5. Rappresentazione e Ragionamento Relazionale

Dispensa ICon

versione: 04/11/2024, 18:01

Struttura Relazionale · Simboli e Semantica · Calcolo dei Predicati · Datalog: Linguaggio Relazionale a Regole · Sostituzioni e Dimostrazioni · Simboli di Funzione e Strutture Dati · Uguaglianza · Assunzione di Conoscenza Completa

1 Struttura Relazionale

Nel rappresentare la *struttura* del mondo, è utile ragionare in termini di individui e relazioni.

Gli **individui** sono entità, cose, oggetti del dominio, e potranno essere *concreti* (ad es. persone, edifici), *immaginari* (ad es. unicorni, programmi che superino il test di Turing), o anche *concetti astratti*, come i processi (ad esempio la lettura di un libro o andare in vacanza) o altri concetti (come il denaro, i corsi di studio, gli istanti di tempo, ecc...).

Le **relazioni**, in generale, specificano verità riguardanti gli individui. Possono essere *proprietà*, vere o false, dei singoli individui oppure anche *proposizioni* che possono essere vere o false indipendentemente da specifici individui, e anche *associazioni* che possono intercorrere tra più individui.

Con questa rappresentazione il ragionamento su più istanze potrà risultare più semplice perché più generale.

> **Esempio** — Nella rappresentazione proposizionale della domotica, si consideravano atomi senza una vera *struttura* interna, come up_s_2, up_s_3 e ok_s_2.
>
> - la rappresentazione non risulta sufficiente a esprimere, ad esempio, che:
>   - up_s_2 e up_s_3 riguardano la *stessa* proprietà ma individui *diversi*;
>   - up_s_2 e ok_s_2 riguardano proprietà *diverse* dello *stesso* individuo.
>
> In *alternativa* si possono rappresentare esplicitamente e separatamente i singoli individui (deviatori) s_1, s_2, s_3 e le rispettive proprietà up e ok:
>
> - ad esempio $up(s_2)$ rappresenta l'enunciato "s_2 si trova in posizione up": sapendo cosa rappresentino up e s_1, non serve una definizione a parte per $up(s_1)$
>
> Inoltre si potrà usare la relazione binaria $connected\_to$ per collegare individui, indipendentemente dalle loro altre proprietà, come in $connected\_to(w_1, s_1)$.

I *vantaggi* della nuova rappresentazione sono molteplici:

- essa è più *naturale* poiché sfrutta la struttura delle feature come proprietà degli individui;
- è utile anche nel caso di *domini sconosciuti* per i quali non vi sia nessuna conoscenza sugli individui, sul loro numero e delle loro caratteristiche: si deve interagire con l'ambiente per apprendere la rappresentazione;

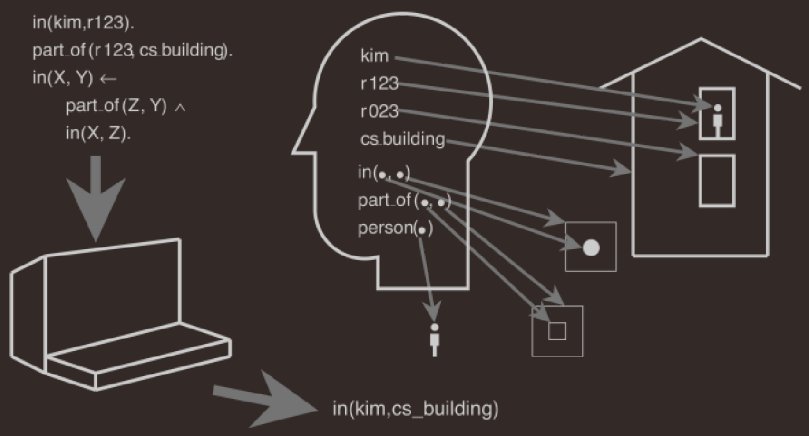
- consente forme *ragionamento* più *generali* e indipendenti dai singoli individui, con obiettivi quali la derivazione di *conclusioni* che valgono *per tutti* gli individui senza conoscerli necessariamente, la scoperta dell'*esistenza* di nuovi individui e loro proprietà, a prescindere dagli altri e, inoltre, la possibilità di rispondere a *query* che riguardano intere popolazioni e non singoli individui;
- rende possibile il trattamento dell'*esistenza incerta* di individui: non si conoscono completamente individui attuali e futuri, nonché le loro proprietà;
- rende possibile il ragionamento su un *infinito* numero di individui (e loro proprietà). Ad esempio si può ragionare su frasi (individui), il cui numero è potenzialmente infinito, pur considerandone un numero finito, quelle utili al compito da svolgere.

2 Simboli e Semantica

La *Logica* può risultare molto utile nella progettazione di basi di conoscenza. I progettisti devono caratterizzare particolare *mondo* formalizzando la cosiddetta **interpretazione intesa**: scegliendo *significati* per i simboli in base a tale interpretazione e descrivendo ciò che dev'essere vero attraverso una base di conoscenza fatta di *clausole*. Chiunque conosca il significato dei simboli può interpretare, rispetto a tale modello, le *conseguenze logiche* derivate dalla base: ciò che è vero in tutti i modelli della base sarà vero, in particolare, nell'interpretazione intesa.

Estendendo il linguaggio delle clausole proposizionali si possono definire atomi con una *struttura interna* descritta in termini di relazioni e individui.

Esempio — Idea della semantica della rappresentazione:



- Nella progettazione della base di conoscenza si attribuisce un *significato* ai simboli: l'utente sa che a cosa si riferiscano nel dominio e il significato delle *frasi* nel linguaggio di rappresentazione (la base di conoscenza) fornite alla macchina, sa formulare *query* usando i simboli nel significato loro attribuito e sa interpretare le risposte rispetto al mondo di riferimento. La macchina è in grado di calcolare le *risposte* alle query pur non comprendendo tale significato.

Una **concettualizzazione** è un'associazione fra simboli nella mente e gli individui e le relazioni da essi denotati. Per il momento sarà trattata solo informalmente o come definita nella mente dell'utente, successivamente si introdurrà la nozione formale di *ontologia*.

Vi è una separazione tra l'aspetto *semantico* e quello *computazionale*: la *correttezza* della base di conoscenza e delle risposte alle query è definita dalla *semantica* indipendentemente dall'*algoritmo* di ragionamento, che si può ottimizzare a patto di rimanere semanticamente corretto.

3 Calcolo dei Predicati

Il **calcolo dei predicati**, o *logica dei predicati* o *logica del primo ordine* estende il calcolo proposizionale in due modi: gli atomi sono dotati di una loro *struttura* con la possibilità di includere variabili; inoltre, le variabili logiche sono *quantificate*.

> Nel seguito si seguiranno le convenzioni del PROLOG

Sintassi

Per definire la sintassi, occorre specificare i seguenti nozioni:

- **simbolo**: stringa alfanumerica comprendente anche _
- **variabile logica**: simbolo che inizia con maiuscola o _
  - ad es. $Stanza$, $\_B_4$, $Lista$ e The_Dude
- **costante**: simbolo che inizia con una minuscola o numero o stringa (tra apici)
- simbolo di **predicato**: simbolo che inizia con una minuscola (costanti e predicati distinguibili in base al *contesto*)
  - ad es. $nico$, $r123$, f_1, $insegna\_a$ e $aula\_IV$, ma 2018 solo costante
- **termine**: variabile o costante
  - ad es. X, $nico$, dib, $uniBA$ o $Docenti$
- **atomo** (*simbolo atomico*): forma p oppure $p(t_1, \ldots, t_n)$ con p simbolo di predicato e ogni t_i è un termine, i-esimo *argomento* del predicato
  - esempi: $aperto$, $giallo(C)$, $insegna(nicoF, 63507)$, $in(vitoR, 522)$, $padre(alanT, Y)$, $inserire(Dessert, menuT)$. In $padre(alanT, Y)$, dal contesto si desume che $padre$ è un simbolo di predicato mentre $alanT$ è una costante.

Una **formula logica** viene definita costruttivamente a partire dagli atomi, usando connettivi logici e quantificatori. Date la variabile X e la formula logica p (in cui possa occorrere X):

- $\forall X\, p$
  - da leggersi *"per ogni X, p"*, si dice che X è *quantificata* **universalmente**;
- $\exists X\, p$
  - da leggersi *"esiste un X tale che p"*, e si dice che X è *quantificata* **esistenzialmente**.

Esempio — possibili formule logiche:

- $\forall G\ (grandfather(sam, G) \leftarrow \exists P\ (father(sam, P) \land parent(P, G)))$.
- $in(X, Y) \leftarrow partof(Z, Y) \land in(X, Z)$.
- $slithy(toves) \leftarrow \forall Raths\ (mimsy \land borogroves \land outgrabe(mome, Raths))$.

dal contesto si capisce che:

- sam, $toves$ e $mome$ sono costanti;
- $grandfather$, $father$, $parent$, in, $part\_of$, $slithy$, $mimsy$, $borogroves$ e $outgrabe$ sono simboli predicativi;
- G, P, X, Y, Z e $Raths$ sono variabili.

Le prime due formule sono intuitive, non serve una specifica formale del significato, ma non per questo sono più significative: il significato verrà dato dalla semantica.

3.1 Semantica delle Formule Logiche di Base

Un'*espressione* **ground**, o di (livello-)*base*, non contiene variabili. Essa è un atomo o, più in generale, una formula che si riferisce solo a precisi individui.

Un'**interpretazione** delle espressioni *ground* è definita da una tripla

$$I = \langle D, \phi, \pi \rangle$$

dove

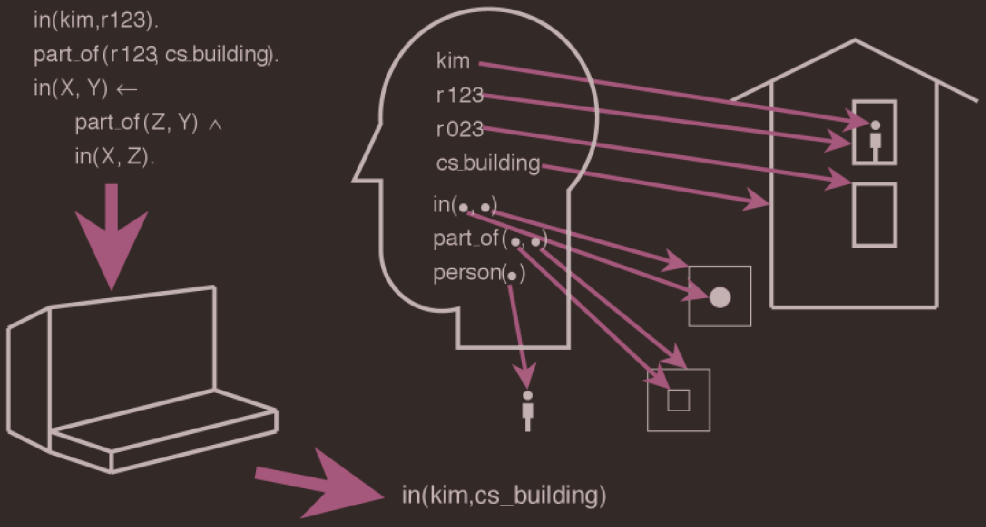
- il **dominio** D è l'insieme non vuoto di **individui**;
- la funzione ϕ: $c \mapsto \phi(c) \in D$ assegna a ogni costante c un individuo di D;
- la funzione π: $p \mapsto \pi(p)$ assegna a ogni simbolo di *predicato* n-ario p una funzione $\pi(p)$: $D^n \to \{true, false\}$ che *per ogni* n-pla di individui, $\pi(p)$ specifica se la relazione p sia vera/falsa;
  - π è una *funzione di ordine superiore* in quanto associa un nome di predicato a una funzione (predicativa);
  - se p non ha argomenti allora $\pi(p)$ sarà una (funzione) costante: $true$ o $false$ (semantica del calcolo proposizionale).

Esempio — Scena da descrivere: tavolo con forbici, modellino e matita

- le *costanti* potrebbero essere $plane$, $pencil$ e $model$
  - si noti l'assenza intenzionale di un nome più preciso (per gli umani) come $scissors$;
- i simboli di *predicato* potrebbero essere $noisy$ (unario) e $left\_of$ (binario);
- una particolare *interpretazione* per la descrizione della scena potrebbe essere costituita dalla tripla $\langle D, \phi, \pi \rangle$ che segue:
  - $D = \{✂, ✈, ✎\}$ (contiene individui: oggetti del mondo reale)
  - $\phi(plane) = ✈$, $\phi(pencil) = ✎$, $\phi(model) = ✈$
    - si noti che i nomi $plane$ e $model$ si riferiscono allo *stesso individuo* (✈) quindi gli *stessi enunciati* che li riguardano risulteranno *veri*;
  - $\pi(noisy)$: arg è rumoroso, definito da
    $\langle ✂ \rangle \mapsto false$, $\langle ✈ \rangle \mapsto true$, $\langle ✎ \rangle \mapsto false$;
    e $\pi(left\_of)$: arg_1 è a sinistra di arg_2, definito da
    $\langle ✂, ✂ \rangle \mapsto false$, $\langle ✂, ✈ \rangle \mapsto true$, $\langle ✂, ✎ \rangle \mapsto true$,
    $\langle ✈, ✂ \rangle \mapsto false$, $\langle ✈, ✈ \rangle \mapsto false$ $\langle ✈, ✎ \rangle \mapsto true$,
    $\langle ✎, ✂ \rangle \mapsto false$, $\langle ✎, ✈ \rangle \mapsto false$, $\langle ✎, ✎ \rangle \mapsto false$

Tornando al caso precedente:

Esempio — Macchina, mente e interpretazione della scena (*a destra*)



nella/dalla macchina — nella mente — scena da interpretare

Nell'interpretazione I, ogni termine ground c denota un individuo $\phi(c)$.

Semantica degli atomi ground: nell'interpretazione I, l'atomo ground $p(t_1, \ldots, t_n)$ è **vero** se $\pi(p)(\langle t'_1, \ldots, t'_n \rangle) = true$ (dove $t'_k = \phi(t_k) \in D$ è l'individuo denotato dal termine t_k) ed è **falso**, altrimenti.

Esempio — Nell'interpretazione della *figura* precedente:

- $in(kim, r123)$ è vero: la persona denotata da kim è proprio nella stanza denotata da $r123$;
- $person(kim)$ e $part\_of(r123, cs\_building)$ sono veri;
- $in(cs\_building, r123)$ e $person(r123)$ sono falsi.

3.2 Semantica: Interpretazione delle Variabili

Un'**assegnazione** *di variabili* ρ è una funzione dall'insieme delle variabili a D che associa un elemento del dominio a ogni variabile.

Data l'interpretazione $I = \langle D, \phi, \pi \rangle$ e l'assegnazione ρ, ogni *termine* denota un individuo in D interpretato da ϕ se costante, da ρ se variabile. La semantica degli *atomi* è definita come visto in precedenza, ottenuti associando individui ai termini.

Semantica della Quantificazione

Una formula *universalmente quantificata* $\forall X\, p$ è **vera** in un'interpretazione se p è vera per ogni assegnazione di X, e sarà *falsa* se c'è un'assegnazione a X per la quale la formula risulti falsa: la formula p è costituisce l'**ambito** (o *scope*) di X.

Una formula esistenzialmente quantificata $\exists X\, p$ è **vera** in un'interpretazione sse p è vera per qualche assegnazione di X: l'individuo assegnato può dipendere da variabili universalmente quantificate nel cui ambito compaia $\exists X\, p$.

- in $\forall Y(\exists X\, p)$, X può dipendere da Y
  e.g. *per ogni numero* Y *c'è un* X *tale che* $X = Y + 1$ (vero);
- in $\exists X(\forall Y\, p)$, X non può dipendere da Y poiché X non è nell'ambito di Y
  e.g. *esiste un numero* X *tale che per ogni numero* Y *risulti* $X = Y + 1$ (falso).

Semantica delle Formule e Conseguenza Logica

Una **formula** con variabili è **vera** in I sse per qualsiasi assegnazione ρ la formula ground ottenuta applicando ρ alle sue variabili risulta vera.

Un formula si dice **aperta** se ha almeno una variabile non quantificata, per cui la sua semantica non può essere determinata.

Esempio — riprendendo l'interpretazione dell'esempio precedente

- $\forall X \forall Y\ part\_of(X, Y) \leftarrow in(X, Y)$. è falsa:
  - assegnando Kim a X e $r123$ a Y: $in(X, Y)$ vera mentre $part\_of(X, Y)$ falsa;
- $\forall X \forall Y \forall Z\ in(X, Y) \leftarrow part\_of(Z, Y) \wedge in(X, Z)$. è vera:
  - essendo $in(X, Y)$ vera per tutte le assegnazioni per le quali $part\_of(Z, Y) \wedge in(X, Z)$ sia vera.

La nozione di modello e la relazione di **conseguenza logica** fra una data base di conoscenza KB e un goal g si estende al caso delle formule logiche del calcolo dei predicati:

$KB \models g$ se g è vera in ogni modello di KB

Esempio — nel caso dell'esempio precedente, un modello di KB con assiomi:

1. $in(kim, r123)$.
2. $part\_of(r123, cs\_building)$.
3. $\forall X \forall Y \forall Z\ in(X, Y) \leftarrow part\_of(Z, Y) \wedge in(X, Z)$.

si ha che:

- $KB \models in(kim, r123)$: essendo $in(kim, r123)$ presente in KB sarà vero in ogni suo modello;
- $KB \not\models in(kim, r023)$: $in(kim, r023)$ è falso nel modello considerato;
- $KB \not\models part\_of(r023, cs\_building)$: $part\_of(r023, cs\_building)$ è vero nel modello dato (ma ci potrebbero essere altri modelli in cui è falso, simili al precedente ma tali che $\pi(part\_of)(\langle \phi(r023), \phi(cs\_building) \rangle) = false$);
- $KB \models in(kim, cs\_building)$: se $in(kim, cs\_building)$ fosse falsa in un modello, esisterebbe un'*istanza* della (3.), ottenuta applicando l'assegnazione in cui $X = kim$ e $Y = r123$ e $Z = cs\_building$, che risulta falsa nel modello
  - impossibile, deve risultare vera nei modelli per tutte le assegnazioni.

3.3 Semantica: la Prospettiva Umana

Estendendo la metodologia introdotta per le basi di conoscenza proposizionali:

1. si seleziona il *dominio* del task ossia il *mondo* da rappresentare, definendo:
   - D insieme di tutti gli *individui* cui si farà riferimento e sui quali ragionare
   - le *relazioni* da rappresentare
     - aspetti specifici del mondo *reale*, ad esempio la struttura dei corsi e studenti presso un'università o un laboratorio in un determinato momento;
     - mondi *immaginari* o *ipotetici*, ad esempio quello di *Alice nel paese delle meraviglie*, o lo stato di un'apparecchiatura elettrica;
     - mondo *astratti*, e.g. numeri e insiemi, contesti finanziari,...
2. si associano le *costanti* del linguaggio a individui del mondo da nominare:
   - a ogni elemento di D si assegna una costante per riferirvisi, ad esempio kim per un particolare docente, $cs322$ per un corso, two per il successore del numero uno e red per il colore delle luci di stop;
3. si associa un simbolo di *predicato* a ogni relazione da rappresentare, associazioni simbolo-significato per costituire l'**interpretazione intesa**:
   - un simbolo (n-ario) denota una funzione $D^n \to \{true, false\}$, basta specificare le n-ple per le quali la relazione sia vera, ad esempio il predicato $teaches$ binario rappresenta una relazione vera tra un individuo (primo argomento) e un corso insegnato (secondo argomento);
   - le relazioni possono avere qualunque arietà (anche nulla), ad es. is_red predicato unario;
4. si definiscono *formule* vere nell'interpretazione intesa: **assiomatizzazione del dominio**
   - **assiomi** del dominio in forma di clausole, ad esempio se la persona denotata da kim insegna il corso denotato da $cs322$, la formula $teaches(kim, cs322)$ sarà vera nell'interpretazione intesa;
5. si possono formulare *query* riguardanti l'interpretazione intesa:
   - risposte del sistema da interpretare nella semantica attribuita ai simboli.

Esempio — Concettualizzazioni diverse di un colore: il rosa

1. colore come simbolo di *predicato* unario: atomo vero se l'individuo dell'argomento è rosa: $pink(i)$
2. colore come *individuo*: $pink$ rappresenta il colore rosa
   - secondo argomento per un predicato binario $color(i, c)$;

3. a basso livello di dettaglio: le diverse sfumature di rosso risultano indistinguibili
    - colore rosa non considerato, ma solo *red*;
4. aumentando il dettaglio: si considera *pink* come termine generale
    - prevedendo anche/invece *coral* o *salmon*.

4 DATALOG: Linguaggio Relazionale a Regole

Il DATALOG estende il linguaggio delle clausole definite proposizionali con la sintassi del calcolo dei predicati.

Una **clausola definita** si scrive nella forma

$$h \leftarrow a_1 \wedge \ldots \wedge a_m$$

e si legge "h se a_1 e ... e a_m", dove:

- l'atomo h è la **testa** della clausola;
- se $m > 0$ si dice **regola** e la congiunzione di atomi $a_1 \wedge \ldots \wedge a_m$ ne costituisce il **corpo**;
- se $m = 0$ *clausola* si chiama clausola **atomica** o **fatto** (e si può omettere la freccia);
- le variabili sono da intendersi implicitamente *universalmente quantificate*, ad esempio $h(X) \leftarrow b(X, Y)$ è da intendersi come $\forall X \forall Y\ (h(X) \leftarrow b(X, Y))$, che è equivalente a $\forall X\ (h(X) \leftarrow \exists Y\ b(X, Y))$.

Esempio — data una relazione in forma di tabella:

Course	Section	Time	Room
cs111	7	830	dp101
cs422	2	1030	cc208
cs502	1	1230	dp202

il predicato corrispondente $scheduled(Course, Section, Time, Room)$ in una base di conoscenza sarà definito dai fatti:

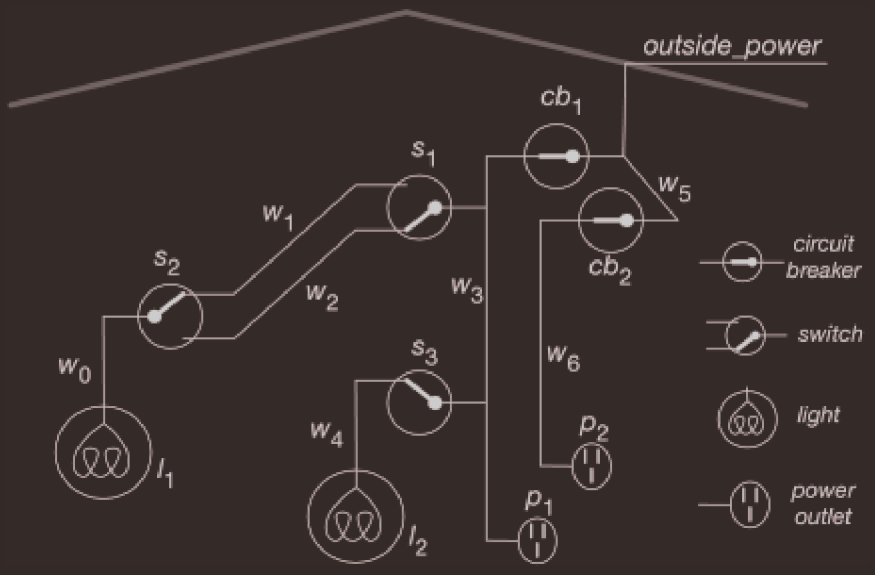
- $scheduled(cs111, 7, 830, dp101)$.
- $scheduled(cs422, 2, 1030, cc208)$.
- $scheduled(cs502, 1, 1230, dp202)$.

Si può considerare anche $enrolled(StudentNum, Course, Section)$, definita attraverso fatti e poi anche $busy(StudentNum, Time)$, ottenuta dal join delle precedenti, proiettato su $StudentNum$ e $Time$:

$$\begin{aligned} busy(StudentNum, Time) \leftarrow \\ \quad enrolled(StudentNum, Course, Section)\ \wedge \\ \quad scheduled(Course, Section, Time, Room). \end{aligned}$$

studente impegnato a un dato orario se iscritto a una sezione d'un corso previsto per quell'ora.

Esempio — diagnosi impianto elettrico: rivisitato con individui e relazioni



1. *individui*: secondo il livello di astrazione scelto
   - ogni deviatore, luce, presa;
   - ogni cavo tra deviatori o tra deviatore e luce;
   - si assume un modello di flusso dell'elettricità in cui l'energia fluisca dall'esterno della casa attraverso i cavi e verso le luci, appropriato dovendo determinare se una luce dev'essere accesa o meno, ma potrebbe non esserlo per altri task
2. *nomi* da dare a ciascun individuo al quale ci si vorrà riferire
   - come in figura, ad es. w_0 è il cavo tra la luce l_1 e il deviatore s_2
3. *relazioni* da rappresentare: predicati con le interpretazioni intese associate
   - $light(L)$ è vero se l'individuo denotato da L è una luce;
   - $lit(L)$ è vero se la luce L è accesa ed emette luce;
   - $live(W)$ è vero se passa corrente attraverso W, ossia se W è in tensione;
   - $up(S)$ è vero se il deviatore S è *su* (acceso);
   - $down(S)$ è vero se il deviatore S è *giù* (spento);
   - $ok(E)$ è vero se E non è guasto; E salvavita o luce;
   - $connected\_to(X, Y)$ è vero se il componente X è connesso a Y in modo che la corrente passi da Y a X.
4. base di *assiomi* per la macchina (che non ne conosce il significato):
   - *regole* generali come $lit(L) \leftarrow light(L) \wedge live(L) \wedge ok(L)$.
   - anche ricorsive: $live(X) \leftarrow connected\_to(X, Y) \wedge live(Y)$., con $live(outside)$.
   - e *fatti* per la sua configurazione d'un impianto specifico: $light(l_1)$, $light(l_2)$, $down(s_1)$, $up(s_2)$, $ok(cb_1)$;
   - inoltre:
     - $connected\_to(w_0, w_1) \leftarrow up(s_2)$.
     - $connected\_to(w_0, w_2) \leftarrow down(s_2)$.
     - $connected\_to(w_1, w_3) \leftarrow up(s_1)$.
     - $connected\_to(w_3, outside) \leftarrow ok(cb_1)$.
5. la macchina saprà quindi rispondere a query su questo impianto domestico in particolare.

Query con Variabili

Una **query** serve a chiedere se un enunciato sia conseguenza logica di una base di conoscenza. Le query *decisionali* sono quelle tipiche della logica proposizionale (cioè di una rappresentazione ground, senza variabili) e hanno come risposta `yes` o `no`. Le query *con variabili* consentono, in aggiunta, di determinare e restituire tuple di *individui* per le quali la query si avvera.

Si dirà **istanza** di una query la formula ottenuta sostituendo termini a variabili: ogni occorrenza di una variabile dev'essere sostituita con uno stesso termine. L'**estrazione della risposta** serve a determinare quali istanze seguano dalla base di conoscenza. Le **risposte** possibili per una query con variabili libere sono:

- *istanze* della query che seguono logicamente dalla base, una per ciascuna distinta tupla di valori assegnabile alle variabili della query, oppure
- no, se nessuna istanza segue logicamente dalla base (ciò non significa che la query sia falsa nell'interpretazione intesa).

Esempio — Dati i seguenti fatti riguardanti i paesi del SudAmerica

- $country(C)$ — vero se C è un paese
  - $country(argentina)$.
  - $country(brazil)$.
  - $country(chile)$.
  - $country(paraguay)$.
  - $country(peru)$.
- $area(C, A)$ — vero se C ha superficie A in Km^2
  - $area(argentina, 2780400)$.
  - $area(brazil, 8515767)$.
  - $area(chile, 756102)$.
  - $area(paraguay, 406756)$.
  - $area(peru, 1285216)$.
- $language(C, L)$ — vero se L lingua principale di C
  - $language(argentina, spanish)$.
  - $language(brazil, portuguese)$.
  - $language(chile, spanish)$.
  - $language(paraguay, spanish)$.
  - $language(peru, spanish)$.
- $borders0(C_1, C_2)$ — vero se C_1 e C_2 confinano e $C_1 \leq C_2$ alfabeticamente
  - $borders0(argentina, brazil)$.
  - $borders0(argentina, chile)$.
  - $borders0(argentina, paraguay)$.
  - $borders0(brazil, paraguay)$.
  - $borders0(brazil, peru)$.
  - $borders0(chile, peru)$.
- $borders(C_1, C_2)$ — vero se C_1 e C_2 confinano
  - $borders(X, Y) \leftarrow borders0(X, Y)$.
  - $borders(X, Y) \leftarrow borders0(Y, X)$.

si possono formulare le seguenti query per le quali si ottengono le risposte riportate:

- ask $language(chile, spanish)$.
  - risposta: yes;
- ask $language(venezuela, spanish)$.
  - no, la query non è conseguenza logica (conoscenza insufficiente) ma non dice che sia falsa in generale;
- ask $language(X, spanish)$.
  - 4 risposte fra cui quella con $X = chile$ per la quale $language(chile, spanish)$ risulta conseguenza logica della base di conoscenza;
- ask $borders0(paraguay, X)$.
  - nessuna risposta;
- ask $borders(paraguay, X)$.
  - risposte $X = argentina$ e $X = brazil$.

- ask $borders(X, Y)$.
  - 12 risposte;
- ask $borders(chile, Y) \wedge area(Y, A) \wedge A > 2000000$.
  - si noti che il predicato $>$ (tipicamente predefinito) ha argomenti numerici ed è qui scritto in forma infissa.

Aggiungendo altri fatti

- $capital(A, B)$ — vero se B capitale di A
  - $capital(argentina, buenos\_aires)$.
  - $capital(chile, santiago)$.
  - $capital(peru, lima)$.
  - $capital(brazil, brasilia)$.
  - $capital(paraguay, asuncion)$.
- $name(E, N)$ — vero se la stringa N è il nome di E
  - $name(buenos\_aires, \text{"Buenos Aires"})$.
  - $name(santiago, \text{"Santiago"})$.
  - $name(lima, \text{"Lima"})$.
  - $name(brasilia, \text{"Brasilia"})$.
  - $name(asuncion, \text{"Asunción"})$.
  - $name(argentina, \text{"Argentina"})$.
  - $name(chile, \text{"Chile"})$.
  - $name(peru, \text{"Peru"})$.
  - $name(brazil, \text{"Brazil"})$.
  - $name(paraguay, \text{"Paraguay"})$.

Una query potrebbe chiedere il nome della capitale del Cile:

- ask $capital(chile, C) \wedge name(C, N)$.

---

5 Sostituzioni e Dimostrazioni

Per implementare il ragionamento su basi di conoscenza e query Datalog si possono generalizzare le procedure BU e TD per basi proposizionali. Si ponga attenzione al fatto che in una clausola Datalog una variabile *libera* indica che tutte le *istanze* della clausola dovranno essere vere, al variare delle possibili assegnazioni. Una dimostrazione potrà anche basarsi su istanze *diverse* di una stessa clausola.

5.1 Istanze e Sostituzioni

Come per le query, un'**istanza** di clausola si ottiene sostituendo variabili con termini uniformemente: ogni occorrenza di una data variabile dev'essere sostituita con lo stesso termine. Una **sostituzione** specifica, per ciascuna variabile, il termine da sostituire:

$$\{V_1/t_1, \ldots, V_n/t_n\}$$

- ogni V_i è una *distinta* variabile e ogni t_i è un termine;
- ogni V_i/t_i si dice **binding** (*associazione*) per la variabile V_i e si dirà in **forma normale** se nessuna V_i occorre in un t_j.

Esempio — Sostituzioni:

- $\{X/Y, Z/chile\}$ è una sostituzione in forma normale;
- $\{X/Y, Z/X\}$ *non* è in forma normale: X occorre sia a sinistra sia a destra di associazioni diverse.

Espressioni

Un'*espressione* è un termine, un atomo o una clausola. Un'**istanza** $e\sigma$ dell'espressione e si ottiene dall'**applicazione** di $\sigma = \{V_1/t_1, \ldots, V_n/t_n\}$ a e: ogni occorrenza di V_i va sostituita con t_i. L'*istanza* si dice **ground** se non contiene variabili.

Esempio — Diverse applicazioni di sostituzioni ad atomi:

- $borders(peru, X)\{X/chile\} = borders(peru, chile)$
- $borders(Y, chile)\{Y/peru\} = borders(peru, chile)$
- $borders(peru, X)\{Y/peru, Z/X\} = borders(peru, X)$
- $p(X, X, Y, Y, Z)\{X/Z, Y/brazil\} = p(Z, Z, brazil, brazil, Z)$

Le sostituzioni si applicano anche alle clausole: data $\sigma = \{X/Y, Z/peru\}$

- $[p(X, Y) \leftarrow q(peru, Z, X, Y, Z)]\,\sigma$
  $= p(Y, Y) \leftarrow q(peru, peru, Y, Y, peru)$

Un **unificatore** delle espressioni e_1 e e_2 è una sostituzione σ tale che

$$e_1\sigma = e_2\sigma$$

i.e. applicata a più espressioni produce un'unica istanza. Quando esiste un tale unificatore si dice che e_1 e e_2 sono *unificabili* o che *si unificano*.

Esempio — unificatori

- $\{X/chile, Y/peru\}$ è un unificatore per $borders(peru, X)$ e $borders(Y, chile)$
  - dalla cui applicazione si ottiene: $borders(peru, chile)$
- $\{X/a, Y/b\}$ unificatore di $t(a, Y, c)$ e $t(X, b, c)$
  - per cui: $t(a, Y, c)\{X/a, Y/b\} = t(X, b, c)\{X/a, Y/b\} = t(a, b, c)$

Diversi unificatori possibili di $p(X, Y)$ e $p(Z, Z)$ sono:

- $\{X/b, Y/b, Z/b\}$, $\{X/c, Y/c, Z/c\}$, $\{X/Z, Y/Z\}$, $\{Y/X, Z/X\}$, ecc.
  - i primi due specificano precisamente i valori sostituiti
  - gli altri due sono *più generali*.

L'**unificatore più generale** (*most general unifier*, MGU) σ di e_1 ed e_2 è l'unificatore tale che, considerato qualunque altro loro unificatore σ', risulta che $e\sigma'$ è un'istanza di $e\sigma$, per ogni espressione e.

Si dirà che e_1 è una **ridenominazione** (*renaming*) di e_2 sse esse sono istanze l'una dell'altra, ossia differiscono solo nei nomi delle variabili. Due espressioni unificabili avranno almeno un MGU e, applicando loro degli MGU, si otterranno renaming reciproci: dati σ e σ' MGU di e_1 ed e_2 si ha che $e_1\sigma$ è un renaming di $e_1\sigma'$ (analogamente per e_2).

Esempio — Due MGU degli atomi $p(X, Y)$ e $p(Z, Z)$ sono $\{X/Z, Y/Z\}$ e $\{Z/X, Y/X\}$.

Applicando entrambi a $p(X, Y)$ si ottengono: $p(X, Y)\{X/Z, Y/Z\} = p(Z, Z)$ e $p(X, Y)\{Z/X, Y/X\} = p(X, X)$, che è un renaming di $p(Z, Z)$.

5.2 Procedura Bottom-up con Variabili

Per estendere la *procedura BU* proposizionale al Datalog, si deve effettuare preventivamente il **grounding** degli assiomi della base di conoscenza ottenendo le *istanze ground* delle clausole, sostituendo uniformemente nelle clausole costanti a variabili. Le costanti da considerare sono quelle che occorrono nella base di conoscenza o nella query. Qualora non ve ne fossero, se ne deve inventare una nuova.

Esempio — data la base di conoscenza con:

- $q(a)$.
- $q(b)$.
- $r(a)$.
- $s(W) \leftarrow r(W)$.
- $p(X, Y) \leftarrow q(X) \wedge s(Y)$.

Tutte le istanze ground saranno:

- $q(a)$.
- $q(b)$.
- $r(a)$.
- $s(a) \leftarrow r(a)$.
- $s(b) \leftarrow r(b)$.
- $p(a, a) \leftarrow q(a) \wedge s(a)$.
- $p(a, b) \leftarrow q(a) \wedge s(b)$.
- $p(b, a) \leftarrow q(b) \wedge s(a)$.
- $p(b, b) \leftarrow q(b) \wedge s(b)$.

Applicando la procedura BU proposizionale si derivano atomi *ground*.

È facile desumere che possibili conseguenze logiche sono: $q(a)$, $q(b)$, $r(a)$, $s(a)$, $p(a, a)$ e $p(b, a)$.

Esempio — data la base di conoscenza con:

- $p(X, Y)$.
- $g \leftarrow p(W, W)$.

e la query ask g, la procedura BU introduce ad hoc la nuova costante c.

Il grounding della base sarà:

- $p(c, c)$.
- $g \leftarrow p(c, c)$.

La BU proposizionale deriva $\{p(c, c), g\}$, per cui la risposta è yes.

Si lascia per **esercizio** la query ask $p(b, d)$.

- *suggerimento*: l'insieme delle istanze ground cambia, dovendo includere le costanti b e d.

Si può dimostrare che la procedura BU, applicata al *grounding* di una base di conoscenza, è **corretta** e **completa**.

La procedura BU è *corretta* (*sound*): ogni istanza di ogni regola è vera in ogni modello. Tale proprietà può essere dimostrata come nel caso proposizionale, considerando l'insieme delle istanze ground delle clausole che sono *vere*, essendo le variabili quantificate universalmente. La procedura *termina* sempre anche nel caso di clausole Datalog essendo *finito* il numero di atomi da rendere ground: a ogni iterata, *un solo* atomo ground viene aggiunto all'insieme delle conseguenze.

La procedura è anche *completa* per *atomi ground* g: se $KB \models g$ allora $KB \vdash g$. Si dimostra costruendo un particolare tipo di modelli, le **interpretazioni di Herbrand** $\langle D, \phi, \pi \rangle$, in cui: D è un *dominio* (simbolico) *fissato*, costituito dalle costanti in KB e g (o se ne introduce una *ad hoc* in caso di mancanza); anche ϕ è *fissato* in quanto ogni costante denota se stessa; infine si definisce π per i predicati tale che sia *vero* ogni atomo che è istanza ground di una relazione derivata dalla procedura (e falsi tutti gli altri). Si dimostra che una tale interpretazione costituisce un *modello* per KB che risulta *minimale*, ossia con il minor numero atomi veri rispetto a ogni altro modello. Se $KB \models g$, con g ground, allora g è vero nel modello minimale, quindi alla fine verrà derivato.

5.3 Unificazione

Dati due termini o atomi, occorre determinare se si unificano e, nel caso, restituire un loro unificatore.

Il seguente algoritmo consente di trovare un eventuale MGU di due atomi o termini e, altrimenti, restituisce $\perp$ per indicare la mancata unificabilità.

```
procedure Unify(t1, t2)
    Inputs
        t1, t2: atomi o termini
    Output
        MGU di t1 e t2 se esiste, altrimenti ⊥
    Local
        E: insieme di uguaglianze
        S: sostituzione
    E ← {t1 = t2}
    S = ∅
    while E ≠ ∅ do
        selezionare e rimuovere α = β da E
        if β non identica ad α then
            if α variabile then
                sostituire α con β ovunque in E e S
                S ← {α/β} ∪ S
            else if β variabile then
                sostituire β con α ovunque in E e S
                S ← {β/α} ∪ S
            else if α è p(α1, ..., αn) e β è p(β1, ..., βn) then
                E ← E ∪ {α1 = β1, ..., αn = βn}
            else return ⊥
    return S
```

L'algoritmo lavora su un insieme E di *uguaglianze* ($\alpha = \beta$) che implicano e sono implicate dall'unificazione e un insieme S di binding che formeranno la *sostituzione* cercata. Se $\alpha/\beta \in S$ allora, per costruzione, α sarà una variabile che non appare altrove in S o in E. Nel caso in cui α e β siano atomi, essi devono avere stesso simbolo di predicato e stessa arietà, altrimenti l'unificazione fallisce.

Esempio — Volendo unificare $p(X, Y, Y)$ e $p(a, Z, b)$:

- $E = \{p(X, Y, Y) = p(a, Z, b)\}$ e $S = \{\ \}$, inizialmente;
- $E = \{X = a, Y = Z, Y = b\}$, dopo la prima iterata;
- $E = \{Y = Z, Y = b\}, S = \{X/a\}$, estratta $X = a$, si aggiunge X/a a S;
- $E = \{Z = b\}, S = \{X/a, Y/Z\}$, estratta $Y = Z$, si aggiunge Y/Z a S dopo la sua applicazione a E;
- $E = \{\ \}, S = \{X/a, Y/b, Z/b\}$, estratta $Z = b$, si aggiunge Z/b a S;
- risultato finale: S (un MGU).

Provando a unificare $p(a, Y, Y)$ e $p(Z, Z, b)$:

- $E = \{p(a, Y, Y) = p(Z, Z, b)\}$, inizialmente;
- $E = \{a = Z, Y = Z, Y = b\}$, successivamente;
- $E = \{Y = a, Y = b\}$, estratta $a = Z$, si applica Z/a a E;
- $E = \{a = b\}$ estratta $Y = a$, si applica Y/a a E;
- resta $a = b$ ma le costanti a e b *non* sono *unificabili* quindi si restituisce $\perp$.

5.4 Risoluzione Definita con Variabili

La *procedura TD* proposizionale può essere estesa estesa per gestire le variabili ammettendo istanze di regole nella derivazione.

Una **clausola di risposta generalizzata** avrà la forma:

$$yes(t_1, \ldots, t_k) \leftarrow a_1 \wedge a_2 \wedge \ldots \wedge a_m$$

dove i $t_1, \ldots, t_k$ sono termini e gli $a_1, \ldots, a_m$ sono atomi.

L'uso del predicato yes consente l'*estrazione della risposta*, ossia di determinare quali istanze della query (con variabili) seguano logicamente da KB.

Data la query q, inizialmente si considera la clausola di risposta:

$$yes(V_1, \ldots, V_k) \leftarrow q$$

dove $V_1, \ldots, V_k$ sono le variabili della query q. Intuitivamente, un'istanza di $yes(V_1, \ldots, V_k)$ risulta vera se la corrispondente istanza di q è vera.

In ogni fase successiva, l'algoritmo

1. seleziona un *atomo* nel *corpo* della clausola di risposta corrente;
2. sceglie una *clausola* di KB la cui *testa* si unifichi con tale atomo.

Nella **risoluzione SLD** della clausola di risposta generalizzata

$$yes(t_1, \ldots, t_k) \leftarrow \underline{a_1} \wedge a_2 \wedge \ldots \wedge a_m$$

avendo *selezionato* a_1, se la clausola *scelta* da KB è

$$a \leftarrow b_1 \wedge \ldots \wedge b_p$$

con a_1 e a unificabili da un MGU σ, la nuova clausola di risposta (*risolvente*) risulterà:

$$(yes(t_1, \ldots, t_k) \leftarrow b_1 \wedge \ldots \wedge b_p \wedge a_2 \wedge \ldots \wedge a_m)\,\sigma$$

Ripetendo passi di risoluzione si ottiene una **derivazione SLD**, ossia una sequenza $\gamma_0, \gamma_1, \ldots, \gamma_n$ di clausole di riposta in cui:

- γ_0 è una clausola di risposta corrispondente alla query originaria q;
  date $V_1, \ldots, V_k$ variabili libere di q:

  $$yes(V_1, \ldots, V_k) \leftarrow q$$

- ogni γ_i si ottiene per risoluzione SLD:
  - selezionando a_1 nel corpo di γ_{i-1};
  - scegliendo della base una *copia*[1] della clausola scelta

    $$a \leftarrow b_1 \wedge \ldots \wedge b_p$$

    la cui testa a si unifichi con a_i tramite un MGU σ;
  - sostituendo a_1 con il corpo $b_1 \wedge \ldots \wedge b_p$;
  - applicando σ alla clausola di risposta risultante.
- γ_n sarà la **risposta** dalla forma:

  $$yes(t_1, \ldots, t_k) \leftarrow$$

  in una dimostrazione terminata con *successo*. In risposta alla query si restituirà:

  $$V_1 = t_1, \ldots, V_k = t_k$$

  dove i t_i sono i termini determinati dai binding delle variabili della query V_i.

```
non-deterministic procedure Prove_datalog_TD(KB,q)
    Input
        KB: insieme di clausole definite
        Query q: insieme di atomi da provare, con variabili V_1,...,V_k
    Output
        sostituzione θ se KB ⊨ qθ altrimenti fallisce
    Local
        G: clausola di risposta generalizzata

    Impostare G a yes(V_1,...,V_k) ← q
    while G non è una risposta do
        Sia G = yes(t_1,...,t_k) ← a_1 ∧ a_2 ∧ ... ∧ a_m
        Selezionare l'atomo a_1 nel corpo di G
        Scegliere una clausola a ← b_1 ∧ ... ∧ b_p in KB
            ridenominando tutte le variabili con nuovi nomi
        Sia σ ← Unify(a_1,a)
        if σ = ⊥ then fallire
        Assegna a G la clausola (yes(t_1,...,t_k) ← b_1 ∧ ... ∧ b_p ∧ a_2 ∧ ... ∧ a_m)σ
    return {V_1 = t_1,...,V_k = t_k} dove G è yes(t_1,...,t_k) ←
```

La procedura TD è *non deterministica*: si possono trovare *tutte* le derivazioni tentando altre scelte che portino al successo. Essa fallisce se tutte le scelte portano al fallimento: non esiste *alcuna derivazione*. La scelta della clausola può essere implementata con una forma di *ricerca*, come nel caso proposizionale.

Esempio — data una base di conoscenza (su *Shakira*):

- $born\_in(shakira, barranquilla)$.
- $born\_in(P, L) \leftarrow part\_of(S, L) \wedge born\_in(P, S)$.
- $part\_of(barranquilla, atlantico)$.
- $part\_of(atlantico, colombia)$.
- $part\_of(colombia, south\_america)$.

per rispondere alla query

- ask $born\_in(P, colombia)$.

si può costruire la *derivazione* seguente:

- $yes(P) \leftarrow born\_in(P, colombia)$.
  - $born\_in(P_1, L_1) \leftarrow part\_of(S_1, L_1) \wedge born\_in(P_1, S_1)$. con $\{P_1/P, L_1/colombia\}$;
- $yes(P) \leftarrow part\_of(S_1, colombia) \wedge born\_in(P, S_1)$.
  - $part\_of(atlantico, colombia)$. con $\{S_1/atlantico\}$;
- $yes(P) \leftarrow born\_in(P, atlantico)$.
  - $born\_in(P_2, L_2) \leftarrow part\_of(S_2, L_2) \wedge born\_in(P_2, S_2)$. con $\{P_2/P, L_2/atlantico\}$;
- $yes(P) \leftarrow part\_of(S_2, atlantico) \wedge born\_in(P, S_2)$.
  - $part\_of(barranquilla, atlantico)$. con $\{S_2/barranquilla\}$;
- $yes(P) \leftarrow born\_in(P, barranquilla)$.
  - $born\_in(shakira, barranquilla)$. con $\{P/shakira\}$;
- $yes(shakira) \leftarrow$.

6 Simboli di Funzione e Strutture Dati

Il Datalog richiede un nome (simbolo di costante) per ogni individuo sul quale si debba ragionare. Spesso risulta più semplice identificare un individuo indirettamente, ossia *in termini di altri individui*.

Considerando una costante per individuo, si può rappresentare solo un numero *finito* di individui, fissato alla costruzione della base di conoscenza. Come fare per ragionare su un dominio potenzialmente *infinito*? Ci vengono in aiuto la *programmazione logica* e, in particolare, il linguaggio Prolog [Llo87, SS94, Bra01, BBS12].

Esempi

1. Riferimenti temporali:
   - *orari*:
     - una costante per ogni orario, e.g. inizio-lezione: `1:30 p.m.`;
     - numero di *ore dopo mezzanotte* e numero di *minuti dopo l'ora*.
   - *date*:
     - una costante per data (sono infinite);
     - più facile in termini di `anno`, `mese`, `giorno`.
2. Sistema QA:
   - se gli individui sono le *frasi* servirebbero troppi nomi;
   - se le *frasi* fossero rappresentate come sequenze di *parole*:
     - nella pratica si potrebbero usare solo un vocabolario *finito* di parole;
     - oppure avere parole costruite come sequenze di *sillabe* o *lettere*.
3. Liste (di studenti):
   - *lista* come individuo con sue proprietà come `lunghezza`, `elemento-i`, ecc.;
   - oppure scegliendo un nome a ciascuna lista, ma è poco pratico;
   - o definendo le liste in termini dei loro elementi.

Le funzioni consentono di descrivere individui indirettamente, in termini di altri individui. Alla sintassi si aggiunge la nozione di *simbolo di* **funzione**, una parola che comincia con una lettera minuscola. Quindi il linguaggio viene *esteso* nella nozione di **termine** che potrà essere una variabile, una costante o avere la forma

$$f(t_1, \ldots, t_n)$$

dove f è un simbolo di funzione (n-aria) e i t_i sono termini. Nelle clausole i termini possono comparire *esclusivamente* all'interno di atomi: le funzioni sono distinte dai predicati.

In ogni interpretazione

$$\langle D, \phi, \pi \rangle$$

la ϕ dev'essere estesa: si assegna una funzione $D^n \mapsto D$ a ogni simbolo di funzione n-aria: essa, per ogni termine ground, deve specificare l'individuo denotato. Le costanti possono essere viste come funzioni 0-arie, i.e. senza argomenti.

Esempio — Per definire delle *date*:

costanti:

- numeri interi naturali (dom. predefinito)
- $jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec$

funzioni: (definizione *estensionale*)

- ce — date della *common era*: $ce(Y, M, D)$ data con anno Y, mese M e giorno D
  - ad es. $ce(2023, oct, 30)$ per il `30 ottobre 2023`
- bce — date prima della *common era*

predicati:

- $month(M, N)$ vero se M mese dell'anno con numero d'ordine N
  - $month(jan, 1).$
  - $month(feb, 2).$
  - $month(mar, 3).$
  - $month(apr, 4).$
  - $month(may, 5).$
  - $month(jun, 6).$
  - $month(jul, 7).$
  - $month(aug, 8).$
  - $month(sep, 9).$
  - $month(oct, 10).$
  - $month(nov, 11).$
  - $month(dec, 12).$
- $before(D_1, D_2)$ vero se D_1 precede D_2
  - $before(ce(Y_1, M_1, D_1), ce(Y_2, M_2, D_2)) \leftarrow Y_1 < Y_2.$
  - $before(ce(Y, M_1, D_1), ce(Y, M_2, D_2)) \leftarrow month(M_1, N_1) \land month(M_2, N_2) \land N_1 < N_2.$
  - $before(ce(Y, M, D_1), ce(Y, M, D_2)) \leftarrow D_1 < D_2.$

Il predicato $<$ rappresenta la relazione `minore_di` sugli interi, può essere descritto in termini di clausole, ma più spesso è *predefinito* (e.g. in Prolog).

Programmazione Logica

Qualunque funzione computabile può essere calcolata usando una base di clausole con simboli di funzione ossia è interpretabile come **programma logico** (adottando una *semantica operazionale*).

Il linguaggio dei programmi logici è *Turing equivalente*. Con un solo simbolo di funzione e una sola costante, si può definire un numero *infinito* di termini o atomi ground che consente di ragionare su un numero di individui potenzialmente *infinito*. Inoltre, con le funzioni si possono definire le tipiche **strutture dati** per i linguaggi di programmazione come, ad esempio, liste, alberi, ecc.

Esempio — Struttura di *albero* (con etichette)

funzioni:

- $node(N, LT, RT)$: denota un *nodo interno* di un albero con il *nome* N e due (sotto-)alberi sinistro LT e destro RT;
- $leaf(L)$: indica un nodo-*foglia* con *etichetta* L;

relazioni (predicati) definiti in forma ricorsiva:

- $at\_leaf(L, T)$ vera se L è l'etichetta di una *foglia* dell'albero T:
  - $at\_leaf(L, leaf(L))$.
  - $at\_leaf(L, node(N, LT, RT)) \leftarrow at\_leaf(L, LT)$.
  - $at\_leaf(L, node(N, LT, RT)) \leftarrow at\_leaf(L, RT)$.
- $in\_tree(L, T)$ vera se L è l'etichetta di un nodo *interno* dell'albero T:
  - $in\_tree(L, node(L, LT, RT))$.
  - $in\_tree(L, node(N, LT, RT)) \leftarrow in\_tree(L, LT)$.
  - $in\_tree(L, node(N, LT, RT)) \leftarrow in\_tree(L, RT)$.

Esempio — *Lista*: sequenza ordinata di elementi sulla quale si può ragionare usando *solo* funzioni e costanti; spesso è gestita attraverso una sintassi e semantica *predefinita* (come in Prolog, Lisp,...);

Una tipica definizione ricorsiva prevede:

- la sequenza vuota, denotabile con una *costante* ad hoc, come ad esempio nil;
- la sequenza composta da un elemento seguito da una lista, rappresentabile tramite la *funzione* $cons(Hd, Tl)$, dove Hd è il primo elemento (in *testa*) e Tl è il resto della lista (*coda*). Ad esempio si definisce la lista con elementi (nell'ordine) a, b, c con $cons(a, cons(b, cons(c, nil)))$;

L'*uso* delle liste tramite *predicati* come, ad esempio:

- $append(X, Y, Z)$ vero quando X, Y e Z sono liste tali che Z è composta dagli elementi di X seguiti da quelli di Y:
  $append(nil, L, L)$.
  $append(cons(Hd, X), Y, cons(Hd, Z)) \leftarrow append(X, Y, Z)$.

Logica del Primo e del Secondo Ordine

La **Logica dei Predicati del Primo Ordine** estende il calcolo proposizionale includendo atomi con simboli di funzione e variabili:

- *variabili* con *quantificazione esplicita* universale ($\forall$) o esistenziale ($\exists$);
- semantica simile a quella dei programmi logici, un sotto-linguaggio utile in termini *pratici*;
- operatori aggiuntivi: disgiunzione e quantificazione esplicite;
- *primo ordine*: quantificazione degli individui del dominio.

La **Logica del Secondo Ordine** ammette la quantificazione e predicati definiti su relazioni del primo ordine:

- *predicati* del *del secondo ordine*, come:
  - *simmetria*: $\forall R\ symmetric(R) \leftrightarrow (\forall X \forall Y\ R(X, Y) \rightarrow R(Y, X))$ vera se R è una relazione *simmetrica*
  - *transitività* di una relazione, non definibile nel primo ordine
    - e.g. $before$ chiusura transitiva di $next$, dove $next(X, s(X))$ è vero

Logica del primo ordine **semi-decidibile**: esiste una procedura completa e corretta capace di provare qualsiasi enunciato vero, ma che potrebbe divergere in caso di enunciato falso.

Logica del secondo ordine **indecidibile**: non esiste alcuna procedura completa e corretta implementabile su una macchina (di Turing).

6.1 Procedure di Dimostrazione con Funzioni

Per gestire i termini con funzioni vanno estese le procedure definite per il Datalog. Il problema principale è che il numero di termini possibili è *infinito*.

Nel definire una *procedura BU* sulla base di quella per la logica proposizionale, se si operasse preventivamente il *grounding* delle clausole, come visto in precedenza, una clausola potrebbe generare un'*infinita* sequenza di conseguenze sulla base del numero di termini formabili usando le funzioni.

Una procedura ingenua di grounding potrebbe innescare un problema di **starvation**, con clausole che vengano sistematicamente trascurate.

Esempio — base di conoscenza con le clausole:

1. $num(0).$
2. $num(s(N)) \leftarrow num(N).$
3. $a \leftarrow b.$
4. $b.$

Senza selezione fair, nel *forward chaining* si selezionerebbe *prima* la 1. e, successivamente, *sempre* la 2. che porta ripetutamente a derivare una nuova conseguenza ad ogni iterata: non arrivando a selezionare anche 3. e 4., non si potrà mai derivare a o b.

Va pertanto considerata una *proprietà* come la **fairness** del criterio di selezione delle clausole da elaborare che assicura che ogni clausola della base prima o poi debba essere selezionata. La procedura BU con un criterio di selezione *equo* (*fair*) risulta *completa*: ogni conseguenza potrà essere generata.

Per estendere la *procedura TD* per il Datalog con il trattamento di termini costruiti da funzioni, occorre una procedura di *unificazione* che possa seguire ricorsivamente la struttura dei termini in profondità, considerando anche le applicazioni di funzioni a diversi livelli (di composizione). Nell'algoritmo esteso per trovare MGU, una variabile X non potrà unificarsi con termini t (con $t \neq X$) in cui essa occorra come, ad esempio, $g(f(X))$. Serve quindi un **controllo di occorrenza** (*occurs-check*) aggiuntivo, senza del quale la procedura non risulterebbe più corretta (sound).

Esempio — Data una base di conoscenza con una sola clausola:

$$\{\ lt(X, s(X)).\ \}$$

nella sua interpretazione intesa il predicato lt corrisponde al classico *minore di* definito sul dominio degli interi (spesso predefinito e indicato con $<$) e la funzione *successore* $s(X)$ serve a denotare l'intero $X+1$.

Per la query $\mathsf{ask}\ lt(Y, Y).$ la procedura dovrebbe *fallire*: l'atomo è falso nell'interpretazione intesa poiché nessun numero è minore di se stesso.

Tuttavia se non si impedisse l'unificazione di X e $s(X)$ (per mancato controllo di occorrenza) la dimostrazione avrebbe *successo* e la correttezza della procedura sarebbe compromessa.

In un *algoritmo di unificazione* con simboli funzione e *occurs-check*, avendo selezionato $\alpha = \beta$, si restituisce $\perp$ se α è una variabile e β è un termine che la contiene essendo diverso da α stesso (o viceversa). Omettendo il controllo, tipicamente per aumentarne l'efficienza (ciò è spesso possibile nelle implementazioni del Prolog), la procedura non sarà più corretta. Una procedura non sound potrebbe derivare un goal g anche quando questo non fosse una conseguenza logica della base di conoscenza.

Esempio — *Risoluzione SLD*: considerata una base di conoscenza con le clausole

1. $append(c(A, X), Y, c(A, Z)) \leftarrow append(X, Y, Z)$.
2. $append(nil, Z, Z)$.

una *derivazione* per la query ask $append(F, c(L, nil), c(l, c(i, c(s, c(t, nil)))))$ è la seguente:

- $yes(F, L) \leftarrow append(F, c(L, nil), c(l, c(i, c(s, c(t, nil)))))$
  - risolvendo con $append(c(A_1, X_1), Y_1, c(A_1, Z_1)) \leftarrow append(X_1, Y_1, Z_1)$ copia della 1. con variabili ridenominate aggiungendo l'indice 1;
  - sostituzione: $\{F/c(l, X_1), Y_1/c(L, nil), A_1/l, Z_1/c(i, c(s, c(t, nil)))\}$;
- $yes(c(l, X_1), L) \leftarrow append(X_1, c(L, nil), c(i, c(s, c(t, nil))))$:
  - risolvendo con $append(c(A_2, X_2), Y_2, c(A_2, Z_2)) \leftarrow append(X_2, Y_2, Z_2)$;
  - sostituzione: $\{X_1/c(i, X_2), Y_2/c(L, nil), A_2/i, Z_2/c(s, c(t, nil))\}$;
- $yes(c(l, c(i, X_2)), L) \leftarrow append(X_2, c(L, nil), c(s, c(t, nil)))$:
  - risolvendo con $append(c(A_3, X_3), Y_3, c(A_3, Z_3)) \leftarrow append(X_3, Y_3, Z_3)$;
  - sostituzione: $\{X_2/c(s, X_3), Y_3/c(L, nil), A_3/s, Z_3/c(t, nil)\}$;
- $yes(c(l, c(i, c(s, X_3))), L) \leftarrow append(X_3, c(L, nil), c(t, nil))$:
  - applicabili entrambe le clausole, scegliendo la 1.: si risolve con $append(c(A_4, X_4), Y_4, c(A_4, Z_4)) \leftarrow append(X_4, Y_4, Z_4)$;
  - sostituzione: $\{X_3/c(t, X_4), Y_4/c(L, nil), A_4/t, Z_4/nil\}$;
- $yes(c(l, c(i, c(s, X_3))), L) \leftarrow append(X_4, c(L, nil), nil)$

qui la dimostrazione *fallisce*: le teste delle clausole non si unificano con l'atomo nel corpo. Si torna al punto di scelta precedente:

- $yes(c(l, c(i, c(s, X_3))), L) \leftarrow append(X_3, c(L, nil), c(t, nil))$
  - scegliendo invece la 2., si risolve con: $append(nil, Z_5, Z_5)$;
  - sostituzione: $\{Z_5/c(t, nil), X_3/nil, L/t\}$;
- $yes(c(l, c(i, c(s, nil))), t) \leftarrow$

e la dimostrazione ha *successo* con risposta $F = c(l, c(i, c(s, nil))), L = t$.

Liste: Notazione Prolog

Abbreviazioni ammesse nella notazione del Prolog:

- $[\,]$ o nil denotano la lista *vuota*;
- $[E \mid R]$ è la lista con primo elemento E e resto della lista R, scritta anche $cons(E, R)$;
- *semplificazione*: $[X \mid [Y]]$ si può scrivere $[X, Y]$, se Y è una sequenza di valori
  - ad esempio: $[a]$ per $[a \mid [\,]]$, $[b, a]$ per $[b \mid [a \mid nil]]$ e $[a, b \mid C]$ per $[a \mid [b \mid C]]$.

Esempio — Definizione che usa la notazione:

- $append([A \mid X], Y, [A \mid Z]) \leftarrow append(X, Y, Z)$.
- $append([\,], Z, Z)$.

Data la query: ask $append(F, [L], [l, i, s, t])$

una risposta che si ottiene è $F = [l, i, s], L = t$.

La dimostrazione resta la stessa: vengono ridenominate una funzione e una costante.

---

7 Uguaglianza

A volte è utile usare più termini per denotare uno *stesso* individuo: ad esempio i termini `3*5`, `F`, `273-258` e `15` possono denotare lo stesso numero. Altre volte serve che ogni nome si riferisca a un diverso individuo: ad esempio un nome distinto per insegnamenti differenti. Spesso non si sa se due nomi denotino lo stesso individuo: ad esempio se `postino_delle_8` e `postino_delle_15` siano la stessa persona. Negli esempi di ragionamento in Datalog presentati in precedenza, tutte le risposte erano valide a prescindere dal fatto che i termini denotassero individui diversi o meno.

Uguaglianza come Predicato

Può essere utile considerare il predicato speciale "$=$" avente un'interpretazione intesa standard *indipendente* dal dominio da rappresentare. Dati due termini t_1 e t_2, si può scrivere l'atomo (in *forma infissa*):

$$t_1 = t_2$$

da leggersi t_1 **uguale** a t_2, sarà vero in un'interpretazione I sse t_1 e t_2 in I denotano lo stesso individuo.

Non si tratta di semplice similarità o identicità ma di *identità*: due nomi (costanti) per un solo individuo, e non due individui simili (o indistinguibili) del dominio.

Esempio — Si consideri il caso in figura:

- $chair1 = chair2$ è falso, pur risultando identiche sotto tutti gli aspetti;
- se non si rappresenta la precisa *posizione*, non sono distinguibili: potrà essere vero che $chairOnRight = chair2$ ma non che la sedia a destra sia *simile* a $chair2$: essa è $chair2$.

7.1 Ammettere Asserzioni d'Uguaglianza

Spesso è utile poter concludere che termini distinti debbano denotare lo *stesso individuo*, ad esempio $chairOnRight = chair2$. Se non si ammettono atomi con predicato d'uguaglianza come teste delle clausole, un termine può essere uguale solo a se stesso in ogni interpretazione. Per poter di derivare relazioni d'uguaglianza, occorrono clausole con questo tipo di atomi nella testa. Le alternative possibili sono:

1. *assiomatizzare* l'uguaglianza come un qualunque altro predicato: aggiungere clausole che codifichino l'interpretazione del predicato $=$;
2. definire *procedure speciali* d'inferenza per l'uguaglianza: attraverso regole di riscrittura, si mappano termini diversi che denotino uno stesso individuo su una forma standard (canonica).

Assiomatizzare l'Uguaglianza

Un'**assiomatizzazione** di base per le variabili è la seguente:

- $X = X.$ *riflessività*
- $X = Y \leftarrow Y = X.$ *simmetria*

- $X = Z \leftarrow X = Y \wedge Y = Z.$ *transitività*

Per funzioni e predicati si devono istanziare **schemi di assiomi** (sostituendo un termine con uno uguale, il valore di una funzione/predicato non cambia):

- *schema* per funzioni n-arie f:

$$f(X_1, \ldots, X_n) = f(Y_1, \ldots, Y_n) \leftarrow X_1 = Y_1 \wedge \cdots \wedge X_n = Y_n$$

- *schema* per predicati n-ari p:

$$p(X_1, \ldots, X_n) \leftarrow p(Y_1, \ldots, Y_n) \wedge X_1 = Y_1 \wedge \cdots \wedge X_n = Y_n$$

Esempio — Istanziazione degli schemi:

- un assioma per la funzione $cons(X, Y)$:

$$cons(X_1, X_2) = cons(Y_1, Y_2) \leftarrow X_1 = Y_1 \wedge X_2 = Y_2$$

- un assioma per relazione $prop(I, P, V)$:

$$prop(I_1, P_1, V_1) \leftarrow prop(I_2, P_2, V_2) \wedge I_1 = I_2 \wedge P_1 = P_2 \wedge V_1 = V_2.$$

Usare questo tipo di assiomi comporta alcuni problemi: il ragionamento diventa *meno efficiente* allorché siano inclusi esplicitamente nella base di conoscenza; inoltre, la *terminazione* delle procedure TD ricerca in profondità *non* è *garantita*. Ad es. l'assioma di *simmetria* può causare un ciclo infinito a che non si aggiungano controlli sulla ripetizione di sotto-goal.

Procedure Speciali di Inferenza per l'Uguaglianza

Con la **paramodulazione** si implementa l'uguaglianza estendendo la procedura di dimostrazione. Quando vale l'uguaglianza fra due termini $t_1 = t_2$, ogni occorrenza di t_1 può essere sostituita con t_2. L'uguaglianza può essere trattata come una *regola di riscrittura* che sostituisce eguali con eguali. Conviene fissare una *rappresentazione canonica* per ogni individuo: termine sul quale vengono mappate tutte le altre sue rappresentazioni.

Esempio — Rappresentazione dei *numeri*:

Si possono usare molti termini per rappresentare lo stesso numero, ad es. `4*4`, `13+3`, `273-257`, `2^4`, `4^2`, `16`.

Una *rappresentazione canonica* tipica è quella delle sequenze di cifre in base 10.

Esempio — *#Matricola*: rappresentazione canonica per ciascuno studente, utile a distinguere, ad esempio, studenti diversi con lo stesso nome.

Anche considerando solo il nome, ci possono essere forme diverse per una stessa persona che andranno mappate sulla sua matricola: ad esempio, "`Cristiano Ronaldo dos Santos Aveiro`", "`Cristiano Ronaldo`", "`Ronaldo, Cristiano`", "`C. Ronaldo`", "`Ronaldo, C.`", "`Ronaldo`" ... possono essere mappate su una matricola come "`CR0007`".

7.2 Unique Names Assumption

In alternativa all'assiomatizzazione dell'uguaglianza si può adottare una *convenzione*. Nella semantica delle clausole definite data in precedenza, la funzione ϕ non è necessariamente iniettiva, quindi termini *diversi* possono denotare lo stesso individuo *o* individui distinti.

Con l'**assunzione di unicità del nome**, o *Unique Names Assumption* [UNA], termini ground diversi denotano individui distinti. Tale assunzione è tipica del contesto delle basi di dati. Per ogni coppia di termini ground distinti t_1 e t_2, si assumerà:

$$t_1 \neq t_2$$

dove $\neq$ significa "non uguale a" e corrisponde ad assumere la verità di $\neg(t_1 = t_2)$ in ogni interpretazione. Sotto UNA, un atomo con $\neq$ può occorrere nel *corpo* delle clausole.

Quindi sotto UNA, termini ground sono *identici* solo se si unificano. Ciò non vale per termini non ground, ad esempio l'atomo $a \neq X$ ha istanze vere nelle quali X assume un certo valore, diciamo b, e un'istanza falsa in cui X assume il valore a. La UNA è utile a integrare conoscenza proveniente da database, senza essere costretti ad esplicitare troppe distinzioni, come ad esempio quelle tra le persone con nomi diversi $kim \neq sam$, $kim \neq chris$, $chris \neq sam$, ... Tuttavia la UNA risulta *inappropriata* in certi altri casi, come ad esempio: $2 + 2 \neq 4$ oppure $clark\_kent \neq superman$.

> **Esempio** — Si consideri una base di dati che contenga n studenti in cui ognuno ha due insegnamenti a scelta da superare.
>
> Si supponga nella base risulti che uno studente abbia scelto e abbia superato l'esame di $math302$ e $psyc303$. Si potrà concludere che lo studente ha superato i due esami a scelta richiesti solo se $math302 \neq psyc303$ ossia se le costanti denotano insegnamenti differenti. Occorre poter distinguere codici che denotino insegnamenti diversi. Invece di aggiungere ben $n(n-1)/2$ assiomi di disuguaglianza, si può adottare la convenzione dell'UNA per cui *codici distinti denoteranno corsi distinti*.

Assiomatizzazione della UNA ⋖

Si aggiunge a quello per l'uguaglianza il seguente *schema di assiomi per definire la disuguaglianza*:

- $c \neq c'$, per ogni c e c'
- $f(X_1, \ldots, X_n) \neq g(Y_1, \ldots, Y_m)$, per ogni coppia di funzioni f e g
- $f(X_1, \ldots, X_n) \neq f(Y_1, \ldots, Y_n) \leftarrow X_i \neq Y_i$, per ogni f n-aria
  - n istanze della clausola, per $i \in \{1, \cdots n\}$
- $f(X_1, \ldots, X_n) \neq c$ per ogni f e c

dove c e c' costanti, f e g simboli di funzione distinti.

7.3 Procedura TD con UNA

Essendoci troppe diversità tra individui da specificare, per incorporare la UNA si può adottare una soluzione alternativa: la disuguaglianza $\neq$ verrà trattata come *predicato speciale*.

Per dimostrare un generico atomo $t_1 \neq t_2$, sono possibili i seguenti casi:

1. t_1 e t_2 non si unificano → *successo*
   - ad esempio $f(X, a, g(X)) \neq f(t(X), X, b)$
2. t_1 e t_2 identici → *fallimento*
   - ad esempio $f(X, a, g(X)) \neq f(X, a, g(X))$
3. successo per alcune istanze, ma fallimento per altre
   - ad esempio $f(W, a, g(Z)) \neq f(t(X), X, Y)$, un loro MGU è $\{X/a, W/t(a), Y/g(Z)\}$. le istanze ground compatibili con l'MGU portano al *fallimento* della dimostrazione, quelle non compatibili con l'MGU, invece, portano al suo *successo*. Va evitato di enumerare tutte le istanze di successo, potrebbero essere troppe.

Si noti che nel caso di coppie di termini ground, gli unici casi possibili sono 1. e 2.

La procedura TD può essere estesa per incorporare la UNA e per dimostrare atomi come $t_1 \neq t_2$ si considerano i tre casi:

1. la mancata unificazione porta al *successo* immediato;
2. l'unificabilità porta al *fallimento* immediato;
3. la procedura può *posticipare* in attesa delle dimostrazioni di altri atomi della query che possano far unificare le variabili, in modo da rientrare nei casi precedenti. Nella *selezione* dell'atomo nel corpo di G, si dovrebbero considerare prima quelli non posticipati. La dimostrazione ha *successo* se non ci sono altri atomi da selezionare e non si rientra nei casi 1-2. Un'istanza di successo assegna a ogni variabile una costante distinta, non usata in precedenza. Le variabili libere nella *risposta* vanno interpretate con cautela: l'atomo è vero *solo* per alcune istanziazioni, non per tutte.

Esempio — Definizione del concetto di studente che abbia superato due esami:

- $passed\_two\_courses(S) \leftarrow C_1 \neq C_2 \wedge passed(S, C_1) \wedge passed(S, C_2)$.
- $passed(S, C) \leftarrow grade(S, C, M) \wedge M \geq 50$.
- $grade(sam, engl101, 87)$.
- $grade(sam, phys101, 89)$.

Data la query: ask $passed\_two\_courses(sam)$

- la verità di $C_1 \neq C_2$ non può essere determinata, quindi va *posticipato*;
- selezionando $passed(sam, C_1)$, che associa $engl101$ a C_1, si dovrà provare $passed(sam, C_2)$, e quindi $grade(sam, C_2, M)$: avrebbe successo con sostituzione $\{C_2/engl101, M/87\}$
  - ma le variabili in $C_1 \neq C_2$ sarebbero legate allo stesso valore, quindi si arriva a un *fallimento* (provvisorio);
- scegliendo l'altra clausola da unificare con $grade(sam, C_2, M)$, si ha la sostituzione $\{C_2/phys101, M/89\}$ e le variabili in $C_1 \neq C_2$ sarebbero legate a costanti distinte, quindi la dimostrazione della diseguaglianza ha *successo*;
- resta solo da verificare $89 > 50$ che è banalmente vera, quindi si può concludere che la query ha *successo*.

Si potrebbe pensare di far si che i test di disuguaglianza ad essere valutati sempre come sotto-goal finali. Il differimento dei goal risulta spesso *efficiente*. Nell'esempio precedente, $C_1 \neq C_2$ differito può essere testato prima di controllare se $87 > 50$. Sebbene questo test possa essere valutato velocemente, spesso molti altri test possono essere evitati anticipando i test di disuguaglianze che sono vengono violate. Se la dimostrazione di un atomo potesse fallire/avere successo prima che le variabili fossero legate (non più libere), si dovrebbe comunque ricordare il vincolo $C_1 \neq C_2$, in modo che future unificazioni che lo violino possano portare al fallimento.

8 Assunzione di Conoscenza Completa

Come visto in precedenza per la Logica proposizionale, l'**assunzione di conoscenza completa**, consente di derivare conclusioni altrimenti non ammissibili rispetto alla semantica standard della Logica. Così un enunciato che *non segua* logicamente dalla base di conoscenza potrà essere considerato *falso*. Come visto in precedenza, questo consente anche di definire una forma di **negazione per fallimento** [NAF].

Per estendere l'assunzione di conoscenza completa al caso dei programmi logici con variabili e funzioni si devono considerare: *assiomi d'uguaglianza*, la proprietà di *chiusura del dominio* (*domain closure*) e una una nozione più sofisticata di *completamento*.

Esempio — sia data la relazione $student$ definita in una base di conoscenza con le sole clausole:

- $student(huan)$.
- $student(manpreet)$.
- $student(karan)$.

Assumendo che la conoscenza sia completa, i tre individui sarebbero gli *unici* studenti:

$$student(X) \leftrightarrow X = huan \vee X = manpreet \vee X = karan.$$

ossia si potrebbe dire che, se X è $huan$, $manpreet$ o $karan$, allora è uno studente e se X è studente allora dev'essere uno dei tre quindi un quarto individuo, ad esempio kim, non può essere uno studente.

L'assunzione di conoscenza completa include la UNA. Ad esempio, la dimostrazione dell'atomo $\neg student(kim)$ richiederebbe di dimostrare $kim \neq huan \wedge kim \neq manpreet \wedge kim \neq karan$, atomi immediatamente provati sotto UNA.

Forma Normale di Clark

Dato un predicato p, la clausola

$$p(t_1, \ldots, t_k) \leftarrow B.$$

in **forma normale di Clark** diventa

$$p(V_1, \ldots, V_k) \leftarrow \exists W_1 \cdots \exists W_m \, V_1 = t_1 \wedge \cdots \wedge V_k = t_k \wedge B.$$

dove, se la clausola è atomica allora B sarà $true$, le variabili $W_1, \ldots, W_m$ sono quelle originarie della clausola mentre le variabili $V_1, \ldots, V_k$ sono *nuove variabili* aggiuntive.

Ponendo tutte le clausole per il dato predicato p in forma normale, usando uno stesso insieme di nuove variabili, si può scrivere:

$$\begin{aligned} p(V_1, \ldots, V_k) &\leftarrow B_1. \\ &\vdots \\ p(V_1, \ldots, V_k) &\leftarrow B_n. \end{aligned}$$

riassumibili in

$$p(V_1, \ldots, V_k) \leftarrow B_1 \vee \cdots \vee B_n.$$

Tale enunciato è equivalente all'insieme di clausole originario.

Il **completamento** *del predicato* p sarà la formula logica:

$$\forall V_1 \cdots \forall V_k \, p(V_1, \ldots, V_k) \leftrightarrow B_1 \vee \cdots \vee B_n$$

dove, nei corpi, la **negazione per fallimento** ($\sim$) viene sostituita dalla negazione logica standard ($\neg$). Dal punto di vista della *semantica*, $p(V_1, \ldots, V_k)$ sarà vero sse è vero almeno uno dei corpi B_i.

Il **completamento** *di una base di conoscenza* comprende i completamenti di ogni predicato che occorre nella base.

Sotto l'assunzione di conoscenza completa, si possono definire relazioni per la cui definizione le clausole definite non sono sufficientemente espressive.

Esempio — date le clausole considerate in precedenza:

- $student(huan).$
- $student(manpreet).$
- $student(karan).$

la forma normale sarà:

- $student(V) \leftarrow V = huan.$
- $student(V) \leftarrow V = manpreet.$
- $student(V) \leftarrow V = karan.$

Questa definizione è equivalente a

$$student(V) \leftarrow V = huan \lor V = manpreet \lor V = karan.$$

quindi il completamento di $student$ è il seguente

$$\forall V\ student(V) \leftrightarrow V = huan \lor V = manpreet \lor V = karan.$$

Esempio — si consideri la definizione ricorsiva di un predicato:

- $passed\_each([\,], St, MinPass).$
- $passed\_each([C \mid R], St, MinPass) \leftarrow passed(St, C, MinPass)\ \land$
  $passed\_each(R, St, MinPass).$

In forma normale, cambiando i nomi delle variabili:

- $passed\_each(L, S, M) \leftarrow L = [\,].$
- $passed\_each(L, S, M) \leftarrow \exists C \exists R\ L = [C \mid R]\ \land$
  $passed(S, C, M)\ \land$
  $passed\_each(R, S, M).$

  sono state rimosse le uguaglianze dei renaming delle variabili e sono state ridenominate opportunamente le variabili.

Il completamento è il seguente:

$$\forall L \forall S \forall M\ passed\_each(L, S, M) \leftrightarrow L = [\,] \lor$$
$$\exists C \exists R\ (L = [C \mid R]\ \land\ passed(S, C, M)\ \land\ passed\_each(R, S, M)).$$

Esempio — Si consideri una base di conoscenza con:

- $course(C)$ vero se C è un corso;
- $enrolled(S, C)$ vero se S è iscritto a C.

Senza assunzione di conoscenza completa, non si potrebbe definire/dimostrare $empty\_course(C)$, vero se non ci sono iscritti al corso C:

- mai vero per un modello della base di conoscenza in cui ogni corso abbia iscritti.

Con l'assunzione, usando la NAF, la definizione sarebbe:

- $empty\_course(C) \leftarrow course(C)\ \land \sim has\_enrollment(C).$
- $has\_enrollment(C) \leftarrow enrolled(S, C).$

da cui il completamento risultante è il seguente:

- $\forall C\ empty\_course(C) \leftrightarrow course(C)\ \land \neg has\_enrollment(C).$
- $\forall C\ has\_enrollment(C) \leftrightarrow \exists S\ enrolled(S, C).$

Occorre cautela nell'inclusione di variabili libere in atomi con la NAF: in genere cambiano il significato inteso. Il predicato $has\_enrollment$ serve ad evitare di avere una variabile libera nel contesto di una negazione tipo $\sim enrolled(S, C)$.

Esempio — Supponendo di definire $empty\_course$ con:

- $empty\_course(C) \leftarrow course(C) \wedge {\sim}enrolled(S, C)$.

Il completamento

- $\forall C\ empty\_course(C) \leftrightarrow \exists S\ course(C) \wedge \neg enrolled(S, C)$.

risulta errato, infatti data una base di conoscenza con:

- $course(cs422)$.
- $course(cs486)$.
- $enrolled(mary, cs422)$.

$enrolled(sally, cs486)$. risulta *falsa*: l'istanza

- $empty\_course(cs422) \leftarrow course(cs422) \wedge {\sim}enrolled(sally, cs422)$

ha il corpo vero e la testa falsa, avendo $cs422$ delle iscrizioni; si crea un conflitto con la verità delle clausole istanziate dalla base di conoscenza.

Difatti il completamento (errato) equivale a $\forall C\ empty\_course(C) \leftrightarrow course(C) \wedge \neg\exists S\ enrolled(S, C)$. ovvero $\forall C\ empty\_course(C) \leftrightarrow course(C) \wedge \forall S\ \neg enrolled(S, C)$.

Procedure Complete di Dimostrazione con NAF

Risulta problematico adottare una procedura TD che possa trattare oltre a variabili e funzioni anche la NAF:

Esempio — Si consideri la base di conoscenza:

- $p(X) \leftarrow {\sim}q(X) \wedge r(X)$.
- $q(a)$.
- $q(b)$.
- $r(d)$.

e la query ask $p(X)$.

Per la semantica, ci sarebbe una sola risposta: $X = d$

- essendo $r(d)$ e ${\sim}q(d)$ vere, $p(d)$ segue logicamente dalla base di conoscenza.

Ma la dimostrazione di $p(X)$ con la procedura TD *fallisce*:

- selezionato ${\sim}q(X)$, la procedura riesce a dimostrare $q(X)$ con $\{X/a\}$;
- mentre $p(X)$ vero se $\{X/d\}$, essendo ${\sim}q(d)$ e $r(d)$.

Questo mostra che la TD risulta *incompleta* e *non corretta*:

- ad esempio, aggiungendo alla base di conoscenza la clausola $s(X) \leftarrow {\sim}q(X)$., dal fallimento di $q(X)$ si arriverebbe a poter derivare *erroneamente* $s(X)$.

Per evitare i problemi causati da *variabili libere nei goal negati* la procedura dovrebbe *posticipare* tali sotto-obiettivi fino a quando le variabili libere non vengano *legate* (in un binding). Qualora ciò non fosse possibile in alcun modo, la dimostrazione del goal *si inceppa* (**flounders**) e non si potrà concludere *nulla* su tale goal.

Esempio — Data la base di conoscenza:

- $p(X) \leftarrow {\sim}q(X)$
- $q(X) \leftarrow {\sim}r(X)$
- $r(a)$

con completamento:

- $p(X) \leftrightarrow \neg q(X)$

- $q(X) \leftrightarrow \neg r(X)$
- $r(X) \leftrightarrow X = a$

e la query: ask $p(X)$.

Sostituendo $X = a$ ad $r(X)$, si ha $q(X) \leftrightarrow \neg(X = a)$ quindi: $p(X) \leftrightarrow X = a$

La risposta giusta sarebbe $X = a$ ma il differimento del goal non aiuterebbe a trovarla. La procedura dovrebbe essere capace di *analizzare i casi* nei quali il goal è fallito (*argomento avanzato* per il quale si rimanda a trattazioni più complete).

Riferimenti Bibliografici

[**BBS12**] P.Blackburn, J. Bos & K. Striegnitz: *Learn Prolog Now!*. College Publications (2012) [online]
[**Bra01**] I. Bratko: *Prolog programming for artificial intelligence*. Pearson (2001)
[**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
[**Llo87**] J.W. Lloyd: *Foundations of logic programming*. 2nd edition, Symbolic Computation Series, Springer-Verlag. 1987
[**PM23**] D. Poole, A. Mackworth: *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press. 3a ed. 2023 (Ch.15)
[**RN20**] S.J. Russell, P. Norvig: *Artificial Intelligence*. Pearson. 4th Ed. 2020
[**SS94**] L.S. Sterling and E.Y. Shapiro: *The art of Prolog: advanced programming techniques*. 2nd edition, MIT Press (1994)

Link

[**UNA**] *Ipotesi di unicità del nome* su wikipedia
[**OWA**] N.Drummond, R.Shearer: *The Open World Assumption*, Univ. of Manchester, (2006) [slide online]
[**CWA**] Ipotesi del mondo chiuso
[**NAF**] Negazione come Fallimento, su Wikipedia
[**GNU-Prolog**] sito ufficiale
[**SWI-Prolog**] sito ufficiale

Note

[1] differenza con la SLD proposizionale: per evitare conflitti tra diverse istanze, copia usando nomi *nuovi* per le variabili