Introduzione alla rappresentazione della conoscenza

ovvero…

Come costruire agenti *basati su conoscenza*
e dotati di *capacità di ragionamento*

Maria Simi, 2014/2015

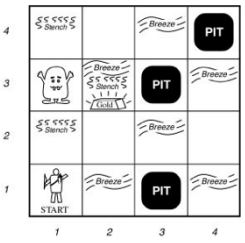
Che cosa abbiamo fatto fin'ora …

- Abbiamo trattato:
  - agenti con stato e con obiettivo in mondi osservabili con stati e azioni descrivibili in maniera semplice
  - enfasi sul processo di ricerca
  - abbiamo brevemente visto come si possono rilassare alcune assunzioni (es. ricerca and-or e insiemi credenza)
- Vogliamo adesso migliorare le capacità razionali dei nostri agenti dotandoli di rappresentazioni di mondi più complessi, non descrivibili semplicemente
- Agenti *basati su conoscenza,* con conoscenza espressa in maniera esplicita e dichiarativa (non cablata)

Perché?

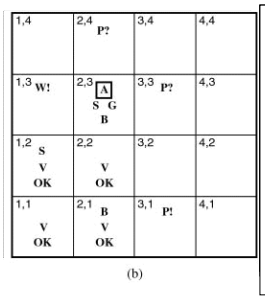
- Il mondo è tipicamente complesso: ci serve una rappresentazione *parziale* e *incompleta* di una astrazione del mondo utile agli scopi dell'agente
- Per ambienti parzialmente osservabili ci servono linguaggi di rappresentazione della conoscenza *più espressivi* e *capacità inferenziali*
- La maggior parte dei problemi di I.A. sono "*knowledge intensive*" tanto che *Sistemi Basati sulla Conoscenza* è quasi sinonimo di sistemi di I.A.

Il mondo del Wumpus: un esempio



- *Misura delle prestazioni*:
  - +1000 se trova l'oro, torna in [1,1] e esce;
  - -1000 se muore;
  - -1 per ogni azione;
  - -10 se usa la freccia.
- Percezioni:
  - *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.
- *Azioni*:
  - *avanti*
  - *a destra* di 90°, *a sinistra* di 90°
  - *afferra* un oggetto
  - *scaglia la freccia* (solo una)
  - Esce
- Ambienti generati a caso ([1,1] safe)

Il mondo del Wumpus: uno scenario



- L'agente si sposta in [2,2] e poi in [2,3]
- Qui percepisce un luccichio, afferra l'oro e torna sui suoi passi, percorrendo caselle OK

Agente basato su conoscenza

- Un agente basato su conoscenza mantiene una *base di conoscenza* (KB): un insieme di *enunciati* espressi in un linguaggio di rappresentazione
- Interagisce con la KB mediante una interfaccia funzionale *Tell-Ask*:
  - *Tell*: per aggiungere nuovi fatti a KB
  - *Ask*: per interrogare la KB
  - … forse *Retract*
- Le risposte α devono essere tali che α è una conseguenza della KB (è *conseguenza logica* di KB)

Il problema da risolvere

- *Il problema*: data una base di conoscenza KB, contenente una rappresentazione dei fatti che si ritengono veri, vorrei sapere se un certo fatto α è vero di conseguenza
  KB $\models \alpha$ (*conseguenza logica*)

Programma di un agente B.C.

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*

Agente basato su conoscenza

- Approccio dichiarativo vs approccio procedurale
- La differenza principale è che la KB racchiude tutta la conoscenza necessaria a decidere l'azione da compiere in forma *dichiarativa*
- L'alternativa (*approccio procedurale*) è scrivere un programma in cui il processo decisionale è cablato, una volta per tutte.
- Più flessibile: più semplice acquisire conoscenza incrementalmente e modificare il comportamento con l'esperienza

Base di conoscenza …

- *Base di conoscenza*: una rappresentazione esplicita, parziale e compatta, in un linguaggio simbolico, che contiene:
  - fatti di tipo specifico (Es. *Socrate è un uomo*)
  - fatti di tipo generale, o regole (Es. *Tutti gli uomini sono mortali*)
- Quello che caratterizza una B.C. è la *capacità di inferire* nuovi fatti da quelli memorizzati esplicitamente (Es. *Socrate è mortale*)

… e base di dati

… e base di dati

- Nelle basi di dati solo fatti specifici e positivi
- Le basi di dati assumono una conoscenza completa del mondo (Closed World Assumption)
- Nessuna capacità inferenziale (vincoli di integrità solo per il controllo, non per la generazione)

Il *trade-off* fondamentale della R.C.

- Il problema 'fondamentale' in R.C. è trovare il giusto compromesso tra:
  - *Espressività* del linguaggio di rappresentazione;
  - *Complessità* del meccanismo inferenziale
- Vogliamo linguaggi *espressivi,* ma anche *efficienti.*
- Questi due obiettivi sono in contrasto e si tratta di mediare tra queste due esigenze

Espressività come *imprecisione*

- Cosa vuol dire *espressivo*? … e perchè l'espressività è in contrasto con l'efficienza?
- Un linguaggio più espressivo ci consente di essere vaghi, imprecisi, di esprimere conoscenze parziali, di omettere dettagli che non si conoscono …
- L'espressività determina non tanto quello che può essere detto ma quello che può essere *lasciato non detto*

Espressività come *imprecisione* : esempi

Nelle B.D. quello che possiamo esprimere sono solo fatti specifici e positivi:

1. *Moglie*(*Rossi*, *Paola*)

Con linguaggi più espressivi …

2. $\exists x$ *Moglie*(*Rossi*, x) *Rossi ha una moglie*
3. $\neg$*Operaio*(*Rossi*) *Rossi non è un operaio*
4. *Moglie*(*Rossi*, *Anna*) $\vee$ *Moglie*(*Rossi*, *Paola*)
   Rossi ha una moglie; si chiama Anna o Paola
5. $\forall y \exists x$ *Moglie*(y, x) $\Rightarrow$ *Coniugato*(y)
   Coloro che hanno una moglie sono ***coniugati***

Espressività e complessità inferenziale

- Nelle basi di dati nessuna deduzione è possibile, solo *recupero*. Si assume una descrizione *completa* del mondo.
- Dai fatti
  - $\exists$ *x Moglie*(*Rossi*, x)
  - $\forall y \exists$ x *Moglie*(y, x) $\Rightarrow$ *Coniugato*(y)

  è possibile dedurre *Coniugato*(*Rossi*)
- Dai fatti
  - *Moglie*(*Rossi*, *Anna*) $\vee$ *Moglie*(*Rossi*, *Paola*)
  - $\forall y \exists$ x *Moglie*(y, x) $\Rightarrow$ *Coniugato*(y)

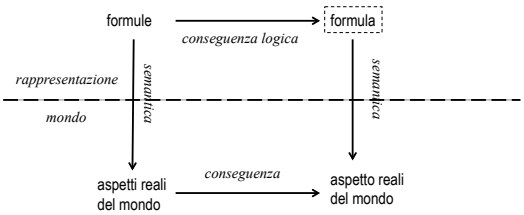
  è possibile dedurre *Coniugato*(*Rossi*) ma è più complicato (richiede un ragionamento per casi)

Formalismi per la R.C.

Un formalismo per la rappresentazione della conoscenza ha tre componenti:

1. Una *sintassi*: un linguaggio composto da un vocabolario e regole per la formazione delle frasi (*enunciati*)
2. Una *semantica*: che stabilisce una corrispondenza tra gli enunciati e fatti del mondo; se un agente ha un enunciato α nella sua KB, crede che il fatto corrispondente sia vero nel mondo
3. Un *meccanismo inferenziale* (codificato o meno tramite regole di inferenza come nella logica) che ci consente di inferire nuovi fatti.

Rappresentazione e mondo



Grounding (radicamento)

- Come sappiamo che la KB è vera nel mondo reale? Come l'agente forma le sue credenze?
- Attraverso i sensori si crea una connessione con il mondo; le credenze sono il risultato di percezioni.
- Non solo: le regole sono il risultato di un processo di apprendimento, che può essere fallibile (es. ragionamento induttivo o altro meccanismo di astrazione o abduzione).

Logica come linguaggio per la R.C.

- I linguaggi logici, calcolo proposizionale (PROP) e logica dei predicati (FOL), sono adatti per la rappresentazione della conoscenza?
- Qual è la complessità computazionale del problema KB $\models \alpha$ nei vari linguaggi?
  - In PROP il problema è decidibile, ma intrattabile (NP)
  - FOL è un linguaggio espressivo, con una semantica ben definita, ma ha un problema: il meccanismo inferenziale non è decidibile

    In FOL il problema KB $\models \alpha$ è semidecidibile

Linguaggi per la R.C.: efficienza

1. Superamento del FOL verso linguaggi ad inferenza limitata: *contrazioni* del FOL alla ricerca di proprietà computazionali migliori (es. i linguaggi di *programmazione logica*, le logiche *descrittive*)
2. Linguaggi di rappresentazione che propongono meccanismi di strutturazione della conoscenza per guadagnare efficienza su forme particolari di inferenza (es. *reti semantiche* e connettività, *frame* e aggregazione, *ereditarietà*). FOL per la semantica.

Limiti in espressività del FOL

Molti linguaggi della R.C. sono *estensioni* [di sottoinsiemi] del FOL per superare limiti di espressività nel ragionamento di "senso comune"
Ne possiamo citare tre importanti:

- Atteggiamenti proposizionali
- Ragionamento incerto
- Ragionamento non monotòno

Atteggiamenti proposizionali

- Atteggiamenti epistemici
  - conoscenze, credenze (convinzioni o opinioni)
- Atteggiamenti motivazionali
  - desideri, obiettivi, intenzioni, …
- L'oggetto del discorso sono le proposizioni

  Bel(P) operatori e logiche modali

  Bel('P') reificazione o meta-livello

Ragionamento incerto

- Nella logica classica le proposizioni sono vere o false (*assunzione epistemologica*)
- Il superamento della dicotomia T|F può avvenire in modi diversi:
  - logiche a più valori (vero, falso, non so)
  - ragionamento probabilistico (vero con probabilità *p*)
  - vero con grado di fiducia c
  - logica *fuzzy* (proprietà sfumate, es. 'alto' in misura *m*)

Ragionamento non monotòno

Nella logica classica vale la proprietà di *monotonia*:
Monotonia: Se KB $\vdash \alpha$ allora KB $\cup \{\beta\} \vdash \alpha$

- Il ragionamento di senso comune è spesso *non monotòno:* si fanno inferenze tentative, anche in mancanza di informazioni complete.
- Esempio 1: ragionamento *default*

  Gli uccelli tipicamente volano. Tweety è un uccello. Quindi Tweety vola.
- Esempio 2: assunzione di mondo chiuso

  Se un fatto non è presente nella KB si assume che non sia vero (come nelle basi di dati). Quando si aggiunge un nuovo fatto può invalidare le vecchie conclusioni.

Assunzioni ontologiche

- Ogni linguaggio per la R. C. fa assunzioni diverse su come è fatto il mondo (ontologico ≅ che riguarda ciò che esiste):
- Nel calcolo proposizionale il mondo è visto come popolato di *fatti* veri o falsi (le proposizioni).
- Il calcolo dei predicati fa una assunzione *ontologica* più sofisticata: il mondo è fatto di *oggetti*, che hanno *proprietà* e tra cui sussistono *relazioni*.
- Logiche specializzate assumono *ontologie* più ricche:
  - gli stati e le azioni nel *calcolo di situazioni*
  - il tempo nelle *logiche temporali*
  - concetti o categorie nelle *logiche descrittive*

Che cosa vedremo

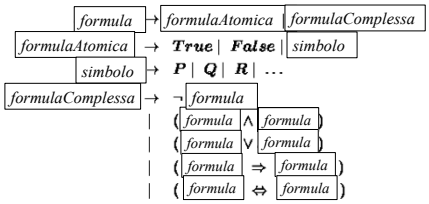
- Rivisitazione di PROP e FOL per la rappresentazione della conoscenza
- … con attenzione agli algoritmi e alla complessità
- Contrazioni ed estensioni
  - I linguaggi a regole
  - La programmazione logica
  - Le logiche descrittive

Agenti logici: calcolo proposizionale

Maria Simi
a.a. 2014/2015

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

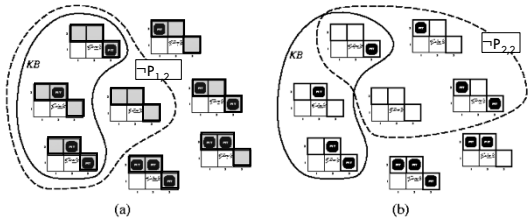
- Una formula A è *conseguenza logica* di un insieme di formule KB se e solo se in ogni modello di KB, anche A è vera (KB $\models$ A)
- Indicando con $M(\alpha)$ l'insieme delle interpretazioni che rendono α vera, i modelli di α, e con M(KB) i modelli di un insieme di formule KB …

$$KB \models \alpha \text{ sse } M(KB) \subseteq M(\alpha)$$

Esempio dal mondo del Wumpus

- KB = {$B_{2,1}$, $\neg B_{1,1}$, + *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 = {$B_{2,1}$, $\neg B_{1,1}$ + *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 ($KB_{\wedge}$ $\wedge$ ¬A) è insoddisfacibile

Tabella di verità

- (¬A ∨ B) ∧ (A ∨ C) $\models$ (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($KB, \alpha, symbols, []$)

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*
  else do
    $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
    return TT-CHECK-ALL($KB, \alpha, rest, [P = true|model]$) **and**
      TT-CHECK-ALL($KB, \alpha, rest, [P = false|model]$)

Esempio di TT-Entails?

(¬A ∨ B) ∧ (A ∨ C) $\models$ (B ∨ C) ?

- TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [A, B, C], [ ])
  - TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [B, C], [A=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
    - 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
      - TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨C), [ ], [A=t; B=f; C=f] OK
  - TT-CHECK-ALL((¬A∨B)∧(A∨C), (B∨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 ∨ B) ∧ (¬B ∨ C ∨ D) ∧ (¬A ∨ 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 ⇔: $(A \Leftrightarrow B) \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$
2. Eliminazione dell' ⇒: $(A \Rightarrow B) \equiv (\neg A \vee B)$
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 ∨ su ∧:
   $(A \vee (B \wedge C)) \equiv (A \vee B) \wedge (A \vee C)$

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 tre miglioramenti rispetto a TTEntails:
  1. Terminazione anticipata
  2. Euristica dei simboli (o letterali) puri
  3. 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 anche una sola clausola è falsa l'interpretazione 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, ¬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. {B} è unitaria ma anche …

  {B, ¬C} è unitaria quando C = *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, [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

Miglioramenti di DPLL

- DPLL è completo
- Alcuni miglioramenti visti per i CSP si applicano anche qui:
  - Analisi di componenti (sotto-problemi indipendenti): se le variabili possono essere suddivise in sotto-insiemi disgiunti (senza simboli in comune)
  - Ordinamento di variabili e valori: scegliere la variabile che compare in più clausole
  - Backtracking intelligente e altri trucchi …

Metodi locali per SAT: formulazione

- Gli stati sono gli assegnamenti completi
- 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]

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

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
- 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

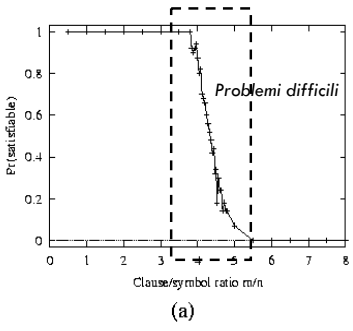
Problemi SAT difficili

- 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

Probabilità di soddisfacibilità in funzione di m/n

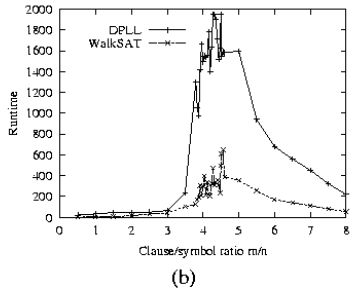


(a)

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

media su 100 problemi generati a caso

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)
- 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

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

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:

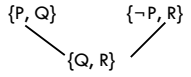
  Da A, B: $A \wedge B$ $A \wedge (A \wedge B)$ … $A \wedge (A \wedge (A \wedge B))$
- Meglio *all'indietro*
  - se si vuole dimostrare $A \wedge B$, si cerchi di dimostrare A e poi B
  - se si vuole dimostrare $A \Rightarrow B$, si assuma A e si cerchi di dimostrare B
  - …

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

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

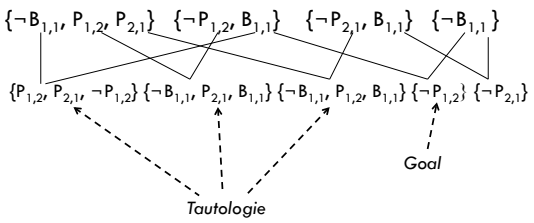
Gli l e m sono letterali, simboli di proposizione positivi o negativi; l_i e m_j *sono uguali e di segno opposto*

Caso particolare

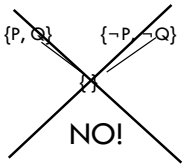
$$\frac{\{P\} \quad \{\neg P\}}{\{\}}$$

clausola vuota

Il grafo di risoluzione

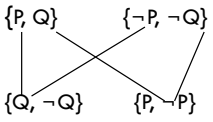


Attenzione!



Non è contradditorio:

Es. Bianco o nero e
non bianco o non nero



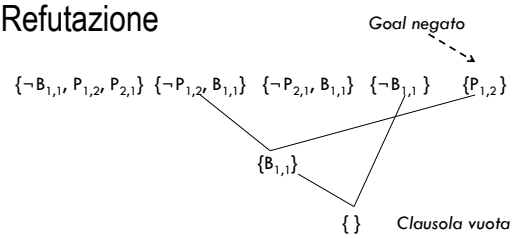
… e qui ci fermiamo

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? Sì, 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 *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 { }.
- 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
  - Se fosse {false, false …, false, $\neg P_i$} e {false, false …, false, P_i}, { } 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!
  - $W_{1,1} \vee W_{1,2} \vee W_{1,3} \vee \ldots W_{4,4}$ 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!!