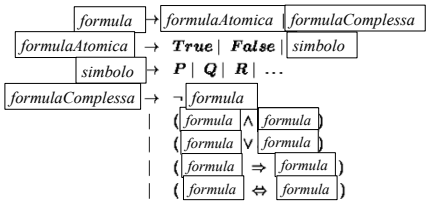
Agenti logici: calcolo proposizionale

Maria Simi
a.a. 2010/2011

Sintassi

- La sintassi definisce quali sono le frasi legittime del linguaggio:



Sintassi: esempi

- ((A ∧ B) ⇒ C)
- Possiamo omettere le parentesi assumendo questa precedenza tra gli operatori:

  ¬ > ∧ > ∨ > ⇒, ⇔
- ¬P ∨ Q ∧ R ⇒ S è la stessa cosa di (((¬P) ∨ (Q ∧ R)) ⇒ S)

Semantica e mondi possibili (modelli)

- La semantica ha a che fare col significato delle frasi: definisce se un enunciato è vero o falso rispetto ad una *interpretazione* (mondo possibile)
- Una interpretazione definisce un valore di verità per tutti i simboli proposizionali.
- Esempio {$P_{1,1}$ *vero*, $P_{1,2}$ *falso*, $W_{2,3}$ *vero*}
- $P_{1,1} \Rightarrow W_{2,3} \vee P_{1,2}$ è vera in questa interpretazione
- Un *modello* è una interpretazione *che rende vera* una formula o un insieme di formule

Semantica composizionale

- Il significato di una frase è determinato dal significato dei suoi componenti, a partire dalle frasi atomiche (i simboli proposizionali)
  - *True* sempre vero; *False* sempre falso
  - P ∧ Q, vero se P e Q sono veri
  - P ∨ Q, vero se P oppure Q, o entrambi, sono veri
  - ¬ P, vero se P è falso
  - P ⇒ Q, vero se P è falso oppure Q è vero
  - P ⇔ Q, vero se entrambi veri o entrambi falsi

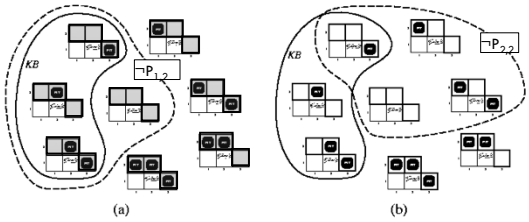
Conseguenza logica

- Una formula A è *conseguenza logica* di un insieme di formule KB se e solo se in ogni modello di KB, anche A è vera (KB ⊨ A)
- Esempio: KB = {$B_{2,1}$, $\neg B_{1,1}$, + *regole del WW*}

  Vogliamo stabilire l'assenza di pozzi in [1,2] (KB ⊨ $\neg P_{1,2}$?) e in [2,2] (KB ⊨ $\neg P_{2,2}$?)

  Ci sono otto possibili interpretazioni o mondi considerando solo l'esistenza di pozzi in 3 caselle …

Conseguenza logica e mondi possibili



KB = {$B_{2,1}$, $\neg B_{1,1}$ + *regole del WW*}

KB $\models \neg P_{1,2}$

KB $\not\models \neg P_{2,2}$

Equivalenza logica

- Equivalenza logica:

  A ≡ B se e solo se A $\models$ B e B $\models$ A

  Esempi:

  A ∧ B ≡ B ∧ A (commutatività di ∧)

  ¬(A ∧ B) ≡ ¬A ∨ ¬B (De Morgan)

  ¬(A ∨ B) ≡ ¬A ∧ ¬B (De Morgan)

Equivalenze logiche

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{de Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Validità, soddisfacibilità

- A valida sse è vera in tutte le interpretazioni (anche detta tautologia)
- A soddisfacibile sse esiste una interpretazione in cui A è vera
- A è valida sse ¬A è insoddisfacibile

Inferenza per Prop

- *Model checking*
  - una forma di inferenza che fa riferimento alla definizione di conseguenza logica (si enumerano i possibili modelli)
  - Tecnica delle tabelle di verità
- Algoritmi per la *soddisfacibilità*
  - KB $\models$ A sse ($KB_\wedge$ ∧ ¬A) è insoddisfacibile

Tabella di verità

- (¬A ∨ B) ∧ (A ∨ C) |= (B ∨ C)

A	B	C	¬A ∨ B	A ∨ C	B ∨ C
T	T	T	T	T	**T**
T	T	F	T	T	**T**
F	T	T	T	T	**T**
F	F	T	T	T	**T**

L'algoritmo TV-Consegue? (TT-entails?)

- KB $\models \alpha$?
- Enumera tutti le possibili interpretazioni di KB (*k* simboli , 2^k possibili modelli)
- Per ciascuna interpretazione
  - Se non soddisfa KB, OK
  - Se soddisfa KB, si controlla che soddisfi anche α

TT-Entails?

function TT-ENTAILS?(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  symbols ← a list of the proposition symbols in KB and α
  return TT-CHECK-ALL(KB, α, *symbols*, [])

function TT-CHECK-ALL(KB, α, *symbols*, *model*) **returns** *true* or *false*
  if EMPTY?(*symbols*) **then**
    if PL-TRUE?(KB, *model*) **then return** PL-TRUE?(α, *model*)
    else return *true*
  else do
    $P \leftarrow$ FIRST(*symbols*); *rest* ← REST(*symbols*)
    return TT-CHECK-ALL(KB, α, *rest*, [P = *true*|*model*]) **and**
      TT-CHECK-ALL(KB, α, *rest*, [P = *false*|*model*])

Esempio di TT-Entails?

$(\neg A \vee B) \wedge (A \vee C) \models (B \vee C)$?

- TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, [A, B, C], [ ])
  - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, [B, C], [A=t])
    - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, [C], [A=t; B=t])
      - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, [ ], [A=t; B=t; C=t]) OK
      - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, [ ], [A=t; B=t; C=f]) OK
    - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, [C], [A=t; B=f])
      - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, [ ], [A=t; B=f; C=t]) OK
      - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, [ ], [A=t; B=f; C=f] OK
  - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, [B, C], [A=f])

  …

Algoritmi per la soddisfacibilità (SAT)

- Usano KB in forma a clausole (insiemi di letterali)
  {A, B} {¬B, C, D} {¬A, F}
- Forma normale congiuntiva (CNF): una congiunzione di disgiunzioni di letterali
  $(A \vee B) \wedge (\neg B \vee C \vee D) \wedge (\neg A \vee F)$
- Non è restrittiva: è sempre possibile ottenerla con trasformazioni che preservano l'equivalenza logica

Trasformazione in forma a clausole

I passi sono:

1. Eliminazione della $\Leftrightarrow$: $(A \Leftrightarrow B) \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$
2. Eliminazione dell' $\Rightarrow$: $(A \Rightarrow B) \equiv (\neg A \vee B)$
3. Negazioni all'interno:
   $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$ (de Morgan)
   $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$
4. Distribuzione di $\vee$ su $\wedge$:
   $(A \vee (B \wedge C)) \equiv (A \vee B) \wedge (A \vee C)$

Esempio di trasformazione

1. $B_{1,1} \Leftrightarrow (P_{12} \vee P_{2,1})$
2. $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
3. $(\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
4. $(\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
5. $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$
6. $\{\neg B_{1,1}, P_{1,2}, P_{2,1}\}\ \{\neg P_{1,2}, B_{1,1}\}\ \{\neg P_{2,1}, B_{1,1}\}$

L'algoritmo DPLL per la soddisfacibilità

- DPLL: Davis, Putman, e poi Lovemann, Loveland
- Parte da una KB in forma a clausole
- È una enumerazione *in profondità* di tutti i possibili modelli, con miglioramenti rispetto a TTEntails:
  - Terminazione anticipata
  - Euristica dei simboli (o letterali) puri
  - Euristica delle clausole unitarie

DPLL: terminazione anticipata

- Si può decidere sulla verità di una clausola anche con modelli parziali: basta che un letterale sia vero
  Se A è vero lo sono anche {A, B} e {A, C} indipendentemente dai valori di B e C
- Se una clausola è falsa l'interpretazione non è un modello

DPLL: simboli puri

- *Simbolo puro*: un simbolo che appare con lo stesso segno in tutte le clausole
  Es. {A, ¬B} {¬B, ¬C} {C, A} A è puro, B anche
- Nel determinare se un simbolo è puro se ne possono trascurare le occorrenze in clausole già rese vere
- I simboli puri possono essere assegnati a *True* se il letterale è positivo, *False* se negativo.
- Non si eliminano modelli utili: se le clausole hanno un modello continuano ad averlo dopo questo assegnamento.

DPLL: clausole unitarie

- *Clausola unitaria*: una clausola con un solo letterale *non assegnato*
  Es. Quando B = *False*, {B, ¬C} è unitaria
- Conviene assegnare prima valori al letterale in clausole unitarie. L'assegnamento è univoco (*True* se positivo, *False* se negativo).
- DPLL è completo

Lo schema dell'algoritmo DPLL

function DPLL-SATISFIABLE?(*s*) **returns** *true* or *false*
  inputs: *s*, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of *s*
  symbols ← a list of the proposition symbols in *s*
  return DPLL(*clauses*, *symbols*, [])

---

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

  if every clause in *clauses* is true in *model* **then return** *true*
  if some clause in *clauses* is false in *model* **then return** *false*
  P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
  if *P* is non-null **then return** DPLL(*clauses*, *symbols*–*P*, [*P* = *value*|*model*])
  P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  if *P* is non-null **then return** DPLL(*clauses*, *symbols*–*P*, [*P* = *value*|*model*])
  P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, [*P* = *true*|*model*]) **or** DPLL(*clauses*, *rest*, [*P* = *false*|*model*])

DPLL: esempio

KB $\{\neg B_{1,1}, P_{1,2}, P_{2,1}\}\ \{\neg P_{1,2}, B_{1,1}\}\ \{\neg P_{2,1}, B_{1,1}\}\ \{\neg B_{1,1}\} \models \{\neg P_{1,2}\}$?

Aggiungiamo $\{P_{1,2}\}$ e vediamo se insoddisfacibile

SAT($\{\neg B_{1,1}, P_{1,2}, P_{2,1}\}\ \{\neg P_{1,2}, B_{1,1}\}\ \{\neg P_{2,1}, B_{1,1}\}\ \{\neg B_{1,1}\}\ \{P_{1,2}\}$)?

1 2 3 4 5

- La *5* è unitaria; $P_{1,2}$=True; la prima clausola e la *5* sono soddisfatte
- La 2 diventa unitaria; $B_{1,1}$=True; 2 e 3 sono soddisfatte, ma la 4 no; Fail

Non esistono modelli quindi $\neg P_{1,2}$ è conseguenza logica della KB

WalkSAT: un metodo di ricerca locale

- È il migliore tra una serie di algoritmi di ricerca locali per la soddisfacibilità
  - Gli stati sono gli assegnamenti
  - L'obiettivo è un assegnamento che soddisfa tutte le clausole
  - Si parte da un assegnamento casuale
  - Ad ogni passo si cambia il valore di una proposizione (*flip*)
  - Gli stati sono valutati contando il numero di clausole soddisfatte (più sono meglio è) [o non soddisfatte]

WalkSAT

- WalkSAT ad ogni passo
  - Sceglie a caso una clausola non ancora soddisfatta
  - Sceglie un simbolo da modificare (*flip*) scegliendo con probabilità *p* (di solito 0,5) tra una delle due:
    - Sceglie un simbolo a caso (passo casuale)
    - Sceglie quello che rende più clausole soddisfatte (passo di ottimizzazione, simile a *min-conflicts*)
- Si arrende dopo un certo numero di *flip* predefinito

WalkSat: l'algoritmo

function WALKSAT(*clauses*, *p*, *max-flips*) **returns** a satisfying model or *failure*
  inputs: *clauses*, a set of clauses in propositional logic
    p, the probability of choosing to do a "random walk" move, typically around 0.5
    max-flips, number of flips allowed before giving up

  model ← a random assignment of *true*/*false* to the symbols in *clauses*
  for *i* = 1 **to** *max-flips* **do**
    if *model* satisfies *clauses* **then return** *model*
    clause ← a randomly selected clause from *clauses* that is false in *model*
    with probability *p* flip the value in *model* of a randomly selected symbol from *clause*
    else flip whichever symbol in *clause* maximizes the number of satisfied clauses
  return *failure*

WalkSAT: un esempio

$\{\neg B_{1,1}, P_{1,2}, P_{2,1}\}\ \{\neg P_{1,2}, B_{1,1}\}\ \{\neg P_{2,1}, B_{1,1}\}\ \{\neg B_{1,1}\}$

$[B_{1,1}=F, P_{1,2}=T, P_{2,1}=T]$ 2, 3 F; scelgo 2; a caso:flip $B_{1,1}$

$[B_{1,1}=T, P_{1,2}=T, P_{2,1}=T]$ 4 F; scelgo 4; flip $B_{1,1}$

$[B_{1,1}=F, P_{1,2}=T, P_{2,1}=T]$ 2, 3 F; scelgo 2; a caso:flip $P_{1,2}$

$[B_{1,1}=F, P_{1,2}=F, P_{2,1}=T]$ 3 F; scelgo 3;
ottimizzazione: flip $P_{2,1}$[4]; flip $B_{1,1}$[3]

$[B_{1,1}=F, P_{1,2}=F, P_{2,1}=F]$ modello

Rosso: passo casuale

Verde: passo di ottimizzazione

Analisi di WalkSAT

- Se *max-flips* = ∞ e l'insieme di clausole è soddisfacibile prima o poi termina
- Ma se è insoddisfacibile non termina: il meglio che si può dire è di averci provato a lungo ...
- Il problema è decidibile ma l'algoritmo non è completo.

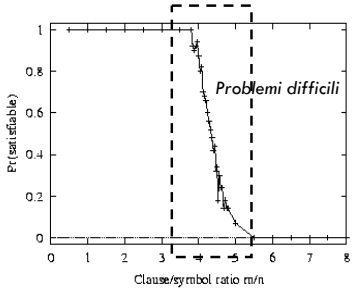
Problemi SAT difficili

- Se un problema ha molte soluzioni è più probabile che WalkSAT ne trovi una.
  - Esempio: 16 soluzioni su 32; un assegnamento ha il 50% di probabilità di essere giusto: 2 passi in media!

$$(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E) \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$$

- Quello che conta è il rapporto *m*/*n* dove *m* è il numero di clausole (vincoli) e *n* il numero di simboli. Es. 5/5=1
- Più grande il rapporto, più vincolato è il problema
  - Le regine sono facili perché il problema è sotto-vincolato

Probabilità di soddisfacibilità in funzione di m/n

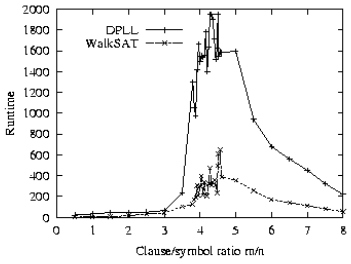


(a)

***m* (n. clausole) varia**
***n* (n. simboli) = 50**

media su 100 problemi generati a caso

Confronto tra DPLL e WalkSAT



(b)

Confronto su problemi soddisfacibili, ma difficili

Inferenza come deduzione

- Un altro modo per decidere se KB $\models$ A è dare delle regole di inferenza
  - Si scrive KB |- A (A è deducibile da KB)
- Le regole di inferenza
  - dovrebbero derivare solo formule che sono conseguenza logica
  - dovrebbero derivare tutte le formule che sono conseguenza logica

Correttezza e completezza

- Correttezza: Se KB |- A allora KB $\models$ A
  Tutto ciò che è derivabile è conseguenza logica. Il meccanismo preserva la verità.
- Completezza: Se KB $\models$ A allora KB |- A
  Tutto ciò che è conseguenza logica è ottenibile tramite il meccanismo di inferenza. Non sempre è possibile.

Alcune regole di inferenza per Prop

- Le regole sono schemi deduttivi del tipo:

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$ *Modus ponens* oppure *Eliminazione dell'implicazione*

$$\frac{\alpha \wedge \beta}{\alpha}$$ *Eliminazione dell'AND*

Eliminazione e introduzione della doppia implicazione

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \quad \text{and} \quad \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Meta-teoremi utili

- A valida sse $\neg$A è insoddisfacibile
- Teorema di deduzione:
  A $\models$ B sse (A $\Rightarrow$ B) è valida
- Teorema di refutazione:
  A $\models$ B sse (A $\wedge$ $\neg$B) è insoddisfacibile
  dimostrazione per assurdo o per *refutazione*

Una rappresentazione per il WW

R_1: $\neg P_{1,1}$ *non ci sono pozzi in [1, 1]*

C'è brezza nelle caselle adiacenti ai pozzi:

R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{1,2} \vee P_{2,1})$

Percezioni:

R_4: $\neg B_{1,1}$ *non c'è brezza in [1, 1]*

R_5: $B_{2,1}$ *c'è brezza in [2, 1]*

$KB = \{R_1\, R_2\, R_3\, R_4\, R_5\}$ $KB \models \neg P_{1,2}$?

Dimostrazione

R_6: $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$ (R_2, $\Leftrightarrow$ E)

R_7: $(P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}$ (R_6, $\wedge$E)

R_8: $\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})$ (R_7, *contrapposizione*)

R_9: $\neg(P_{1,2} \vee P_{2,1})$ (R_4 e R_8, *Modus Ponens*)

R_{10}: $\neg P_{1,2} \wedge \neg P_{2,1}$ (R_9, De Morgan)

R_{11}: $\neg P_{1,2}$ (R_{10}, $\wedge$E)

Dimostrazione come ricerca

- *Problema*: come decidere ad ogni passo qual'è la regola di inferenza da applicare? … e a quali premesse? Come evitare l'esplosione combinatoria?
- È un problema di esplorazione di uno spazio di stati
- Una *procedura di dimostrazione* definisce:
  - la direzione della ricerca
  - la strategia di ricerca

Direzione della ricerca

- Nella dimostrazione di teoremi conviene procedere all'indietro. Con una lettura *in avanti* delle regole:

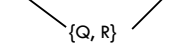
  Da A, B: $A \wedge B$ $A \wedge (A \wedge B)$ … $A \wedge (A \wedge (A \wedge B))$
- Meglio *all'indietro*
  - se si vuole dimostrare $A \wedge B$, si cerchi di dimostrare A e poi B
  - se si vuole dimostrare $A \Rightarrow B$, si assuma A e si cerchi di dimostrare B
  - …

Strategia di ricerca

- Completezza
  - Le regole della deduzione naturale sono un insieme di regole di inferenza completo (2 per ogni connettivo)
  - Se l'algoritmo di ricerca è completo siamo a posto
- Efficienza
  - La complessità è alta: è un problema decidibile ma NP-completo

Regola di risoluzione per prop

- E se avessimo un'unica regola di inferenza (senza rinunciare alla completezza)?
- Regola di risoluzione (presuppone forma a clausole)



$$\frac{P \vee Q,\ \neg P \vee R}{Q \vee R}$$

- Corretta? Basta pensare ai modelli
- Preferita la notazione insiemistica
- NOTA: gli eventuali duplicati si eliminano

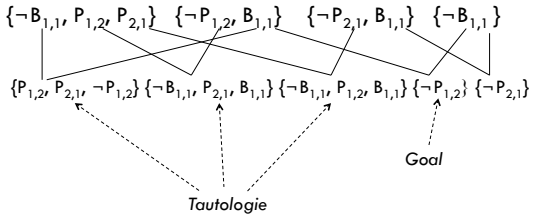
La regola di risoluzione in generale

$$\frac{\{l_1, l_2, \ldots l_i, \ldots l_k\} \quad \{m_1, m_2, \ldots m_j, \ldots m_n\}}{\{l_1, l_2, \ldots l_{i-1}, l_{i+1}, \ldots l_k\ m_1\ m_2, \ldots m_{j-1}, m_{j+1}, \ldots m_n\}}$$

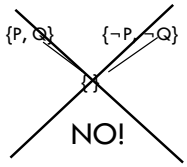
Gli l e m sono letterali, simboli di proposizione positivi o negativi; l_i e m_j *sono uguali e di segno opposto*

Caso particolare $\dfrac{\{P\} \quad \{\neg P\}}{\{\,\}}$ *clausola vuota*

Il grafo di risoluzione

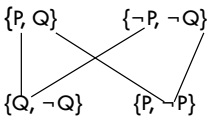


Attenzione!



Non è contradditorio:

Es. Bianco o nero e
non bianco o non nero



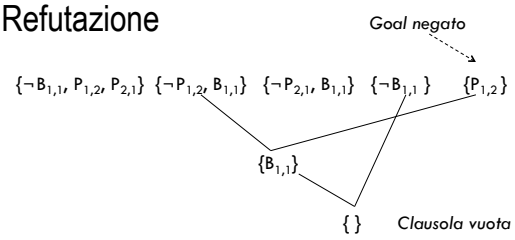
… e qui ci fermiamo

Un passo alla volta !!!

Ma siamo sicuri che basti una regola?

- Completezza: se KB $\models \alpha$ allora KB $\vdash_{res} \alpha$? Non sempre:
  Es. KB $\models$ {A, ¬A} ma non è vero che KB $\vdash_{res}$ {A, ¬A}
- Teorema di risoluzione [ground]:
  Se KB insoddisfacibile allora KB $\vdash_{res}$ { } *completezza*
- Teorema di refutazione offre un modo alternativo:
  KB $\models \alpha$ sse (KB $\cup$ {¬ α}) insoddisfacibile
- Nell'esempio:
  KB $\cup$ *FC*(¬(A $\vee$¬A)) è insoddisfacibile? Si, perché …
  KB $\cup$ {A} $\cup$ {¬A} $\vdash$ { } in un passo e la regola di risoluzione è corretta
  Quindi KB $\models$ {A, ¬A}

Refutazione



Il teorema di risoluzione

- Sia RC(*S*) l'insieme (*chiusura per risoluzione*) ottenuto applicando in tutti i modi possibili la regola di risoluzione ad *S*.
- RC(*S*) è finito
- Teorema di risoluzione *ground*:
  Se *S* è insoddisfacibile allora RC(*S*) contiene { }.
- Se RC(*S*) non contenesse { } potremmo costruire un modello di *S*
- Sia $P_1, P_2 \ldots P_k$ un ordinamento delle proposizioni. Assegniamo valori procedendo con i=1,…k in questo modo:
  - se in una clausola c'è $\neg P_i$ e gli altri letterali sono falsi in base agli assegnamenti già fatti, assegna *False* a P_i
  - altrimenti assegna *True* a P_i

Il Wumpus World con Prop: regole

- Regole generali: "C'è *brezza nelle caselle adiacenti ai pozzi*"

  $B_{x,y} \Leftrightarrow P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y}$ per ogni x e y

  16 asserzioni di questo tipo in un mondo 4 X 4
- C'è esattamente un Wumpus!
  - $W_{1,1} \vee W_{1,2} \vee W_{1,3} \vee \ldots W_{4,4}$ almeno uno
  - $\neg W_{1,1} \vee \neg W_{1,2}$ per ogni coppia di caselle

    16X15/2 = 155 asserzioni per dire che ce n'è al più uno!!!

Wumpus World: locazione, orientamento

- Se si vuole tenere traccia della locazione
  - $L_{1,1} \wedge \text{FacingRight} \wedge \text{Forward} \Rightarrow L_{2,1}$

  Non va bene, serve una dimensione temporale
  - $L^1_{1,1} \wedge \text{FacingRight}^1 \wedge \text{Forward}^1 \Rightarrow L^2_{2,1}$
- Stessa cosa per l'orientamento …
  - $\text{FacingRight}^1 \wedge \text{TurnLeft}^1 \Rightarrow \text{FacingUp}^2$

Il Wumpus World con Prop

- Una casella [i, j] è sicura se $KB \models (\neg P_{i,j} \wedge \neg W_{i,j})$
- Una casella [i, j] potrebbe essere considerata sicura se $KB \not\models (P_{i,j} \vee W_{i,j})$
- Con tutti questi simboli di proposizione
  - servono procedure di inferenza efficienti (TTEntails e DPLL non sono praticabili)
  - serve un linguaggio più espressivo!!