BASE DI CONOSCENZA

È UNA RAPPRESENTAZIONE ESPLICITA DI INFORMAZIONI & REGOLE, CONSENTE DEDUZIONI & INFERENZA. PUÒ CONTENERE:

- FATTI SPECIFICI, es. "SOCRATE È UN UOMO"
- REGOLE GENERALI, es. "TUTTI GLI UOMINI SONO MORTALI"

LA CAPACITÀ INFERENZIALE CONSENTE DI DEDURRE NUOVI FATTI BASANDOSI SU QUELLI ESISTENTI AD ESEMPIO DA "SOCRATE È UN UOMO" E "TUTTI GLI UOMINI SONO MORTALI", SI DEDUCE "SOCRATE È MORTALE".

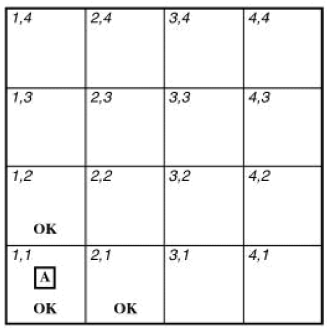
LA DIFFERENZA CON LE BASI DI DATI È CHE LE BASI DI DATI CONTENGONO SOLO FATTI SPECIFICI E NON CONSENTONO L'INFERENZA.

SCENARIO MONDO WUMPUS

SAPPIAMO CHE NON C'È UNA BREZZA NÉ UNA PUZZA IN 1,1

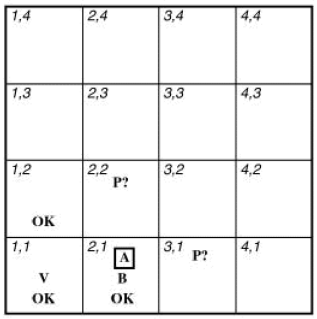
- *puzzo* nelle caselle adiacenti al Wumpus;
- *brezza* nelle caselle adiacenti alle buche;
- *luccichio* nelle caselle con l'oro;
- *bump* se sbatte in un muro;
- *urlo* se il Wumpus viene ucciso.
- L'agente non percepisce la sua locazione.

CONOSCENDO LE REGOLE PER CONSEGUENZA ABBIAMO CHE [1,2] & [2,1] SONO SAFE.



SPOSTANDOSI IN [2,1] SENTE UNA BREZZA

CONOSCENDO LE REGOLE PER CONSEGUENZA ABBIAMO CHE [2,2] & [3,1] UNA / TUTTE E DUE HANNO UNA BUCA



AGENTI BASATI SU CONOSCENZA

GLI AGENTI BASATI SU CONOSCENZA UTILIZZANO UN ARCHITETTURA BASATA SU:

- TELL → INSERISCE NUOVI FATTI NELLA KB
- ASK → INTERROGA LA KB PER DEDURRE AZIONI O VERIFICARE IPOTESI

Function Agente-KB (*percezione*) **returns** *un'azione*
  persistent: *KB*, una base di conoscenza
    t, un contatore, inizialmente a 0, che indica il tempo
  TELL(*KB*, Costruisci_Formula_Percezione(*percezione*, *t*))
  azione ← ASK(*KB*, Costruisci_Query_Azione(*t*))
  TELL(*KB*, Costruisci_Formula_Azione(*azione*, *t*))
  t ← *t* + 1
  return *azione*

→ PROGRAMMA DI UN AGENTE BASATO SU CONOSCENZA, UTILIZZA TELL, ASK PER DECIDERE LA PROSSIMA AZIONE

CONSEGUENZA LOGICA

DATO UN IN INSIEME DI FORMULE KB E UN ENUNCIATO α, SI DICE CHE α È CONSEGUENZA DI KB ($KB \models \alpha$) S.E.S. IN OGNI MODELLO IN CUI KB È VERA ANCHE α È VERA.

IN ALTRE PAROLE, α DEVE ESSERE VERA OGNI VOLTA CHE TUTTE LE PROPOSIZIONI IN KB SONO VERE.

ESEMPIO

$$KB = \{\neg A, \underset{*}{B}, C\}\{\neg B, A\}\{\neg C, A\}\{\neg A\} \models \underset{\alpha}{\{\neg B\}}$$

CLAUSOLE DELLA KB

A	B	C	¬A	(A∨¬C)	(A∨¬B)	*	KB	¬B
0	0	0	1	1	1	1	1	1
0	0	1	1	0	1	1	0	1
0	1	0	1	1	0	1	0	0
0	1	1	1	0	0	1	0	0
1	0	0	0	1	1	0	0	1
1	0	1	0	1	1	1	0	1
1	1	0	0	1	1	1	0	0
1	1	1	0	1	1	1	0	0

NELLA PRIMA RIGA TUTTE LE CLAUSOLE SONO VERE QUINDI LA KB È VERA E ANCHE α È VERA.
QUINDI α È CONSEGUENZA LOGICA DELLA KB ($KB \models \alpha$)

DEDUZIONE LOGICA

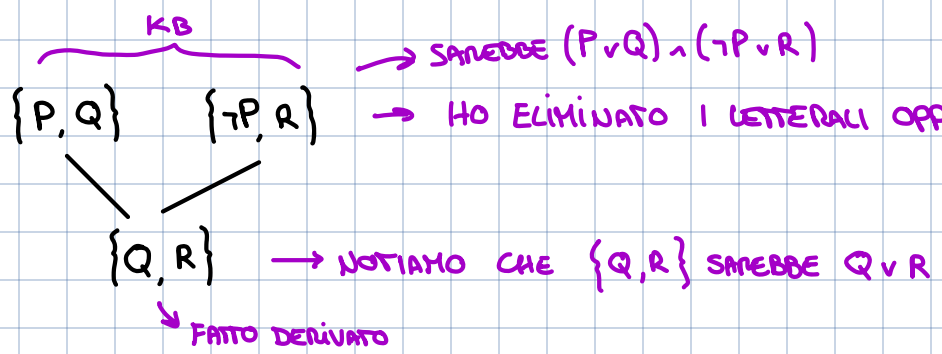
SI BASA SULL' APPLICAZIONE DI REGOLE DI INFERENZA PER DERIVARE UNA FORMULA A PARTENDO DA UN INSIEME DI FORMULE KB. SI USANO PROCEDURE DI INFERENZA COME IL MODUS PONENS E L'ELIMINAZIONE DELL' AND.

DEDUZIONE → SEQUENZA DI PASSAGGI IN CUI OGNI PASSO È GIUSTIFICATO DA UNA REGOLA D'INFERENZA, LA CONCLUSIONE FINALE È LA FORMULA CHE SI VUOLE DIMOSTRARE. SI INDICA CON:

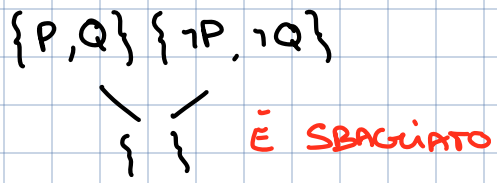
$KB \vdash A$

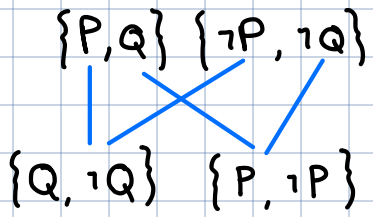
REGOLA DI RISOLUZIONE PER PROP

REGOLA DI INFERENZA CHE SI PUÒ UTILIZZARE PER DERIVARE NUOVI FATTI (PER INFER.) USA LA FORMA A CLAUSOLE.



NOTIAMO CHE POSSIAMO SVOLGERE SOLO UNA "ELIMINAZIONE" ALLA VOLTA. ESEMPIO:





- P e $\neg P$: genera $\{Q, \neg Q\}$.
- Q e $\neg Q$: genera $\{P, \neg P\}$.

GROUNDING

È IL PROCESSO CON IL QUALE L'AGENTE COLLEGA I SIMBOLI PRESENTI NELLA KB AL MONDO REALE. COME L'AGENTE FORMA LE SUE CREDENZE?

- PERCEZIONI → I SENSORI DELL'AGENTE CATTURANO INFORMAZIONI DAL MONDO REALE PER AGGIORNARE LA KB.
- APPRENDIMENTO → LE REGOLE NELLA KB POSSONO ESSERE APPRESE ATTRAVERSO INDUZIONE O ESPERIENZE PASSATE.

ONTOLOGIE

RAPPRESENTAZIONE FORMALE DI UN DOMINIO DI CONOSCENZA, INCLUDE:

- ENTITÀ → OGGETTI RILEVANTI DEL DOMINIO
- RELAZIONI → LEGAMI E INTERAZIONI TRA QUESTE ENTITÀ
- PROPRIETÀ → ATTRIBUTI CHE CARATTERIZZANO LE ENTITÀ
- VINCOLI → REGOLE CHE DEFINISCONO I COMPORTAMENTI ACCETTABILI

APPLICAZIONE ONTOLOGIE IN NPL

SONO FONDAMENTALI PER:

- COMPRENDERE IL LINGUAGGIO NATURALE → FORNENDO UNA STRUTTURA PER MAPPARE PAROLE E FRASI A CONCETTI BEN DEFINITI.
- DISAMBIGUAZIONE SEMANTICA → AIUTANDO A DISTINGUERE I SIGNIFICATI DI PAROLE POLISEMICHE, BASANDOSI SUL CONTESTO
- INFERENZA → CONSENTE DI DEDURRE INFORMAZIONI IMPLICITE A PARTIRE DAI FATTI (es. "ROSSI È PADRE" → "ROSSI È UNA PERSONA")

INGEGNERIZZAZIONE DELLE ONTOLOGIE NELLA PROGETTAZIONE DI UN AGENTE

DEFINIZIONE DELLO SCOPO DELL'ONTOLOGIA:

- IDENTIFICARE IL DOMINIO E I PROBLEMI CHE L'AGENTE DEVE RISOLVERE
  es. PER IL WUMPUS IL DOMINIO È IL LABIRINTO E LE ENTITÀ SONO: AGENTE, ORO, WUMPUS, BUCHE.

CREAZIONE DI UN'ONTOLOGIA DI ALTO LIVELLO (UPPER ONTOLOGY): DEFINIRE CATEGORIE ASTRATTE APPLICABILI A PIÙ DOMINI: OGGETTO, ENTITÀ AZIONE, RELAZIONE SPAZIALE.

SPECIFICAZIONE DEI DETTAGLI DEL DOMINIO: IDENTIFICARE ENTITÀ SPECIFICHE, PROPRIETÀ & RELAZIONI PARTICOLARI:

- ENTITÀ: ORO, WUMPUS, BUCA.
- RELAZIONI: "VICINO A", "CONTENUTO IN", "PERCEPITO DA"

SEMANTICA E MONDI POSSIBILI

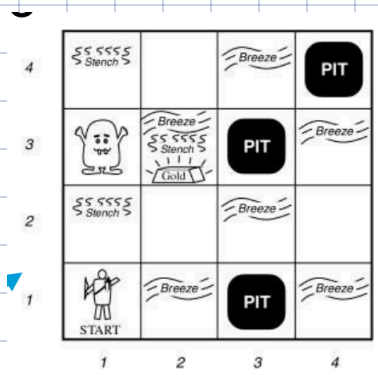
INTERPRETAZIONE → DEFINISCE UN VALORE DI VERITÀ PER TUTTI I SIMBOLI
V F

MODELLO → È UN INTERPRETAZIONE CHE RENDE VERA UNA FORMULA / INSIEME DI FORMULE

es.

$$\underbrace{P_{3,1} \rightarrow W_{1,3} \vee P_{1,2}}_{\text{INSIEME DI FORMULE}}$$

VERA IN QUESTA INTERPRETAZIONE



INTERPRETAZIONE DEL MONDO

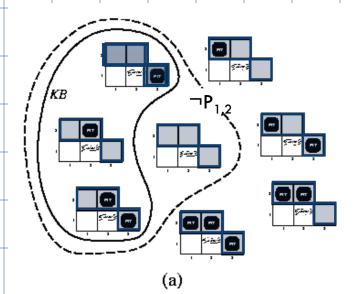
CONSEGUENZA LOGICA E MONDI POSSIBILI

SIA LA SEGUENTE KB $\{\underbrace{B_{2,1}}_{1}, \underbrace{\neg B_{1,1}}_{2}\}$

1 SAPPIAMO CHE C'È UNA BREZZA IN 2,1, CONOSCENDO LE REGOLE DEL WUMPUS SAPPIAMO CHE NELLE CASELLE ADIACENTI PUÒ ESSERCI UNA BUCA. CASELLE ADIACENTI $\{[1,1],[2,2],[3,1]\}$

② SAPP. CHE NON C'È UNA BREZZA IN 1,1, QUINDI NELLE CASELLE ADIACENTI NON PUÒ ESSERCI UNA BUCA. CASELLE ADIACENTI SENZA BUCA $\{[1,2]\}$

PER CONSEGUENZA LOGICA POSSIAMO SAPERE CHE NON C'È UNA BUCA IN [1,2] CHE RISCRIVIAMO COME $\neg P_{1,2}$. (P SAREBBE PIT = BUCA IN INGLESE)



(a)

LINEA CONTINUA = MONDI POSSIBILI IN BASE AI FATTI DELLA KB
→ LINEA TRATTEGGIATA = MONDI DEDOTTI PER CONSEGUENZA LOGICA

MODEL CHECKING

METODO PER VERIFICARE SE UNA CERTA PROPOSIZIONE LOGICA È VERA IN UN DATO MODELLO (STATO DEL MONDO).

SI VERIFICA SE UNA FORMULA, AD ESEMPIO α, È SODDISFATTA IN UN MODELLO M, CIOÈ SE $M \models \alpha$.
PER MODELLO INTENDIAMO UNA RAPPRESENTAZIONE FORMALE DEL MONDO CHE ASSEGNA VALORI DI VERITÀ ALLE PROPOSIZIONI.

SIA KB LA BASE DI CONOSCENZA, $KB \models \alpha$?

↳ SI ENUMERANO TUTTE LE POSSIBILI INTERPRETAZIONI DI KB, SE k SONO I SIMBOLI USATI CI SARANNO 2^k POSSIBILI MODELLI.

↳ PER CIASCUNA INTERPRETAZIONE (MODELLO):

- SE NON SODDISFA KB → VIENE IGNORATA
- SE SODDISFA KB SI CONTROLLA CHE SODDISFA ANCHE α.

POSSIAMO INDIVIDUARE 3 ALGORITMI PER VERIFICARE SE $KB \models \alpha$:

- TT-ENTAILS
- DPLL
- WALKSAT

→ ALGORITMI PER LA SODDISFACIBILITÀ (KB IN FORMA A CLAUSOLE)

TUTTI RITORNANO TRUE o FALSE SE $KB \models \alpha$

TT-ENTAILS

TT-ENTAILS COSTRUISCE LE TABELLE DI VERITÀ CON TUTTE LE COMBINAZIONI POSSIBILI DEI VALORI DI VERITÀ, CONTROLLA IN QUALI RIGHE KB È VERA, PER LE RIGHE TROVATE VERIFICA SE ANCHE α È VERA.

```
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 //  when KB is false, always return true
  else do
     P ← FIRST(symbols)
     rest ← REST(symbols)
     return (TT-CHECK-ALL(KB, α, rest, model ∪ {P = true})
             and
             TT-CHECK-ALL(KB, α, rest, model ∪ {P = false }))
```

INPUT:

- BASE DI CONOSCENZA KB
- FORMULA α
- ELENCO SIMBOLI PROPOSIZIONALI ANCORA DA ASSEGNARE
- INTERPRETAZ. PARZIALE DEI SIMBOLI

PROCEDURA RICORSIVA:

- QUANDO HA FINITO DI ASSEGNARE I SIMBOLI:
  - CONTROLLA SE L'INTERPRETAZIONE SODDISFA KB

    NO → IGNORA IL MODELLO

    SI → CONTROLLA SE SODDISFA α —NO→ RETURN FALSE

- SE CI SONO SIMBOLI DA ASSEGNARE
  - GENERA 2 RAMI, UNO VERO & UNO FALSO
  - ESPLORA ENTRAMBI I RAMI

- TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [A, B, C], [ ])
  - A=t • TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [B, C], [A=t])
    - B=t • TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [C], [A=t; B=t])
      - TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [ ], [A=t; B=t; C=t]) OK
      - TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [ ], [A=t; B=t; C=f]) OK
    - B=f • TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [C], [A=t; B=f])
      - TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [ ], [A=t; B=f; C=t]) OK %not a model for KB
      - TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [ ], [A=t; B=f; C=f] OK %not a model for KB
  - A=f • TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [B, C], [A=f])
    - TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [C], [A=f; B=t])
      - TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [ ], [A=f; B=t; C=t]) OK
      - TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [ ], [A=f; B=t; C=f]) OK %not a model for KB
    - TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [C], [A=f; B=f])
      - TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [ ], [A=f; B=f; C=t]) OK
      - TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [ ], [A=f; B=f; C=f]) OK %not a model for KB

NON EFFICIENTE

DPLL

UTILIZZA KB IN FORMA A CLAUSOLE. es. $\{A, B\}\ \{\neg B, C, D\}\ \{\neg A, F\}$

SI UTILIZZA PER DETERMINARE SE UNA FORMULA IN CNF È SODDISFACIBILE, $(A \vee B) \wedge (\neg B \vee C \vee D) \wedge (\neg A \vee F)$

TRASFORMAZIONE IN FORMA A CLAUSOLE

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}

IN DPLL CI SONO 3 MIGLIORAMENTI RISPETTO A TT-ENTAILS:

- TERMINAZIONE ANTICIPATA
  ↳ SE ALMENO UNA CLAUSOLA È FALSA, L'ALGORITMO TERMINA (XCHÉ È IN CNF).
- SIMBOLI PURI
  ↳ UN SIMBOLO CON STESSO SEGNO IN TUTTE LE CLAUSOLE $\{A, \neg B\}\{\neg B, \neg C\}\{C, A\}$

  PURO

  NON PURO
- EURISTICA CLAUSOLE UNITARIE
  ↳ QUANDO LA CLAUSOLA DIPENDE DA UN SOLO LETTERALE NON ASSEGNATO.

  {B, ¬C} è unitaria quando C = *True*

  SE C=TRUE ALLORA ¬C QUINDI È FALSA, ORA LA CLAUSOLA DIPENDE SOLO DA B DEVE ESSERE TRUE PER FAR SI CHE LA CLAUSOLA SIA VERA.

  SI PUÒ DECIDERE SULLA VERITÀ DI UNA CLAUSOLA ANCHE CON MODELLI PARZIALI, SE A È VERA LO SONO ANCHE LE CLAUSOLE $\{A, B\}\ \{A, C\}$ INDIPENDENTEMENTE DAI VALORI DI B & C.
  SE ANCHE UNA SOLA CLAUSOLA È FALSA ALLORA L'INTERPRETAZIONE COMPLESSIVA È FALSA.

L'ALGORITMO COMBINA LA RICERCA SISTEMATICA CON LA SEMPLIFICAZIONE PER RIDURRE IL NUMERO DI COMBINAZIONI DA ESPLORARE:

- ## PROPAGAZIONE UNITARIA

  - Se una clausola ha un **solo letterale non assegnato** (detta **unit clause**), quel letterale deve essere vero per soddisfare la clausola.
  - Ad esempio, se hai la clausola (A), allora A deve essere True.
  - L'algoritmo assegna il valore richiesto al letterale e semplifica la formula:
    - Elimina tutte le clausole che contengono quel letterale (perché sono già soddisfatte).
    - Rimuove le occorrenze del letterale negato $(\neg A)$ da altre clausole.

RICORDIAMOCI CHE SIAMO IN CNF

$(A \vee B) \wedge (\neg A \vee C \vee D)$ CON A NON ASSEGNATO

↓

ASSEGNO A = TRUE

↓

RIMUOVO $(A \vee B)$ XCHÉ CON A=TRUE LA CLAUSOLA È SODDISFATTA

↓

RIMUOVO $\neg A$ DA $(\neg A \vee C \vee D)$, ESSENDO IN OR O C O D DEVONO ESSERE TRUE

- ## ELIMINAZIONE CLAUSOLE VUOTE

  - Se durante il processo di semplificazione compare una **clausola vuota** (cioè una clausola senza letterali), la formula è **insoddisfacibile** (UNSAT).
  - In tal caso, l'algoritmo termina immediatamente restituendo "non soddisfacibile".

→ IN CNF SE UNA CLAUSOLA È VUOTA È AUTOMATICAMENTE FALSA. LA DISGIUNZIONE RICHIEDE ALMENO UN ELEMENTO VERO.

- ## ASSEGNAZIONE EURISTICA (SCELTA DEL LETTERALE)

  - Se non ci sono clausole unitarie, l'algoritmo sceglie un letterale arbitrario (di solito con una strategia euristica, come scegliere la variabile più frequente) e assegna un valore (vero o falso).
  - L'algoritmo esplora ricorsivamente le due possibilità:
    - Prima suppone che il letterale sia True.
    - Se non trova una soluzione, torna indietro (backtracking) e prova con False.

- ## BACKTRACKING

  - Se un'assegnazione porta a una contraddizione (ad esempio, a una clausola vuota), l'algoritmo "torna indietro" per provare un'altra assegnazione.

- ## TERMINAZIONE

  - L'algoritmo termina quando:
    1. Tutte le clausole sono soddisfatte (formula soddisfacibile, SAT).
    2. Viene dimostrato che non esiste alcuna assegnazione possibile che soddisfi la formula (formula insoddisfacibile, UNSAT).

WALKSAT

UTILIZZATO PER RISOLVERE PROBLEMI DI SODDISFACIBILITÀ BOOLEANA. ESPLORA LO SPAZIO DELLE SOLUZIONI PASSEGGIANDO NELLO SPAZIO DELLE POSSIBILI ASSEGNAZIONI.

INPUT → FORMULA IN CNF $(\vee\ \vee) \wedge (\vee\ \vee) \ldots$

1. INIZIALIZZAZIONE

   ↳ ASSEGNA UN VALORE DI VERITÀ T o F A CIASCUNA VARIABILE. (SOLUZIONE CORRENTE)

2. CICLO PRINCIPALE, RIPETI FINCHÉ NON SODDISFA TUTTE LE CLAUSOLE O RAGGIUNGE UN LIMITE DI ITERAZIONI.

- 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*)

SE MAX-FLIPS = ∞ E L'INSIEME DELLE CLAUSOLE È SODDISFACIBILE PRIMA O POI TERMINA, SE INSODDISFACIBILE NON TERMINA.

NON PUÒ ESSERE USATO PER VERIFICARE L'INSODDISFACIBILITÀ.

```
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
```