

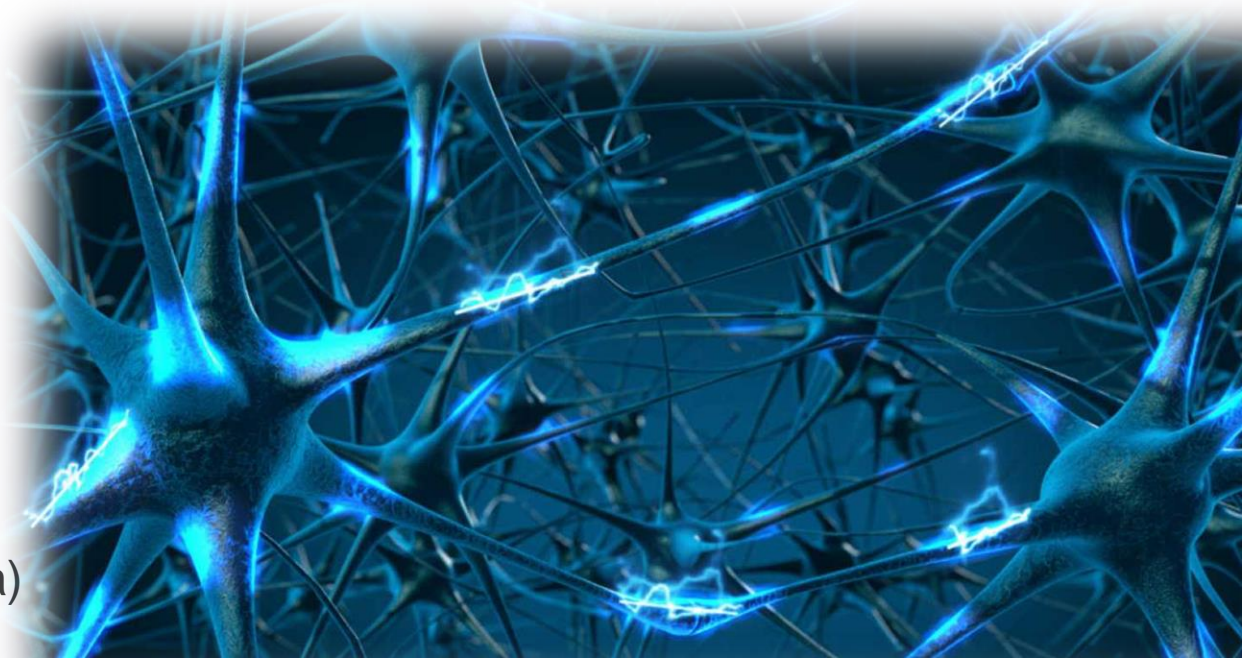
INTELLIGENZA ARTIFICIALE

KR: CALCOLO PROPOSIZIONALE ()*

Corsi di Laurea in Informatica, Ing. Gestionale, Ing. Informatica,
Ing. di Internet
(a.a. 2023-2024)

Roberto Basili

(*) alcune *slides* sono di
Maria Simi (Univ. Pisa)



Overview

- La rappresentazione della conoscenza in CPROR
- Sintassi e semantica del CPROR
- Inferenza nel CPROR
 - Algoritmo naive
 - Algoritmi di random walk
 - Deduzione e Inferenza: Risoluzione
- Applicazione del CPROR al mondo del Wumpus

Sintassi

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

$formula \rightarrow formulaAtomica \mid formulaComplessa$

$formulaAtomica \rightarrow \mathbf{True} \mid \mathbf{False} \mid simbolo$

$simbolo \rightarrow \mathbf{P} \mid \mathbf{Q} \mid \mathbf{R} \mid \dots$

$formulaComplessa \rightarrow \neg formula$

$\mid (formula \wedge formula)$

$\mid (formula \vee formula)$

$\mid (formula \Rightarrow formula)$

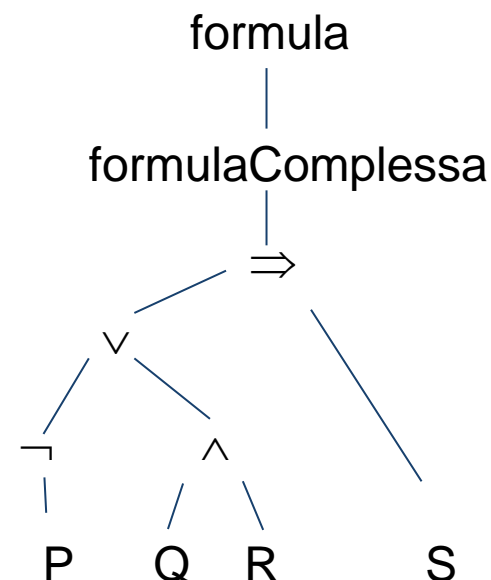
$\mid (formula \Leftrightarrow formula)$

Sintassi: esempi

- $((A \wedge B) \Rightarrow C)$
- Possiamo omettere le parentesi assumendo questa precedenza tra gli operatori:

$\neg > \wedge > \vee > \Rightarrow, \Leftrightarrow$

- $\neg P \vee Q \wedge R \Rightarrow S$ è la stessa
formula di
 $((\neg P) \vee (Q \wedge R)) \Rightarrow S$



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.

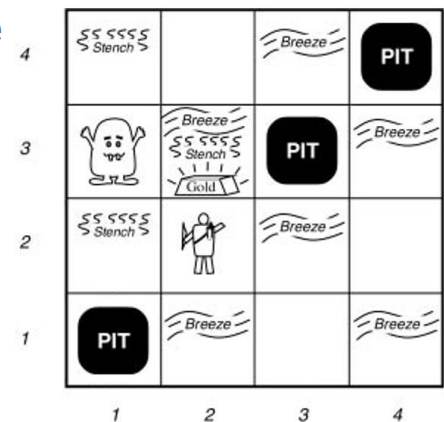
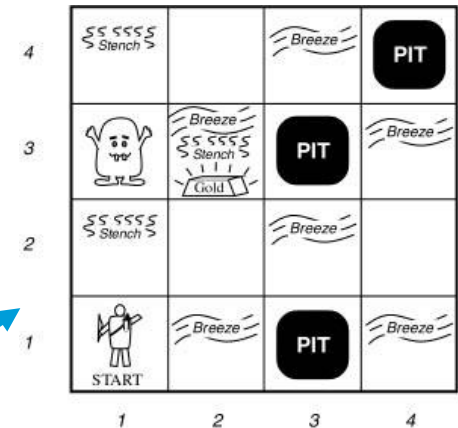
- Esempio:

- $\{P_{3,1} \text{ vero}, P_{1,2} \text{ falso}, W_{1,3} \text{ vero}\}$

- $P_{3,1} \Rightarrow W_{1,3} \vee P_{1,2}$ è vera in questa interpretazione

- $\{P_{3,1} \text{ falso}, P_{1,1} \text{ vero}, W_{1,3} \text{ vero}\}$

- 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
 - $P \wedge Q$, vero se P e Q sono veri
 - $P \vee Q$, vero se P oppure Q , o entrambi, sono veri
 - $\neg P$, vero se P è falso
 - $P \Rightarrow Q$, vero se P è falso oppure Q è vero
 - $P \Leftrightarrow Q$, 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 ($KB \models A$)
- Indicando con $M(\alpha)$ l'insieme delle interpretazioni che rendono α vera, cioè i **modelli** di α , e con $M(KB)$ i modelli di un insieme di formule KB ...
$$KB \models \alpha \quad \text{sse} \quad M(KB) \subseteq M(\alpha)$$

Esempio dal mondo del Wumpus

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



Vogliamo stabilire l'assenza di pozzi in $[1,2]$ e in

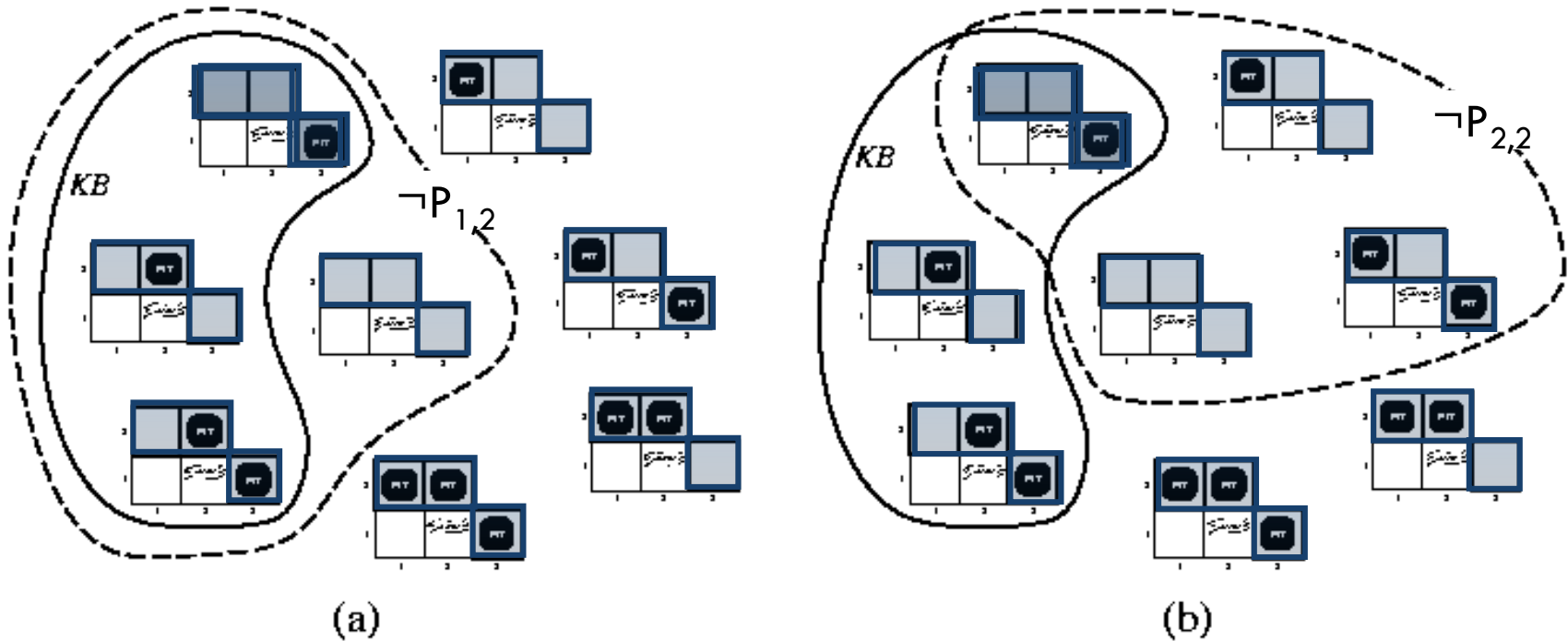
$[2,2]$ $KB \models \neg P_{1,2}?$

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

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

$P_{1,2}, P_{2,2}$ e $P_{3,1}$

Conseguenza logica e mondi possibili



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

$$KB \models \neg P_{1,2} \quad \text{poichè } M(KB) \subseteq M(\neg P_{1,2})$$

$$KB \not\models \neg P_{2,2} \quad \text{poichè } M(KB) \not\subseteq M(\neg P_{2,2})$$

Equivalenza logica

- Equivalenza logica:

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

Esempi:

$A \wedge B \equiv B \wedge A$ (commutatività di \wedge)

$\neg(A \wedge B) \equiv \neg A \vee \neg B$ (De Morgan)

$\neg(A \vee B) \equiv \neg A \wedge \neg B$ (De Morgan)

Equivalenze logiche

Two sentences are **logically equivalent** iff true in same models:

$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

Validità, soddisfacibilità

- Una formula A è **valida** sse è vera in tutte le interpretazioni (A è anche detta **tautologia**)
- A è **soddisfacibile** sse esiste (almeno) una interpretazione in cui A è vera
- A è valida sse $\neg A$ non è soddisfacibile
(o A è insoddisfacibile)

Inferenza per Prop

- *Model checking*
 - una forma di inferenza che sfrutta la definizione di conseguenza logica
 - Si enumerano i possibili modelli
 - Tecnica delle tabelle di verità
- Algoritmi per la *soddisfacibilità*
 - $KB \models A$ sse $(KB \wedge \neg A)$ è insoddisfacibile

Tabella di verità

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

A	B	C	$\neg A \vee B$	$A \vee C$	$B \vee C$
T	T	T	T	T	T
T	T	F	T	T	T
F	T	T	T	T	T
F	F	T	T	T	T

Tabella di verità

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

Soddisfatto



2³

A	B	C	$\neg A \vee B$	$A \vee C$	$B \vee C$
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?)

- Problema: $KB \models \alpha$?
- Enumera tutti le possibili interpretazioni di KB
(Se k sono i simboli usati da KB, sono 2^k i 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,  $\alpha$ ) returns true or false  
  inputs: KB, the knowledge base, a sentence in propositional logic  
            $\alpha$ , the query, a sentence in propositional logic  
  
  symbols  $\leftarrow$  a list of the proposition symbols in KB and  $\alpha$   
  return TT-CHECK-ALL(KB,  $\alpha$ , symbols, { })
```

```
function TT-CHECK-ALL(KB,  $\alpha$ , symbols, model) returns true or false  
  if EMPTY?(symbols) then  
    if PL-TRUE?(KB, model) then return PL-TRUE?( $\alpha$ , model)  
    else return true // when KB is false, always return true  
  else do  
    P  $\leftarrow$  FIRST(symbols)  
    rest  $\leftarrow$  REST(symbols)  
    return (TT-CHECK-ALL(KB,  $\alpha$ , rest, model  $\cup$  {P = true})  
            and  
            TT-CHECK-ALL(KB,  $\alpha$ , rest, model  $\cup$  {P = false })))
```

Figure 7.10 A truth-table enumeration algorithm for deciding propositional entailment. (TT stands for truth table.) PL-TRUE? returns *true* if a sentence holds within a model. The variable *model* represents a partial model—an assignment to some of the symbols. The keyword “**and**” is used here as a logical operation on its two arguments, returning *true* or *false*.

Esempio di TT-Entails?

KB

Formula alfa

Symbols

Model

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

- 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($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, $[]$, $[A=t; B=t; \underline{C=f}]$) 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 %not a model for KB
 - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, $[]$, $[A=t; B=f; C=f]$) OK %not a model for KB
 - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, $[B, C]$, $[A=f]$)
 - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, $[C]$, $[A=f; B=t]$)
 - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, $[]$, $[A=f; B=t; C=t]$) OK
 - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, $[]$, $[A=f; B=t; C=f]$) OK %not a model for KB
 - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, $[C]$, $[A=f; B=f]$)
 - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, $[]$, $[A=f; B=f; C=t]$) OK
 - TT-CHECK-ALL($(\neg A \vee B) \wedge (A \vee C)$, $(B \vee C)$, $[]$, $[A=f; B=f; C=f]$) OK %not a model for KB

Algoritmi per la soddisfacibilità (SAT)

- Usano KB in forma a clausole (insiemi di letterali)

$$\{A, B\} \quad \{\neg B, C, D\} \quad \{\neg A, F\}$$

- 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. Distribuzione delle 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 delle disgiunzioni \vee su \wedge (**CNF**):
 $(A \vee (B \wedge C)) \equiv (A \vee B) \wedge (A \vee C)$

Notazione insemistica: $\{A, B\}$ $\{A, C\}$

Esempio di trasformazione

1. $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
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 *tre miglioramenti* rispetto a TT-Entails:
 1. Terminazione anticipata
 2. Euristiche dei simboli (o letterali) puri
 3. Euristiche 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
 - (a causa della disgiunzione logica, OR)
- Se anche una sola clausola è falsa, allora l'interpretazione complessiva è falsa ed essa non è un modello dell'insieme di clausole

DPLL: simboli puri

- *Simbolo puro*: un simbolo che appare con lo stesso segno in tutte le clausole
 - Es. $\{A, \neg B\} \{\neg B, \neg C\} \{C, A\}$ A è puro, B anche, C no
- 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. $\{B\}$ è unitaria ma anche ...
 - $\{B, \neg C\}$ è unitaria quando $C = \text{True}$
- Conviene assegnare prima valori al letterale in clausole unitarie. L'assegnamento è univoco (*True* se positivo, *False* se negativo).

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*, *model* ∪ {*P=**value*})

P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)

if *P* is non-null **then return** DPLL(*clauses*, *symbols* - (*P*), *model* ∪ {*P=**value*})

P ← FIRST(*symbols*); *rest* ← REST(*symbols*)

return DPLL(*clauses*, *rest*, *model* ∪ {*P=true*}) **or**

DPLL(*clauses*, *rest*, *model* ∪ {*P=false*})

Figure 7.17 The DPLL algorithm for checking satisfiability of a sentence in propositional logic. The ideas behind FIND-PURE-SYMBOL and FIND-UNIT-CLAUSE are described in the text; each returns a symbol (or null) and the truth value to assign to that symbol. Like TT-ENTAILS?, DPLL operates over partial models.

Metodi locali per SAT: formulazione

- Gli stati sono gli assegnamenti completi
- L'obiettivo è un assegnamento alle proposizioni di valori di verità che soddisfa **tutte** le clausole
- Si parte da un **assegnamento casuale**
- Ad ogni passo si cambia il valore di una proposizione (*flip*)
- **Euristica**: Gli stati sono valutati **contando il numero di clausole soddisfatte** (più sono meglio è) [o non soddisfatte]

Metodi locali per SAT: algoritmi

- Ci sono *molti minimi locali*, per sfuggire ai quali serve introdurre perturbazioni casuali
- *Hill climbing* con riavvio casuale
- *Simulated Annealing*
- Molta sperimentazione per trovare il miglior compromesso tra il grado di avidità e casualità
- WALK-SAT è uno degli algoritmi più semplici ed efficaci

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

Conflitti tra variabili

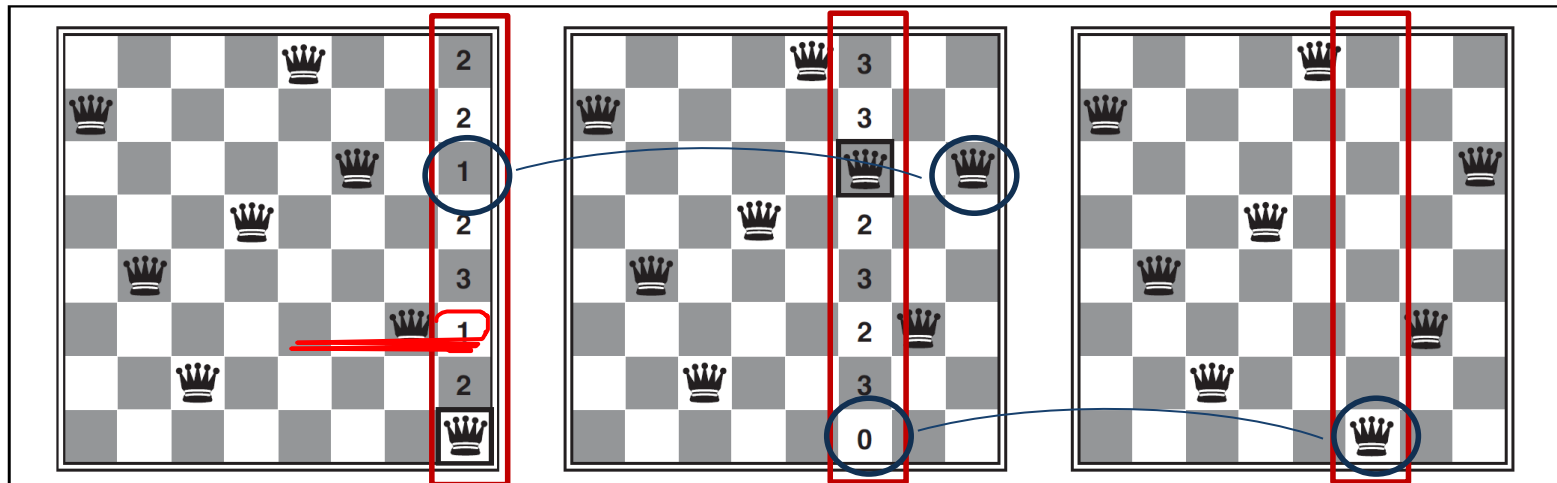


Figure 6.9 A two-step solution using min-conflicts for an 8-queens problem. At each stage, a queen is chosen for reassignment in its column. The number of conflicts (in this case, the number of attacking queens) is shown in each square. The algorithm moves the queen to the min-conflicts square, breaking ties randomly.

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  
         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
```


Analisi di WalkSAT

- Se *max-flips* = ∞ e l'insieme di clausole è soddisfacibile prima o poi termina
- Va bene per cercare un modello, sapendo che c'è, ma se è insoddisfacibile non termina
- Non può essere usato per verificare l'insoddisfacibilità
- Il problema è decidibile ma l'algoritmo non è completo

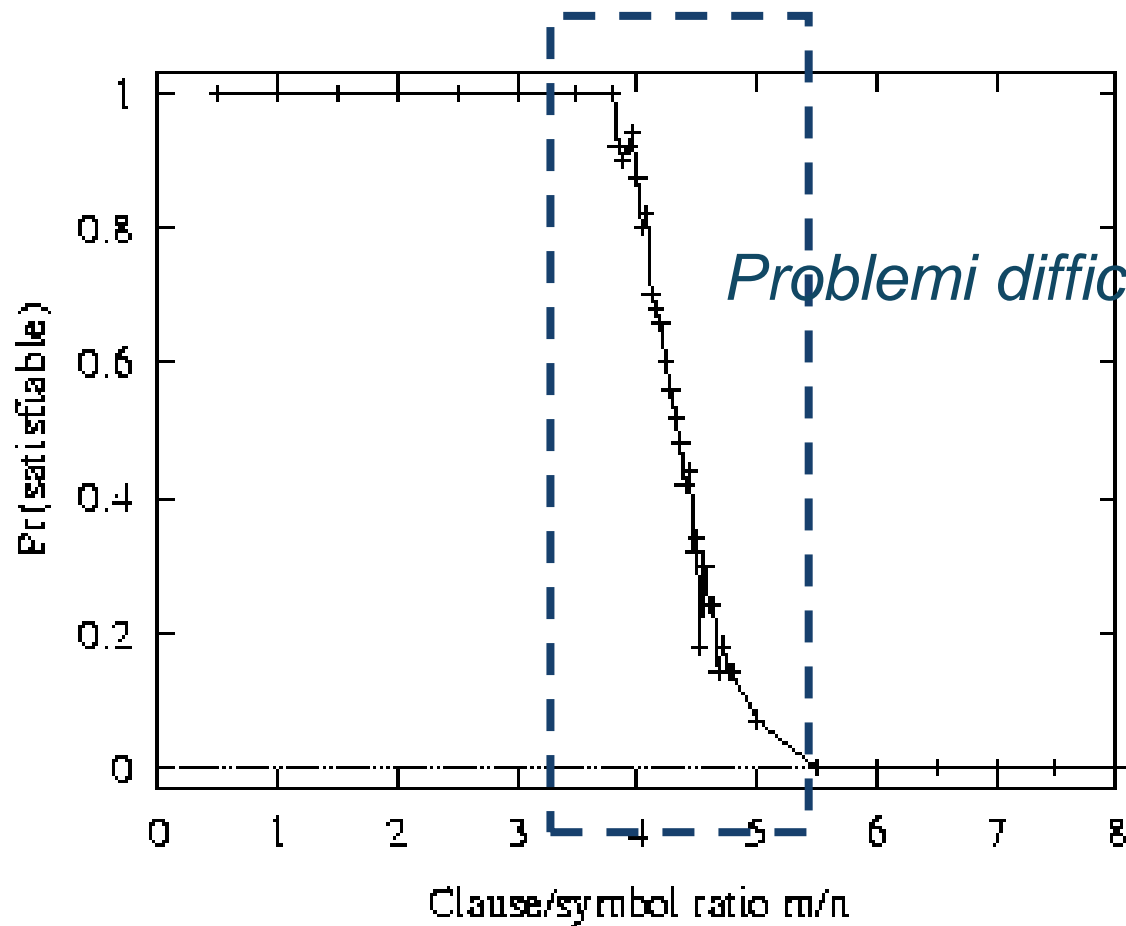
Problemi SAT difficili

- Se un problema ha molte soluzioni è più probabile che WalkSAT ne trovi una (problema sotto-vincolato)
 - 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 (*poche regine e molte caselle*)

Probabilità di soddisfacibilità in funzione di m/n

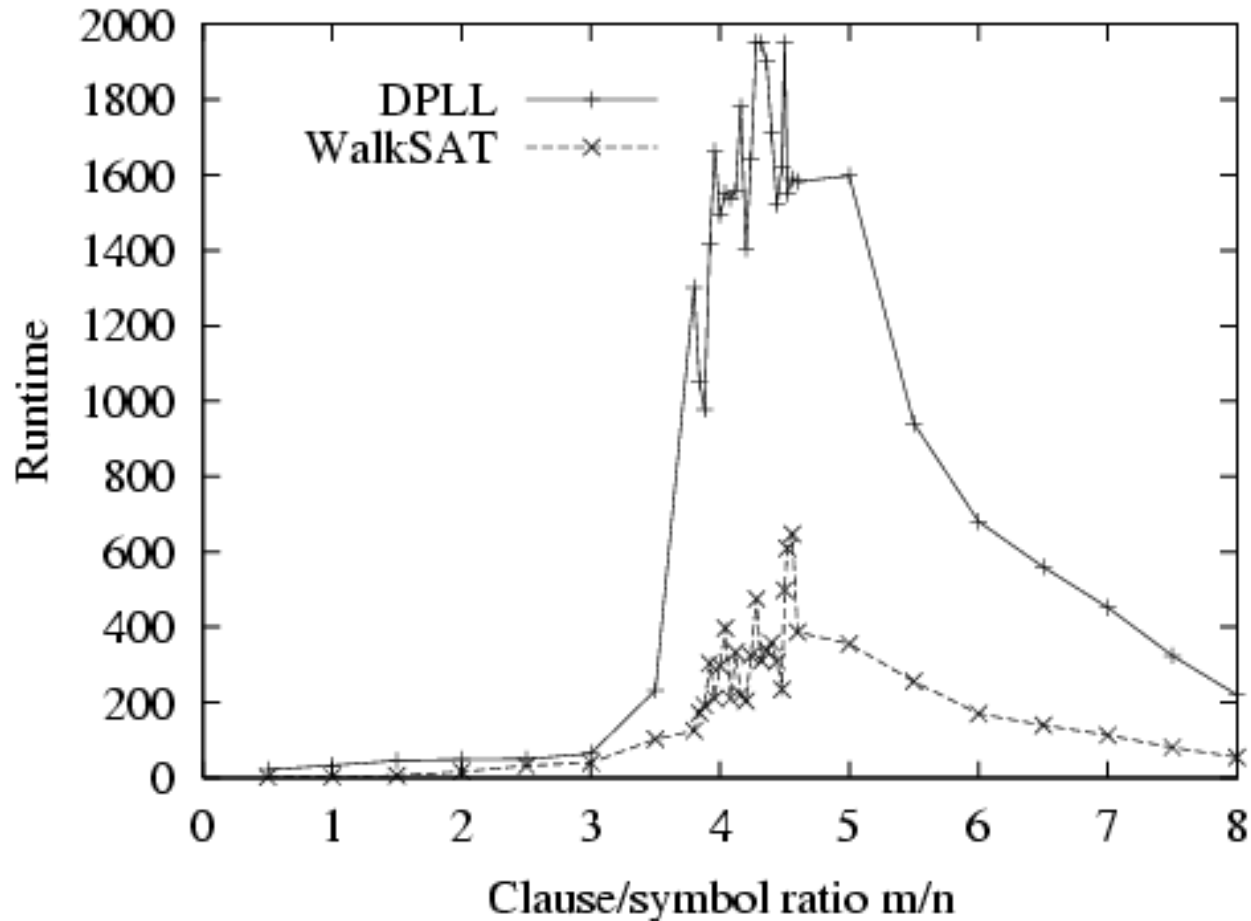


m (n. clausole) varia
 n (n. simboli) = 50
3 letterali per clausola

media su 100 problemi
generati a caso

(a)

Confronto tra DPLL e WalkSAT



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 $KB \vdash A$ (A è deducibile da KB)
 - Nota la differenza con:
 - $KB \models A$ (A è conseq. logica 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 $KB \vdash A$ allora $KB \models A$
Tutto ciò che è derivabile è conseguenza logica.
Le regole preservano la verità.
- **COMPLETEZZA:** Se $KB \models A$ allora $KB \vdash A$
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

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

and

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

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**

(corrisponde alla 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:

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

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

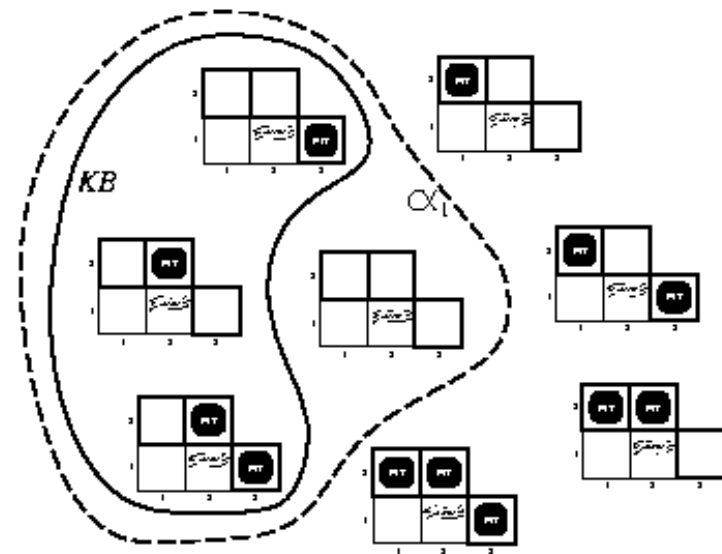
Percezioni:

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

$R_5: B_{2,1}$ *c'è brezza in [2, 1]*

$KB = \{R_1 R_2 R_3 R_4 R_5\}$

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



(a)

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:

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

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

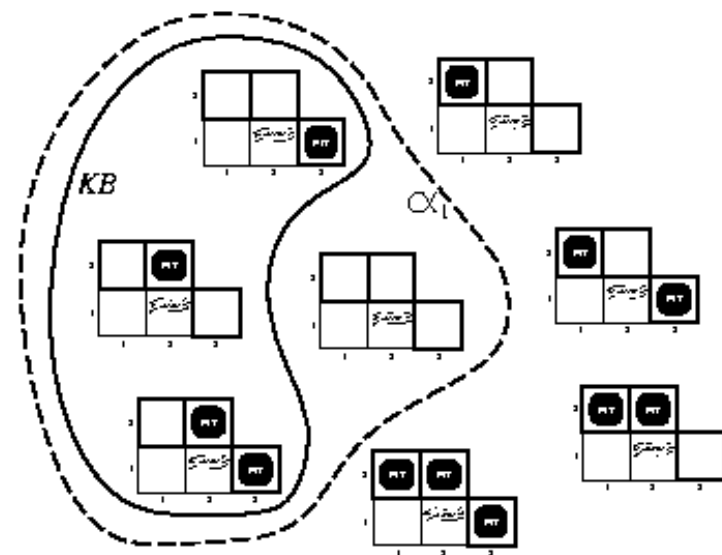
Percezioni:

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

$R_5: B_{2,1}$ *c'è brezza in [2, 1]*

$KB = \{R_1 R_2 R_3 R_4 R_5\}$

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



(a)

Dimostrazione

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

KB:

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

Brezza

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

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

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]*

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

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

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

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

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

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

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.

- Infatti, con una lettura *in avanti* delle regole:

Da $A, B: A \wedge B > A \wedge (A \wedge B) > \dots > A \wedge (A \wedge (A \wedge B)) \dots$

- 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)
 - Introduzione (ad es. $A, B \vdash A \wedge B$)
 - Eliminazione (ad es. $A \wedge B \vdash A$, e pure $A \wedge B \vdash B$)
 - 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 PROLOG

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

$$\begin{array}{ccc} \{P, Q\} & & \{\neg P, R\} \\ & \searrow \quad \swarrow & \\ & \{Q, R\} & \end{array} \qquad \frac{P \vee Q, \neg P \vee R}{Q \vee R}$$

- Corretta? SI: basta pensare ai modelli
- Preferita la notazione insiemistica delle clausole
- NOTA: gli *eventuali duplicati* si eliminano

La regola di risoluzione in generale

$$\{l_1, l_2, \dots, l_i, \dots, l_k\} \quad \{m_1, m_2, \dots, m_j, \dots, m_n\}$$

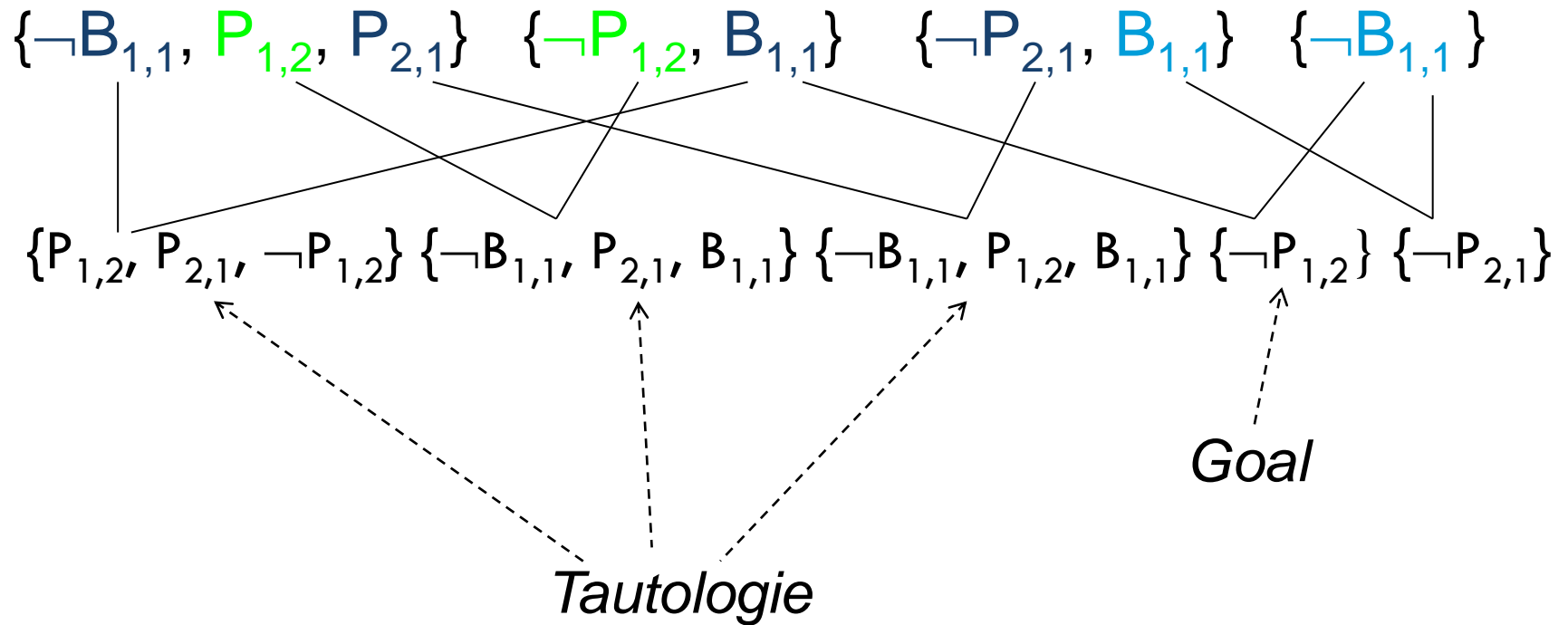
$$\{l_1, l_2, \dots, l_{i-1}, l_{i+1}, \dots, l_k, m_1, m_2, \dots, m_{j-1}, m_{j+1}, \dots, m_n\}$$

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

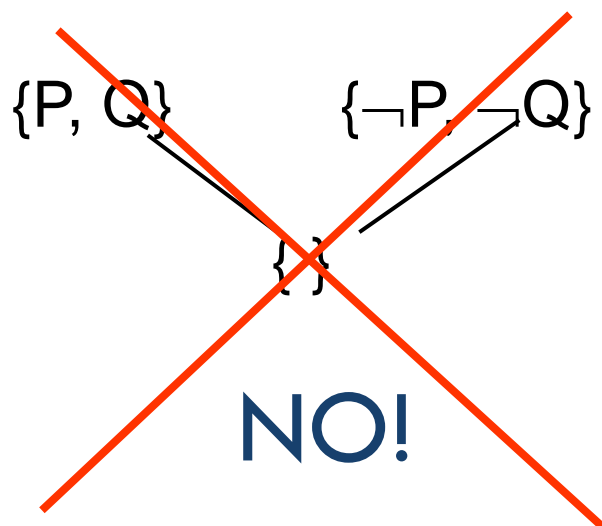
Caso particolare:

$$\frac{\{P\} \quad \{\neg P\}}{\{\}} \quad \text{clausola vuota}$$

Il grafo di risoluzione

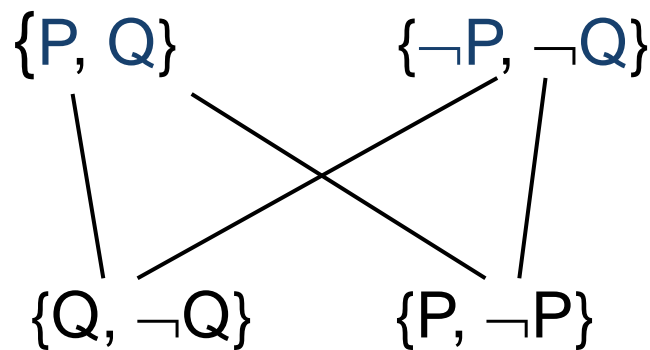


Attenzione!



Non è contraddittorio:

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

(*) *FC = Forma Clausale*

Refutazione

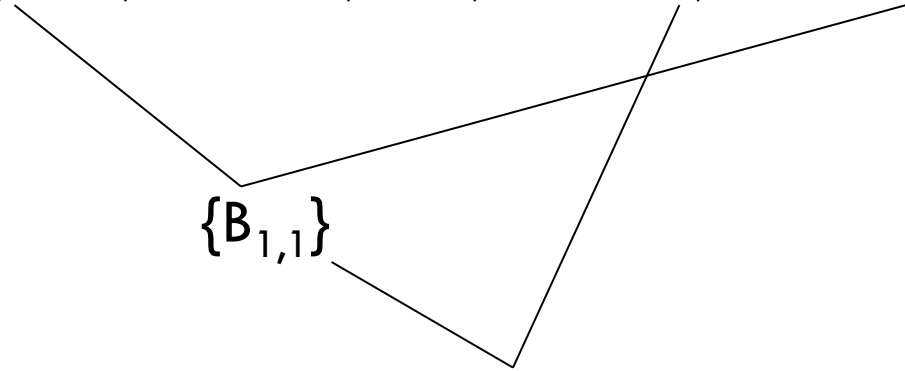
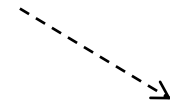
Goal negato

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

$\{B_{1,1}\}$

$\{\}$

Clausola vuota



Il teorema di risoluzione *ground*

- 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 $\{ \}$.
- Infatti:
- Se $RC(S)$ non contenesse $\{ \}$ potremmo costruire un modello di S
- Come? Sia $P_1, P_2 \dots P_k$ un ordinamento delle proposizioni. Assegniamo valori procedendo con $i=1, \dots, k$ 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
 - Se fosse $\{false, false \dots, false, \neg P_i\}$ e $\{false, false \dots, false, P_i\}$, allora la clausola vuota $\{ \}$ sarebbe in $RC(S)$

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!
 - $W_{1,1} \vee W_{1,2} \vee W_{1,3} \vee \dots \vee W_{4,4}$ *%almeno uno*
 - $\neg W_{i,j} \vee \neg W_{k,m}$ per ogni coppia di caselle *%esattam. uno*
- 16X15/2 = 120 asserzioni per dire che ce n’è al più uno!!!

Wumpus World: locazione, orientamento

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

Ma ... non va bene ($L_{2,1} \wedge L_{1,1}$ è inconsistente!)
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 ...
 - $\text{FacingRight}^1 \wedge \text{TurnLeft}^1 \Rightarrow \text{FacingUp}^2$

La rappresentazione del tempo

- **Fluenti** (Fatti che variano nel tempo)

$$L_{x,y}^t \Rightarrow (Breeze^t \Leftrightarrow B_{x,y})$$

$$L_{x,y}^t \Rightarrow (Stench^t \Leftrightarrow S_{x,y}) .$$

$$L_{1,1}^0 \wedge FacingEast^0 \wedge Forward^0 \Rightarrow (L_{2,1}^1 \wedge \neg L_{1,1}^1) .$$

- **Assiomi di stato**

$$F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t) .$$

- **Es.** $HaveArrow^{t+1} \Leftrightarrow (HaveArrow^t \wedge \neg Shoot^t) .$

$$L_{1,1}^{t+1} \Leftrightarrow (L_{1,1}^t \wedge (\neg Forward^t \vee Bump^{t+1}))$$

$$\vee (L_{1,2}^t \wedge (South^t \wedge Forward^t))$$

$$\vee (L_{2,1}^t \wedge (West^t \wedge Forward^t)) .$$

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 $KB \cup \{P_{i,j}, W_{i,j}\}$ è insoddisfacibile
- Con tutti questi simboli di proposizione
 - n fluenti $\Rightarrow 2^n$ stati fisici al tempo t quindi 2^{2^n} credenze possibili
 - servono procedure di inferenza efficienti (TTEntails e DPLL non sono praticabili)
 - serve un linguaggio più espressivo!!

Planning as search in Wumpus

function HYBRID-WUMPUS-AGENT(*percept*) **returns** an *action*
inputs: *percept*, a list, [*stench*,*breeze*,*glitter*,*bump*,*scream*]
persistent: *KB*, a knowledge base, initially the atemporal “wumpus physics”
t, a counter, initially 0, indicating time
plan, an action sequence, initially empty

TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))
 TELL the *KB* the temporal “physics” sentences for time *t*
 $safe \leftarrow \{[x, y] : \text{ASK}(KB, OK_{x,y}^t) = true\}$
if ASK(*KB*, $Glitter^t$) = true **then**
 $plan \leftarrow [Grab] + \text{PLAN-ROUTE}(current, \{[1,1]\}, safe) + [Climb]$
if *plan* is empty **then**
 $unvisited \leftarrow \{[x, y] : \text{ASK}(KB, L_{x,y}^{t'}) = false \text{ for all } t' \leq t\}$
 $plan \leftarrow \text{PLAN-ROUTE}(current, unvisited \cap safe, safe)$
if *plan* is empty and ASK(*KB*, $HaveArrow^t$) = true **then**
 $possible_wumpus \leftarrow \{[x, y] : \text{ASK}(KB, \neg W_{x,y}) = false\}$
 $plan \leftarrow \text{PLAN-SHOT}(current, possible_wumpus, safe)$
if *plan* is empty **then** // no choice but to take a risk
 $not_unsafe \leftarrow \{[x, y] : \text{ASK}(KB, \neg OK_{x,y}^t) = false\}$
 $plan \leftarrow \text{PLAN-ROUTE}(current, unvisited \cap not_unsafe, safe)$
if *plan* is empty **then**
 $plan \leftarrow \text{PLAN-ROUTE}(current, \{[1,1]\}, safe) + [Climb]$
action \leftarrow POP(*plan*)
 TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
 $t \leftarrow t + 1$
return *action*

function PLAN-ROUTE(*current*, *goals*, *allowed*) **returns** an action sequence

inputs: *current*, the agent’s current position

function HYBRID-WUMPUS-AGENT(*percept*) **returns** an *action*

inputs: *percept*, a list, [*stench*, *breeze*, *glitter*, *bump*, *scream*]

persistent: *KB*, a knowledge base, initially the atemporal “wumpus physics”

t, a counter, initially 0, indicating time

plan, an action sequence, initially empty

TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))

TELL the *KB* the temporal “physics” sentences for time *t*

$safe \leftarrow \{[x, y] : \text{ASK}(KB, OK_{x,y}^t) = true\}$

if ASK(*KB*, $Glitter^t$) = *true* **then**

$plan \leftarrow [Grab] + \text{PLAN-ROUTE}(current, \{[1,1]\}, safe) + [Climb]$

if *plan* is empty **then**

$unvisited \leftarrow \{[x, y] : \text{ASK}(KB, L_{x,y}^{t'}) = false \text{ for all } t' \leq t\}$

$plan \leftarrow \text{PLAN-ROUTE}(current, unvisited \cap safe, safe)$

if *plan* is empty and ASK(*KB*, $HaveArrow^t$) = *true* **then**

$possible_wumpus \leftarrow \{[x, y] : \text{ASK}(KB, \neg W_{x,y}) = false\}$

$plan \leftarrow \text{PLAN-SHOT}(current, possible_wumpus, safe)$

if *plan* is empty **then** // no choice but to take a risk

$not_unsafe \leftarrow \{[x, y] : \text{ASK}(KB, \neg OK_{x,y}^t) = false\}$

$plan \leftarrow \text{PLAN-ROUTE}(current, unvisited \cap not_unsafe, safe)$

if *plan* is empty **then**

$plan \leftarrow \text{PLAN-ROUTE}(current, \{[1,1]\}, safe) + [Climb]$

action \leftarrow POP(*plan*)

TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))

$t \leftarrow t + 1$

return *action*


```

plan ← [Climb] + PLAN-ROUTE(current, {[1,1]}, safe) + [Climb]
if plan is empty then
    unvisited ← {[x, y] : ASK(KB, Lx,yt') = false for all t' ≤ t}
    plan ← PLAN-ROUTE(current, unvisited ∩ safe, safe)
if plan is empty and ASK(KB, HaveArrowt) = true then
    possible_wumpus ← {[x, y] : ASK(KB, ¬ Wx,y) = false}
    plan ← PLAN-SHOT(current, possible_wumpus, safe)
if plan is empty then // no choice but to take a risk
    not_unsafe ← {[x, y] : ASK(KB, ¬ OKx,yt) = false}
    plan ← PLAN-ROUTE(current, unvisited ∩ not_unsafe, safe)
if plan is empty then
    plan ← PLAN-ROUTE(current, {[1, 1]}, safe) + [Climb]
action ← POP(plan)
TELL(KB, MAKE-ACTION-SENTENCE(action, t))
t ← t + 1
return action

```

function PLAN-ROUTE(*current*, *goals*, *allowed*) **returns** an action sequence

inputs: *current*, the agent's current position

goals, a set of squares; try to plan a route to one of them

allowed, a set of squares that can form part of the route

problem ← ROUTE-PROBLEM(*current*, *goals*, *allowed*)

return A*-GRAPH-SEARCH(*problem*)

Figure 7.20 A hybrid agent program for the wumpus world. It uses a propositional knowledge base to infer the state of the world, and a combination of problem-solving search and domain-specific code to decide what actions to take.

Come usare il PROP per pianificare

- Creare una formula che descriva
 - Stato iniziale
 - Possibil continuazioni
 - Il Goal
- Fornire la formula ad un SAT solver
- SE la soluzione esiste allora il modello di essa contiene
 - Assegnazioni di valori di verità alle azioni che in ordine costituiscono il piano
 - Oppure la soluzione ed il piano sono impossibili

Un algoritmo per il SATplan

```

function SATPLAN(init, transition, goal,  $T_{\max}$ ) returns solution or failure
  inputs: init, transition, goal, constitute a description of the problem
            $T_{\max}$ , an upper limit for plan length

  for  $t = 0$  to  $T_{\max}$  do
    cnf  $\leftarrow$  TRANSLATE-TO-SAT(init, transition, goal,  $t$ )
    model  $\leftarrow$  SAT-SOLVER(cnf)
    if model is not null then
      return EXTRACT-SOLUTION(model)
  return failure

```

Figure 7.22 The SATPLAN algorithm. The planning problem is translated into a CNF sentence in which the goal is asserted to hold at a fixed time step t and axioms are included for each time step up to t . If the satisfiability algorithm finds a model, then a plan is extracted by looking at those proposition symbols that refer to actions and are assigned *true* in the model. If no model exists, then the process is repeated with the goal moved one step later.

SummarAlzing

- Sintassi e semantica del Calcolo proposizionale
- Una rappresentazione proposizionale del mondo
 - Esempio: il mondo del Wumpus
- Conseguenza Logica e Inferenza nel PROP
- Algoritmi per la soddisfacibilità delle formule
 - Enumerazione delle tabelle di verità (TTEntails)
 - L'algoritmo di Davis Putnam
 - Il WalkSat
- Dimostrazione automatica:
 - Deduzione Naturale e backward chaining
 - La risoluzione
- Il mondo di Wumpus e l'uso della risoluzione