4. Rappresentazione e Ragionamento Proposizionale

Dispensa ICon

versione: 23/10/2024, 17:40

Proposizioni · Vincoli Proposizionali · Clausole Definite Proposizionali · Interrogare l'Utente · Debugging a Livello di Conoscenza · Dimostrazione per Contraddizione · Assunzione di Conoscenza Completa · Abduzione

1 Proposizioni

Le **proposizioni** sono *enunciati* che possono essere veri o falsi. Esse sono espresse in un *linguaggio* formale che si può descrivere in termini di *sintassi* e *semantica*, la **Logica Proposizionale**.

Il *ragionamento*, che potrà essere definito in termini di proposizioni, si propone obiettivi quali trovare modelli, trovare conseguenze logiche, formulare ipotesi e fare debugging della conoscenza descritta per correggere eventuali errori (mancata cattura dell'interpretazione intesa).

1.1 Sintassi del Calcolo Proposizionale

A ciascun enunciato, se formulato correttamente in un *linguaggio* basato su variabili booleane e connettivi logici, potrà essere associato successivamente un valore di verità.

Una **proposizione atomica**, o **atomo** viene rappresentata/o da un simbolo, *convenzionalmente* una stringa alfanumerica, contenente eventuali _, che inizi con una lettera *minuscola*. Esempi di atomi leciti sono i seguenti: *ai_divertente*, *lampadina1_accesa*, *piove*, *frusco*[1], *soleggiato*.

Una **proposizione** o **formula logica** è costituita da *atomo oppure* da una *proposizione* **composta** formata usando **connettivi** logici e proposizioni *più semplici*. Nella seguente tabella sono elencate le diverse di proposizioni composte che si possono formare a partire da due generiche proposizioni p e q:

sintassi	lettura	definizione
$\neg p$	"non p"	*negazione* di p
$p \wedge q$	"p e q"	*congiunzione* di p e q
$p \vee q$	"p o q"	*disgiunzione* di p e q
$p \rightarrow q$	"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
$p \oplus q$	"p XOR q"	*or esclusivo* di p e q

Vale il classico ordine di *precedenza* fra connettivi: $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow, \oplus$. Si possono usare le *parentesi* per disambiguare e avere una maggiore leggibilità, ad esempio la proposizione $\neg a \vee b \wedge c \rightarrow d \wedge \neg e \vee f$ si potrà scrivere anche nella forma $((\neg a) \vee (b \wedge c)) \rightarrow ((d \wedge (\neg e)) \vee f)$.

1.2 Semantica del Calcolo Proposizionale

La **semantica** definisce il *significato* degli enunciati, ossia delle proposizioni, fornendo la *corrispondenza* tra simboli e stato del mondo che viene rappresentato. Ogni proposizione sarà *interpretabile* come vera o falsa rispetto allo stato del mondo: le proposizioni vere determinano il mondo conosciuto dal sistema, essendo un *atomo* l'unità *minima* che possa essere dotata di significato. Il significato delle proposizioni più complesse discenderà da quello degli atomi.

Un'**interpretazione** è definita come funzione $\pi : \text{Atomi} \rightarrow \{true, false\}$

- se $\pi(a) = true$, si dice che a **vero** nell'interpretazione π;
- se $\pi(a) = false$, si dice che a **falso** nell'interpretazione π.

Un'interpretazione si può anche definire come l'*insieme* I di tutti e soli gli atomi veri, ossia associati al valore di verità $true$. Tutti gli altri atomi saranno interpretati come associati a $false$.

Per associare i valori di verità alle proposizioni non atomiche, si dovranno considerare le *tavole di verità* dei connettivi, mostrate di seguito:

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

Si noti che la verità di una proposizione dipende solo dall'interpretazione dei suoi atomi. Una stessa proposizione ammette diverse interpretazioni per gli atomi contenuti, quindi potrà avere assegnati valori di verità diversi.

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 $light\_on$

ovvero una funzione 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 è vera e $\neg ai\_is\_fun$ è falsa;
- $happy$ è falsa e $\neg happy$ è vera;
- $ai\_is\_fun \vee happy$ è vera;
- $ai\_is\_fun \leftarrow happy$ è vera;
- $ai\_is\_fun \rightarrow happy$ è falsa;
- $ai\_is\_fun \leftarrow happy \wedge light\_on$ è vera.

Data l'interpretazione I_2 che assegna $false$ a ai_is_fun, $true$ a $happy$, e $false$ a $light\_on$, allora in I_2:

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

Una **Base di Conoscenza** (KB, *knowledge base*) è costituita da un insieme di proposizioni dette **assiomi**: essi sono dichiarati, ossia assunti, come veri. La base di conoscenza può essere vista come una congiunzione implicita dei assiomi, quindi sarà vera se tutti sono veri.

Gli assiomi servono a specificare il *mondo* da rappresentare attraverso quella che si dice la sua **interpretazione intesa**, ossia quella che dovrebbe risiedere nella mente dell'*esperto di dominio* e che va formalizzata dall'*ingegnere della conoscenza* che fornisce la sua *assiomatizzazione*. Un **modello** di una base di conoscenza KB è un'interpretazione per la quale sia *vero* ogni assioma di KB.

La proposizione g (*goal*) è **conseguenza logica** di KB, ovvero g *segue logicamente* da KB sse g vera in ogni modello di KB, in simboli:

$$KB \models g$$

In inglese questa relazione fra KB e g si dice *entailment*.

Quindi, se vale $KB \models g$, non esistono modelli di KB nei quali g sia falsa. Si può scrivere $KB \not\models g$ per dire che "g non è conseguenza logica di KB", ciò significa che esiste (almeno) un modello di KB in cui g è falsa. Un caso da distinguere è quello in cui $KB \models \neg g$.

> ⚠ Si noti che non vengono prese in considerazione altre possibili interpretazioni di g che non siano modelli di KB.

Esempio — Data la KB contenente gli assiomi

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

si può scrivere

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

mentre

- $KB \not\models switch\_1\_is\_up$ ossia esiste un modello di KB in cui $switch\_1\_is\_up$ è falso; tale modello, dovendo risultare vero l'ultimo assioma [☾] in KB, può essere definito assumendo che $sam\_is\_in\_room$ sia falso.

1.3 Semantica: Punto di Vista del/la Progettista

Per caratterizzare un dominio / mondo (il suo stato), il/la *progettista* della KB lo definisce come **interpretazione intesa**: il significato dei simboli viene definito attraverso proposizioni, ossia assiomi che si possono assumere veri rispetto a tale stato del mondo, quindi le conseguenze logiche di KB saranno *vere* rispetto all'*interpretazione intesa*. Il significato viene così *condiviso* e potrà essere usato per interpretare le risposte a interrogazioni alla KB.

Metodologia di Costruzione di Basi di Conoscenza

Il *progetto* della KB si sviluppa nelle seguenti fasi:

1. Si sceglie il *dominio*/mondo da rappresentare, ossia l'*interpretazione intesa*.
   - Questo può comprendere sia aspetti del mondo *reale* come, ad es., la struttura dei corsi universitari, un laboratorio in un particolare momento, oppure anche un mondo *immaginario* come, ad es., quello di Alice nel paese delle meraviglie o lo stato di un sistema in casi-limite, infine anche un dominio *astratto* come quello numerico e/o dell'insiemistica.
2. Si scelgono gli *atomi* per rappresentare le proposizioni d'interesse, la cosiddetta **concettualizzazione**, cui associare un significato preciso rispetto all'interpretazione intesa.
3. Si definiscono gli *assiomi*, ossia *proposizioni* vere nell'interpretazione intesa, l'**assiomatizzazione** del dominio applicativo per il sistema.
4. Si pongono al sistema **domande** sull'interpretazione intesa; le sue **risposte** dovranno essere interpretate in base ai significati associati ai simboli.

1.4 Semantica: Punto di Vista della Macchina

Il sistema elabora la KB per decidere se g vera nell'interpretazione intesa. I modelli di KB rappresentano tutti e soli i modi in cui il mondo potrebbe essere fatto. Data una concettualizzazione e un'assiomatizzazione corrette, l'interpretazione intesa sarà uno dei modelli di KB, ma il sistema non sa determinarlo esattamente.

Il sistema deve determinare se $KB \models g$: se $KB \models g$, esso potrà concludere che g è vera in tale modello, essendo g vera in tutti i modelli di KB. Se, invece, $KB \not\models g$, allora non può trarre conclusioni: g *potrebbe* essere falsa proprio nell'interpretazione intesa ma il sistema non può stabilirlo dato che non la conosce.

Queste forme di inferenza sono implementate dalle procedure di dimostrazione trattate nel seguito.

> ⚠ 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.:
  - $KB \models apple\_is\_eaten$ quindi $apple\_is\_eaten$ è vera nell'interpretazione intesa
    - $bird\_eats\_apple. \in KB$
    - $apple\_is\_eaten \leftarrow bird\_eats\_apple. \in KB$
  - $KB \not\models switch\_1\_is\_up$ è falsa in qualche modello (interpretazione intesa?)
    - $switch\_1\_is\_up \leftarrow sam\_is\_in\_room \wedge night\_time. \in KB$
    - $night\_time. \in KB$
    - $sam\_is\_in\_room$: vero o falso nell'interpretazione intesa?

> ⚠ Il/la progettista può sbagliare e codificare nella KB assiomi falsi nell'interpretazione intesa. In tali casi non può essere garantito che le risposte della macchina siano vere in tale interpretazione.

---

2 Vincoli Proposizionali

Si considereranno ora CSP con vincoli rappresentati come *formule logiche* dalla struttura concisa e formale per consentire l'elaborazione automatica.

Un *problema* di **soddisfacibilità proposizionale** (*SAT*) è definito da:

- un insieme di *variabili booleane*, i.e. con dominio $\{true, false\}$; si usa la sintassi $X = true$ abbreviata in x, e $X = false$ oppure $\neg x$ quindi, ad esempio, data la variabile booleana $Happy$, $happy$ sta per $Happy = true$ e $\neg happy$ sta per $Happy = false$;
- un insieme di *vincoli clausali* o **clausole**, ossia espressioni logiche della forma:

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

  in cui ogni **letterale** l_i è un atomo a o la sua negazione $\neg a$, ossia un'assegnazione a una variabile booleana; si dirà che a *occorre negativamente* (risp. *positivamente*) nel letterale $\neg a$ (risp. a).

La clausola $l_1 \vee l_2 \vee \cdots \vee l_k$ è vera in un'interpretazione (*mondo possibile*) sse $\exists i \in \{1, \ldots, k\}: l_i$ vero in tale interpretazione. In tal caso, si dice anche che il vincolo clausale è *soddisfatto*.

Clausole come Proposizioni o Vincoli

In *Logica Proposizionale*, una clausola è una formula logica in una forma ristretta (normale). Esiste un algoritmo [conversione] per cui *ogni proposizione può essere convertita in forma di clausola*.

Nei *CSP*, una clausola è un *vincolo* su un insieme di variabili booleane che è *soddisfatto* se almeno uno dei sui letterali è vero: $l_1 \vee l_2 \vee \cdots \vee l_k$ equivale a $\neg(\neg l_1 \wedge \neg l_2 \wedge \cdots \wedge \neg l_k)$, ciò esclude le assegnazioni in cui *tutti* i letterali risultino falsi.

Esempio — Clausola

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

- vista come come vincolo su $Happy$, Sad e $Living$, è soddisfatto se si assegna $true$ a $Happy$ o Sad oppure $false$ a $Living$, dato che $happy$ e sad vi occorrono positivamente e $living$ negativamente;
- un'assegnazione che viola il vincolo è: $\neg happy, \neg sad, living$; precisamente essa è l'unica assegnazione che coinvolge le tre variabili che la violi.

2.1 Forma Clausale per CSP

Un CSP finito si può trasformare in un problema di soddisfacibilità di una KB proposizionale in cui:

- per ogni variabile Y del CSP, con $dom(Y) = \{v_1, \dots, v_k\}$ si considerano le **variabili indicatrici** booleane $\{Y_1, \dots, Y_k\}$ tali che $\forall i = 1, \dots, k$: Y_i *vera* se $Y = v_i$ e *falsa* altrimenti; quindi per rappresentare Y in KB si considereranno gli atomi $y_1, \dots, y_k$;
- si aggiungeranno a KB le seguenti *clausole* (vincoli):
  - $\neg y_i \vee \neg y_j$ per ogni $i, j \in \{1, \dots, k\}$ con $i < j$, poiché y_i e y_j non possono essere entrambi veri (si può assegnare un solo valore a Y);
  - $y_1 \vee \dots \vee y_k$, poiché uno degli y_i dev'essere necessariamente vero, ossia Y avrà un certo valore i nell'assegnazione totale;
- per ogni vincolo del CSP, si definirà una clausola per (negare) ogni *assegnazione* che lo violi:
  - $\neg(l_1 \wedge l_2 \wedge \dots \wedge l_k)$, che equivale a $\neg l_1 \vee \neg l_2 \vee \dots \vee \neg l_k$, i.e. letterali per le *assegnazioni* alle Y_i *non ammesse* dal vincolo;
  - per semplificare, si possono combinare le clausole; ad esempio, combinando $a \vee b \vee c$ con $a \vee b \vee \neg c$ si ha $a \vee b$.

Esempio — Considerando le variabili A e B con dominio $\{1, 2, 3, 4\}$

- per A, le variabili indicatrici saranno a_1, a_2, a_3, e a_4, con a_i vera sse $A = i$, variabili disgiunte che coprono tutti i casi per cui, in tutto, 7 clausole:
  $\neg a_1 \vee \neg a_2$, $\neg a_1 \vee \neg a_3$, $\neg a_1 \vee \neg a_4$, $\neg a_2 \vee \neg a_3$, $\neg a_2 \vee \neg a_4$, $\neg a_3 \vee \neg a_4$, $a_1 \vee a_2 \vee a_3 \vee a_4$;
- analogamente per B;
- il vincolo $A < B$ implica che a_4 sia falsa, b_1 falsa e le coppie che lo violano siano parimenti false quindi un totale di 5 clausole:
  $\neg a_4$, $\neg b_1$, $\neg a_2 \vee \neg b_2$, $\neg a_3 \vee \neg b_3$, $\neg a_3 \vee \neg b_2$;
  NB non serve $\neg a_1 \vee \neg b_1$ implicata da $\neg b_1$.

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 base di conoscenza

- 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 $Y = false$

Algoritmi

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

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

L'algoritmo risulta 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
  - le clausole dove compare positivamente *potrebbero* non essere più soddisfatte
    - le clausole dove occorre negativamente sono soddisfatte

Osservazioni — Rispetto ai CSP:

- Spazio di ricerca *esteso*
  - prima di trovare una soluzione:
    più di una Y_i vera $\rightarrow$ Y ha più valori
    oppure, Y_i tutte false $\rightarrow$ 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

3 Clausole Definite Proposizionali

Il linguaggio delle **clausole definite proposizionali** è un sottolinguaggio della logica proposizionale con la stessa semantica (e non consente ambiguità o incertezza nella rappresentazione). Esso ammette una forma più specifica di proposizioni, le clausole con un unico letterale positivo della forma seguente:

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

Sintassi

Le *proposizioni atomiche* o *atomi* sono gli stessi della logica proposizionale. Valendo l'equivalenza $A \vee \neg B \equiv A \leftarrow B$, una **clausola definita** proposizionale si può scrivere come segue:

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

(con h, a_i atomi) formula che si legge "h se a_1 e $\cdots$ e a_m".

In essa, h si dice **testa** (*head*) e $a_1 \wedge \cdots \wedge a_m$ si dice **corpo** della clausola. Una clausola definita è una **regola** se $m > 0$ (ossia ha un corpo che rappresenta una condizione), altrimenti è un **fatto** o **clausola atomica** quando $m = 0$ (*corpo vuoto*) nel qual caso si può omettere "$\leftarrow$".

Una **base di conoscenza** può essere specificata come insieme di clausole definite.

Esempio — Nella base di conoscenza vista in precedenza tutte clausole sono 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 $happy \vee sad \leftarrow alive.$

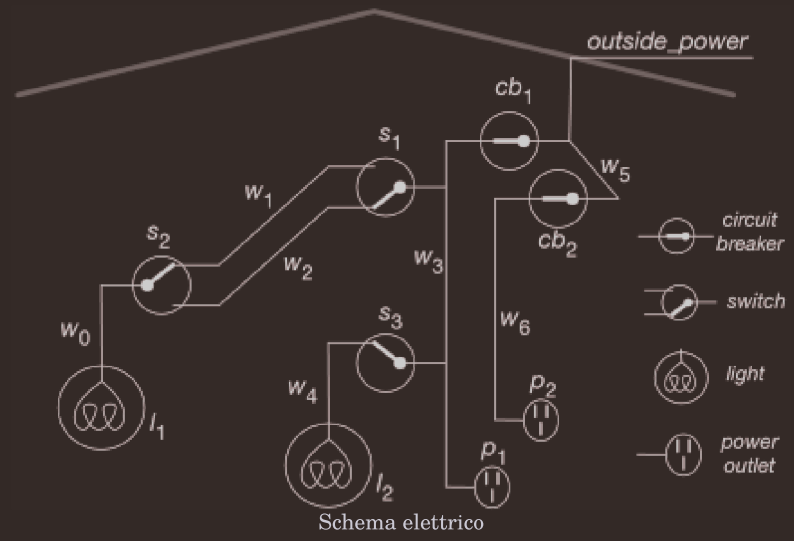
Semantica

La clausola $h \leftarrow a_1 \wedge \cdots \wedge a_m$ è *falsa* nell'interpretazione I se $a_1, \ldots, a_m$ sono tutti veri in I ma h è falsa, altrimenti essa è *vera* in I (cfr. tavola di verità del connettivo '$\leftarrow$').

La clausola definita è una forma *ristretta* di clausola, definita in precedenza, nella quale compare *esattamente* un letterale positivo. Ad esempio, $a \leftarrow b \wedge c \wedge d.$ è equivalente a

$a \vee \neg b \vee \neg c \vee \neg d$. Essa non può rappresentare una qualunque disgiunzione di atomi (come $a \vee b$), nemmeno se tutti negati (ad esempio, $\neg c \vee \neg d$).

Esempio — Domotica: ambiente intelligente per la simulazione (diagnostica)



Schema elettrico

- si definisce una *rappresentazione* dello stato di cavi (*wire*), prese (*outlet*), luci (*light*), deviatori (*switch*) e fusibili (*circuit breaker*), ... I dettagli trascurabili non verranno modellati (ad es. colori, peso, lunghezze, altezze, ecc.);
- bisogna decidere il *livello di astrazione* per la rappresentazione: per *specialisti* (i voltaggi, le correnti, le frequenze, ...) oppure comprensibile anche a non specialisti (in base al *senso comune*);
- poi si decide *cosa* rappresentare: proposizioni su accensione luci, funzionamento cavi, stato deviatori, ecc.;
- si definiscono *atomi* con significati precisi per il mondo da modellare: nomi descrittivi come up_s_2 per indicare che un deviatore/switch s_2 è in posizione up oppure $live\_l_1$ che indica che il punto-luce l_1 è *sotto tensione*.
- si definisce anche la *conoscenza di fondo* (BK) che fornisce altri fatti veri riguardo il mondo, ed è rappresentata nel sistema attraverso altre clausole definite, spesso in forma semplice, come fatti:
  - $light\_l_1$.
  - $light\_l_2$.
  - ok_l_1.
  - ok_l_2.
  - ok_cb_1.
  - ok_cb_2.
  - $live\_outside$.
- 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"
  - $live\_w_0 \leftarrow live\_w_1 \wedge up\_s_2$.
  - $live\_w_0 \leftarrow live\_w_2 \wedge down\_s_2$.
  - $live\_w_1 \leftarrow live\_w_3 \wedge up\_s_1$.
  - $live\_w_2 \leftarrow live\_w_3 \wedge down\_s_1$.
  - $live\_l_2 \leftarrow live\_w_4$.
  - $live\_w_4 \leftarrow live\_w_3 \wedge up\_s_3$.
  - $live\_p_1 \leftarrow live\_w_3$.
  - $live\_w_3 \leftarrow live\_w_5 \wedge ok\_cb_1$.
  - $live\_p_2 \leftarrow live\_w_6$.
  - $live\_w_6 \leftarrow live\_w_5 \wedge ok\_cb_2$.
  - $live\_w_5 \leftarrow live\_outside$.
  - $lit\_l_1 \leftarrow light\_l_1 \wedge live\_l_1 \wedge ok\_l_1$.
  - $lit\_l_2 \leftarrow light\_l_2 \wedge live\_l_2 \wedge ok\_l_2$.
- a run-time, si forniranno al sistema delle *osservazioni*, ad es. fatti sullo stato dei deviatori, come:
  - $down\_s_1$.
  - up_s_2.

- up_s_3.

3.1 Domande e Risposte

Si deve determinare la verità di fatti riguardanti il mondo ossia il dominio rappresentato:

- un utente può porre *domande* sulle conseguenze logiche della base di conoscenza fornita alla macchina;
- il sistema è in grado di dare *risposte* decidendo se date proposizioni seguano logicamente dalla base di conoscenza o meno;
- l'utente potrà comprendere il *significato* di tali risposte, se conosce la semantica degli atomi utilizzati.

A tale scopo si definisce una **query**, ossia una domanda utile a sapere se una data proposizione segua logicamente da una base di conoscenza, denotata con: ask b.

- dove b è un atomo o una congiunzione di atomi (ciò vale solo per basi di clausole definite) il che corrisponde al *corpo* di una regola: $\leftarrow b_1 \wedge \cdots \wedge b_m$;

Le *risposte* possibili sono:

- yes se b è conseguenza logica, $KB \models b$;
- no altrimenti, $KB \not\models b$: ciò non significa che b sia falso nell'interpretazione intesa bensì che non è possibile determinarlo con la conoscenza disponibile.

Esempio — data la base di conoscenza 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
  - l_1 acceso, atomo vero in tutti i modelli

L'*utente* potrà interpretare le risposte rispetto all'interpretazione intesa.

3.2 Dimostrazioni

La relazione $\models$ definisce le risposte corrette secondo la cosiddetta *model theory*. La **proof theory** si occupa invece di come *calcolare* tali risposte, ossia della dimostrazione delle conseguenze logiche. Si tratta di un problema di *deduzione*, una forma di *inferenza*.

Una **prova** (*proof*) è una *dimostrazione*, derivabile anche *automaticamente*, del fatto che una proposizione segua logicamente da un insieme di assiomi. Una proposizione dimostrabile si dice anche **teorema**.

Una **procedura di dimostrazione** (*proof procedure*) è un algoritmo, eventualmente non-deterministico, utile a costruire prove. Data una procedura di dimostrazione si denota con

$$KB \vdash g$$

la possibilità di produrre una prova del fatto che g sia conseguenza di KB.

Le proprietà più importanti delle procedure riguardano le relazioni tra prove e modelli:

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

$$\text{se } KB \vdash g \text{ allora } KB \models g$$

- una procedura **completa** sse ogni conseguenza logica di KB può essere dimostrata dalla procedura:

$$\text{se } KB \models g \text{ allora } KB \vdash g$$

Esistono due tipi principali di procedure: le *Bottom-Up* (BU) e le *Top-Down* (TD).

3.3 Procedura Bottom–Up

L'*obiettivo* di una procedura *bottom-up* è quello di derivare tutte le conseguenze logiche di una base di conoscenza. A tal fine si costruisce la prova gradualmente, basandosi su quanto è stato già dimostrato (atomi). Si procede per **concatenazione in avanti** (*forward chaining*) su clausole definite, derivando fatti nuovi a partire da quanto è già stato provato.

Serve una sola *regola di derivazione* che costituisce una generalizzazione della regola d'inferenza detta **modus ponens**

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

Se $h \leftarrow a_1 \wedge \cdots \wedge a_m$ è in KB e ogni a_i è già stato provato, allora si potrà derivare h. Ogni fatto ($m = 0$) viene considerato *immediatamente* provato.

La procedura trova C, l'**insieme delle conseguenze** di KB:

```
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 risposta restituita è `yes` ossia $KB \vdash g$ quando $g \in C$, se g atomica, ovvero quando $\{g_1, \ldots, g_k\} \subseteq C$ se g è una congiunzione di atomi $g_1 \wedge \ldots \wedge g_k$.

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.
- $f \leftarrow a \wedge g$.

La traccia dei valori assegnati via via a C è la seguente:

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

Da $C = \{a, b, c, e, d\}$ si può concludere che $KB \vdash a$, $KB \vdash b$, ecc.

Note:

- 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 BU

- *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 è un modello *minimale* di KB: ha il minor numero di proposizioni vere tra tutti i suoi modelli (ogni altro modello può avere anche altri atomi veri oltre a quelli di C);

- *Correttezza*: 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 $KB \vdash g$.

3.4 Procedura Top–Down

Tale procedura si basa su una ricerca *backward* ossia *top-down* a partire da una query per determinare se questa segua logicamente dalle clausole di KB. Essa si basa su una diversa regola di ragionamento (inferenza su clausole) detta **risoluzione**:

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

si noti che si applica alla forma generale delle clausole: basta individuare un atomo (a) compare positivamente in una delle due premesse e negativamente nell'altra.

In particolare, si usa la **risoluzione SLD** (*S*elezione di un atomo usando una strategia *L*ineare su clausole *D*efinite):

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

- versione *proposizionale* di una forma più generale presentata in seguito.

Procedura Top–Down con Clausole di Risposta

Una **clausola di risposta** ha la forma

$$yes \leftarrow a_1 \wedge a_2 \wedge \cdots \wedge a_m$$

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

Data la query ask $q_1 \wedge \cdots \wedge q_m$, si parte dalla clausola di risposta *iniziale* $yes \leftarrow q_1 \wedge \cdots \wedge q_m$; successivamente si ripete:

- la *selezione*[2] di un atomo del corpo (*sotto-goal*) da dimostrare, (ad esempio a_1)
- la *scelta* di una clausola definita in KB avente come testa l'atomo selezionato (a_1) per effettuare un passo di **risoluzione**; qualora non vi fossero altre clausole per dimostrare l'atomo selezionato, la dimostrazione *fallisce* (risposta: `no`).

Il **risolvente** della clausola di risposta e della clausola scelta sarà una nuova clausola di risposta:

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

in sostanza l'atomo *goal* selezionato (a_1) viene sostituito dal corpo della clausola scelta.

In una dimostrazione di successo si raggiunge la **risposta**, ossia una clausola di risposta con corpo vuoto (tutti i goal sono stati dimostrati):

$$yes \leftarrow$$

Nella **derivazione SLD** per una query ask $q_1 \wedge \cdots \wedge q_k$ da KB si avrà una sequenza di clausole di risposta $\gamma_0, \gamma_1, \ldots, \gamma_n$ tali che:

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

Procedura Top–Down Alternativa

Si parte da un insieme G (*goal*) di *atomi da dimostrare*, quindi:

- si inizializza G con tutti gli atomi della query (*sotto-goal*) quindi G corrisponde a $yes \leftarrow \bigwedge_{g_i \in G} g_i$;
- selezionato a, la clausola scelta $a \leftarrow b_1 \wedge \cdots \wedge b_p$ indica che a può essere sostituito in G dai sotto-goal $b_1, \ldots, b_p$;

- si ha una terminazione con successo se $G = \emptyset$.

```
non-deterministic procedure Prove_DC_TD(KB, Query)
    Input
        KB: insieme di clausole definite
        Query insieme di atomi da dimostrare
    Output
        yes se KB ⊨ Query, altrimenti fallisce (no)
    Local
        G insieme di atomi

    G ← Query
    repeat
        selezionare un atomo a ∈ G
        scegliere da KB una clausola definita a ← B con a come testa
        // se possibile, altrimenti il tentativo fallisce
        G ← (G \ {a}) ∪ B
    until G = ∅
    return yes
```

Qualsiasi atomo di G può essere *selezionato* (tutti, prima o poi, lo saranno). Se una selezione non porta a terminare la prova, non serve tentare di selezionarne un altro, l'intera dimostrazione fallisce

La procedura di *scelta* della clausola per la risoluzione (che porti al successo del tentativo) è *non-deterministica*. Se ci sono scelte che portano a G vuoto, l'algoritmo ha successo (rispondendo yes) *altrimenti* il tentativo fallisce (e si può rispondere no).

Il corpo della clausola B e G sono considerati come *insiemi* di atomi. In alternativa G può essere costituito da una *lista* ordinata di atomi che possono occorrere più volte (tipicamente implementata negli interpreti Prolog, considerati più avanti).

Esempio — Data la base di conoscenza 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 si usa G (come insieme) in luogo del corpo della clausola di risposta $yes \leftarrow G$, inizialmente gli atomi (della query) da dimostrare: $G = \{a\}$;
- selezionato l'atomo *più a sinistra* del corpo:

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
$yes \leftarrow$	

- Sequenza *alternativa* di scelte in cui si sceglie la seconda clausola per b:
  - $yes \leftarrow a$

- $yes \leftarrow b \wedge c$
- $yes \leftarrow g \wedge e \wedge c$
  ma selezionando g non ci sono regole da scegliere per cui *questo tentativo* fallisce: occorre scegliere altre alternative, se ve ne sono.

Algoritmo TD su Grafo di Ricerca

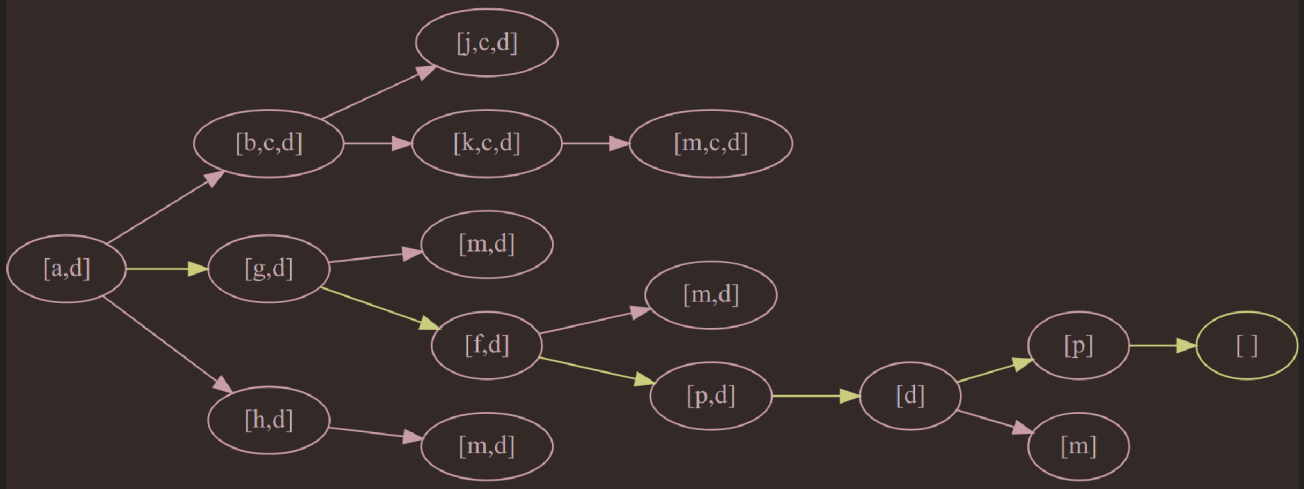
Algoritmo alternativo indotto dalla strategia di selezione basato su un *grafo di ricerca* in cui:

- i *nodi* contengono clausole di risposta (bastano gli atomi nei loro corpi);
- i *vicini* di $yes \leftarrow \underline{a_1} \wedge \cdots \wedge a_m$ rappresentano tutte le possibili clausole di risposta ottenute risolvendo su a_1 e possono essere generati *dinamicamente*, uno per ogni clausola definita con testa a_1;
- i *nodi obiettivo* sono del tipo $yes \leftarrow$
- si può usare qualsiasi metodo di ricerca su grafo: basta trovare *un* percorso di successo perché la query sia conseguenza logica della base di conoscenza; per ogni nodo, lo spazio viene determinato dalla query e dall'atomo selezionato.

Esempio — Data la seguente base di conoscenza e la query ask $a \wedge d$.

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

da cui il grafo (nei nodi, lista di atomi del *corpo* del risolvente):



Confronto fra procedure

Le dimostrazioni TD e BU sono interscambiabili: ciò è utile a provare consistenza e completezza della TD.

La BU prova ogni atomo *una sola* volta mentre TD potrebbe provare lo stesso atomo *più volte*, ma si concentra su quelli *rilevanti* per la query.

Con TD sono 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.

- le uniche conseguenze logiche (atomiche) sono: 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.

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

- negli esempi precedenti: atomo *più a sinistra*, il che risulta problematico senza controllo sui cicli;
- una strategia migliore seleziona l'atomo che porti al *fallimento* più rapidamente;
- una strategia comune prevede l'*ordinamento* degli atomi per la selezione: ciò consente l'uso di un'*euristica*.

4 Interrogare l'Utente

A run-time il sistema riceve **osservazioni** dagli utenti, da sensori o altre sorgenti. Esse sono rappresentate come *insiemi di atomi* (non direttamente come regole), ad esempio nella diagnosi medica il paziente comunica i propri sintomi. Le osservazioni interagiscono con la **conoscenza di fondo** (*background knowledge*, BK): possono esservi *aggiunte* o essere tenute *separate* dalla BK.

Le informazioni fornite dall'**utente** sono incomplete: è possibile che non sappia usare il *vocabolario* o non sappia giudicarne la *rilevanza*. Per questo servirebbero: 1) un'**ontologia** che specifichi il significato dei simboli e 2) un'*interfaccia* che possa aiutare a fornire informazioni rilevanti.

Analogamente, i *sensori* possono essere *passivi*, quando raccolgono osservazioni dirette (congiunzioni di atomi) oppure *attivi*, se atti a raccogliere risposte a esplicite richieste di informazioni necessarie.

Per l'*acquisizione* si può pensare a un meccanismo **ask-the-user** integrato nella procedura di ragionamento: si considerano atomi detti **askable** per rappresentare fatti veri acquisibili a run-time dall'utente (o da sensore). Nella procedura TD, selezionato un atomo da dimostrare, si usa una clausola della base di conoscenza oppure, se askable, si chiede se sia vero o falso.
NB Ciò vale solo per atomi *rilevanti*, quindi si considerano tre diverse *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.

È possibile adottare un meccanismo simile anche con procedure BU, ma vanno evitate *troppe richieste* all'utente.

Esempio — Diagnosi elettrica: non tutto può essere fornito dal progettista (KB + BK) quindi si può ricorrere ad atomi *askable*.

Possibile *dialogo utente-sistema* (con un'interfaccia minimale):

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

NB: si tratta solo di domande rilevanti cui l'utente può rispondere.

---

5 Debugging a Livello di Conoscenza

Data una semantica esplicita si possono realizzare spiegazione e debugging a livello di conoscenza. Il **knowledge-level debugging** consiste nel trovare *errori* nelle basi di conoscenza facendo riferimento *solo* al significato dei simboli e quanto risulta vero nel mondo codificato e non ai passi di ragionamento.

I diversi tipi di *Errore* (non sintattici) *possibili* sono:

- risposta *errata*: derivazione di atomo falso nell'interpretazione intesa;
- risposta *mancante*: dimostrazione fallita per qualche atomo vero nell'interpretazione intesa;
- *loop infinito*: solo con procedure TD, in base alla strategia di selezione. Ma si può evitare con meccanismi di *cycle pruning* (simili a quelli visti in precedenza).

5.1 Risposte Errate

Si tratta di risposte fornite dalla procedura di dimostrazione che sono *false* nell'interpretazione intesa, cioè casi di **falsi positivi**. Se la procedura è *sound* allora nella dimostrazione sarà stata impiegata una *clausola errata*.

Per il *debugging dei falsi positivi*, si considerano l'atomo g dimostrato (conseguenza logica) che, invece, dovrebbe essere falso nell'interpretazione intesa e la clausola usata per dimostrarlo $g \leftarrow a_1 \wedge \cdots \wedge a_k$. I casi possibili sono i seguenti:

- tutti gli a_i sono veri nell'interpretazione intesa, quindi la clausola $g \leftarrow a_1 \wedge \cdots \wedge a_k$ è *errata*;
- uno o più a_i dimostrati sono a loro volta falsi nell'interpretazione intesa, per cui si ha un analogo problema da risolvere (ricorsivamente).

L'algoritmo da usare può essere schematizzato come segue:

```
procedure Debug_false(g, KB)

    Input
        KB base di conoscenza
        g atomo: KB ⊢ g ma falso nell'interpretazione intesa
    Output
        clausola in KB che risulta falsa

    Trovare g ← a_1 ∧ ··· ∧ a_k ∈ KB usata per dimostrare g
    Chiedere all'utente se ogni a_i sia vero o meno
    if l'utente indica un a_i falso then
        return Debug_false(a_i, KB)
    else if l'utente risponde che tutti gli a_i sono veri then
        return g ← a_1 ∧ ··· ∧ a_k
```

Esempio — C'è un bug nella base di conoscenza precedente: 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, interagendo con l'utente e facendo riferimento alla dimostrazione:

- lit_l_1 è falso nell'interpretazione intesa derivato da $lit\_l_1 \leftarrow light\_l_1 \wedge live\_l_1 \wedge ok\_l_1$.
- $live\_l_1$ è falso, derivato usando $live\_l_1 \leftarrow live\_w_0$.

- $live\_w_0$ è falso, derivato usando $live\_w_0 \leftarrow live\_w_1 \wedge up\_s_2$.
- $live\_w_1$ è falso, derivato usando $live\_w_1 \leftarrow live\_w_3 \wedge up\_s_3$.
- essendo $live\_w_3$, up_s_3 veri nell'interpretazione intesa, la *regola difettosa* sarà proprio $live\_w_1 \leftarrow live\_w_3 \wedge up\_s_3$.

5.2 Risposte Mancanti

In tal caso una risposta attesa (ossia goal vero nell'interpretazione intesa) non viene derivata dal sistema. Il *fallimento* dipende da un atomo g *vero* nell'interpretazione intesa ma tale che $KB \nvdash g$, si tratta, cioè, di un caso di **falso negativo**.

Tale atomo segnala il caso di una o più *clausole mancanti* tra gli assiomi della base di conoscenza e si può individuare con una procedura, come quella riportata di seguito, per trovare atomi cui manchino clausole utili alla loro dimostrazione:

```
procedure Debug_missing(g, KB)

    Input
        KB base di conoscenza
        g atomo: KB ⊬ g ma g vero nell'interpretazione intesa
    Output
        atomo per il quale manchi una clausola utile

    if esiste g ← a_1 ∧ ··· ∧ a_k ∈ 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
```

Esempio — Si supponga che $down\_s_2$ sia vero ma manchi la clausola per provarlo: in particolare, data la base di conoscenza, non si riesce a dimostrare lit_l_1. In tal caso:

- si trovano tutte le regole con lit_l_1 in testa, nel caso in esame 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, si trova che $light\_l_1$ e ok_l_1 sono dimostrabili, ma non $live\_l_1$ (caso da indagare);
- anche $live\_l_1$ occorre come testa in una sola regola: $live\_l_1 \leftarrow live\_w_0$. tuttavia $live\_w_0$ non può essere dimostrato, mentre dovrebbe risultare vero nell'interpretazione intesa;
- ci sono due regole per dimostrare $live\_w_0$: $live\_w_0 \leftarrow live\_w_1 \wedge up\_s_2$. e $live\_w_0 \leftarrow live\_w_2 \wedge down\_s_2$.; l'utente sa che il corpo della seconda regola è vero nell'interpretazione intesa ma, mentre $live\_w_2$ è dimostrabile, mancano clausole per $down\_s_2$ che potrà quindi essere restituito come *difettoso*;
- *correzione*: aggiunta di una clausola (fatto o regola) per provare tale atomo.

6 Dimostrazione per Contraddizione

6.1 Clausole di Horn

Si intende estendere il ragionamento ammettendo regole che producano *contraddizioni*, ossia utili a specificare casi *impossibili*. Ciò ha anche lo scopo di implementare successivamente forme di *diagnostica*. Ad esempio, nel caso della domotica, attraverso il

ragionamento (e le osservazioni) si potrà dedurre che alcuni componenti *non* siano *funzionanti*.

A tale scopo il linguaggio delle clausole definite può essere *esteso* per esplicitare le contraddizioni. Definito un atomo speciale, detto $false$, che codifica la *contraddizione*, ossia che è falso in tutte le interpretazioni, saranno ammesse regole, dette **vincoli d'integrità**, rappresentate da clausole con l'atomo $false$ come testa:

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

Una **clausola di Horn** è una clausola definita o un vincolo d'integrità, ossia la sua testa è un atomo normale oppure l'atomo speciale $false$.

Da una base di conoscenza di clausole di Horn si possono derivare **negazioni** di atomi tramite la regola di ragionamento detta *modus tollens*:

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

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, si può scrivere: $KB_1 \models \neg c$.

Inoltre, valendo le seguenti equivalenze:

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

dimostrare che una *congiunzione di atomi* sia *falsa* in tutti i modelli di KB equivale a 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: $KB_2 \models \neg c \vee \neg e$.

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 la procedura $\vdash$ è *corretta e completa* allora KB inconsistente sse insoddisfacibile.

Una base di conoscenza di clausole definite è *sempre* soddisfacibile, per esempio basta considerare l'interpretazione che assegna il valore $true$ a tutti gli atomi. Ciò non vale per basi di clausole di Horn.

Esempio — La semplice base di conoscenza

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

non è soddisfacibile: nessuna interpretazione soddisfa entrambe le clausole, non possono essere entrambe vere in un'interpretazione.

Per provare l'inconsistenza si può usare una procedura TD o BU per dimostrare la contraddizione, rispondendo alla query ask $false$.

6.2 Assumibili e Conflitti

Le *contraddizioni* sono utili a individuare combinazioni di *assunzioni incompatibili*, ossia supposizioni fatte (per mancanza di conoscenza) che possono dimostrarsi false. Ad esempio, nella *diagnostica* di un sistema, il funzionamento *normale* delle componenti potrebbe risultare inconsistente rispetto a nuove osservazioni (aggiuntive rispetto alla base di conoscenza). In caso di malfunzionamento serve una diagnosi del guasto.

Si definisce **assumibile** un atomo che può essere assunto (come vero) in una *dimostrazione per contraddizione*. In diagnostica, sarà utile a determinare una *disgiunzione di negazioni* di assumibili, provando la contraddizione a partire da una base di conoscenza e una combinazione di assumibili.

Si dice **conflitto** di KB l'insieme di assumibili $C = \{c_1, \ldots, c_r\}$ tale che $KB \cup C \models false$. Si tratta di una **risposta** per una dimostrazione per contraddizione del tipo: $KB \models \neg c_1 \vee \cdots \vee \neg c_r$. Un conflitto è **minimale** se nessun suo sottoinsieme costituisce, a sua volta, 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 $\{c, e\}$ sono conflitti minimali;
- $\{c, d, e, h\}$ è un conflitto non minimale.

6.3 Diagnosi Basata sulla Consistenza

Lo *scopo* della *diagnosi* è quello di determinare possibili guasti in base a un *modello del sistema*, una base di conoscenza, e a *osservazioni* aggiuntive sul sistema stesso.

In sostanza si fanno assunzioni sul funzionamento *normale*, considerando quindi *assumibili* in assenza di guasti. Si derivano (per contraddizione) i componenti *anormali*, ossia i *conflitti*

(inconsistenza), la *anomalie* nel sistema.

Esempio — Si riconsideri la base di conoscenza precedente (sistema per la domotica).

Si aggiungono:

- *assunzioni* di normalità tramite *assumibili*: atomi $ok*$ per ogni componente (dev'essere in buono stato per funzionare);
- *osservazioni* fornite dall'utente che specificano le posizioni effettive (up / $down$) dei deviatori, lo stato delle luci (lit / $dark$),...;
- *vincoli d'integrità* utili a definire stati reciprocamente incompatibili come, ad esempio, che sia possibile un solo stato per i punti-luce: $false \leftarrow dark\_l_1 \wedge lit\_l_1$.

KB risultante:

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

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

Tutto considerato, ci sono due conflitti minimali:

1. $\{ok\_cb_1, ok\_s_1, ok\_s_2, ok\_l_1\}$
2. $\{ok\_cb_1, ok\_s_3, ok\_l_2\}$

Pertanto:

1. $KB \models \neg ok\_cb_1 \vee \neg ok\_s_1 \vee \neg ok\_s_2 \vee \neg ok\_l_1$
   quindi almeno un componente fra cb_1, s_1, s_2, l_1 ha un guasto;

2. $KB \models \neg ok\_cb_1 \vee \neg ok\_s_3 \vee \neg ok\_l_2$
   ossia almeno uno tra cb_1, s_3, l_2 non funziona.

Dato l'insieme dei conflitti, l'utente può diagnosticare il problema: ciò è anche utile a farsi un'idea della numerosità dei possibili guasti.

Nella **diagnosi basata su consistenza** (CBD) dato un insieme di conflitti, cioè un insieme di assumibili con almeno un elemento in ogni conflitto, si costruisce una *diagnosi* che sarà **minimale** se nessun suo sotto-insieme è una diagnosi. Intuitivamente, una sola delle diagnosi minimali è quella giusta: quella contenente solo conflitti falsi nell'interpretazione intesa.

Esempio — Nell'esempio precedente, vi sono 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*) che può essere trasformata in una equivalente ma in **forma normale disgiuntiva** (*DNF*), ossia in una disgiunzione di congiunzioni:

$$\neg ok\_cb_1 \vee (\neg ok\_s_1 \wedge \neg ok\_s_3) \vee (\neg ok\_s_1 \wedge \neg ok\_l_2) \vee (\neg ok\_s_2 \wedge \neg ok\_s_3) \vee (\neg ok\_s_2 \wedge \neg ok\_l_2) \vee (\neg ok\_l_1 \wedge \neg ok\_s_3) \vee (\neg ok\_l_1 \wedge \neg ok\_l_2)$$

cioè cb_1 è guasto o ci sono sei possibili malfunzionamenti di coppie di componenti.

Le proposizioni corrispondono 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 di esse dev'essere causa del malfunzionamento.

6.4 Ragionamento con Assunzioni e Clausole di Horn

L'*obiettivo* è la ricerca dei *conflitti* in basi di conoscenza di clausole di Horn attraverso dimostrazioni per contraddizione dei due tipi considerati in precedenza.

6.4.1 Implementazione Bottom–Up

Si estende l'algoritmo bottom-up per clausole definite:

- le conclusioni sono *coppie* $\langle a, A \rangle$, con A insieme di assumibili che implicano l'atomo a rispetto a KB;
- nell'inizializzazione, l'insieme di conclusioni sarà $C \leftarrow \{\langle a, \{a\}\rangle \mid a \text{ assumibile}\}$;

- si usano poi le clausole in KB 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 basterà aggiungere $\langle h, \{\} \rangle$).

---

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/insieme di assumibili

  $C \leftarrow \{\langle a, \{a\} \rangle \mid a \in As\}$
  repeat
    selezionare la clausola $h \leftarrow b_1 \wedge \cdots \wedge b_m \in KB$ 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 $\{A \mid \langle false, A \rangle \in C\}$

Esempio — 5.23 in [PM23]

6.4.2 Implementazione Top–Down

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

- la query da provare è data dall'atomo speciale $false$;
- nella dimostrazione, gli atomi assumibili non vanno dimostrati ma semplicemente raccolti via via essendo assunti come veri.

Esempio — 5.24 in [PM23]

---

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 $false$)

  $G \leftarrow \{false\}$
  repeat
    selezionare un atomo $a \in G$ tale che $a \notin As$
    scegliere una clausola $a \leftarrow B$ di KB con a come testa
    $G \leftarrow (G \setminus \{a\}) \cup B$
  until $G \subseteq As$
  return G

Si osservi che l'algoritmo contiene diverse scelte *non deterministiche* in corrispondenza delle quali si potranno trovare diversi conflitti: in mancanza di scelte, l'algoritmo fallisce.

7 Assunzione di Conoscenza Completa

Nella semantica delle *basi di dati* quello che non è derivabile viene considerato falso: la conoscenza su un dominio viene considerata *completa*. In *Logica* la mancanza di conoscenza *non consente* di trarre conclusioni dal fallimento delle dimostrazioni.

Esempio — diagnosi del sistema elettrico:

- il sistema potrebbe richiedere di specificare *esplicitamente* deviatori up e fusibili non funzionanti (*broken*) *assumendo* per gli altri che siano in condizioni *normali*, ad esempio come *default*: $down\_s_i$ per i deviatori e ok_cb_j per i fusibili indicati con i e j, rispettivamente.

Assumendo la **completezza della conoscenza**, le clausole con uno stesso atomo come testa coprono *tutti e soli* i casi in cui esso è vero. Tale assunzione si dice anche **closed-world assumption** (CWA). In base ad essa, se non si riesce a dimostrare che a sia vero (nei modelli della base), si conclude che è falso, ossia che la sua negazione $\neg a$ è vera. Si assume quindi che tutto quello che è rilevante sia stato asserito *esplicitamente* nella base di conoscenza.

In Logica normalmente non si può assumere che una base di conoscenza possa definire tutta la conoscenza su un dato dominio nella sua completezza, si parla pertanto di (tacita) **open-world assumption** (OWA). Quindi, non potendosi trarre conclusioni dalla mancanza di conoscenza, nel caso precedente ($KB \nvdash a$), la base di conoscenza di clausole definite *non può* avere $\neg a$ come conseguenza logica.

Completare la Base di Conoscenza

Per codificare l'assunzione di conoscenza completa negli assiomi della base di conoscenza essa dev'essere trasformata ossia *completata*.

Dato l'atomo a e tutte le clausole della base di conoscenza che lo definiscono, del tipo:

$$\begin{aligned} a &\leftarrow b_1. \\ &\vdots \\ a &\leftarrow b_n. \end{aligned}$$

dove b_i (body) è una congiunzione di atomi oppure $true$ (per clausole atomiche). In mancanza di clausole per a si considera: $a \leftarrow false$.

Tali clausole possono essere riassunte da un'unica proposizione:

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

(si noti che non è una clausola).

Se a è vero (nei modelli di KB) allora, per la CWA, dovrà *necessariamente* essere vero uno dei corpi b_i quindi si può aggiungere:

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

Il significato delle clausole per l'atomo a sotto CWA è la congiunzione delle due proposizioni, ossia l'*equivalenza*:

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

detta *completamento di Clark* delle clausole per a. Si noti che se non ci sono regole per a, il completamento sarà $a \Leftrightarrow false$.

Il **completamento di Clark** della base di conoscenza ricomprende i completamenti di tutti gli atomi della base.

Esempio — Si consideri la base di conoscenza 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 $down\_s_2$.

Completamento della base di conoscenza:

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

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

Negazione come Fallimento

Un **letterale** è un atomo o la negazione di un atomo. Con il completamento si possono dimostrare *negazioni* di proposizioni, quindi la negazione può essere ammessa anche nel corpo delle clausole. Si considereranno clausole definite *estese* contenenti nel corpo letterali (e non atomi). La *negazione sotto assunzione di conoscenza completa*, detta anche **negation as failure** (NaF), verrà denotata con $\sim a$ per distinguerla dalla negazione classica. Sotto CWA, il corpo g è una conseguenza di KB sse $KB' \models g$, dove KB' è il completamento di KB. Si noti come $\sim a$ nel corpo di una clausola diventi $\neg a$ nel completamento.

Esempio — Si consideri l'assiomatizzazione precedente:

Si semplifica la rappresentazione chiedendo all'utente di indicare solo i deviatori up, gli altri saranno $down$ per default. Analogamente, si potrebbe specificare che i fusibili sono *funzionanti* a meno che non risultino *rotti*

- Regole da aggiungere:
  - $down\_s_1 \leftarrow \sim up\_s_1$.
  - $down\_s_2 \leftarrow \sim up\_s_2$.
  - $down\_s_3 \leftarrow \sim up\_s_3$.
  - $ok\_cb_1 \leftarrow \sim broken\_cb_1$.
  - $ok\_cb_2 \leftarrow \sim broken\_cb_2$.
- Per rappresentare lo stato della figura, l'utente specifica:

    - up_s_2.
    - up_s_3.
- Il sistema può inferire che s_1 dev'essere in posizione $down$ e i due fusibili non sono guasti.
- Completamento:
    - $down\_s_1 \Leftrightarrow \neg up\_s_1$.
    - $down\_s_2 \Leftrightarrow \neg up\_s_2$.
    - $down\_s_3 \Leftrightarrow \neg up\_s_3$.
    - $ok\_cb_1 \Leftrightarrow \neg broken\_cb_1$.
    - $ok\_cb_2 \Leftrightarrow \neg broken\_cb_2$.
    - $up\_s_1 \Leftrightarrow false$.
    - $up\_s_2 \Leftrightarrow true$.
    - $up\_s_3 \Leftrightarrow true$.
    - $broken\_cb_1 \Leftrightarrow false$.
    - $broken\_cb_2 \Leftrightarrow false$.

Con la NAF, le basi di conoscenza non acicliche diventano *problematiche* dal punto di vista semantico. Ad esempio, data la base non aciclica:

- $a \leftarrow \sim b$.
- $b \leftarrow \sim a$.

il completamento equivalente a $a \Leftrightarrow \neg b$, indica che a e b hanno valori di verità opposti: solo uno può essere vero.

Altro esempio di base di conoscenza non aciclica è una con l'assioma:

- $a \leftarrow \sim a$.

con completamento:

- $a \Leftrightarrow \neg a$

che risulta *inconsistente* dal punto di vista logico.

Il completamento di una *base aciclica* è sempre *consistente* e prevede *un solo* valore di verità per ogni atomo.

7.1 Ragionamento Non Monotono

In una *logica* **monotòna** ogni proposizione derivabile da una base di conoscenza rimane derivabile dopo l'aggiunta di altre proposizioni: aggiungendo assiomi *non* si riduce l'insieme delle proposizioni derivabili. Ad esempio si consideri la *Logica delle clausole definite*. In una *Logica* **non monotona** alcune conclusioni possono essere invalidate con l'aggiunta di altri assiomi. Ad esempio si consideri la *Logica delle clausole definite* con l'aggiunta della *NAF*.

Default ed Eccezioni

Una logica non monotona utile a rappresentare casi predefiniti. Si dice **default** una regola che resta valida fintantoché non si verifichi un'*eccezione*. Ad esempio per asserire che *normalmente* b è vera se c è vera si può definire la regola:

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

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, in quanto l'aggiunta di ab_a *inibisce* la possibilità di concludere b. Le regole che implicano ab_a possono inibire il default, sotto le condizioni del corpo della regola. Ad esempio, nella maggior parte dei casi gli uccelli (c) sono volatili (b), ma i pinguini fanno eccezione (ab_a).

7.2 Procedure di Dimostrazione per la NAF

Procedura Bottom–Up + NAF

La procedura BU per clausole definite va modificata: quando p *fallisce* va aggiunto il letterale $\sim p$ all'insieme C delle conseguenze derivate.

Si ha una definizione ricorsiva del **fallimento**: p fallisce se fallisce il corpo di *ogni clausola* che abbia p come testa; un corpo fallisce se *almeno uno* dei suoi *letterali* fallisce; un atomo b_i fallisce se si può derivare $\sim b_i$ e la negazione $\sim b_i$ fallisce se si può derivare b_i.

```
procedure Prove_NAF_BU(KB)

    Input
        KB: insieme di clausole che possono includere NAF
    Output
        insieme di letterali che seguono dal completamento di KB
    Local
        C insieme di letterali

    C ← {}
    repeat
        either
            selezionare (h ← b_1 ∧ ··· ∧ b_m) ∈ KB tale che: h ∉ C e ∀i: b_i ∈ C
            C ← C ∪ {h}
        or
            selezionare h con ∼h ∉ C tale che:
                per ogni (h ← b_1 ∧ ··· ∧ b_m) ∈ KB: ∃i: ∼b_i ∈ C oppure b_i = ∼g e
                g ∈ C
            C ← C ∪ {∼h}
    until non ci sono altre selezioni possibili
```

Osservazione — Sono ricompresi i casi di *corpo vuoto* e *atomo non definito*

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. $s \leftarrow w$.

Una possibile sequenza di letterali aggiunti a C è la seguente:

- $C = \{t\}$
  - 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

La procedura TD è analoga a quella delle clausole definite, ma procede per NAF. È non-deterministica e, come tale, si può implementare 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 base di conoscenza precedente e la query ask p:

- $G = \{p\}$ — inizializzazione
- $G = \{q, \sim r\}$ — sostituzione con il corpo di 1.;
- $G = \{\sim s, \sim r\}$ — sostituendo q con il corpo della 3.;
- $G = \{\sim r\}$ — selezionato ed eliminato $\sim s$ la dim. di s *fallisce*;
- $G = \{\}$ — selezionato $\sim r$, la dim. di r *fallisce* poiché, usando la 4., si deve provare il sotto-goal $\sim t$, si tenta quindi di dimostrare t; data 5., tale dim. ha *successo* immediato, e quella di $\sim t$ *fallisce*. non ci sono altre regole per r. da cui il successo di $\sim r$ che può essere eliminato.

Essendo G vuoto l'output sarà yes.

Quanto detto vale solo nel caso di *fallimento finito*: nessuna conclusione è possibile in caso di divergenza. Ad esempio, data una base di conoscenza con la sola regola $p \leftarrow p$ e la query ask p, l'algoritmo non converge: il completamento è $p \Leftrightarrow p$ ma, pur potendosi accertare l'indimostrabilità di p, una procedura *corretta* non può concludere $\sim p$ poiché non segue logicamente dal completamento.

8 Abduzione

L'**Abduzione** (che si deve a Peirce) è una forma di ragionamento nella quale si fanno *ipotesi* ossia *assunzioni* al fine di *spiegare* osservazioni. Le ipotesi da farsi *possono* implicare le

osservazioni fatte: se verificate, esse giustificano quanto osservato ma non si devono introdurre *contraddizioni* poiché queste giustificherebbero qualsiasi conclusione (si ricordi la massima latina "*ex falso quodlibet*"). Ad esempio se si osserva che qualche punto-luce non sta funzionando, si fanno ipotesi per spiegare le ragioni di quanto sta succedendo.

L'abduzione si differenzia dalla *deduzione* la quale mira alle conseguenze logiche di un set di assiomi e dall'*induzione* che comporta l'inferenza di relazioni generali da casi particolari (esempi).

Scenari e Spiegazioni

Possiamo schematizzare questa forma di ragionamento come segue:

Dati:

- una base KB di Horn;
- un insieme A di atomi utili a costruire ipotesi, detti *assumibili* o anche *abducibili*,

Trovare: spiegazioni per delle osservazioni.

Si dice **scenario** di $\langle KB, A\rangle$ un insieme di abducibili $H \subseteq A$ tale che $KB \cup H$ sia soddisfacibile (ossia ammetta modelli). In tal caso deve esistere un modello in cui tutti gli elementi di KB e di H sono veri, quindi nessun sottoinsieme di H dev'essere un conflitto per KB.

Una **spiegazione** (*ipotesi*) della proposizione g da $\langle KB, A\rangle$ è uno scenario $H \subseteq A$ che, assieme a KB, abbia g come conseguenza, quindi:

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

Essa è **minimale** se H non contiene altre spiegazioni: per ogni altra H' di g da $\langle KB, A\rangle$, deve risultare $H' \not\subset H$.

Esempio — Si consideri una base di conoscenza 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).

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 base di conoscenza:

- $alarm \leftarrow tampering.$

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

Se si osserva $alarm$, spiegazioni minimali sono: $\{tampering\}$ (manomissione) e $\{fire\}$ (incendio).

Se si osservano $alarm \wedge smoke$, la spiegazione minimale è solo $\{fire\}$. Difatti avendo osservato $smoke$ non c'è bisogno di ipotizzare $tampering$ per spiegare $alarm$, è già spiegato da $fire$.

Diagnosi Abduttiva

La *diagnosi abduttiva* (AbD) formula ipotesi su problemi riscontrati e funzionamento normale.

Confrontandola con quella basata su consistenza (CbD):

- al livello di *rappresentazione*, la CbD mira *solo* al comportamento normale quindi le ipotesi saranno assunzioni sul comportamento normale, mentre con la AbD si modella anche il comportamento anomalo, si devono prevedere assumibili anche per ciascun guasto (anomalia);
- per quanto riguarda le *osservazioni*, la CbD le *aggiunge* alla base di conoscenza per poi dimostrare $false$ mentre la AbD le considera come goal da *spiegare* attraverso ipotesi;
- in termini di *qualità*, la AbD risulta più dettagliata, le diagnosi sono migliori: si distinguono base di conoscenza e assunzioni consentono di *dimostrare* le osservazioni, anche casi per cui non sia definibile la normalità.

Ragionamento Abduttivo: Procedure

Il ragionamento abduttivo è basato su *assunzioni* e si realizza mediante le procedure su *clausole di Horn*:

- con la procedura BU si calcolano spiegazioni minimali per ogni atomo, essa è ulteriormente specializzabile con il pruning delle spiegazioni *dominate*;
- con la procedura TD si trovano spiegazioni di un g generando i conflitti, ma mirando a dimostrare g anziché $false$: la spiegazione minimale di g sarà un insieme minimale di assumibili raccolti nella prova.

Riferimenti Bibliografici

[**CCM14**] I.M. Copi, C. Cohen, V. Rodych: *Introduction to Logic*. Routledge. 15th ed. 2019
[**GHR94**] D.M. Gabbay, C.J. Hogger, J.A. Robinson: *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford University Press. 1993
[**PM23**] D. Poole, A. Mackworth: *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press. 3a ed. 2023 (Ch.5)
[**PMG86**] D. Poole, A. Mackworth, R. Goebel: *Computational Intelligence: A Logical Approach*. Oxford University Press. 1986
[**RN20**] S.J. Russell, P. Norvig: *Artificial Intelligence*. Pearson. 4th Ed. 2020
[**Sow99**] J. Sowa: *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole/Cengage. 1999

Link

[**Abduzione**] it.wikipedia
[**Backward_chaining**] it.wikipedia

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

Note

[1] aggettivo inventato, cfr. "*Il Lonfo*" di F. Maraini
https://www.treccani.it/magazine/lingua_italiana/speciali/nonsensi/4.html

[2] Sulle nozioni di *scelta* e *selezione* si veda lo specchietto nel libro di testo.