]$	$\forall x\ \forall y\ \neg Zoppo(y) \vee\ \neg VaCon(x, y) \vee IZ(x)$
$\forall x\ \forall y\ (\neg Zoppo(y) \vee \neg VaCon(x, y) \vee IZ(x))$	$\{\neg Zoppo(y), \neg VaCon(x, y), IZ(x)\}$
$\{\neg Zoppo(y), \neg VaCon(x, y), IZ(x)\}$	

Si, ma solo perché y non compare nel conseguente.

Il miglior voto a IA è stato migliore del miglior voto a PA.

Vocabolario: >, la funzione Voto(x, y) per il voto di x nella materia y

$\exists x\ \forall y\ (Voto(x, IA) \geq Voto(y, IA)) \wedge (Voto(x, IA) \geq Voto(y, PA))$
Esiste un voto a IA che è maggiore di tutti i voti a IA e che è maggiore di tutti i voti a PA

$\exists x\ \forall y\ (Voto(x, IA) \geq Voto(y, PA))$
Esiste un voto a IA che è maggiore di tutti i voti a PA (quindi anche del migliore)

Interpretare formalizzazioni

Quale/i delle seguenti sono formalizzazioni corrette della seguente frase:

Ogni cane che ama uno dei suoi fratelli è felice

a. $\forall x\ \text{Cane}(x) \wedge (\exists y\ \text{Fratello}(y, x) \wedge \text{Ama}(x, y)) \Rightarrow \text{Felice}(x)$
b. $\forall x\ \forall y\ \text{Cane}(x) \wedge \text{Fratello}(y, x) \wedge \text{Ama}(x, y) \Rightarrow \text{Felice}(x)$
c. $\forall x\ \text{Cane}(x) \wedge (\forall y\ \text{Fratello}(y, x) \wedge \text{Ama}(x, y)) \Rightarrow \text{Felice}(x)$
d. $\forall x\ \text{Cane}(x) \wedge (\forall y\ \text{Fratello}(y, x) \Rightarrow \text{Ama}(x, y)) \Rightarrow \text{Felice}(x)$

Se qualcuna di queste è scorretta si spieghi perché, dandone una lettura in linguaggio naturale.

a. OK. È sufficiente che ne ami uno.
b. OK. Anche questa va bene. Di fatto è equivalente alla prima. Hanno la stessa FC.
    Nota.
    $\forall x\ \forall y\ (F(y) \Rightarrow G(x))$ -> $\{\neg F(x), G(x)\}$
    $\forall x\ (\exists y\ F(y) \Rightarrow G(x))$ -> $\forall x\ (\neg \exists y\ F(y) \vee G(x))$ -> $\forall x\ \forall y\ \neg F(y) \vee G(x)$ ->$\{\neg F(x), G(x)\}$
c. NO. Troppo forte
d. NO. Troppo forte

Unificazione

Potete esercitarvi con l'unificazione anche utilizzando il Notebook che trovate all'indirizzo http://attardi-4.di.unipi.it:8000 . Per usufruire del servizio autenticatevi con l'account gSuite di Ateneo con le credenziali da studente.

a) P(A, B, B) , P(x, y, z)

```
>>> unify(P(A, B, B), P(x, y, z), {})
unify(P(A, B, B), P(x, y, z), {})
unify(P(A, B, B), P(x, y, z), {})
unify('P', 'P', {})
unify([A, B, B], [x, y, z], {})
unify(A, x, {})
unify_var(x, A, {})
unify([B, B], [y, z], {x: A})
unify(B, y, {x: A})
unify_var(y, B, {x: A})
unify([B], [z], {x: A, y: B})
unify(B, z, {x: A, y: B})
unify_var(z, B, {x: A, y: B})
unify([], [], {z: B, x: A, y: B})
{z: B, x: A, y: B}
```

a) Q(y, G(A, B)), Q(G(x, x), y)

```
>>> unify(Q(y, G(A, B)), Q(G(x, x),y), {})
```

```
unify(Q(y, G(A, B)), Q(G(x, x), y), {})
unify(Q(y, G(A, B)), Q(G(x, x), y), {})
unify('Q', 'Q', {})
unify([y, G(A, B)], [G(x, x), y], {})
unify(y, G(x, x), {})
unify_var(y, G(x, x), {})
unify([G(A, B)], [y], {y: G(x, x)})
unify(G(A, B), y, {y: G(x, x)})
unify_var(y, G(A, B), {y: G(x, x)})
unify(G(x, x), G(A, B), {y: G(x, x)})
unify('G', 'G', {y: G(x, x)})
unify([x, x], [A, B], {y: G(x, x)})
unify(x, A, {y: G(x, x)})
unify_var(x, A, {y: G(x, x)})
unify([x], [B], {x: A, y: G(x, x)})
unify(x, B, {x: A, y: G(x, x)})
unify_var(x, B, {x: A, y: G(x, x)})
unify(A, B, {x: A, y: G(x, x)})
unify('A', 'B', {x: A, y: G(x, x)})
unify([], [], None)
unify([], [], None)
unify([], [], None)
```

Le due espressioni non sono unificabili.

b) Older(Father(y), y), Older(Father(x), john)

```
>>> unify(Older(Father(y), y), Older(Father(x), John), {})
unify(Older(Father(y), y), Older(Father(x), John), {})
unify(Older(Father(y), y), Older(Father(x), John), {})
unify('Older', 'Older', {})
unify([Father(y), y], [Father(x), John], {})
unify(Father(y), Father(x), {})
unify('Father', 'Father', {})
unify([y], [x], {})
unify(y, x, {})
unify_var(y, x, {})
unify([], [], {y: x})
```

unify([y], [John], {y: x})

unify(y, John, {y: x})

unify_var(y, John, {y: x})

unify(x, John, {y: x})

unify_var(x, John, {y: x})

unify([], [], {x: John, y: x})

{x: John, y: John}

c) Knows(Father(y), y), Knows(x, x)

>>> **unify(Knows(Father(y), y), Knows(x, x), {})**

unify(Knows(Father(y), y), Knows(x, x), {})

unify(Knows(Father(y), y), Knows(x, x), {})

unify('Knows', 'Knows', {})

unify([Father(y), y], [x, x], {})

unify(Father(y), x, {})

unify_var(x, Father(y), {})

unify([y], [x], {x: Father(y)})

unify(y, x, {x: Father(y)})

unify_var(y, x, {x: Father(y)})

unify([], [], None)

Le due espressioni non sono unificabili. L'unificazione fallisce a causa dell'*occur check*.

Attenzione

Ama(x, Gelato) *Tutti amano il gelato*
Ama(Peter, x) *Peter ama qualunque cosa*

Sono unificabili? Dovrebbero esserlo ... Ma l'unificazione fallisce. Come mai?

>>> **unify(Ama(x, Gelato), Ama(Peter, x), {})**

unify(Ama(x, Gelato), Ama(Peter, x), {})

unify(Ama(x, Gelato), Ama(Peter, x), {})

unify('Ama', 'Ama', {})

unify([x, Gelato], [Peter, x], {})

unify(x, Peter, {})

unify_var(x, Peter, {})

unify([Gelato], [x], {x: Peter})

unify(Gelato, x, {x: Peter})

```
unify_var(x, Gelato, {x: Peter})
unify(Peter, Gelato, {x: Peter})
unify('Peter', 'Gelato', {x: Peter})
unify([], [], None)
unify([], [], None)
```

Bisogna rinominare. In questo caso le variabili nelle due clausole non sono opportunamente separate.

```
>>> unify(Ama(x, Gelato), Ama(Peter, y), {})
unify(Ama(x, Gelato), Ama(Peter, y), {})
unify(Ama(x, Gelato), Ama(Peter, y), {})
unify('Ama', 'Ama', {})
unify([x, Gelato], [Peter, y], {})
unify(x, Peter, {})
unify_var(x, Peter, {})
unify([Gelato], [y], {x: Peter})
unify(Gelato, y, {x: Peter})
unify_var(y, Gelato, {x: Peter})
unify([], [], {x: Peter, y: Gelato})
{x: Peter, y: Gelato}
```

Ama(x, Gelato) e Ama(Peter, y) ha successo con MGU={x/Peter, y/Gelato}

“Mary ama John” con il metodo di risoluzione

Dimostrare con il metodo di risoluzione che “Mary ama John” è conseguenza logica delle premesse: “Tutti amano chi ama qualcuno” e “John ama Mary”.

Possiamo scriverlo in FOL nel modo seguente:

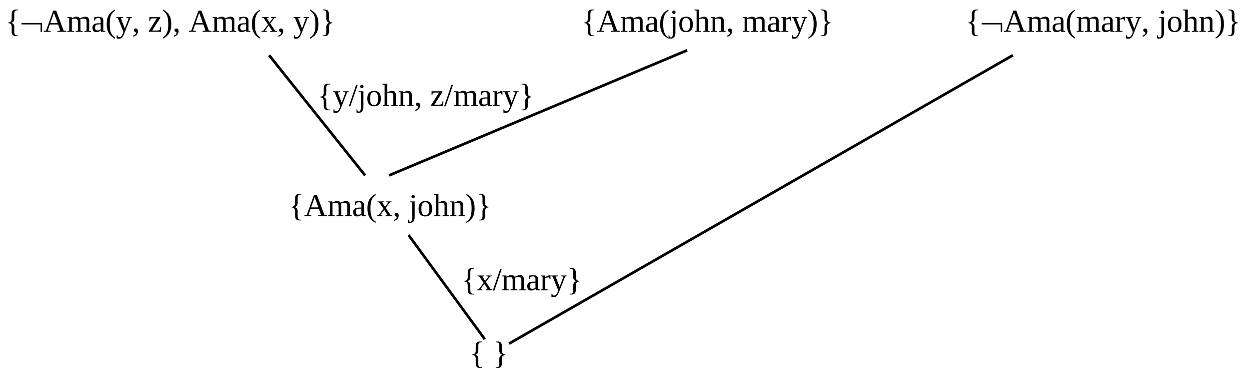
$\forall x\ \forall y\ (\exists z\ Ama(y, z)) \Rightarrow Ama(x, y)$
Ama (John, Mary)

In forma a clausole abbiamo:

$\forall x\ \forall y\ \neg(\exists z\ Ama(y, z)) \vee Ama(x, y)$
$\forall x\ \forall y\ \forall z\ \neg Ama(y, z) \vee Ama(x, y)$
$\{\neg Ama(y, z), Ama(x, y)\}$
{Ama(John, Mary)}

Il goal negato è $\{\neg Ama(Mary, John)\}$

Usando la refutazione, con strategia lineare, abbiamo:



La virgola che fa la differenza

a. Si formalizzino in logica del prim'ordine le seguenti frasi in linguaggio naturale:
   A1. *Odio i film violenti, che mi disturbano.*
   A2. *Odio i film violenti che mi disturbano.*
   Si noti che le due frasi, che differiscono per la presenza della virgola, hanno un significato diverso.
b. Sapendo inoltre che
   B1. *"Django" è un film diretto da Tarantino*
   B2. *Tarantino ha diretto solo film violenti*
   formalizzare e dimostrare con il metodo di risoluzione che "Odio *Django*" è conseguenza logica di una KB costituita da B1, B2 insieme con A1 oppure A2, scegliendo tra le due quella che vi sembra più utile per completare la dimostrazione.
c. Dire se con l'altra assunzione, "Odio *Django*" è conseguenza logica o meno, motivando la risposta.

Nella formalizzazione si utilizzi il seguente vocabolario.

Odio(x) : io odio x
Violento(x): x è un film violento
Disturba(x): x mi disturba
Regista(x, y): x ha diretto y
T : Tarantino
D: il fim 'Django'

a) Formalizzazione

A1. ∀x Violento(x) => Odio(x) ∧ Disturba(x)
{¬Violento(x), Odio(x)}
{¬Violento(y), Disturba(y)}

A2. ∀x Violento(x) ∧ Disturba(x) => Odio(x)
{¬Violento(z), ¬Disturba(z), Odio(z)}

B1. Regista(T, D)
{ Regista(T, D)}

B2. ∀y Regista(T, y) => Violento(y)
{¬Regista(T, w), Violento(w)}

b) Dimostrazione

Goal negato: {¬Odio(D)}

Le due dimostrazioni a confronto:

1. {¬Violento(x), Odio(x)}	1. {¬Violento(z), ¬Disturba(z), Odio(z)}
2. {¬Violento(y), Disturba(y)}	2. {Regista(T, D)}
3. {Regista(T, D)}	3. {¬Regista(T, w), Violento(w)}
4. {¬Regista(T, w), Violento(w)}	4. {¬Odio(D)}
5. {¬Odio(D)}	
6. {¬Violento(D)} [5, 1]	5. {¬Violento(D), ¬Disturba(D)} [4, 1]
7. {¬Regista(T, D)}	6. {¬Regista(T, D), ¬Disturba(D)} [5, 3]
8. { }	7. {¬Disturba(D)} [7, 2] *non riesco a procedere*

c) La colonna di destra usa la seconda assunzione che è troppo "debole" per ottenere la clausola vuota. In sostanza sto dicendo che non necessariamente odio tutti i film violenti ma solo quelli che mi disturbano. In particolare non ho informazioni sul fatto che il film Django disturba. Mancando Disturba(D) non riesco a ricavare la contraddizione.

Riesco a scrivere un programma logico per la versione A1?

1. regista(t, d).
2. odio(X) :- violento(X).
3. disturba(X) :- violento(X).
4. violento(W) :- regista(t, W).

Traccia Prolog per la query odio(d).

```
?- trace, odio(d).
   Call: (7) odio(d) ? creep
   Call: (8) violento(d) ? creep
   Call: (9) regista(t, d) ? creep
   Exit: (9) regista(t, d) ? creep
   Exit: (8) violento(d) ? creep
   Exit: (7) odio(d) ? creep
true.

?- trace, odio(Z).
   Call: (7) odio(_G517) ? creep
   Call: (8) violento(_G517) ? creep
   Call: (9) regista(t, _G517) ? creep
```

Exit: (9) regista(t, d) ? creep

Exit: (8) violento(d) ? creep

Exit: (7) odio(d) ? creep

Z = d.