Rappresentazione e Ragionamento Proposizionale

Nicola Fanizzi
Ingegneria della Conoscenza
CdL in Informatica • *Dipartimento di Informatica*
Università degli studi di Bari Aldo Moro

indice

Proposizioni

Proposizioni

Proposizioni: *enunciati* riguardanti il mondo

- pongono vincoli su quello che potrebbe essere vero, definibili:
  - *estensionalmente* — tabelle di assegnazioni lecite a variabili
  - *intensionalmente* — formule

Logica Proposizionale (*propositional calculus*)
linguaggio atto a rappresentare intensionalmente *vincoli* e *interrogazioni*

- *formule*: relazioni tra variabili
  - più concise e leggibili dell'equivalente estensionale
  - utili a rendere il *ragionamento più efficiente*
- *modularità* → *debugging* più facile
  - piccoli cambiamenti nel problema → poche modifiche alla base di conoscenza
- *risposte* alle interrogazioni *più ricche* delle semplici assegnazioni
- *estendibile* per ragionare su individui e relazioni fra di essi

Sintassi

Frasi formulate in un *linguaggio* con variabili booleane e connettivi logici (simboli poi associati a una semantica)

Proposizione atomica, o **Atomo** — simbolo/stringa di bit

- per *convenzione*: stringa alfanumerica, con eventuali "_", che inizia con una lettera *minuscola*
  - ad es. ai_is_fun, $accesa\_l_1$, $piove$, $terminato$, $nuvoloso$

Proposizione o **formula logica**: *atomica oppure proposizione* **composta**

- formata usando *connettivi* logici e proposizioni *più semplici*

Connettivi logici, in ordine di *precedenza*: $\neg \land \lor \to \leftarrow \leftrightarrow$

- date le proposizioni p e q:

sintassi	lettura	definizione
$\neg p$	"non p"	*negazione* di p
$p \land q$	"p e q"	*congiunzione* di p e q
$p \lor q$	"p o q"	*disgiunzione* di p e q
$p \to q$	"p implica q"	*implicazione* di q da p
$p \leftarrow q$	"p se q"	*implicazione* di p da q
$p \leftrightarrow q$	"p se e solo se q"	*equivalenza* di p e q

- *parentesi* per disambiguare / maggiore leggibilità
  - ad es. $\neg a \lor b \land c \to d \land \neg e \lor f$
    abbreviazione di $((\neg a) \lor (b \land c)) \to ((d \land (\neg e)) \lor f)$

Semantica — definisce il *significato* delle frasi/proposizioni

- *corrispondenza* tra simboli e stato del mondo rappresentato
- proposizioni: *interpretabili* come vere o false
  - *atomo*, unità *minima* dotabile di significato
  - significato di proposizioni più complesse, discende da quello degli atomi

Interpretazione $\pi : \text{Atomi} \to \{true, false\}$

- se $\pi(a) = true$, a **vero** nell'interpretazione π
- se $\pi(a) = false$, a **falso** nell'interpretazione π

definibile anche come insieme I di atomi associati a $true$

- tutti gli altri interpretati come $false$

Estensione dell'interpretazione a proposizioni (non atomiche):

- determinata dalle *tavole di verità*:

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \leftarrow q$	$p \rightarrow q$	$p \leftrightarrow q$
$true$	$true$	$false$	$true$	$true$	$true$	$true$	$true$
$true$	$false$	$false$	$false$	$true$	$true$	$false$	$false$
$false$	$true$	$true$	$false$	$true$	$false$	$true$	$false$
$false$	$false$	$true$	$false$	$false$	$true$	$true$	$true$

- verità delle proposizioni dipende solo dall'interpretazione degli atomi
  - diverse interpretazioni → possibili diversi valori di verità delle proposizioni

Esempio — Considerati ai_is_fun, $happy$ e $light\_on$

sia data I_1 che assegna

- $true$ a ai_is_fun,
- $false$ a $happy$,
- $true$ a $light\_on$

ossia π_1 definita da

- $\pi_1(ai\_is\_fun) = true$
- $\pi_1(happy) = false$
- $\pi_1(light\_on) = true$

allora in I_1:

- ai_is_fun vera e $\neg ai\_is\_fun$ falsa
- $happy$ falsa e $\neg happy$ vera
- $ai\_is\_fun \vee happy$ vera
- $ai\_is\_fun \leftarrow happy$ vera
- $ai\_is\_fun \rightarrow happy$ falsa
- $ai\_is\_fun \leftarrow happy \wedge light\_on$ vera

(..cont.)

Data l'interpretazione I_2 che assegna:

- $false$ a ai_is_fun,
- $true$ a $happy$
- $false$ a $light\_on$

allora in I_2:

- ai_is_fun falsa
- $\neg ai\_is\_fun$ vera
- $happy$ vera
- $\neg happy$ falsa
- $ai\_is\_fun \vee happy$ vera
- $ai\_is\_fun \leftarrow happy$ falsa
- $ai\_is\_fun \leftarrow light\_on$ vera
- $ai\_is\_fun \leftarrow happy \wedge light\_on$ vera

Basi di Conoscenza

Base di Conoscenza KB — insieme di proposizioni dette assiomi:

- da considerare come vere (senza dim.)
- servono a caratterizzare il mondo da rappresentare attraverso la sua **interpretazione intesa**
  - quella nella mente dell'*esperto di dominio*
  - da formalizzare: *assiomatizzazione*
    compito dell'*ingegnere della conoscenza*

Teoria dei Modelli e Conseguenze Logiche

Modello di KB: interpretazione per la quale sia *vero* ogni assioma di KB

La proposizione g (*goal*) è **conseguenza logica** di KB,
sse è vera in ogni modello di KB:

$$KB \models g$$

si dice anche che "g segue logicamente da KB" (in Inglese: KB entails g)

Quindi:

- se $KB \models g$, non esistono modelli di KB per i quali g sia falsa
- $KB \not\models g$, i.e. g non è conseguenza logica di KB,
  comporta che esiste un modello di KB in cui g è falsa

⚠ altre interpretazioni di g che non siano modelli di KB non contano

Esempio — Data la KB contenente:

- $sam\_is\_happy.$
- $ai\_is\_fun.$
- $worms\_live\_underground.$
- $night\_time.$
- $bird\_eats\_apple.$
- $apple\_is\_eaten \leftarrow bird\_eats\_apple.$
- $switch\_1\_is\_up \leftarrow sam\_is\_in\_room \wedge night\_time.$ [☾]

allora:

- $KB \models bird\_eats\_apple.$
- $KB \models apple\_is\_eaten.$

mentre:

- $KB \not\models switch\_1\_is\_up$ **ossia** $\exists$ **modello di** KB **in cui** $switch\_1\_is\_up$ **falsa:**
  - **in tale modello l'implicazione [☾] resta vera anche qualora** $sam\_is\_in\_room$ **fosse falsa**

La Prospettiva dell'Ingegnere della Conoscenza

Per caratterizzare un dominio tramite la KB,
il/la *progettista* deve definire il mondo come *interpretazione intesa*:

- significato dei simboli definite attraverso proposizioni:
  assiomi che si possono assumere veri su tale mondo
  → conseguenze logiche di KB *vere* rispetto all'*interpretazione intesa*
- significato *condiviso*: per interpretare le risposte a interrogazioni alla KB

$\models$ *relazione semantica* fra KB e g

- KB e g *simboliche*: rappresentabili nelle macchine
  ma significato riferito a un mondo esterno tipicamente *non simbolico*
- $\models$ NON comporta computazione o dimostrazione
  - specifica solo la verità di quello che consegue da certe assunzioni

Metodologia di *progetto* della KB: fasi

1. si decide il *dominio*/mondo da rappresentare → interpretazione intesa
   - (aspetti del) mondo *reale*
     - ad es. la struttura dei corsi e gli studenti universitari,
       un laboratorio in un particolare momento
   - mondo *immaginario*
     - ad es. mondo di Pinocchio in letteratura,
       o situazioni-limite in domotica
   - mondo *astratto*
     - ad es. numeri e insiemi
2. si scelgono gli *atomi* per rappresentare le proposizioni d'interesse
   - con significato preciso rispetto all'interpretazione intesa
3. si definiscono le *proposizioni* che saranno vere nell'interpretazione intesa:
   assiomatizzazione del dominio
4. si pongono al sistema **domande** sulla KB
   - **risposte** da interpretare in base ai significati associati ai simboli

Osservazioni

- Solo al passo 3. si comunica con la *macchina*
  - prima solo progettazione
- *Documentazione* dei simboli utile agli altri per
  - *ricordare* il loro significato
  - *controllare* la veridicità delle proposizioni
- Specifica dei significati dei simboli: **ontologia**
  - informalmente, come commenti
  - oppure attraverso linguaggi formali ai fini dell'*interoperabilità semantica*
    - capacità di usare insieme diverse KB
- Fase 4. utile a chi comprende il significato dei simboli
  e sa interpretarli rispetto al mondo considerato
  - nelle domande e nelle risposte
  - conta l'attendibilità degli assiomi definiti dal progettista

La Prospettiva della Macchina

Un KBS lavora sugli assiomi della KB, assunti come veri

- modelli di KB: tutti e soli i modi in cui il mondo potrebbe essere fatto
  - se il significato dei simboli è stato codificato correttamente, l'interpretazione intesa è un modello di KB
  - il sistema lo sa ma non sa quale modello sia
- per determinare la verità di g nell'interpretazione intesa, il sistema deve decidere se g è *conseguenza logica* di KB
  - inferenza tramite procedure di dimostrazione trattate in seguito
  - se $KB \models g$, potrà concludere che g è vera anche in tale modello
    - g vera in tutti i modelli di KB, quindi anche nell'interpretazione intesa
  - se $KB \not\models g$, allora non può trarre conclusioni
    - g *potrebbe* essere falsa proprio nell'interpretazione intesa ma il sistema non conoscendola non può stabilirlo

⚠ Si noti che $KB \not\models g$ non comporta necessariamente $KB \models \neg g$

Esempio data KB dell'esempio precedente:

- l'utente saprebbe interpretare il significato dei simboli
- la macchina non comprende il significato,
  ma può trarre conclusioni basandosi su quello che è asserito, ad es.:
  - $KB \models apple\_is\_eaten$ (i.e. vera nell'interpretazione intesa)
    - $bird\_eats\_apple. \in KB$
    - $apple\_is\_eaten \leftarrow bird\_eats\_apple. \in KB$
  - $KB \not\models switch\_1\_is\_up$: falsa in qualche modello
    - $switch\_1\_is\_up \leftarrow sam\_is\_in\_room \wedge night\_time. \in KB$
    - $night\_time. \in KB$
    - $sam\_is\_in\_room$ non è certo:
      - potrebbe essere falso in qualche suo modello

Osservazioni

- Se il/la progettista commette errori, codifica nella KB assiomi falsi nell'interpretazione intesa, *non è garantito* che le risposte della macchina siano vere in tale interpretazione
- Le macchine *non comprendono* il significato dei simboli: sono gli umani ad attribuirglielo
  - sanno solo quanto viene loro detto del mondo
  - ma da questo sono in grado di trarre conclusioni vere per quel mondo

Vincoli Proposizionali

Vincoli Proposizionali

Formule logiche: vincoli con struttura concisa ed elaborabile

Classe dei **CSP proposizionali** caratterizzata da:

- *Variabili booleane*: dominio $\{true, false\}$
  - sintassi: $X = true$ oppure x, e $X = false$ oppure $\neg x$
    - ad es. data $Happy$ booleana,
      $happy$ sta per $Happy = true$ e
      $\neg happy$ sta per $Happy = false$
- *Vincoli clausali* o **clausole** — espressioni logiche della forma:

$$l_1 \vee l_2 \vee \cdots \vee l_k,$$

  - ogni **letterale** l_i atomo a o negazione $\neg a$
    - i.e. un'assegnazione a una variabile booleana
    - a *occorre negativamente* (risp. *positivamente*) nel letterale $\neg a$ (risp. a)
  - $l_1 \vee l_2 \vee \cdots \vee l_k$, **soddisfatta** da un *mondo possibile* (i.e. interpretazione sse (almeno) un l_i è vero in tale mondo è un modello)

Clausole come Proposizioni o Vincoli

In *Logica Proposizionale*

- clausola: formula logica in una forma ristretta (normale)
  - ogni proposizione può essere convertita in forma di clausola
    - algoritmo [conversione]

Nei *CSP*

- clausola: *vincolo* su un insieme di variabili booleane
  - soddisfatto se almeno uno dei sui letterali è vero
  - i.e. esclude assegnazioni per cui *tutti* i letterali risultino falsi

Esempio — clausola

$$happy \vee sad \vee \neg living$$

- come vincolo su $Happy$, Sad e $Living$
  - soddisfatto assegnando $true$ a $Happy$ o Sad oppure $false$ a $Living$
    - poiché $happy$ e sad vi occorrono positivamente e $living$ negativamente
- assegnazione totale che viola il vincolo: $\neg happy, \neg sad, living$
  - unica assegnazione che violi la clausola

Conversione CSP → Logica proposizionale

CSP finito → problema di soddisfacibilità di una KB proposizionale:

- variabile Y del CSP, $dom(Y) = \{v_1, \ldots, v_k\}$ → **variabili indicatrici** booleane $\{Y_1, \ldots, Y_k\}$: Y_i ***vera*** se $Y = v_i$ e ***falsa*** altrimenti
  - per rappresentare Y in KB: atomi $y_1, \ldots, y_k$
- ***clausole*** (vincoli) da inserire in KB
  - $y_1 \vee \cdots \vee y_k$
    - uno degli y_i dev'essere vero
  - $\neg y_i \vee \neg y_j$ per ogni $i, j \in \{1, \ldots, k\}$ con $i < j$
    - y_i e y_j non possono essere entrambi veri
- per ogni vincolo del CSP, una clausola per ogni ***assegnazione*** che lo violi
  - i.e. ***assegnazioni*** alle Y_i ***non ammesse*** dal vincolo
  - per semplificare, si possono combinare le clausole
    - ad es. combinando $a \vee b \vee c$ e $a \vee b \vee \neg c$ si ha: $a \vee b$

Algoritmi Basati su Consistenza per Clausole

Specifici per la *soddisfacibilità* di clausole: spesso superano quelli per CSP

- dominio binario → eliminando un valore si assegna l'altro
  ad es. togliendo $true$ da D_X resta solo $X = false$

Consistenza (degli archi) usata per restringere insiemi di valori / di vincoli:

- assegnando un valore a una variabile si *semplifica* l'insieme dei vincoli:
  1. assegnando $true$ a X, tutte le clausole con $X = true$ diventano ridondanti: possono essere eliminate perché soddisfatte (analogamente, assegnando $false$)
  2. **risoluzione** *unitaria*: assegnando $true$ a X, in ogni clausola con $X = false$ si può eliminare tale letterale (analogamente, assegnando $false$)
- se, dopo la semplificazione, resta una clausola con una sola assegnazione, $Y = v$, si può rimuovere l'altro valore da D_Y

una clausola senza atomi (assegnazioni possibili) rappresenta la *contraddizione*: vincoli insoddisfacibili

Esempio — Si consideri la clausola $\neg x \vee y \vee \neg z$

- assegnando $true$ a X, la si può semplificare in $y \vee \neg z$
  - assegnando poi $false$ a Y, la si può semplificare ancora in $\neg z$
  - infine $true$ può essere rimosso dal dominio di Z
- invece assegnando $false$ a X, l'intera clausola può essere rimossa (perché soddisfatta)

Letterale puro:
atomo che occorre solo positivamente o negativamente nella KB

- se serve trovare un solo modello (soddisfacibilità) e,
  dopo le semplificazioni, si ha un letterale puro,
  si può ***fissare*** l'assegnazione corrispondente
  - ad es. se compare solo y (i.e. $\neg y$ non compare mai)
    a Y può essere assegnato $true$
  - ciò semplifica il problema senza eliminare tutti i modelli:
    - le clausole che restano sono un sottoinsieme di quelle che rimarrebbero fissando $Y = false$

Algoritmo DPLL *Davis-Putnam-Logemann-Loveland*, 1962

- riduzione dei domini (pruning) e dei vincoli
- separazione dei domini
- assegnazione di letterali puri

efficiente con strutture dati indicizzate ad hoc

Esercizio — trovare *implementazioni* di risolutori di problemi di soddisfacibilità e *problemi* per testarle

Ricerca Locale su Struttura Proposizionale

Ricerca locale stocastica metodi semplici ed efficienti per problemi di soddisfacibilità proposizionali con vincoli clausali:

- *un solo valore* alternativo per ogni assegnazione
- clausola non soddisfatta soddisfacibile con il *cambio* di valore di una sola variabile, ma conseguenze sulle altre:
  - assegnando $true$ a una variabile
    - clausole dove occorre *negativamente* potrebbero non essere più soddisfatte
    - clausole dove occorre *positivamente* soddisfatte
  - ponendo una variabile a $false$: caso duale simmetrico
- favorisce un'*efficiente indicizzazione* delle clausole

Osservazioni — Rispetto ai CSP:

- Spazio di ricerca *esteso*
  - prima di trovare una soluzione:
    più di una Y_i vera → Y ha più valori
    oppure, Y_i tutte false → Y non ha valori ammissibili
  - assegnazioni che sono *minimi locali* nel problema originario potrebbero non esserlo più in questa rappresentazione
- Problemi di soddisfacibilità proposizionale *molto più investigati*
  - esistono *risolutori più efficienti*
  - la ricerca ha esplorato uno spazio di algoritmi più ampio

Clausole Definite Proposizionali

Clausole Definite Proposizionali

Linguaggio delle **clausole definite proposizionali**
sottolinguaggio della logica proposizionale

- non ammette ambiguità o incertezza nella rappresentazione
- stessa semantica della logica proposizionale
- forma più specifica di proposizioni ammesse:

clausole con un unico letterale positivo

$$a \vee \neg b_1 \vee \cdots \vee \neg b_m$$

SINTASSI

Di base: *proposizioni atomiche* o *atomi*

- **clausola definita** proposizionale: NB: $A \vee \neg B \equiv A \leftarrow B$

$$h \leftarrow a_1 \wedge \cdots \wedge a_m.$$

  che si legge "h se a_1 e $\cdots$ e a_m":
  - h **testa** (*head*), atomo
  - $a_1 \wedge \cdots \wedge a_m$ **corpo**, con a_i atomi
    - si dice **regola** se $m > 0$
    - si chiama **clausola atomica** o **fatto** se $m = 0$ (*corpo vuoto*) e si può omettere "$\leftarrow$"
- **base di conoscenza**: insieme di clausole definite

Esempio — Nella KB vista nella sezione precedente tutte clausole definite

Le seguenti proposizioni non sono clausole definite:

- $\neg apple\_is\_eaten.$
  - manca la testa
- $apple\_is\_eaten \wedge bird\_eats\_apple.$
  - testa non atomica
- $sam\_is\_in\_room \wedge night\_time \leftarrow up\_s_1.$
  - idem
- $Apple\_is\_eaten \leftarrow Bird\_eats\_apple.$
  - nomi degli atomi in maiuscolo non ammessi
- $happy \vee sad \vee \neg alive.$
  - testa non atomica: equivale a $happy \vee sad \leftarrow alive.$

SEMANTICA

$$h \leftarrow a_1 \wedge \cdots \wedge a_m$$

Nell'interpretazione I, essa è

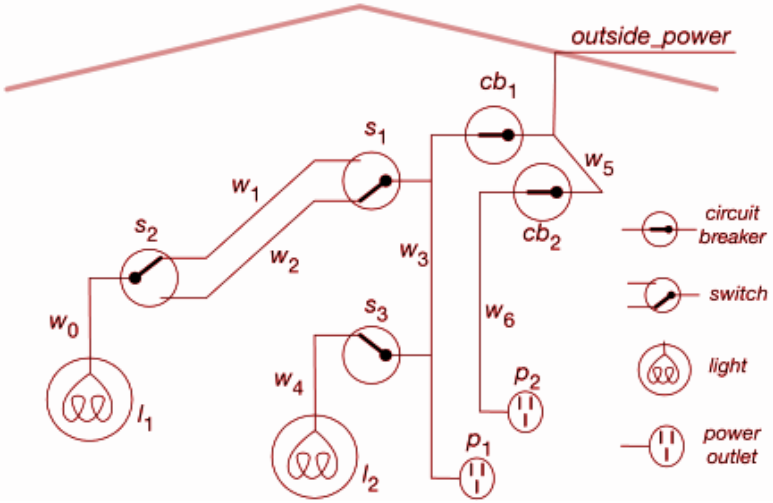
cfr. tavola di verità di $\leftarrow$

- *falsa* se $a_1, \ldots, a_m$ tutti veri ma h falsa;
- *vera* altrimenti

Clausola definita forma *ristretta* di clausola:

- clausola con *esattamente* un letterale positivo
  - ad es. $a \leftarrow b \wedge c \wedge d.$ equivalente a $a \vee \neg b \vee \neg c \vee \neg d.$
  - non può rappresentare una disgiunzione di atomi (e.g. $a \vee b$), nemmeno se tutti negati (e.g. $\neg c \vee \neg d$)

Esempio — assiomatizzazione dell'ambiente intelligente per la simulazione (diagnostica)



Schema domotica

- *Rappresentazione* dello stato di cavi (*wire*), prese (*outlet*), luci (*light*), commutatori (*switch*) e fusibili (*circuit breaker*), ...
  - dettagli trascurabili non verranno modellati (ad es. colori, peso, lunghezze, altezze, ecc.)

(..cont.)

- *livello di astrazione* per la rappresentazione?
  - per *specialisti*: voltaggi? correnti? frequenze?
  - *senso comune*: comprensibile anche a non specialisti
- *cosa* rappresentare ?
  - proposizioni su accensione luci, funzionamento cavi, stato commutatori, ecc.
- *atomi* con significati precisi nel mondo
  - nomi descrittivi
    - up_s_2: (commutatore/switch) s_2 in posizione up
    - $live\_l_1$: (punto-luce) l_1 *in tensione*

(..cont.)

- *conoscenza di fondo* (BK) sulle cose vere nel mondo, fornita al sistema attraverso altre clausole definite
  - forma semplice: fatti (clausole senza corpo)
    - $light\_l_1.$
    - $light\_l_2.$
    - $ok\_l_1.$
    - $ok\_l_2.$
    - $ok\_cb_1.$
    - $ok\_cb_2.$
    - $live\_outside.$

(..cont.)

- si può anche considerare una parte del dominio e definire altre *regole*
  - $live\_l_1 \leftarrow live\_w_0.$ "(punto-luce) l_1 *in tensione* se (cavo) w_0 in tensione"
  - $live\_w_0 \leftarrow live\_w_1 \wedge up\_s_2.$
  - $live\_w_0 \leftarrow live\_w_2 \wedge down\_s_2.$
  - $live\_w_1 \leftarrow live\_w_3 \wedge up\_s_1.$
  - $live\_w_2 \leftarrow live\_w_3 \wedge down\_s_1.$
  - $live\_l_2 \leftarrow live\_w_4.$
  - $live\_w_4 \leftarrow live\_w_3 \wedge up\_s_3.$
  - $live\_p_1 \leftarrow live\_w_3.$
  - $live\_w_3 \leftarrow live\_w_5 \wedge ok\_cb_1.$
  - $live\_p_2 \leftarrow live\_w_6.$
  - $live\_w_6 \leftarrow live\_w_5 \wedge ok\_cb_2.$
  - $live\_w_5 \leftarrow live\_outside.$
  - $lit\_l_1 \leftarrow light\_l_1 \wedge live\_l_1 \wedge ok\_l_1.$
  - $lit\_l_2 \leftarrow light\_l_2 \wedge live\_l_2 \wedge ok\_l_2.$

(..cont.)

- A run-time, si forniranno al sistema *osservazioni*
  - ad es. sullo stato dei commutatori, come:
    - $down\_s_1$.
    - up_s_2.
    - up_s_3.

Domande e Risposte

Determinare fatti veri riguardanti il mondo/dominio rappresentato:

- un utente può porre *domande*
  - sulle conseguenze logiche della KB fornita alla macchina
- la macchina può dare *risposte*
  - decidendo se proposizioni seguano logicamente oppure no
- l'utente può comprendere il *significato* di tali risposte
  - conoscendo la semantica degli atomi

Query: domanda tesa a sapere se una data proposizione segua da una KB

- ***sintassi***

  ask b.

  - b atomo o congiunzione di atomi, per KB di clausole definite
    i.e. il ***corpo*** di una regola: $\leftarrow b_1 \wedge \cdots \wedge b_m$
- ***risposte*** possibili:
  - `yes` se b è conseguenza logica, $KB \models b$
  - `no` altrimenti, $KB \not\models b$
    - non significa che b sia falso (per i modelli di KB)
      bensì che con la conoscenza disponibile è impossibile determinarne la verità

Esempio — data la KB precedente,
la *macchina* sa rispondere a query come:

- ask $light\_l_1$.
  risposta: `yes`
  - fatto asserito
- ask $light\_l_6$.
  riposta: `no`
  - non c'è info sufficiente a sapere se l_6 sia un punto luce
- ask lit_l_2.
  risposta: `yes`
  - acceso, atomo vero in tutti i modelli

L'*utente* sa interpretare le risposte sulla base dell'interpretazione intesa

Dimostrazioni

Come *calcolare* le risposte? *Proof Theory*

- $\models$ non specifica COME calcolare conseguenze logiche:
  problema della *deduzione* forma di *inferenza*

Prova (*proof*): dimostrazione, derivabile anche *automaticamente*,
del fatto che una proposizione segua logicamente da un insieme di assiomi

- **teorema**: proposizione dimostrabile

Procedura di dimostrazione (*proof procedure*):
algoritmo, eventualmente non-deterministico, per costruire prove

- data una procedura di dimostrazione,

$$KB \vdash g$$

indica che si può produrre una prova del fatto che g sia conseguenza di KB

Proprietà e Tipi di procedure

Relazioni tra prove e modelli:

- procedura **corretta** (*sound*) se ogni proposizione dimostrata dalla procedura è conseguenza logica di KB:

  se $KB \vdash g$ allora $KB \models g$
- procedura **completa** se ogni conseguenza logica di KB può essere dimostrata dalla procedura:

  se $KB \models g$ allora $KB \vdash g$

Due tipi di procedure:

- *Bottom-Up* (BU)
- *Top-Down* (TD)

Procedura Bottom-Up

Scopo — derivare tutte le conseguenze logiche di una KB

- si costruisce la prova gradualmente,
  basandosi su quanto è stato già dimostrato (atomi)
- **concatenazione in avanti** (*forward chaining*) su clausole definite:
  - derivare fatti nuovi a partire da quanto è già stato provato

Una sola *regola di derivazione*,
generalizzazione della regola d'inferenza detta **modus ponens**:

$$\frac{p \rightarrow q \qquad p}{q}$$

- se $h \leftarrow a_1 \wedge \cdots \wedge a_m$ è in KB e ogni a_i è già stato provato,
  allora si può derivare h
  - ogni fatto ($m = 0$) si considera *immediatamente* provato

```
procedure Prove_DC_BU(KB)
  Input
     KB: insieme di clausole definite
  Output
     insieme di tutte le conseguenze logiche di KB
  Local
     C insieme di atomi

  C ← ∅
  repeat
     selezionare h ← a_1 ∧ ··· ∧ a_m da KB
     tale che ∀i: a_i ∈ C e h ∉ C
     C ← C ∪ {h}
  until nessun'altra clausola selezionabile
  return C
```

La procedura trova C, insieme delle conseguenze di KB

Risposta = `yes` **ossia** $KB \vdash g$ **quando:**

- $g \in C$ — g **atomica**
- $\{g_1, \ldots, g_k\} \subseteq C$ — $g = g_1 \wedge \ldots \wedge g_k$ **congiunzione**

Esempio — KB con gli assiomi:

- $a \leftarrow b \wedge c.$
- $b \leftarrow d \wedge e.$
- $b \leftarrow g \wedge e.$
- $c \leftarrow e.$
- $d.$
- $e.$
- $f \leftarrow a \wedge g.$

Traccia dei valori via via assegnati a C dalla procedura:

- $\{\}$
- $\{d\}$
- $\{e, d\}$
- $\{c, e, d\}$
- $\{b, c, e, d\}$
- $\{a, b, c, e, d\}$

(..cont.)

L'algoritmo termina con $C = \{a, b, c, e, d\}$

- quindi $KB \vdash a$, $KB \vdash b$, ecc.

Non viene mai usata l'ultima regola di KB

Non si riesce a provare né f né g

- difatti c'è un modello della KB in cui f e g sono entrambe false

Proprietà della Procedura

Consistenza — ogni atomo in C è conseguenza logica di KB:

- se $KB \vdash g$ allora $KB \models g$

Completezza — tutte le conseguenze logiche sono derivabili:

- se $KB \models g$ allora g è vera in ogni modello di KB
- quindi anche in quello *minimo*, ovvero in C
- per cui $KB \vdash g$

Complessità $\not\propto$

L'algoritmo si ferma: itera per un numero limitato di volte determinato dal numero di clausole in KB

- ogni clausola viene usata al più una volta
- complessità *lineare* nelle dimostrazioni da KB
  - indicizzando le clausole
    in modo che il ciclo interno sia eseguito in tempo costante

Punti Fissi

C restituito è un **minimo punto fisso**:
ogni ulteriore applicazione della regola di derivazione non lo cambia

- detta I l'interpretazione in cui ogni atomo in C è vero e gli altri sono falsi
  I dev'essere un modello di KB
- I modello *minimale* di KB:
  ha il minor numero di proposizioni vere tra tutti i modelli
  - ogni altro modello può avere anche altri atomi veri oltre a quelli di C

Procedura Top-Down

Ricerca *backward* o *top-down* a partire da una query
per determinare se segua logicamente dalle clausole di KB

Risoluzione (inferenza su clausole)

$$\frac{l_1 \vee \cdots \vee a \vee \cdots \vee l_r \qquad l'_1 \vee \cdots \vee \neg a \vee \cdots \vee l'_m}{l_1 \vee \cdots \vee l_r \vee l'_1 \vee \cdots \vee l'_m}$$

Risoluzione SLD
*S*elezione di un atomo usando una strategia
*L*ineare su clausole
*D*efinite proposizionali

$$\frac{a \leftarrow a_1 \wedge \cdots \wedge \underline{a_i} \wedge \cdots \wedge a_r \qquad a_i \leftarrow b_1 \wedge \cdots \wedge b_m}{a \leftarrow a_1 \wedge \cdots \wedge \underline{b_1 \wedge \cdots \wedge b_m} \wedge \cdots \wedge a_r}$$

- versione *proposizionale* di un metodo più generale

Procedura Top-Down con Clausole di Risposta

Clausola di risposta: $yes \leftarrow a_1 \wedge a_2 \wedge \cdots \wedge a_m$

- dove yes è un atomo *speciale*, vero se la risposta alla query è "`yes`"

Data la query: ask $q_1 \wedge \cdots \wedge q_m$
→ clausola di risposta *iniziale*: $yes \leftarrow q_1 \wedge \cdots \wedge q_m$

Procedura — data una clausola di risposta

- si *seleziona* un atomo del corpo (*sotto-goal*) da dimostrare
  - ad es. a_1
- si procede quindi attraverso passi di **risoluzione**
  - per cui si *sceglie* una clausola definita in KB avente come testa l'atomo selezionato (a_1)
  - se non ne esistono, si *fallisce*

Scelta vs Selezione (cfr. specchietto nel testo)

Risolvente della clausola di risposta e clausola scelta:

$$\frac{yes \leftarrow \underline{a_1} \wedge a_2 \wedge \cdots \wedge a_m \qquad a_1 \leftarrow b_1 \wedge \cdots \wedge b_p}{yes \leftarrow b_1 \wedge \cdots \wedge b_p \wedge a_2 \wedge \cdots \wedge a_m}$$

- sotto-goal selezionato sostituito dal corpo della clausola scelta

Risposta: clausola di risposta con corpo vuoto:

$$yes \leftarrow$$

Derivazione SLD di una query ask $q_1 \wedge \cdots \wedge q_k$ da KB:
sequenza di clausole di risposta $\gamma_0, \gamma_1, \ldots, \gamma_n$ tali che

- γ_0 ***query originaria*** $yes \leftarrow q_1 \wedge \cdots \wedge q_k$
- γ_i ***risolvente*** di γ_{i-1} con una clausola definita in KB
- γ_n ***risposta***

Procedura Top-Down Alternativa

Parte da un insieme G (*goal*) di *atomi da dimostrare*:

- inizializzazione di G con tutti gli atomi della query (*sotto-goal*)
- G corrisponde a $yes \leftarrow \bigwedge_{g \in G} g$
- la clausola

$$a \leftarrow b_1 \wedge \cdots \wedge b_p$$

indica che a può essere sostituito dai sotto-goal $b_1, \ldots, b_p$

non-deterministic **procedure** Prove_DC_TD(KB, $Query$)

Input

KB: insieme di clausole definite

$Query$ insieme di atomi da dimostrare

Output

yes se $KB \models Query$, altrimenti fallisce (no)

Local

G insieme di atomi

$G \leftarrow Query$

repeat

selezionare un atomo $a \in G$

scegliere da KB una clausola con a come testa: $a \leftarrow B$

// se possibile, altrimenti il tentativo fallisce

$G \leftarrow (G \setminus \{a\}) \cup B$

until $G = \emptyset$

return yes

Osservazioni

- qualsiasi atomo del corpo può essere *selezionato* (tutti, prima o poi)
  - se una selezione non porta a terminare la prova,
    non serve tentare di selezionarne un altro
- procedura *non-deterministica*: *scelta* della clausola per la risoluzione che porti al successo del tentativo
  - se ci sono scelte che portano a G vuoto, l'algoritmo ha successo (rispondendo `yes`) *altrimenti* il tentativo fallisce (e può rispondere `no`)
- corpo della clausola: *insieme* di atomi, come G
  - in alternativa, G *lista* ordinata di atomi che possono comparire più volte

Esempio — Data la KB seguente:

- $a \leftarrow b \wedge c.$
- $b \leftarrow d \wedge e.$
- $b \leftarrow g \wedge e.$
- $c \leftarrow e.$
- $d.$
- $e.$
- $f \leftarrow a \wedge g.$

e la query: ask $a.$

- qui G come clausola di risposta
  - con selezione dell'atomo *più a sinistra* del corpo

(..cont.)

risolventi γ_i	clausole da KB
$yes \leftarrow \underline{a}$	$a \leftarrow b \wedge c$
$yes \leftarrow \underline{b} \wedge c$	$b \leftarrow d \wedge e$
$yes \leftarrow \underline{d} \wedge e \wedge c$	d
$yes \leftarrow \underline{e} \wedge c$	e
$yes \leftarrow \underline{c}$	$c \leftarrow e$
$yes \leftarrow \underline{e}$	e
$yes \leftarrow$	

(..cont.)

Sequenza *alternativa* di scelte
in cui si sceglie la seconda clausola per b:

- $yes \leftarrow a$
- $yes \leftarrow b \wedge c$
- $yes \leftarrow g \wedge e \wedge c$
  - selezionando g, non ci sono regole da scegliere
  - *questo tentativo* fallisce: scegliere altre alternative

Algoritmo TD su Grafo di Ricerca

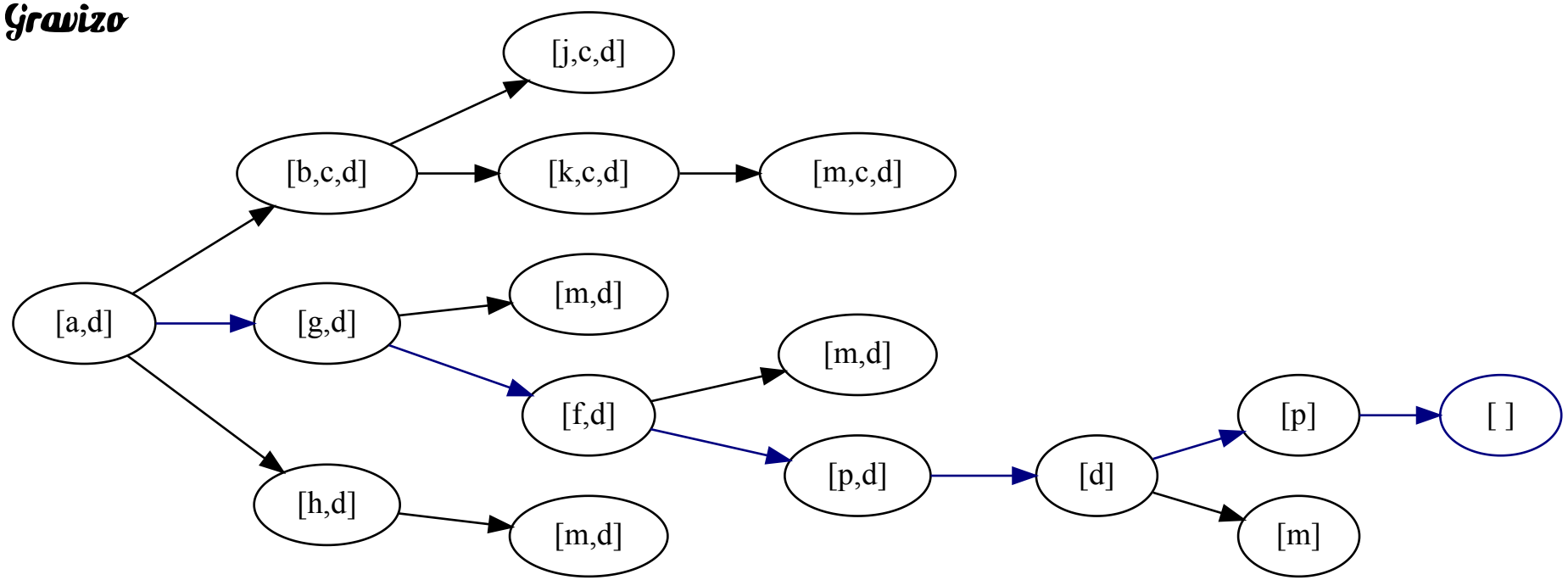
indotto dalla strategia di selezione

- *nodi*: clausole di risposta
- *vicini* di $yes \leftarrow \underline{a_1} \wedge \cdots \wedge a_m$ rappresentano tutte le possibili clausole di risposta ottenute risolvendo su a_1
  - generati *dinamicamente*
  - uno per ogni clausola definita con testa a_1
- *nodi obiettivo*: $yes \leftarrow$
- si può usare qualsiasi metodo di ricerca su grafo
  - basta *un* percorso di successo perché la query sia conseguenza logica della KB
  - per ogni nodo, spazio determinato dalla query e dall'atomo selezionato

Esempio — Data la seguente KB e la query ask $a \wedge d$.

$$KB = \left\{ \begin{array}{lll} a \leftarrow b \wedge c. & a \leftarrow g. & a \leftarrow h. \\ b \leftarrow j. & b \leftarrow k. & d \leftarrow m. \\ d \leftarrow p. & f \leftarrow m. & f \leftarrow p. \\ g \leftarrow m. & g \leftarrow f. & k \leftarrow m. \\ h \leftarrow m. & p. & \end{array} \right\}$$

→ grafo (nei nodi, lista di atomi del *corpo* del risolvente):



Confronto

- Dimostrazioni TD e BU interscambiabili
  - utile a provare consistenza e completezza della TD
- BU prova ogni atomo *una sola* volta;
  TD potrebbe provare lo stesso atomo *più volte*,
  ma si concentra su quelli *rilevanti* per la query
- Con TD possibili *cicli infiniti*

Esempio — Data

$$KB = \{g \leftarrow a. \quad a \leftarrow b. \quad b \leftarrow a. \quad g \leftarrow c. \quad c.\}$$

e la query ask g.

- uniche conseguenze logiche (atomiche): g e c
- BU termina con il punto fisso $\{c, g\}$
- TD con DFS semplice potrebbe continuare *indefinitamente*
  - a meno che non sia prevista la *potatura* dei cicli

Strategia di Selezione

La *strategia di selezione* dell'atomo per la risoluzione condiziona *efficienza* e *terminazione*:

- negli es. precedenti: atomo *più a sinistra*
  - problematica senza controllo sui cicli
- strategia migliore: atomo che porti al *fallimento* più facilmente
- strategia comune: *ordinamento* degli atomi per la selezione
  - consente l'uso di un'*euristica*

Questioni di Rappresentazione della Conoscenza

Osservazioni e Conoscenza di Fondo

Osservazioni — ricevute (a run-time) da utenti, sensori o altre sorgenti:

- rappresentate come *insiemi di atomi* (non direttamente come regole)
  - es. diagnosi medica: il paziente comunica i sintomi
- interagiscono con la **conoscenza di fondo** (*background knowledge*, BK)
  - *aggiunte* alla BK / *separate* dalla BK

Incompletezza delle informazioni fornite dall'**utente**:
1. non sa usare il *vocabolario*; *2.* non sa giudicare la *rilevanza*

- servirebbero:
  1. un'**ontologia** che specifichi il significato dei simboli
  2. un'*interfaccia* può aiutare a fornire informazioni rilevanti

Analogamente:

- *sensori passivi* ← osservazioni dirette (congiunzioni di atomi)
- *sensori attivi* ← risposte a richieste di info necessarie

Interrogare l'Utente

Acquisizione: meccanismo **ask-the-user** nella procedura di ragionamento:

- atomo **askable**: verità acquisibile dall'utente a run-time
- la TD, selezionato un atomo da dimostrare
  - usa una clausola della KB
  - oppure, se askable, chiede all'utente se sia vero o falso
    NB solo per atomi *rilevanti*
- *classi* di atomi:
  1. non askable
  2. askable senza risposta-utente
     - si può chiedere e memorizzare
  3. askable con risposta-utente memorizzata
     - usati senza chiedere di nuovo
- possibile anche con procedure BU
  - ma vanno evitate *troppe richieste* all'utente

Simmetria dei ruoli utente-sistema:
entrambi possono fare domande e dare risposte

- inizio: l'utente fa una domanda al sistema (query)
- a ogni passo: il sistema può porre domande la cui risposta viene acquisita
  - ritrovando le clausole rilevanti
    oppure
  - chiedendo all'utente

Interazione caratterizzata da un *protocollo*
di domande e risposte tra utente e sistema

Esempio — Smart home:
non tutto può essere fornito dal progettista (KB + BK) → atomi *askable*

Possibile *dialogo utente-sistema* (interfaccia minimale):

- `ailog:` ask lit_l_1.
- `Is` up_s_1 `true? no.`
- `Is` $down\_s_1$ `true? yes.`
- `Is` $down\_s_2$ `true? yes.`
- `Answer:` lit_l_1.

solo domande rilevanti cui l'utente può rispondere

A volte sufficiente/preferibile che l'utente segnali *casi inusuali*:

- es. un paziente segnala un problema di salute
  - (dove) avverte dolore
- es. un sensore indica una scena cambiata
  - senza altri dettagli

Eventi *eccezionali*
→ possibili inferenze anche in mancanza di conoscenza:

- situazione di normalità, di *default*,
  superata a seguito di tali segnalazioni
  - esistono forme specifiche di ragionamento (cfr. abduzione)

Spiegazioni a Livello di Conoscenza

Uso esplicito della semantica

→ *spiegazione*/*debugging* a livello di conoscenza

- rende il sistema *usabile* da parte di utenti comuni
  - esso deve saper giustificare le risposte
    - ad es. una diagnosi medica
- utile a:
  - *spiegare* come è stato trovato un risultato
  - *correggere* la KB

Domande al Sistema

Interrogazioni dell'utente per avere spiegazioni:

1. **Come?** (*how question*):
   si chiede *come* sia stata provata una risposta
   - il sistema fornisce le *clausole* utilizzate per dedurre la risposta
   - si può poi chiedere anche per ogni atomo nel corpo d'una clausola
2. **Perché?** (*why question*):
   si chiede il *motivo* di una domanda all'utente
   - il sistema mostra la *regola* che ha prodotto la domanda
   - l'utente può chiedere perché sia stata dimostrata la testa
   - utile a *navigare* la dimostrazione (anche parzialmente)
3. **Perché no?** (*whynot question*):
   si chiede perché *non* sia stato possibile dimostrare un atomo

Domande al Sistema — Come?

Procedura di spiegazione alla domanda how

- se g ammette una dimostrazione:
  - g fatto
    oppure
  - esiste $g \leftarrow a_1 \wedge \cdots \wedge a_k$ tale che ogni a_i sia stato dimostrato
- provato g, se l'utente chiede
  how.
  il sistema è in grado di fornire la clausola usata per provare g
- se questa è una regola, l'utente potrebbe chiedere
  how i.
  per avere la regola usata per dimostrare a_i
- continuando con altri how si può esplorare la dimostrazione di g

Esempio — KB precedente, query ask lit_l_2

- utente: how — in risposta alla dimostrazione
- sistema: $lit\_l_2 \leftarrow light\_l_2 \wedge live\_l_2 \wedge ok\_l_2$ — regola finale della dim.
- utente: how 2 — com'è stato provato $live\_l_2$?
- sistema (regola usata): $live\_l_2 \leftarrow live\_w_4$
- utente: how 1 — com'è stato provato $live\_w_4$?
- sistema: $live\_w_4 \leftarrow live\_w_3 \wedge up\_s_3$
- utente: how 1 — com'è stato provato $live\_w_3$?
- sistema: $live\_w_3 \leftarrow live\_w_5 \wedge ok\_cb_1$ — regola usata per provarlo
- utente: how 2 — per il secondo atomo
- sistema: ok_cb_1 — fatto già presente nella KB

spiegazione a *livello di conoscenza* e con sole clausole rilevanti:
all'utente basta conoscere la semantica dei simboli (non la procedura)

Domande al Sistema — Perché?

Spiegazione del perché una domanda sia stata posta

- aumenta la *credibilità* del sistema
  - appare più intelligente / trasparente / affidabile
- *misura di complessità* di sistemi interattivi:
  numero di domande richieste (da minimizzare)
  - conoscere le motivazioni aiuta a ridurre la complessità
- *domanda irrilevante*: sintomo di problema più grave
- si *impara* qualcosa dal sistema capendo i motivi delle sue azioni
  - come un apprendista

Uso di why

Il sistema pone all'utente una domanda q,
ci dev'essere una regola che contiene q nel corpo:

- utente: why.
  - "*perché mi stai ponendo questa domanda?*"
- sistema: fornisce la regola per cui serve sapere di q
- se l'utente chiede ancora why,
  il sistema spiega perché si sia domandato dell'atomo nella testa della regola...

why ripetute danno un *percorso* di sotto-goal fino alla query originaria:

- se tutte le regole sono ragionevoli, si ha una *giustificazione* dell'appropriatezza della domanda iniziale all'utente

Esempio — Dialogo con l'uso di why

- `ailog:` ask lit_l_1.
- `Is` up_s_1 `true?` why.
- up_s_1 `is used in the rule`
  $live\_w_1 \leftarrow live\_w_3 \wedge up\_s_1$: why.
- $live\_w_1$ `is used in the rule`
  $live\_w_0 \leftarrow live\_w_1 \wedge up\_s_2$: why.
- $live\_w_0$ `is used in the rule`
  $live\_l_1 \leftarrow live\_w_0$: why.
- $live\_l_1$ `is used in the rule`
  $lit\_l_1 \leftarrow light\_l_1 \wedge live\_l_1 \wedge ok\_l_1$: why.
- `Because that is what you asked me!`

Navigazione della Dimostrazione

In genere how e why usate insieme:

- how sposta da sotto-goal a livello più alto a uno più basso
- why viceversa

Permettono all'utente di navigare l'**albero di dimostrazione** (*proof tree*):

- nodi ← atomi
- nodo + figli ← regola

Debugging a Livello di Conoscenza

Le KB possono presentare *errori* e *omissioni*

- occorre saperla correggere e/o completare
  - strumenti standard per il debugging del SW non appropriati
  - non tutti comprendono il funzionamento interno del sistema
    - utenti/esperti di dominio non conoscono le *procedure* di ragionamento

Knowledge-level Debugging: richiede di conoscere *significato* dei simboli e *verità* degli atomi nell'interpretazione intesa

- *obiettivo*: coinvolgere gli esperti di dominio
  - ad es. medicina, domotica
  - senza aspettarsi conoscenze sull'AI

Knowledge-level Debugging

Errori (non sintattici) nei sistemi a regole:

1. *risposta non corretta* derivata
   - derivato qualche atomo falso nell'interpretazione intesa
2. *risposta omessa* — non derivata
   - dimostrazione fallita per un atomo vero nell'interpretazione intesa
     - avrebbe dovuto avere successo
3. *ciclo infinito*
4. *domande irrilevanti* poste dal sistema

Risposte Errate

Risposte della dimostrazione *false* nell'interpretazione intesa:

- casi di **falsi positivi**
- ma procedura *sound* → *clausola errata* impiegata nella dimostrazione

Si assume che chi opera il debugging conosca il *significato* dei simboli e il valore di *verità* di un atomo nell'interpretazione intesa

Per il *debugging dei falsi positivi*:

- Sia g l'atomo dimostrato, ma falso nell'interpretazione intesa e $g \leftarrow a_1 \wedge \cdots \wedge a_k$ la clausola usata per provarlo
- Casi possibili:
  - uno o più a_i falsi nell'interpretazione intesa → stesso problema
    - da risolvere ricorsivamente
  - tutti gli a_i veri nell'interpretazione intesa
    - clausola $g \leftarrow a_1 \wedge \cdots \wedge a_k$ *errata*

```
procedure Debug_false(g, KB)
  Input
     KB base di conoscenza
     g atomo: KB ⊢ g ma falso nell'interpretazione intesa
  Output
     clausola in KB falsa

  Trovare g ← a_1 ∧ ··· ∧ a_k ∈ KB usata per provare g
  for each a_i do
     Chiedere all'utente se a_i sia vero
     if risponde che a_i è falso then
        return Debug_false(a_i, KB)
  return g ← a_1 ∧ ··· ∧ a_k
```

Strategia da attuare usando il comando how:

- data una prova di g, si può chiedere come sia stato dimostrato
  - sarà restituita la clausola usata
- se è una regola, si può usare how per cercare eventuali atomi nel corpo falsi nell'interpretazione intesa
  - nel caso, si restituiranno le regole usate per dimostrarli
- si ripete il procedimento fino a trovare una clausola con tutti gli atomi del corpo veri (o corpo vuoto) che è quella *errata*

Esempio — Bug nella KB: è specificato erroneamente che la connessione di w_1 a w_3 dipende da s_3 anziché da s_1, con la regola $live\_w_1 \leftarrow live\_w_3 \wedge up\_s_3$.

- Per trovare tale regola, da KB si può derivare:
  - lit_l_1
    falso nell'interpretazione intesa, si può quindi chiedere how.
  - con riposta: $lit\_l_1 \leftarrow light\_l_1 \wedge live\_l_1 \wedge ok\_l_1$.
    $live\_l_1$ falso, perciò si chiede how 2.
  - risposta: $live\_l_1 \leftarrow live\_w_0$.
    ma $live\_w_0$ è falso per cui si chiede how 1.
  - risposta: $live\_w_0 \leftarrow live\_w_1 \wedge up\_s_2$.
    ma $live\_w_1$ è falso, per cui si chiede how 1.
  - risposta: $live\_w_1 \leftarrow live\_w_3 \wedge up\_s_3$.
    atomi del corpo veri nell'interpretazione intesa → *regola difettosa*

Risposte Mancanti

Una risposta attesa non viene derivata dal sistema

- *fallimento*:
  - atomo g *vero* nell'interpretazione intesa ma $KB \nvdash g$
  - caso di **falso negativo**

Segnala un caso di *clausole mancanti* tra gli assiomi della KB

- individuabile attraverso una procedura
  - per trovare atomi cui manchino clausole utili alla loro dimostrazione:

procedure Debug_missing(g,KB)

Input

KB base di conoscenza

g atomo: $KB \nvdash g$ ma g vero nell'interpretazione intesa

Output

atomo per il quale c'è una clausola mancante

if esiste $g \leftarrow a_1 \wedge \cdots \wedge a_k \in KB$ con tutti gli a_i veri nell'interpretazione intesa **then**

Selezionare a_i che non può essere dimostrato

return Debug_missing(a_i,KB)

else

return g

Debugging nel caso di risposte mancate

Se la dimostrazione di g fallisce,
questo dipenderà dai *corpi* di *tutte* le clausole con g come *testa*:

- almeno per una di esse, tutti gli atomi del corpo dovevano risultare veri nell'interpretazione intesa
  - uno di tali atomi non è dimostrabile come vero:
    caso analogo da investigare *ricorsivamente*
- altrimenti: non ci sono clausole utili a provare g
  → occorre *aggiungerla* alla KB

Domanda whynot per chiedere perché g non venga dimostrato

- per implementare `Debug_missing`
  il sistema deve saper porre domande rilevanti

Esempio — Si supponga che $down\_s_2$ sia vero ma manchi la clausola per provarlo: in particolare, data la KB, non si riesce a dimostrare lit_l_1

- si trovano tutte le regole con lit_l_1 in testa, ossia la sola $lit\_l_1 \leftarrow light\_l_1 \wedge live\_l_1 \wedge ok\_l_1.$
- verificando che tutti gli atomi del corpo siano veri
  - $light\_l_1$ e ok_l_1 dimostrabili, ma non $live\_l_1$ (da indagare)
- anche $live\_l_1$ occorre come testa solo in una regola: $live\_l_1 \leftarrow live\_w_0.$
  - ma $live\_w_0$ non può essere dimostrato, mentre dovrebbe risultare vero nell'interpretazione intesa
- 2 regole per $live\_w_0$: $live\_w_0 \leftarrow live\_w_1 \wedge up\_s_2.$ e $live\_w_0 \leftarrow live\_w_2 \wedge down\_s_2.$
  - l'utente sa determinare che il corpo della seconda regola è vero
  - ma, mentre $live\_w_2$ è dimostrabile, mancano clausole per $down\_s_2$ che viene quindi restituito come ***difettoso***
- ***correzione***: aggiunta di una clausola (fatto o regola) per provare l'atomo

Cicli Infiniti

KB **ciclica**: contiene un atomo a per il quale esiste in KB una sequenza

$$\begin{array}{lllll} a & \leftarrow & \ldots & a_1 & \ldots \\ a_1 & \leftarrow & \ldots & a_2 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_n & \leftarrow & \ldots & a & \ldots \end{array}$$

- ad es. una KB precedente mostra un caso di loop (infinito) per TD
  $KB = \{g \leftarrow a. \quad a \leftarrow b. \quad b \leftarrow a. \quad g \leftarrow c. \quad c.\}$
- per $n = 0$, solo la clausola $a \leftarrow \ldots a \ldots$
- spesso indice di ***bug*** nella definizione della KB

KB **aciclica**: atomi della KB numerabili in modo che, per ogni clausola, i numeri degli atomi del corpo siano ***minori*** di quello dell'atomo in testa

BU non affetta dal problema:

- regola selezionabile solo se la testa non è già stata derivata

Controllo di Ciclicità in procedure TD

Si gestisce, una lista di **antenati** nella dimostrazione per ogni atomo:

- inizio $\forall i\colon\ ancestors(a_i) \leftarrow \emptyset$
- quando si usa $a \leftarrow a_1 \wedge \ldots \wedge a_k$ per dimostrare a:
  $ancestors(a_i) = ancestors(a) \cup \{a\}$
- una dimostrazione di un atomo fallisce
  quando esso risulta nell'insieme dei propri antenati → *KB ciclica*
  - versione specializzata della *potatura dei cicli*

Dimostrazione per Contraddizione

Contraddizioni

Cosa si può concludere quando dalla KB deriva una proposizione contraria a quanto si osserva nella realtà?

- ad es. si osserva che un punto luce è spento
  mentre dovrebbe essere acceso secondo la KB

Si può ragionare ammettendo regole che producano *contraddizioni*

- specifica di casi *impossibili*
- utilità: *diagnostica*
  - ad es. nel caso precedente può servire a dedurre che alcuni componenti *non* siano *funzionanti*

Clausole di Horn

Linguaggio delle clausole definite esteso per esplicitare contraddizioni:

- $false$: atomo speciale che codifica la *contraddizione*
  - falso in tutte le interpretazioni
- **vincolo d'integrità**: clausola con $false$ come testa

$$false \leftarrow a_1 \wedge \cdots \wedge a_k$$

Clausola di Horn: clausola definita o vincolo d'integrità

- testa = atomo normale oppure $false$

Conclusioni Negative

Da una KB di clausole di Horn, si possono derivare **negazioni** di atomi *(pur non essendo ammesse come input)* tramite *modus tollens*:

$$\frac{p \to q \qquad \neg q}{\neg p}$$

Esempio — Base di conoscenza KB_1:

1. $false \leftarrow a \wedge b.$
2. $a \leftarrow c.$
3. $b \leftarrow c.$

c è falso in tutti i modelli di KB_1

- Dim. (per contraddizione)

Pertanto: $KB_1 \models \neg c$

Conclusioni Disgiuntive

$$false \leftarrow a_1 \wedge \cdots \wedge a_k \equiv \neg a_1 \vee \cdots \vee \neg a_k \equiv \neg(a_1 \wedge \cdots \wedge a_k)$$

si può quindi dimostrare che *congiunzioni di atomi* siano *false* in tutti i modelli di KB ovvero provare una **disgiunzione** di *negazioni di atomi*

Esempio — Sia KB_2:

1. $false \leftarrow a \wedge b.$
2. $a \leftarrow c.$
3. $b \leftarrow d.$
4. $b \leftarrow e.$

Una fra c e d falsa in ogni modello di KB_2: $KB_2 \models \neg c \vee \neg d$

- Dim. (per contraddizione)

Analogamente, una fra c ed e è falsa rispetto a KB_2: $KB_2 \models \neg c \vee \neg e$

Soddisfacibilità e Consistenza

Un insieme di clausole KB è **insoddisfacibile** se non ha modelli:

$$KB \models false$$

Un insieme di clausole KB è **inconsistente** *rispetto a una procedura di dimostrazione* $\vdash$ se consente di derivare $false$:

$$KB \vdash false$$

- Se $\vdash$ *corretta e completa* allora KB inconsistente sse KB insoddisfacibile

---

Una KB di clausole definite è *sempre* soddisfacibile

- e.g. interpretazione per cui tutti gli atomi siano veri

ciò non vale per KB di clausole di Horn

Esempio — La semplice KB

$$\left\{ \begin{array}{l} a. \\ false \leftarrow a. \end{array} \right\}$$

non è soddisfacibile:

- nessuna interpretazione soddisfa entrambe le clausole
- non possono essere entrambe vere in un'interpretazione

Per provare l'inconsistenza si può usare TD o BU:

- ask $false.$

Assumibili e Conflitti

Contraddizioni: utili a individuare combinazioni di *assunzioni incompatibili*

- supposizioni che possono dimostrarsi false
  - ad es. in *diagnostica*: funzionamento normale delle componenti inconsistente rispetto a osservazioni fatte
    - in caso di malfunzionamento → diagnosi del guasto

Assumibile: atomo che può essere assunto (come vero)
in una *dimostrazione per contraddizione*

- diagnostica: determinare *disgiunzioni di negazioni* di assumibili

Conflitto di KB: $C = \{c_1, \ldots, c_r\}$ di assumibili tale che $KB \cup C \models false$

- i.e. **risposta** a dim. per contraddizione: $KB \models \neg c_1 \vee \cdots \vee \neg c_r$

Conflitto **minimale** se nessun suo sottoinsieme è un conflitto

Esempio — Data KB_2 e l'insieme di atomi assumibili $\{c, d, e, f, g, h\}$:

$$KB_2 = \left\{ \begin{array}{c} false \leftarrow a \wedge b. \\ a \leftarrow c. \\ b \leftarrow d. \\ b \leftarrow e. \end{array} \right\}$$

- $\{c, d\}$ e $\{c, e\}$ conflitti minimali
- $\{c, d, e, h\}$ conflitto non minimale

Diagnosi Basata sulla Consistenza

Scopo: determinare possibili guasti

- in base a un *modello del sistema* e a *osservazioni* su di esso

Diagnosi:

- fa assunzioni sul funzionamento *normale*
  - *assumibile*: assenza di guasti
- deriva i componenti *anormali*: *conflitti* (inconsistenza)
  - *anomalie* nel sistema

Esempio — Si riconsideri la KB precedente (smart home)

si aggiungono:

- assunzioni di normalità i.e. *assumibili*
  - atomi $ok*$: ogni componente dev'essere in buono stato per funzionare
- l'utente può *osservare* (e specificare):
  - posizioni effettive dei commutatori (up/$down$)
  - stato delle luci (lit/$dark$)
- *vincoli d'integrità*: definiscono stati reciprocamente incompatibili
  - es. possibile un solo stato d'accensione per le luci: $false \leftarrow dark\_l_1 \wedge lit\_l_1.$

(..cont.)

KB

- $light\_l_1.$
- $light\_l_2.$
- $live\_outside.$
- $live\_l_1 \leftarrow live\_w_0.$
- $live\_w_0 \leftarrow live\_w_1 \wedge up\_s_2 \wedge ok\_s_2.$
- $live\_w_0 \leftarrow live\_w_2 \wedge down\_s_2 \wedge ok\_s_2.$
- $live\_w_1 \leftarrow live\_w_3 \wedge up\_s_1 \wedge ok\_s_1.$
- $live\_w_2 \leftarrow live\_w_3 \wedge down\_s_1 \wedge ok\_s_1.$
- $live\_l_2 \leftarrow live\_w_4.$
- $live\_w_4 \leftarrow live\_w_3 \wedge up\_s_3 \wedge ok\_s_3.$
- $live\_p_1 \leftarrow live\_w_3.$
- $live\_w_3 \leftarrow live\_w_5 \wedge ok\_cb_1.$
- $live\_p_2 \leftarrow live\_w_6.$
- $live\_w_6 \leftarrow live\_w_5 \wedge ok\_cb_2.$
- $live\_w_5 \leftarrow live\_outside.$
- $lit\_l_1 \leftarrow light\_l_1 \wedge live\_l_1 \wedge ok\_l_1.$
- $lit\_l_2 \leftarrow light\_l_2 \wedge live\_l_2 \wedge ok\_l_2.$

vincoli d'integrità:

- $false \leftarrow dark\_l_1 \wedge lit\_l_1.$
- $false \leftarrow dark\_l_2 \wedge lit\_l_2.$

osservazioni:

- $up\_s_1.$
- $up\_s_2.$
- $up\_s_3.$
- $dark\_l_1.$
- $dark\_l_2.$

dichiarazioni:

- assumable $ok\_cb_1, ok\_cb_2, ok\_s_1, ok\_s_2, ok\_s_3, ok\_l_1, ok\_l_2.$

(..cont.)

- Tutto considerato, ci sono due conflitti minimali:
  - $\{ok\_cb_1, ok\_s_1, ok\_s_2, ok\_l_1\}$
  - $\{ok\_cb_1, ok\_s_3, ok\_l_2\}$
- Pertanto:
  - $KB \models \neg ok\_cb_1 \vee \neg ok\_s_1 \vee \neg ok\_s_2 \vee \neg ok\_l_1$
  - $KB \models \neg ok\_cb_1 \vee \neg ok\_s_3 \vee \neg ok\_l_2$
- Quindi almeno una fra cb_1, s_1, s_2, l_1 non dev'essere ok,
  e almeno una tra cb_1, s_3, l_2 non è ok

CBD

Dato l'insieme dei conflitti, l'utente può diagnosticare il problema

- utile a farsi un'idea della numerosità dei possibili guasti

Diagnosi basata su consistenza (CBD): dato un insieme di conflitti,
è un insieme di assumibili con almeno un elemento in ogni conflitto

- *diagnosi* **minimale** se nessun suo sotto-insieme è una diagnosi
  - intuitivamente, una sola delle diagnosi minimali è quella giusta:
    solo conflitti falsi nell'interpretazione intesa

Esempio — Nell'esempio precedente, due conflitti:
$KB \models \neg ok\_cb_1 \vee \neg ok\_s_1 \vee \neg ok\_s_2 \vee \neg ok\_l_1$ e
$KB \models \neg ok\_cb_1 \vee \neg ok\_s_3 \vee \neg ok\_l_2$

Quindi, da KB segue:
$(\neg ok\_cb_1 \vee \neg ok\_s_1 \vee \neg ok\_s_2 \vee \neg ok\_l_1) \wedge (\neg ok\_cb_1 \vee \neg ok\_s_3 \vee \neg ok\_l_2)$

- proposizione in **forma normale congiuntiva** (*CNF*)
- distribuibile in **forma normale disgiuntiva** (*DNF*): disgiunzione di congiunzioni (qui di atomi negati): $\neg ok\_cb_1 \vee (\neg ok\_s_1 \wedge \neg ok\_s_3) \vee (\neg ok\_s_1 \wedge \neg ok\_l_2)$
  $\vee (\neg ok\_s_2 \wedge \neg ok\_s_3) \vee (\neg ok\_s_2 \wedge \neg ok\_l_2)$
  $\vee (\neg ok\_l_1 \wedge \neg ok\_s_3) \vee (\neg ok\_l_1 \wedge \neg ok\_l_2)$
  - i.e. cb_1 guasto o 6 possibili malfunzionamenti di coppie di componenti
- proposizioni corrispondenti a 7 diagnosi minimali:
  $\{ok\_cb_1\}, \{ok\_s_1, ok\_s_3\}, \{ok\_s_1, ok\_l_2\}, \{ok\_s_2, ok\_s_3\}, \{ok\_s_2, ok\_l_2\}$,
  $\{ok\_l_1, ok\_s_3\}, \{ok\_l_1, ok\_l_2\}$
  una dev'essere causa del malfunzionamento

Obiettivo: ricerca dei *conflitti* in KB di clausole di Horn

Implementazione Bottom-Up

Estensione dell'algoritmo bottom-up per clausole definite:

- le conclusioni sono *coppie* $\langle a, A \rangle$
  - a atomo
  - A insieme di assumibili che implicano a rispetto a KB
- inizializzazione, set di conclusioni: $C \leftarrow \{\langle a, \{a\}\rangle \mid a \text{ assumibile}\}$
- si usano le clausole per derivare nuove conclusioni:
  - se $\exists h \leftarrow b_1 \wedge \cdots \wedge b_m \in KB$ tale che ogni $\langle b_i, A_i \rangle \in C$, allora si può aggiungere $\langle h, A_1 \cup \ldots \cup A_m \rangle$ a C
    - per i fatti ($m = 0$) si aggiunge $\langle h, \{\}\rangle$

procedure Prove_conflict_BU(KB, As)
  Input
    KB: insieme di clausole di Horn
    As: insieme di atomi che si possono assumere veri
  Output
    insieme di conflitti
  Local
    C: insieme di coppie atomo/ins. di assumibili

  $C \leftarrow \{\langle a, \{a\}\rangle \mid a \in As\}$
  repeat
    selezionare la clausola $h \leftarrow b_1 \wedge \cdots \wedge b_m$ tale che:
      per ogni $\langle b_i, A_i\rangle \in C$ per ogni i e
      $\langle h, A\rangle \notin C$, dove $A = A_1 \cup \cdots \cup A_m$
    $C \leftarrow C \cup \{\langle h, A\rangle\}$
  until nessun'altra selezione possibile
  return $\{A \mid \langle false, A\rangle \in C\}$

Implementazione Top-Down

Anche l'implementazione top-down deriva dall'omologa per clausole definite, con 2 *differenze*:

- query principale da provare: $false$
- nella dimostrazione: atomi assumibili non vanno dimostrati,
  ma solo raccolti per essere assunti come veri

```
non-deterministic procedure Prove_conflict_TD(KB, As)
  Input
     KB: insieme di clausole di Horn
     As: insieme di atomi che si possono assumere veri
  Output
     un conflitto
  Local
     G insieme di atomi (che implicano false)

  G ← {false}
  repeat
     selezionare un atomo a ∈ G tale che a ∉ As
     scegliere una clausola a ← B di KB con a come testa
     G ← (G \ {a}) ∪ B
  until G ⊆ As
  return G
```

Osservazioni

- diverse scelte ND → diversi conflitti trovati
- in mancanza di scelte, l'algoritmo fallisce

Assunzione di Conoscenza Completa

Spesso la conoscenza su un dominio viene considerata *completa*:

- semantica tipica dei DB

Esempio — smart home

- Il sistema potrebbe richiedere di specificare *esplicitamente* i commutatori che sono up e i fusibili $broken$ e assumere per quelli non specificati che siano in condizioni *normali* / di *default*, ad es.:
  - $down\_s_i$ per gli commutatori
  - ok_cb_j per i fusibili

OWA vs. CWA

In logica normalmente non si assume che una KB definisca tutta la conoscenza su un dato dominio — **open-world assumption** (OWA)

- la mancanza di conoscenza *non consente* di trarre conclusioni: fallimento delle dimostrazioni
  - $\neg a$ *non può* essere conseguenza logica di una KB di clausole definite

Assunzione di completezza della conoscenza:

- le clausole con uno stesso atomo come testa coprono *tutti e soli* i casi in cui esso è vero — **closed-world assumption** (CWA)
  - tutto quanto sia rilevante è asserito *esplicitamente* nella KB
  - se non si riesce a provare che un atomo a è vero, si può concludere che sia falso: vera $\neg a$

Completare la Base di Conoscenza

Dato l'atomo a e tutte le clausole della KB che lo definiscono:

$$a \leftarrow b_1 .$$
$$\vdots$$
$$a \leftarrow b_n .$$

- con b_i (body): congiunzione di atomi oppure $true$ (per clausole atomiche)

esse si possono essere riassunte da un'<u>unica</u> proposizione ~~clausola~~

$$a \leftarrow b_1 \vee \cdots \vee b_n$$

Se a vero allora, per la CWA, dovrà ***necessariamente*** essere vero uno dei b_i (rispetto a qualunque interpretazione) quindi si può aggiungere:

$$a \rightarrow b_1 \vee \cdots \vee b_n .$$

Completamento di Clark

Significato delle clausole per l'atomo a sotto CWA:

- congiunzione delle due proposizioni, i.e. *equivalenza*:

$$a \Leftrightarrow b_1 \vee \cdots \vee b_n$$

  detta **completamento di Clark** delle clausole per a
  - se non ci sono regole per a, completamento: $a \Leftrightarrow false$!

Completamento della KB

ricomprende i completamenti di tutti gli atomi della KB

Esempio — Si consideri la KB precedente con le clausole:

- $down\_s_1.$
- $up\_s_2.$
- $ok\_cb_1.$
- $live\_l_1 \leftarrow live\_w_0.$
- $live\_w_0 \leftarrow live\_w_1 \wedge up\_s_2.$
- $live\_w_0 \leftarrow live\_w_2 \wedge down\_s_2.$
- $live\_w_1 \leftarrow live\_w_3 \wedge up\_s_1.$
- $live\_w_2 \leftarrow live\_w_3 \wedge down\_s_1.$
- $live\_w_3 \leftarrow live\_outside \wedge ok\_cb_1.$
- $live\_outside.$

Si noti che non sono definite clausole per up_s_1 e $down\_s_2$

(..cont.)

***Completamento*:**

- $down\_s_1 \Leftrightarrow true.$
- $up\_s_1 \Leftrightarrow false.$
- $up\_s_2 \Leftrightarrow true.$
- $down\_s_2 \Leftrightarrow false.$
- $ok\_cb_1 \Leftrightarrow true.$
- $live\_l_1 \Leftrightarrow live\_w_0.$
- $live\_w_0 \Leftrightarrow (live\_w_1 \wedge up\_s_2) \vee (live\_w_2 \wedge down\_s_2).$
- $live\_w_1 \Leftrightarrow live\_w_3 \wedge up\_s_1.$
- $live\_w_2 \Leftrightarrow live\_w_3 \wedge down\_s_1.$
- $live\_w_3 \Leftrightarrow live\_outside \wedge ok\_cb_1.$
- $live\_outside \Leftrightarrow true.$

Così, ad es., up_s_1 è falso, $live\_w_1$ è falso e $live\_w_2$ è vero

Negation as Failure

Letterale: atomo o negazione di un atomo

1. Con il completamento *negazioni* dimostrabili,
   quindi possono essere ammesse anche nei corpi delle clausole
   - clausole definite *estese*: nel corpo letterali (non atomi)
   - *negazione sotto assunzione di conoscenza completa* o **negation as failure** (NAF)
     denotata con $\sim a$
     - per distinguerla dalla negazione classica
2. sotto NAF, il corpo g è una conseguenza di KB se $KB' \models g$,
   dove KB' è il completamento di KB
   - $\sim a$ nel corpo di una clausola diventa $\neg a$ nel completamento

Esempio — Si consideri l'assiomatizzazione precedente

- Semplifica la rappresentazione chiedendo all'utente di indicare solo i commutatori up
  - gli altri $down$ per default
- Regole da aggiungere:
  - $down\_s_1 \leftarrow \sim up\_s_1.$
  - $down\_s_2 \leftarrow \sim up\_s_2.$
  - $down\_s_3 \leftarrow \sim up\_s_3.$
- Analogamente, si potrebbe specificare che i fusibili sono *funzionanti* a meno che non risultino *rotti*:
  - $ok\_cb_1 \leftarrow \sim broken\_cb_1.$
  - $ok\_cb_2 \leftarrow \sim broken\_cb_2.$

(..cont.)

- Per rappresentare lo stato della figura, l'utente specifica:
  - up_s_2.
  - up_s_3.
- Il sistema può inferire che s_1 dev'essere `down` e i due fusibili funzionanti
- Completamento:

- $down\_s_1 \Leftrightarrow \neg up\_s_1$.
- $down\_s_2 \Leftrightarrow \neg up\_s_2$.
- $down\_s_3 \Leftrightarrow \neg up\_s_3$.
- $ok\_cb_1 \Leftrightarrow \neg broken\_cb_1$.
- $ok\_cb_2 \Leftrightarrow \neg broken\_cb_2$.
- $up\_s_1 \Leftrightarrow false$.
- $up\_s_2 \Leftrightarrow true$.
- $up\_s_3 \Leftrightarrow true$.
- $broken\_cb_1 \Leftrightarrow false$.
- $broken\_cb_2 \Leftrightarrow false$.

NB atomi che non occorrono mai come testa considerati falsi

NB con la NAF, KB con cicli *problematiche* dal punto di vista semantico

- es. data la KB non aciclica:
  - $a \leftarrow \sim b.$
  - $b \leftarrow \sim a.$

    completamento equivalente a $a \Leftrightarrow \neg b$,
    - indica che a e b hanno valori di verità opposti: solo uno è vero
- es. KB non aciclica:
  - $a \leftarrow \sim a.$

    completamento $a \Leftrightarrow \neg a$, logicamente *inconsistente*!

Completamento di una *base aciclica*:

- sempre *consistente*
- prevede *un solo* valore di verità per ogni atomo

Ragionamento Non Monotono

Logica **monotòna**: ogni proposizione derivabile da una KB rimane derivabile dopo l'aggiunta di altre proposizioni

- aggiungendo assiomi *non* si riduce l'insieme delle proposizioni derivabili
- ad es. *Logica delle clausole definite*

Logica **non monotona**: alcune conclusioni possono essere invalidate con l'aggiunta di altri assiomi

- ad es. *Logica delle clausole definite + NAF*

Default ed Eccezioni

Logica non monotona utile a rappresentare casi predefiniti:

- **default**: regola che resta valida fintantoché non si verifichi un'*eccezione*
  - ad es. per asserire che *normalmente* b è vera se c è vera:

$$b \leftarrow c \wedge \sim ab\_a.$$

    con ab_a che indica l'*anormalità* rispetto a qualche aspetto di a
    - dato c si può derivare b, a meno che non venga asserito ab_a, l'aggiunta di ab_a *inibisce* la conclusione di b
      - regole che implicano ab_a possono inibire il default, sotto le condizioni del corpo della regola

Procedura Bottom-Up + NAF

Fallimento (def. ricorsiva):

- p fallisce se, fallisce il corpo di *ogni clausola* che ha p come testa
- un corpo fallisce se *almeno uno* dei suoi *letterali* fallisce
- il letterale b_i / $\sim b_i$ fallisce se si può derivare $\sim b_i$ / b_i

Procedura BU per clausole definite modificata:

- quando p fallisce va aggiunto letterale $\sim p$ all'insieme C
  - conseguenze derivate

procedure Prove_NAF_BU(KB)

Input

KB: insieme di clausole che possono includere NAF

Output

insieme di letterali che seguono logicamente dal completamento di KB

Local

C insieme di letterali

$C \leftarrow \{\}$

repeat

either

selezionare $(h \leftarrow b_1 \wedge \cdots \wedge b_m) \in KB$ tale che:

$h \notin C$ e $\forall i : b_i \in C$

$C \leftarrow C \cup \{h\}$

or

selezionare h con $\sim h \notin C$ tale che:

per ogni $(h \leftarrow b_1 \wedge \cdots \wedge b_m) \in KB$ si abbia che

$\exists i : \sim b_i \in C$ oppure $b_i = \sim g$ e $g \in C$

$C \leftarrow C \cup \{\sim h\}$

until non ci sono altre selezioni possibili

Osservazione

Sono ricompresi i casi

- *corpo vuoto*: $m = 0$ e atomo della testa aggiunto a C
- *atomo non definito* come testa di alcuna clausola:
  si aggiunge a C la sua negazione

Esempio — Si considerino le clausole:

1. $p \leftarrow q \wedge \sim r.$
2. $p \leftarrow s.$
3. $q \leftarrow \sim s.$
4. $r \leftarrow \sim t.$
5. $t.$
6. $s \leftarrow w.$

possibile sequenza di letterali aggiunti a C:

- $C = \{t\}$
  - 5. (fatto)
- $C = \{t, \sim r\}$
  - $t \in C$ e 4.
- $C = \{t, \sim r, \sim w\}$
  - non ci sono clausole per w
- $C = \{t, \sim r, \sim w, \sim s\}$
  - $\sim w \in C$ e 6.
- $C = \{t, \sim r, \sim w, \sim s, q\}$
  - $\sim s \in C$ e 3.
- $C = \{t, \sim r, \sim w, \sim s, q, p\}$
  - $q \in C, \sim r \in C$ e 1.

Procedura Top-Down + NAF

Procedura TD analoga a quella delle clausole definite,
ma procede per NAF

- non-deterministica
- implementabile come ricerca di scelte di successo:
  - selezionato un atomo $\sim a$, parte una nuova dimostrazione per a
    - se questa fallisce allora
      - $\sim a$ ha successo
    - altrimenti l'algoritmo fallisce:
      - deve tentare altre scelte se possibile

```
non-deterministic procedure Prove_NAF_TD(KB, Query)
   Input
      KB: insieme di clausole che includono la NAF
      Query: insieme di letterali da dimostrare
   Output
      yes se Query segue dal completamento di KB, fail altrimenti
   Local
      G: insieme di letterali

   G ← Query
   repeat
      selezionare il letterale l ∈ G
      if l = ∼a then
         if Prove_NAF_TD(KB,a) = fail then
            G ← G \ {l}
         else
            return fail
      else
         scegliere una clausola l ← B ∈ KB
         G ← G \ {l} ∪ B
   until G = ∅
   return yes
```

Esempio — Data la KB precedente e la query ask p:

- $G = \{p\}$ — inizializzazione
- $G = \{q, \sim r\}$ — sostituiti con il corpo di 1.
- $G = \{\sim s, \sim r\}$ — sostituito q con il corpo della 3.
- $G = \{\sim r\}$ — selezionato ed eliminato $\sim s$
  - la dim. di s *fallisce*
- $G = \{\}$ — $\sim r$, selezionato, dimostrato ed eliminato:
  - la dim. di r *fallisce*:
    - usando la 4. sotto-goal $\sim t$: si tenta di dimostrare t
      - data 5. la dim. di t ha *successo* immediato, quindi quella di $\sim t$ *fallisce*
    - non ci sono altre regole per r

G vuoto → output `yes`

Osservazione

Vale solo nel caso di *fallimento finito*:
nessuna conclusione in caso di divergenza

- **ad es., data la sola regola $p \leftarrow p$ e la query ask p,**
  l'algoritmo non converge
  - **completamento: $p \Leftrightarrow p$**
  - **pur potendosi accertare l'indimostrabilità di p, una procedura *corretta* non può concludere $\sim p$: non segue logicamente dal completamento**

Abduzione

Abduzione

Abduzione [Peirce] forma di ragionamento nella quale si fanno *ipotesi*/*assunzioni* per *spiegare* osservazioni

- invece di aggiungerle semplicemente alla KB
  - ad es. se si osserva che qualche luce non sta funzionando, si fanno ipotesi su quanto sta succedendo

Si differenzia dalla *deduzione*

- che mira alle conseguenze logiche di un set di assiomi

e dall'*induzione*

- che comporta l'inferenza di relazioni generali da casi particolari (esempi)

Dato un caso osservato, si fanno ipotesi che potrebbero risultare vere:

- ipotesi che *possono* implicare le osservazioni fatte
  - ossia ciò che, se verificato, rende vere le osservazioni
  - senza *contraddizioni*: giustificherebbero qualsiasi conclusione
    - *ex falso quodlibet*

Formalizzazione

Dati:

- KB, insieme di clausole di Horn
- A, insieme di atomi, detti **assumibili**, per costruire ipotesi
  - o anche *abducibili*

Trovare: *spiegazioni* per le osservazioni

Scenari e Spiegazioni

Scenario di $\langle KB, A \rangle$: $H \subseteq A$ tale che $KB \cup H$ soddisfacibile

- i.e. esiste un modello in cui tutti gli elementi di KB e di H siano veri
- per questo, nessun sottoinsieme di H dev'essere un conflitto di KB

Spiegazione (*ipotesi*) della proposizione g da $\langle KB, A \rangle$:
scenario $H \subseteq A$ che, assieme a KB, implichi g:

- $KB \cup H \models g$
- $KB \cup H \not\models false$

Spiegazione minimale di g da $\langle KB, A \rangle$:
spiegazione H di g da $\langle KB, A \rangle$ tale che nessuna sua parte lo sia

Esempio — Si consideri una KB con assumibili per la *diagnosi medica*:

- $bronchitis \leftarrow influenza.$
- $bronchitis \leftarrow smokes.$
- $coughing \leftarrow bronchitis.$
- $wheezing \leftarrow bronchitis.$
- $fever \leftarrow influenza.$
- $fever \leftarrow infection.$
- $sore\_throat \leftarrow influenza.$
- $false \leftarrow smokes \wedge nonsmoker.$
- assumable $smokes, nonsmoker, influenza, infection.$

Se si osserva $wheezing$ (respiro affannoso), due spiegazioni minime:

- $\{influenza\}$ e $\{smokes\}$
  - che implicano $bronchitis$ e $coughing$ (tosse)

(..cont.)

Se si osservano $wheezing \wedge fever$, spiegazioni minimali:

- $\{influenza\}$ e $\{smokes, infection\}$

Se si osservano $wheezing \wedge nonsmoker$, spiegazione minimale:

- $\{influenza, nonsmoker\}$
  - anche $\{smokes\}$ potrebbe spiegare $wheezing$,
    ma il vincolo la rende inconsistente rispetto a $nonsmoker$ osservato

Esempio — Si consideri la KB:

- $alarm \leftarrow tampering.$
- $alarm \leftarrow fire.$
- $smoke \leftarrow fire.$

Se si osserva $alarm$, spiegazioni minimali:

- $\{tampering\}$ (manomissione) e $\{fire\}$

Se si osservano $alarm \wedge smoke$, spiegazione minimale:

- $\{fire\}$
  - avendo osservato $smoke$ non c'è bisogno di ipotizzare $tampering$ per spiegare $alarm$: già spiegato da $fire$

Diagnosi Abduttiva

AD — *diagnosi* dei problemi della KB in base a osservazioni sul comportamento

Diagnosi Consistency-Based vs Diagnosi Abduttiva

- *rappresentazione*
  - CBD: modellato *solo* il comportamento normale
    - ipotesi: assunzioni di comportamento normale
  - AD: modellato anche il comportamento anomalo
    - assumibili anche per ciascun guasto (o caso anomalo)
- *osservazioni*
  - CBD: sono semplicemente *aggiunte* alla KB e si dimostra $false$
  - AD: vanno anche *spiegate*
- *dettaglio*
  - AD modellazione più dettagliata → diagnosi migliori
    - KB e assunzioni consentono di *dimostrare* le osservazioni
    - anche per casi in cui non sia definibile un comportamento normale

Abduzione per la Progettazione

- *Query* da spiegare → obiettivo di progettazione
- *Assumibili* → mattoni per la progettazione
- *Spiegazione* → progetto
- *Consistenza* → progetto possibile
- *Implicazione* dell'obiettivo di progetto → il progetto ha raggiunto dimostrabilmente l'obiettivo

Ragionamento Abduttivo: Procedure

Ragionamento basato su *assunzioni* mediante le procedure su *clausole di Horn*

- con BU si calcolano spiegazioni minimali per ogni atomo
  - specializzabile con il pruning delle spiegazioni *dominate*
- con TD si trovano spiegazioni di un g generando i conflitti
  ma dimostrando g (anziché $false$)
  - spiegazione minimale di g:
    insieme minimale di assumibili raccolti nella prova

Modelli Causali

Atomo primitivo definito attraverso fatti

Atomo derivato definito usando regole

- In genere il progettista scrive assiomi per atomi derivati
  e si aspetta che vengano specificati atomi primitivi veri
  - un atomo derivato viene *inferito* come necessario dai primitivi e altri derivabili

Molte *decisioni* per la *definizione* della KB

Esempio — due proposizioni per la KB (devono essere vere)

- definibili in vari modi:
  1. a e b come clausole atomiche (atomi primitivi)
  2. a primitivo e b derivato:
     - a clausola atomica
     - e regola $b \leftarrow a$
  3. oppure, viceversa
     - b primitivo/clausola atomica
     - $a \leftarrow b$
- rappresentazioni logicamente equivalenti ma effetti diversi quando si modifica la KB:
  - se a per qualche ragione non è più vero
    - con la 1. e la 3. b ancora vera
    - con la 2. b non sarebbe più vero

Modelli Causali e di Evidenza

Modello causale, o di *causalità*:
rappresentazione che predice i risultati di un intervento

- **intervento** azione che forza una variabile ad avere un dato valore
  - in modi diversi dalla modifica di altre variabili del modello
- un modello causale *rappresenta* come una causa implichi l'effetto per predire i risultati dell'intervento
  - se cambia una causa devono essere cambiati gli effetti

Modello d'evidenza: rappresenta il dominio nell'altra direzione,
dall'effetto alla causa

- non necessariamente una causa precisa ma un insieme di proposizioni che, nel loro insieme, possono causare l'avverarsi dell'effetto

Modello causale spesso preferibile a uno d'evidenza:
più trasparente, stabile e modulare

Riferimenti

[1] D. Poole, A. Mackworth: *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press [Ch.5]
[2] D. Poole, A. Mackworth, R. Goebel: *Computational Intelligence: A Logical Approach*. Oxford University Press
[3] S. J. Russell, P. Norvig: *Artificial Intelligence* Pearson. 4th Ed. - cfr. anche ed. Italiana
[4] J. Sowa: *Knowledge Representation: Logical, Philosophical, and Computational Foundations* Brooks Cole/Cengage
[5] I.M. Copi, C. Cohen: *Introduction to Logic*. Pearson
[6] D.M. Gabbay, C.J. Hogger, J.A. Robinson: *Handbook of Logic in Artificial Intelligence and Logic Programming* Oxford University Press

LINK

[Logica_Proposizionale] it.wikipedia
[conversione] algoritmo di conversione dalle formule proposizionali alle clausole
[Modus_ponens] it.wikipedia
[Modus_tollens] it.wikipedia
[Forward_chaining] it.wikipedia
[Backward_chaining] it.wikipedia
[Proof_by_Contradiction] en.wikipedia e (*reductio ad absurdum*) it.wikipedia
[SAT] cfr. Treccani
[Abduzione] it.wikipedia

NOTE

[⋖] consigliata la lettura
[versione] 17/10/2022, 08:59:55

Figure tratte da [1] salvo diversa indicazione