Rappresentazione e Ragionamento Relazionale

Nicola Fanizzi

Ingegneria della Conoscenza

CdL in Informatica • *Dipartimento di Informatica*
Università degli studi di Bari Aldo Moro

indice

Struttura Relazionale

Struttura: Individui e Relazioni

Nel rappresentare la *struttura* del mondo, utile ragionare in termini di:

- **Individui**: entità, cose, oggetti del mondo:
  - *concreti*: persone, edifici
  - *immaginari*: unicorni, programmi che superino il test di Turing
  - *concetti astratti*:
    - processi: la lettura di un libro, andare in vacanza
    - concetti: denaro, corsi, istanti di tempo
- **Relazioni**: in generale, specificano le verità riguardanti gli individui
  - *proprietà*: vere o false, su singoli individui
  - *proposizioni*: vere / false indipendentemente da qualunque individuo
  - *associazioni*: intercorrono tra più individui

Ragionamento più semplice perché più generale

Esempio — Nella rappresentazione proposizionale della smart house, atomi come up_s_2, up_s_3 e ok_s_2 non hanno ***struttura*** interna

- <u>non sufficiente</u> a esprimere 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

Alternativa: rappresentare esplicitamente e separatamente i singoli deviatori s_1, s_2, s_3 e le proprietà up e ok

- es. $up(s_2)$ per l'enunciato "s_2 si trova in posizione up"
  - sapendo cosa rappresentino up e s_1,
    non serve una definizione a parte per $up(s_1)$
- es. relazione binaria $connected\_to$ per collegare individui
  - come in $connected\_to(w_1, s_1)$

Vantaggi della nuova rappresentazione:

- più *naturale*
  - sfrutta la struttura delle feature come proprietà degli individui
- utile per *domini sconosciuti*: nessuna conoscenza sugli individui, sul loro numero e delle loro caratteristiche
  - si deve interagire con l'ambiente
- *ragionamento* più *generale*, indipendente dai singoli individui, circa
  - *conclusioni* che valgono *per tutti* gli individui senza conoscerli
  - *esistenza* di individui e loro proprietà, a prescindere dagli altri
  - *query* che riguardano intere popolazioni e non singoli individui
- *esistenza incerta* di individui
  - non si conoscono completamente individui attuali/futuri e loro proprietà
- ragionamento su un *infinito* numero di individui/proprietà
  - ad es. individui = frasi (potenzialmente infinite)
    - si ragiona su un numero finito di frasi: quelle utili al compito da svolgere

Simboli e Semantica

Simboli e Semantica

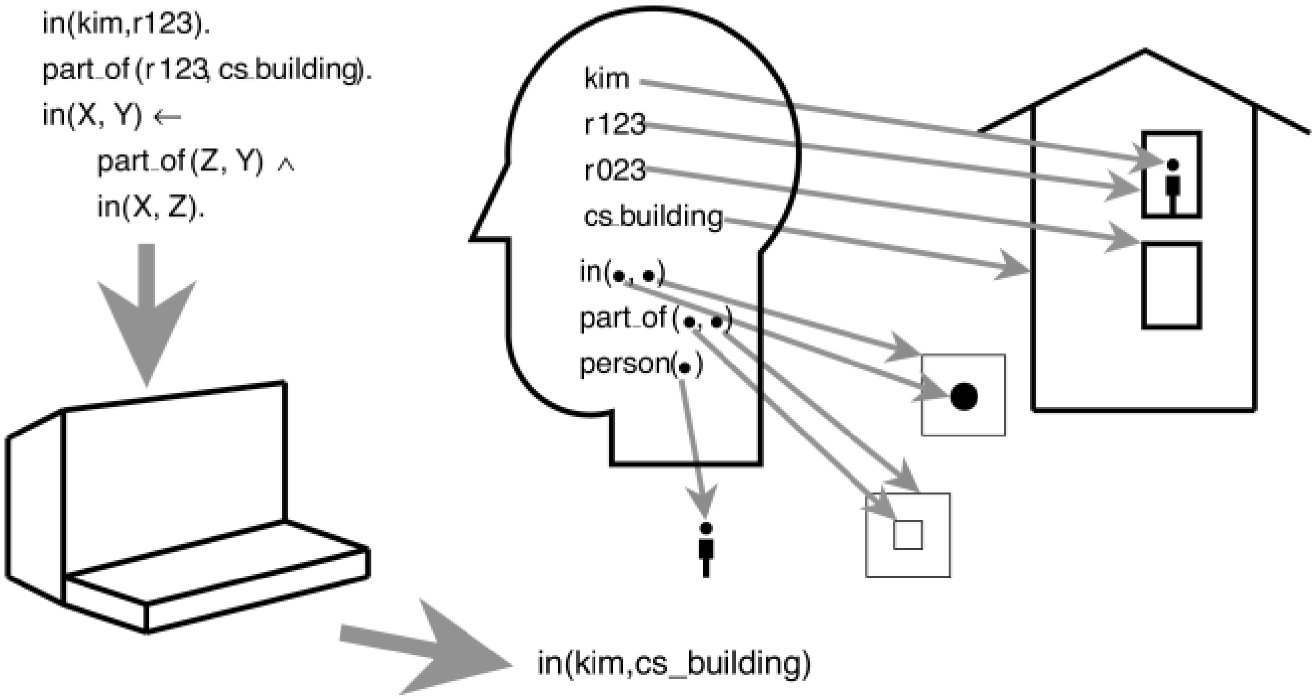
Utilità della *logica* nella progettazione di KB:

- i progettisti hanno un particolare *mondo* da caratterizzare in base a un'**interpretazione intesa**:
    - scegliendo *significati* per i simboli in base a tale interpretazione
    - descrivendo ciò che dev'essere vero attraverso una KB di *clausole*
- chiunque conosca il significato dei simboli può interpretare, rispetto a tale modello, *conseguenze logiche* derivate dalla KB
    - vera in tutti i modelli della KB → vera nell'interpretazione intesa

Estendendo il linguaggio delle clausole proposizionali:

- atomi con una *struttura interna* descritta in termini di relazioni e individui

Esempio — Idea della semantica della rappresentazione



- chi progetta la base di conoscenza attribuisce un *significato* ai simboli
  - sa che a cosa si riferiscano nel dominio e il significato delle *frasi* nel linguaggio di rappresentazione (la KB) 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 calcola *risposte* alle query pur ignorando tale significato

CONCETTUALIZZAZIONE

Associazione

simboli nella mente → individui e relazioni da essi denotati

- qui solo informale o nella mente dell'utente
- successivamente: *ontologie formali*

Separazione tra l'aspetto *semantico* e quello *computazionale*:

- *correttezza* della base di conoscenza e delle risposte alle query definita dalla *semantica* indipendentemente dall'*algoritmo* di ragionamento
  - ottimizzabile a patto di rimanere semanticamente corretto
- relazione tra linguaggi logici e di programmazione (cfr. [1] §13.2)

Datalog: Linguaggio Relazionale a Regole

DATALOG: estende il linguaggio delle clausole definite proposizionali con sintassi del calcolo dei predicati e convenzioni del PROLOG [6, 7, 9]

- KB di enunciati positivi e negativi
- contesto *statico*
- numero *finito* di individui di interesse nel dominio:
    - ognuno con un *nome* univoco assegnato

SINTASSI

- **variabile**: parola (alfanumerica con _) che inizia con maiuscola o _
  - es. $Stanza$, $B4$, $Lista$ e The_Dude
- **costante**: parola che inizia con una minuscola o numero o stringa tra apici
- **predicato**: parola che inizia con una minuscola
  - costanti o predicati distinguibili in base al ***contesto***
    - es. $nico$, $r123$, f_1, $insegna\_a$ e $aula\_IV$, ma 2018 solo costante
- **termine**: variabile o costante
  - es. X, $nico$, dib, $uniBA$ o $Docenti$
- **atomo** (***simbolo atomico***): forma p oppure $p(t_1, \ldots, t_n)$
  - p simbolo di predicato
  - t_i termine, i-esimo ***argomento*** del predicato, $\forall i$
    - es. $insegna(nicoF, 63507)$, $in(vitoR, 522)$, $padre(alan, Y)$, $giallo(C)$, $inserire(Dessert, menuT)$, $aperto$
      - nell'atomo $inserire(Dessert, menuT)$, dal contesto: $inserire$ predicato mentre $menuT$ costante

Sintassi: Clausole Definite

- **clausola definita**:

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

  si legge "h se a_1 e ... e a_m", fatta di atomi
  - h **testa** della clausola
  - se $m > 0$ è una **regola** e $a_1 \wedge \ldots \wedge a_m$ **corpo**
  - se $m = 0$ *clausola* **atomica** o **fatto**
    - corpo vuoto, si può omettere la freccia
- **base di conoscenza**: insieme di clausole definite
- **query**:

$$\text{ask } a_1 \wedge \ldots \wedge a_m$$

  - corrisponde a $\leftarrow a_1 \wedge \ldots \wedge a_m$

Esempio — Base di conoscenza:

1. $grandfather(sam, X) \leftarrow father(sam, Y) \wedge parent(Y, X).$
2. $in(kim, R) \leftarrow teaches(kim, cs422) \wedge in(cs422, R).$
3. $slithy(toves) \leftarrow mimsy \wedge borogroves \wedge outgrabe(mome, Raths).$
   dove
   - $sam, kim, cs422, toves$ e $mome$ costanti
   - $grandfather$, $father$, $parent$, in, $teaches$, $slithy$, $mimsy$, $borogroves$ e $outgrabe$ predicati
   - X, Y, R e $Raths$ variabili

Clausole 1. e 2. su Kim (kim) e Sam (sam) dal significato intuitivo:

- per una macchina non sono più intelligibili della 3. (da testi di L.Carrol)

Espressioni: Semantica e Livelli di Generalità

Significato fornito attraverso una specifica formale

- **Espressione**: termine, atomo, clausola definita o query
- *Espressione* **ground** (livello-*base*) non contiene variabili
    - si riferisce solo a precisi individui
    - ad es., atomo $insegna(nicola\_fanizzi, 63507)$ ground,
      invece $insegna(Prof, 63507)$ o $insegna(Prof, C\_Ins)$ non ground

Definizione della *semantica*

1. si definisce per le espressioni ground
2. si estende al caso di espressioni con variabili

Interpretazione delle espressioni *ground*:

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

- **dominio** D insieme non vuoto di **individui**
- ϕ associazione: $c \mapsto \phi(c) \in D$
  - a ogni nome di *costante* c assegna un individuo di D
    - fisico, del mondo reale (una persona o un virus)
    - astratto (un concetto, un corso, un'emozione, il numero 2 o un simbolo)
- π associazione: $p \mapsto \pi(p)$
  - a ogni simbolo di *predicato* n-ario p assegna una funzione booleana $\pi(p): \ D^n \to \{true, false\}$
  - *per ogni* n-pla di individui, $\pi(p)$ specifica se la relazione p sia vera/falsa
    - se p non ha argomenti allora $\pi(p)$ sarà costante: $true$ o $false$
      - come nella semantica del calcolo proposizionale

Esempio — Mondo semplice: tavolo con forbici, telefono e matita

✂☎ ✎

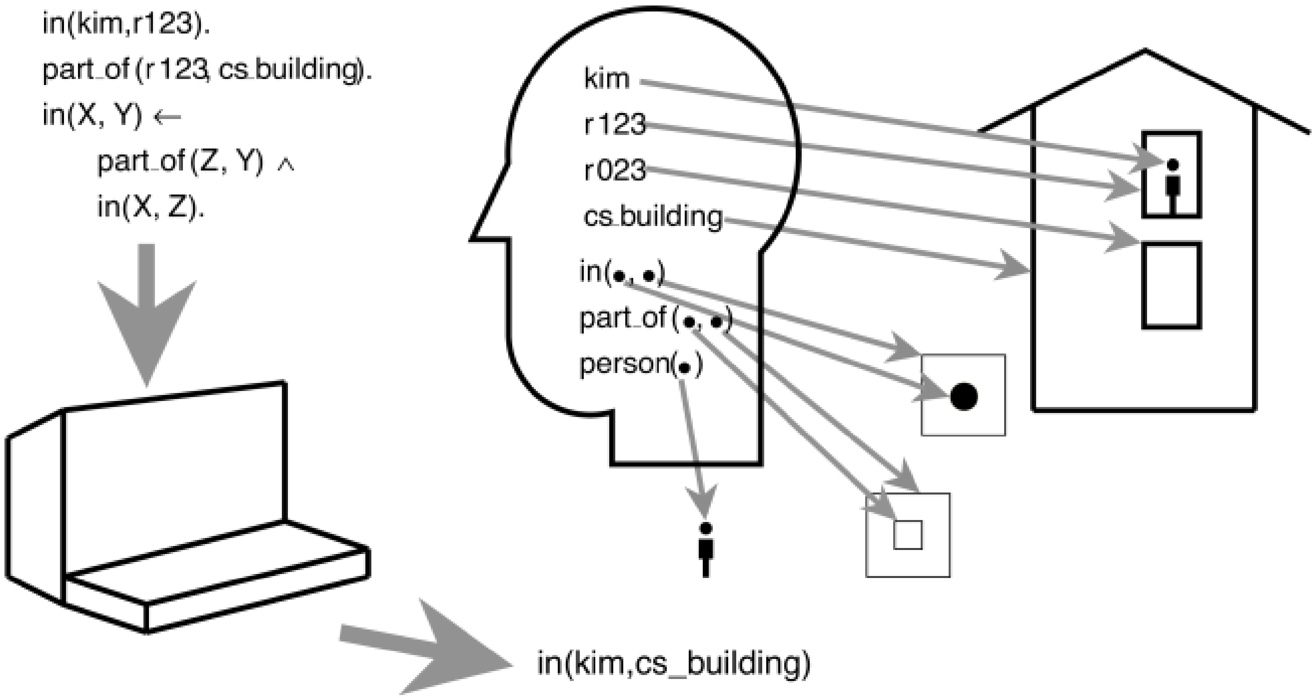
- nomi di ***costanti***: $phone$, $pencil$ e $telephone$
  - assenza di $scissors$ intenzionale
- simboli di ***predicato***: $noisy$ (unario) e $left\_of$ (binario)
- una particolare ***interpretazione***:
  - $D = \{✂, ☎, ✎\}$ contiene individui: oggetti del mondo reale
  - $\phi(phone) = ☎$, $\phi(pencil) = ✎$, $\phi(telephone) = ☎$
    - i nomi $phone$ e $telephone$ si riferiscono allo ***stesso individuo***

      → ***stessi enunciati*** che li riguardano risulteranno ***veri***
  - $\pi(noisy)$: fa rumore
    - $\langle ✂ \rangle \mapsto false,\ \langle ☎ \rangle \mapsto true,\ \langle ✎ \rangle \mapsto false$
  - $\pi(left\_of)$: a sinistra di
    - $\langle ✂, ✂ \rangle \mapsto false,\ \langle ✂, ☎ \rangle \mapsto true,\ \langle ✂, ✎ \rangle \mapsto true,$

      $\langle ☎, ✂ \rangle \mapsto false, \langle ☎, ☎ \rangle \mapsto false\ \langle ☎, ✎ \rangle \mapsto true,$

      $\langle ✎, ✂ \rangle \mapsto false, \langle ✎, ☎ \rangle \mapsto false,\ \langle ✎, ✎ \rangle \mapsto false$

Esempio — macchina, mente e interpretazione della situazione (*a destra*)



nella/dalla macchina — nella mente — scena da interpretare

Semantica: Atomi Ground

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

Semantica degli atomi ground:

- atomo ground $p(t_1, \ldots, t_n)$
  - **vero** in I se $\pi(p)(\langle i_1, \ldots, i_n \rangle) = true$
    - $i_k = \phi(t_k)$ individuo denotato dal termine t_k
  - **falso** in I altrimenti

Esempio — Nell'interpretazione della *figura* precedente:

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

Semantica: Estensioni

Per *connettivi* logici, *modelli* e *conseguenza logica*
stessa semantica del calcolo proposizionale

Data un'interpretazione I:

- *clausola ground* — cfr. tabella di verità di $\leftarrow$
  - **falsa** in I se la testa è falsa mentre il corpo è vero (o vuoto)
  - **vera** in I altrimenti

Considerata KB di sole clausole ground: — come proposizionale

- I **modello** di KB sse *ogni* clausola di KB vera in I
- $KB \models g$, ossia g **conseguenza logica** di KB,
  se g è vera in ogni modello di KB
  - $KB \not\models g$ indica che esiste un modello di KB in cui g è falsa

Interpretare le Variabili

Clausole con variabili da intendersi come *universalmente quantificate*

- **assegnazione** *di variabili* ρ: funzione dall'insieme delle variabili a D
  - 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
  - semantica degli *atomi* e delle *clausole* ground come da def. precedenti
    - ottenuti associando individui ai termini

Clausola con variabili **vera** in I sse $\forall \rho$:

- ogni clausola ground ottenuta applicando ρ alle sue variabili risulta vera

Esempio — Nell'interpretazione dell'esempio precedente:

$part\_of(X, Y) \leftarrow in(X, Y).$ è ***falsa***

- considerando l'assegnazione di X a Kim e di Y alla stanza 123
  - corpo $in(kim, r123)$ vero
  - testa $part\_of(kim, r123)$ falsa

$in(X, Y) \leftarrow part\_of(Z, Y) \wedge in(X, Z).$ è ***vera***

- istanza del corpo $part\_of(r123, cs\_building) \wedge in(kim, r123)$ vera solo sotto l'assegnazione per la quale anche l'istanza della testa è vera
  - per le altre: istanze del corpo false → clausole ground sempre vere

Conseguenze Logiche

Come per le clausole definite proposizionali:

- congiunzione di atomi (corpo) ground g **conseguenza logica** di KB, denotato con $KB \models g$, sse g è vero in ogni modello di KB

Esempio — Supponendo che KB contenga:

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

l'interpretazione (cfr. figura) degli esempi precedenti è un modello di KB, essendo le clausole vere in tale interpretazione:

(..cont.)

- $KB \models in(kim, r123)$
  perché compare esplicitamente nella KB
  - ogni clausola di KB è vera in quell'interpretazione
- $KB \not\models in(kim, r023)$:
  - perché falso nell'interpretazione, modello di KB
- $KB \not\models part\_of(r023, cs\_building)$:
  sebbene $part\_of(r023, cs\_building)$ sia vero nell'interpretazione,
  è ammessa l'esistenza di un altro modello di KB in cui l'atomo è falso
  - $\pi(part\_of)(\langle \phi(r023), \phi(cs\_building) \rangle) = false$
- $KB \models in(kim, cs\_building)$:
  altrimenti ci sarebbe un'istanza della 3. falsa in I (testa falsa, corpo vero)
  - impossibile perché I modello per KB quindi per la 3. e tutte le sue istanze

Caso: clausola con variabili che occorrono solo nel corpo

Esempio — Nell'esempio precedente:

$$in(X, Y) \leftarrow part\_of(Z, Y) \wedge in(X, Z).$$

- Y è universalmente quantificata a livello di clausola
  - clausola vera per tutte le assegnazioni di variabili
- assegnando c_1 a X e c_2 a Y, la clausola

  $$in(c_1, c_2) \leftarrow part\_of(Z, c_2) \wedge in(c_1, Z).$$

  è vera per tutte le assegnazioni a Z
  - se, assegnando c_3 a Z, $part\_of(Z, c_2) \wedge in(c_1, Z)$ risulta vera in una interpretazione, allora $in(c_1, c_2)$ vera nella stessa interpretazione
  - si può leggere la clausola come "per ogni X e per ogni Y, $in(X, Y)$ è vera se <u>esiste</u> una Z tale che $part\_of(Z, Y) \wedge in(X, Z)$ è vera"

QUANTIFICAZIONE

Quantificazione universale *implicita* nel linguaggio delle clausole (def.):

- a volte utile renderla *esplicita*

Quantificatori in logica:

- $\forall X \; p(X)$, da leggersi "*per ogni* X, $p(X)$",
  significa che $p(X)$ è vera per ogni assegnazione a X
  - X si dice **universalmente** *quantificata*
- $\exists X \; p(X)$, da leggersi "*esiste* un X tale che $p(X)$",
  significa che $p(X)$ è vera per qualche assegnazione a X
  - X si dice **esistenzialmente** *quantificata*

La clausola

$$P(X) \leftarrow Q(X, Y)$$

significa

$$\forall X \forall Y (P(X) \leftarrow Q(X, Y))$$

equivalente a

$$\forall X (P(X) \leftarrow \exists Y Q(X, Y))$$

- variabili *libere* che appaiono solo nel corpo: *esistenzialmente* quantificate in tale ambito

Esempio — Caso particolare:

$$in(cs422, love) \leftarrow part\_of(cs422, sky) \wedge in(sky, love).$$

- $cs422$ denota un corso, $love$ un concetto astratto e sky denota il cielo

vera nell'interpretazione intesa secondo la tavola di verità di $\leftarrow$

- premessa (a destra) falsa nell'interpretazione intesa (*ex falso quodlibet*)

Osservazione — una regola con testa e corpo poco significativi non potrà essere usata per provare qualcosa di utile:

- è un fatto *convenzionale* che la clausola sia vera quando il corpo è falso: semplifica la semantica senza causare problemi tecnici
- ci si può preoccupare solo dei casi in cui il corpo è vero

Semantica: Punto di Vista del Progettista/Esperto

Estendendo al DATALOG la metodologia per KB proposizionali:

1. selezionare il *dominio* del task o *mondo* da rappresentare
    - D insieme di tutti gli *individui* cui si farà riferimento e sui quali ragionare
    - *relazioni* da rappresentare
        - aspetti specifici del mondo *reale*, e.g. struttura dei corsi e studenti presso un'università o un laboratorio in un determinato momento
        - mondi *immaginari* o *ipotetici*, e.g. quello di *Alice nel paese delle meraviglie*, o lo stato di un'apparecchiatura elettrica
        - mondo *astratti*, e.g. numeri e insiemi, contesti finanziari,...
2. associare le *costanti* del linguaggio a individui del mondo da nominare
    - a ogni elemento di D si assegna una costante per riferirvisi
        - ad es. 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. associare un simbolo di *predicato* a ogni relazione da rappresentare
    - un simbolo (n-ario) denota una funzione $D^n \rightarrow \{true, false\}$
        - basta specificare le n-ple per le quali la relazione è vera
        - e.g. $teaches$ binario: relazione vera tra un individuo (primo argomento) e un corso insegnato (secondo argomento)
    - relazioni con qualunque arietà, anche 0
        - e.g. is_red predicato unario
    - associazioni simbolo-significato per costituire l'**interpretazione intesa**
4. definire *clausole* vere nella stessa: **assiomatizzazione del dominio**
    - clausole = **assiomi** del dominio
        - e.g. se la persona denotata da kim insegna il corso denotato da $cs322$, la clausola $teaches(kim, cs322)$ sarà vera nell'interpretazione intesa
5. formulare *query* riguardanti l'interpretazione intesa
    - risposte del sistema da interpretare nella semantica attribuita ai simboli

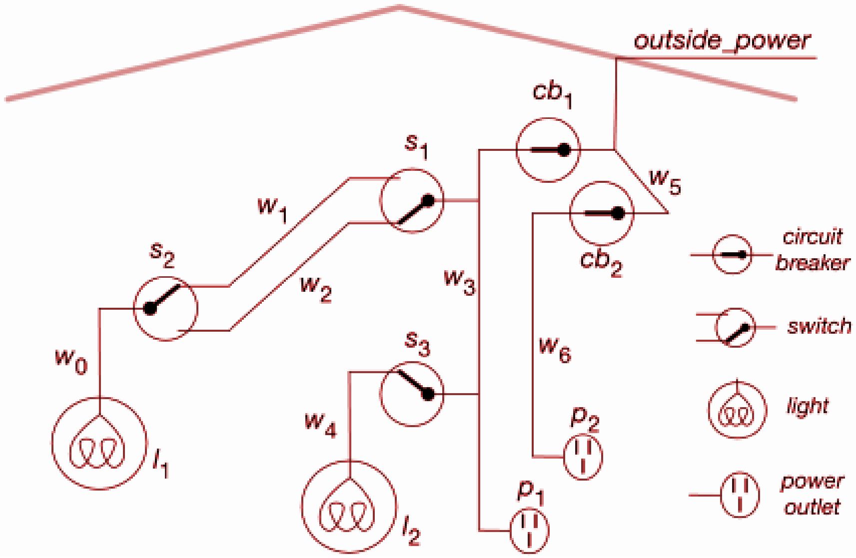
Osservazioni — seguendo la metodologia:

- il *progettista* prescinde dalla *macchina* fino al passo 4.
  - primi tre passi svolti “sulla carta”
    - con strumenti dell'ingegneria del SW
- le notazioni vanno *documentate*
  per rendere la KB comprensibile ad altre persone
  - fissa la notazione di ogni simbolo
  - permette di verificare la verità della clausole
- il mondo di per sé non impone niente sulla natura degli individui

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. basso livello di dettaglio:
    - diverse sfumature di rosso indistinguibili
    - colore rosa non considerato, ma solo red
4. aumentare il dettaglio:
    - si considera $pink$ come termine generale
    - prevedendo anche/invece $coral$ o $salmon$

Esempio — (Smart Home) 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
       - potrebbe non esserlo per altri task

2. ***nomi*** **da dare a ciascun individuo al quale ci si vorrà riferire**
   - come in figura
     - e.g. 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 vivo
   - $up(S)$ 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:
     - $live(X) \leftarrow connected\_to(X, Y) \wedge live(Y).$
     - $live(outside).$
   - 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 questa casa in particolare

Query con Variabili

Query: per chiedere se un enunciato sia conseguenza logica di una KB

query decisionali	caso proposizionale (ground, senza variabili) con risposta yes o no
query con variabili	consentono di determinare e restituire (tuple di) *individui* per i quali la query si avvera

Istanza di una query: ottenuta sostituendo termini alle variabili

- ogni occorrenza di una variabile sostituita con uno stesso termine
- **estrazione della risposta**: determinare quali istanze seguano dalla KB
- **risposte** possibili a una query con variabili libere:
  - *istanze* della query che seguono logicamente dalla KB
    - specificate dai valori per le variabili nella query
  - no: nessuna istanza segue logicamente dalla KB
    - <u>non</u> significa che sia falsa nell'interpretazione intesa

Esempio — KB riguardante le stanze di un edificio con le seguenti clausole:

- $imm\_west(W, E)$: W è immediatamente a ovest di E
    - $imm\_west(r101, r103).$
    - $imm\_west(r103, r105).$
    - $imm\_west(r105, r107).$
    - $imm\_west(r107, r109).$
    - $imm\_west(r109, r111).$
    - $imm\_west(r131, r129).$
    - $imm\_west(r129, r127).$
    - $imm\_west(r127, r125).$
- $imm\_east(E, W)$: E è immediatamente a est di W
    - $imm\_east(E, W) \leftarrow imm\_west(W, E).$

(..cont.)

- $next\_door(R1, R2)$: $R1$ è a una porta da $R2$
  - $next\_door(E, W) \leftarrow imm\_east(E, W).$
  - $next\_door(W, E) \leftarrow imm\_west(W, E).$
- $two\_doors\_east(E, W)$: E è a due porte a est da W
  - $two\_doors\_east(E, W) \leftarrow imm\_east(E, M), imm\_east(M, W).$
- $west(W, E)$: W è a ovest di E
  - $west(W, E) \leftarrow imm\_west(W, E).$
  - $west(W, E) \leftarrow imm\_west(W, M), west(M, E).$

Clausole scritte per essere vere in un mondo anche ipotetico

(..cont.)

La macchina non sa altro: conosce solo le clausole della KB
e sa calcolarne le *conseguenze logiche*

Risposte a query proposizionali:

- ask $imm\_west(r105, r107).$
  - yes
- ask $imm\_east(r107, r105).$
  - yes
- ask $imm\_west(r205, r207).$
  - no, i.e. non è conseguenza logica (non significa che sia falso):
    - informazione <u>non sufficiente</u> a determinare se $r205$ sia o meno immediatamente a ovest di $r207$

(..cont.)

Risposte a query con variabili:

- ask $next\_door(R, r105)$.
  1. $R = r107$
     - i.e. $next\_door(r107, r105)$ conseguenza logica
  2. $R = r103$
- ask $west(R, r105)$.
  - $R = r103$
  - $R = r101$
- ask $west(r105, R)$.
  - $R = r107$
  - $R = r109$
  - $R = r111$
- ask $next\_door(X, Y)$.
  - $X = r103, Y = r101$
  - $X = r105, Y = r103$
  - $X = r101, Y = r103$
  - ...e altre 13 risposte

Sostituzioni e Dimostrazioni

Estensione delle Procedure di Dimostrazione

A partire dalle procedure BU e TD per KB proposizionali, generalizzare per considerare KB e query DATALOG:

- variabile *libera* in una clausola DATALOG:
  - indica che tutte le *istanze* della clausola dovranno essere vere
    - al variare delle assegnazioni
- una dimostrazione può basarsi su istanze *diverse* di una stessa clausola

Un'*istanza* di una clausola si ottiene sostituendo variabili con termini uniformemente:

- ogni occorrenza di una sua variabile 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
- V_i/t_i **binding** (*associazione*) per V_i
  - in **forma normale** se nessuna V_i occorre in un t_j con $i \neq j$

Esempio — Sostituzioni:

- $\{X/Y, Z/a\}$ in forma normale che associa X a Y e Z ad a:
- $\{X/Y, Z/X\}$ *non* in forma normale
  - X occorre sia a sinistra sia a destra di binding

Istanza $e\sigma$ dell'espressione e (termine, atomo o clausola) tramite σ:

- ottenuta dall'*applicazione* di $\sigma = \{V_1/t_1, \ldots, V_n/t_n\}$ a e:
  ogni occorrenza di V_i sostituita da t_i
  - *istanza* **ground** se non contiene variabili

Esempio — applicazioni di sostituzioni ad atomi:

- $p(a, X)\{X/c\} = p(a, c)$
- $p(Y, c)\{Y/a\} = p(a, c)$
- $p(a, X)\{Y/a, Z/X\} = p(a, X)$
- $p(X, X, Y, Y, Z)\{X/Z, Y/t\} = p(Z, Z, t, t, Z)$

si applicano anche a clausole:
data $\sigma = \{X/Y, Z/a\}$

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

Unificazione

Unificatore delle espressioni e_1 e e_2: sostituzione σ tale che

$$e_1\sigma = e_2\sigma$$

- i.e. applicata a più espressioni produce un'unica istanza

Esempio — Unificatori:

- $\{X/a, Y/b\}$ unificatore di $t(a, Y, c)$ e $t(X, b, c)$:
  $t(a, Y, c)\{X/a, Y/b\} = t(X, b, c)\{X/a, Y/b\} = t(a, b, c)$

due espressioni possono avere *diversi* unificatori:

- $p(X, Y)$ e $p(Z, Z)$ hanno diversi unificatori, tra i quali:
  $\{X/b, Y/b, Z/b\}$, $\{X/c, Y/c, Z/c\}$, $\{X/Z, Y/Z\}$ e $\{Y/X, Z/X\}$
  - i primi due specificano precisamente i valori sostituiti
  - gli altri due *più generali*

MGU

Unificatore Più Generale (*most general unifier*, MGU) σ di e_1 ed e_2: unificatore tale che, per qualunque altro loro unificatore σ' si ha che $e\sigma'$ istanza di $e\sigma$, per ogni espressione e

- e_1 **ridenominazione** (*renaming*) di e_2 sse istanze l'una dell'altra
  - differiscono solo nei nomi delle variabili

Due espressioni unificabili avranno almeno un MGU

- applicando loro un MGU risulteranno ridenominazioni reciproche:
  - σ e σ' MGU di e_1 ed e_2 → $e_1\sigma$ renaming di $e_1\sigma'$ (analogamente per e_2)

Esempio — MGU di $p(X, Y)$ e $p(Z, Z)$: $\{X/Z, Y/Z\}$ e $\{Z/X, Y/X\}$

- con la loro applicazione a $p(X, Y)$:
  - $p(X, Y)\{X/Z, Y/Z\} = p(Z, Z)$ e
  - $p(X, Y)\{Z/X, Y/X\} = p(X, X)$ renaming

Problema dell'**unificazione:**

- dati due termini o atomi, determinare se si unificano
  - nel caso, restituire un loro unificatore

Algoritmo per trovare un eventuale MGU di due atomi/termini (altrimenti restituisce $\perp$)

- lavora su
  - E insieme di uguaglianze ($\alpha = \beta$) che implicano l'unificazione
  - S insieme di binding per una *sostituzione*
    - se $\alpha/\beta \in S$ allora, per costruzione,
      α è una variabile che <u>non</u> appare altrove in S o in E
    - nel caso degli atomi, α e β devono avere stesso simbolo di predicato e stessa arietà, altrimenti l'unificazione fallisce

```
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
```

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

- $E = \{p(X,Y,Y) = p(a,Z,b)\}, S = \{\ \}$ inizialmente
- $E = \{X = a, Y = Z, Y = b\}$ dopo la prima iterata
- $E = \{Y = Z, Y = b\}, S = \{X/a\}$ estratta $X = a$, X/a aggiunto a S
- $E = \{Z = b\}, S = \{X/a, Y/Z\}$
  selezionata $Y = Z$, Y/Z applicato a E e aggiunto a S
- $E = \{\ \}, S = \{X/a, Y/b, Z/b\}$ selezionata $Z = b$, Z/b aggiunto a S
- risultato: S (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
- $E = \{Y = a, Y = b\}$ selezionata $a = Z$, Z/a applicata a E
- $E = \{a = b\}$ selezionata $Y = a$, Y/a applicata a E
- a e b ***non unificabili*** → $\perp$

Procedura Bottom-up con Variabili

Procedura BU proposizionale estesa al DATALOG
basata sulle *istanze ground* delle clausole:

- ottenute sostituendo uniformemente nella clausola costanti alle variabili
  → **grounding**
- costanti che occorrono nella KB o nella query
  - qualora non ve ne fossero, se ne deve inventare una

Esempio — Base di conoscenza:

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

<u>tutte</u> le istanze ground: →

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

Applicando la procedura BU proposizionale si derivano atomi *ground*

- conseguenze logiche: $q(a)$, $q(b)$, $r(a)$, $s(a)$, $p(a, a)$ e $p(b, a)$

Esempio — Base di conoscenza:

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

Query ask g

- la procedura BU introduce ad hoc la nuova costante c
- grounding della base:
  - $p(c, c).$
  - $g \leftarrow p(c, c).$
- la BU proposizionale deriva $\{p(c, c), g\}$
- risposta: yes

Query ask $p(b, d)$ per **Esercizio**

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

BU: Proprietà

La procedura BU applicata al *grounding* della KB è **corretta** (*sound*):
ogni istanza di ogni regola è vera in ogni modello

- come nel caso senza variabili,
  usando l'insieme delle istanze ground delle clausole
  - *vere* perché le variabili in una clausola sono quantificate universalmente
- *converge* anche con clausole Datalog
  - numero *finito* di atomi da rendere ground
  - e a ogni iterata:
    - *un solo* atomo ground aggiunto all'insieme delle conseguenze

La procedura è **completa** per *atomi ground*:
dato g ground, se $KB \models g$ allora $KB \vdash g$

- si mostra come costruire un particolare tipo di modello:
  interpretazione di Herbrand $\langle D, \phi, \pi \rangle$
  - D *dominio* (simbolico) *fissato*: le costanti del linguaggio (in KB e g)
  - ϕ *fissato*: ogni costante denota se stessa
    - ne viene introdotta una in caso di mancanza
  - π *da definire* per i predicati: vero ogni atomo che è istanza ground di una relazione derivata dalla procedura (falsi tutti gli altri)
- tale interpretazione è un *modello* per KB ed è *minimale*
  - con il minor numero atomi veri di ogni altro modello
- se $KB \models g$, con g ground, allora g è vero nel modello minimale,
  quindi alla fine viene derivato

Esempio — Si consideri l'esempio precedente

- la procedura deriva immediatamente le istanze di imm_west date come fatti
- può quindi aggiungere gli atomi imm_east alle conseguenze:
  - $imm\_east(r103, r101)$
  - $imm\_east(r105, r103)$
  - $imm\_east(r107, r105)$
  - $imm\_east(r109, r107)$
  - $imm\_east(r111, r109)$
  - $imm\_east(r129, r131)$
  - $imm\_east(r127, r129)$
  - $imm\_east(r125, r127)$
- si possono poi aggiungere le seguenti relazioni $next\_door$:
  - $next\_door(r101, r103)$
  - $next\_door(r103, r101)$
- e anche relazioni su two_door_east, come:
  - $two\_door\_east(r105, r101)$
  - $two\_door\_east(r107, r103)$
- infine anche relazioni su $west$...

Procedura TD estesa per gestire le variabili:

- ammette istanze di regole nella derivazione

Clausola di risposta generalizzata:

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

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

- l'uso di yes consente l'*estrazione della risposta*
  - determinare quali istanze della query (con variabili) seguano logicamente da KB

La procedura TD gestisce una clausola di risposta generalizzata *corrente*

- Inizialmente, data la query q, si considera

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

  dove $V_1, \ldots, V_k$ variabili di q
  - intuitivamente: istanza di $yes(V_1, \ldots, V_k)$ vera se corrispondente istanza di q vera
- In ogni fase, l'algoritmo:
  - seleziona un *atomo* nel *corpo* della clausola di risposta corrente
  - sceglie una *clausola* di KB la cui *testa* <u>si unifichi</u> con tale atomo

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, con la clausola *scelta* nella KB

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

essendo a_1 e a unificabili da σ MGU

Nuova clausola di risposta (*risolvente*):

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

Derivazione SLD: sequenza $\gamma_0, \gamma_1, \ldots, \gamma_n$ di clausole di riposta in cui:

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

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

- γ_i ottenuta per risoluzione SLD:
    - selezionando a_1 nel corpo di γ_{i-1}
    - scegliendo della base una ***copia***[1] di

$$a \leftarrow b_1 \land \ldots \land b_p$$

      la cui testa a si unifichi con a_i tramite MGU σ
    - sostituendo a_1 con il corpo $b_1 \land \ldots \land b_p$
    - applicando σ alla clausola di risposta risultante

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

γ_n **risposta** dalla forma:

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

in una dimostrazione terminata con *successo*

- si restituirà la risposta alla query

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

  - t_i determinati dai binding delle variabili della query V_i

Algoritmo TD basato su derivazione SLD: ⥀

- *non deterministico*: si possono trovare tutte le derivazioni tentando altre scelte che portino al successo
- fallisce se tutte le scelte portano al fallimento: *nessuna derivazione*
  - scelta della clausola implementabile come *ricerca*

```
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 fail
  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 rinominando tutte
    le variabili con nuovi nomi
    Sia σ ← Unify(a_1,a)
    if σ = ⊥ then
      termina con un fallimento
    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) ←
```

Esempio — Si consideri la base precedente e la query

$$\text{ask } two\_doors\_east(R, r107).$$

- derivazione di successo con risposta $R = r111$:
  - $yes(R) \leftarrow \underline{two\_doors\_east(R, r107)}$
    - risolve con $two\_doors\_east(E_1, W_1) \leftarrow imm\_east(E_1, M_1) \wedge imm\_east(M_1, W_1)$
    - sostituzione: $\{E_1/R, W_1/r107\}$
  - $yes(R) \leftarrow \underline{imm\_east(R, M_1)} \wedge imm\_east(M_1, r107)$
    - risolve con $imm\_east(E_2, W_2) \leftarrow imm\_west(W_2, E_2)$
    - sostituzione: $\{E_2/R, W_2/M_1\}$
  - $yes(R) \leftarrow \underline{imm\_west(M_1, R)} \wedge imm\_east(M_1, r107)$
    - risolve con $imm\_west(r109, r111)$
    - sostituzione: $\{M_1/r109, R/r111\}$

(..cont.)

- **derivazione:**
  - $yes(r111) \leftarrow \underline{imm\_east(r109, r107)}$
    - risolve con $imm\_east(E_3, W_3) \leftarrow imm\_west(W_3, E_3)$
    - sostituzione: $\{E_3/r109, W_3/r107\}$
  - $yes(r111) \leftarrow \underline{imm\_west(r107, r109)}$
    - risolve con $imm\_west(r107, r109)$
    - sostituzione: $\{\}$
  - $yes(r111) \leftarrow$

nota: usate 2 istanze di $imm\_east(E, W) \leftarrow imm\_west(W, E)$

- una sostituendo $r111$ a E_2 (via R), l'altra sostituendo $r109$ a E_3
- ***selezionato*** $imm\_west(M_1, R)$, con altre ***scelte*** della clausola con cui risolvere non si sarebbe potuto completare la dimostrazione

Simboli di Funzione

Identificazione Indiretta di Individui

DATALOG richiede un nome (simbolo di costante) per ogni individuo sul quale si deve ragionare

- spesso più semplice identificare un individuo *in termini di altri individui*

Una costante per individuo
→ si può rappresentare solo un numero *finito* di individui:

- fissato alla costruzione della KB

e se si volesse ragionare su un dominio potenzialmente *infinito*?

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
            - infinite
        - più facile in termini di `anno`, `mese`, `giorno`
2. Sistema QA:
    - individui = *frasi* → troppi nomi!
    - *frasi* come sequenza di *parole*
        - vocabolario *finito*
            - più pratico
        - parole in funzione di *sillabe* | *lettere*
3. Liste (di studenti):
    - *lista* individuo con proprietà:
        - `lunghezza`, `elemento-i`,...
    - un nome per lista: poco pratico
    - liste in termini dei loro elementi

Funzioni: Sintassi

Consentono di descrivere individui indirettamente:

- in termini di altri individui

Simbolo di **funzione**: parola che comincia con una lettera minuscola

- linguaggio *esteso* definendo
  termine: variabile, costante o della forma

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

  con f è un simbolo di funzione (n-aria) e t_i termini
  - i termini compaiono nelle clausole *esclusivamente* all'interno di atomi
    - non direttamente: funzioni distinte dai predicati

Funzioni: Semantica

In ogni interpretazione

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

- ϕ estesa:
  assegna una funzione $D^n \mapsto D$ a ogni simbolo di funzione n-aria
  - per ogni termine ground specifica l'individuo denotato
  - costanti come funzioni 0-arie
    - i.e. senza argomenti

Esempio — Definire le *date*:

costanti:

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

funzioni: (definizione *estensionale*)

- ce — date della *common era*:
  $ce(Y, M, D)$ data con anno Y, mese M e giorno D
  - ad es. $ce(2021, oct, 21)$ per il `21 ottobre 2021`
- bce — date prima della *common era*

predicati:

- $month(M, N)$ vera se M mese dell'anno con numero d'ordine N
  - $month(jan, 1).$
  - $month(feb, 2).$
  - $month(mar, 3).$
  - $month(apr, 4).$
  - $month(may, 5).$
  - $month(jun, 6).$
  - $month(jul, 7).$
  - $month(aug, 8).$
  - $month(sep, 9).$
  - $month(oct, 10).$
  - $month(nov, 11).$
  - $month(dec, 12).$

(..cont.)

- $before(D1, D2)$ vera se $D1$ precede $D2$
  - $before(ce(Y1, M1, D1), ce(Y2, M2, D2)) \leftarrow Y1 < Y2.$
  - $before(ce(Y, M1, D1), ce(Y, M2, D2)) \leftarrow month(M1, N1) \wedge$
    $month(M2, N2) \wedge$
    $N1 < N2.$
  - $before(ce(Y, M, D1), ce(Y, M, D2)) \leftarrow D1 < D2.$
    - predicato $<$ rappresenta la relazione `minore_di` sugli interi
      - descritto in termini di clausole, ma spesso *predefinito* (e.g. in PROLOG)

Programmazione Logica

Qualunque funzione computabile può essere calcolata usando una KB di clausole con simboli di funzione:

- interpretabile come **programma logico** *semantica operazionale*

Il linguaggio dei programmi logici è *Turing completo*

- con un solo simbolo di funzione e una sola costante, definibile un numero *infinito* di termini/atomi ground
  - si può ragionare su un *infinito* numero di individui

Con le funzioni si possono definire **strutture dati**:

- ad es. liste, alberi

Esempio — *Albero* (con etichette)

Può servire a descrivere la sintassi di una frase in un sistema NLP

- ***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***:
  - $at\_leaf(L, T)$ vera se L è l'etichetta di una ***foglia*** dell'albero T:
    - $at\_leaf(L, leaf(L)).$
    - $at\_leaf(L, node(N, LT, RT)) \leftarrow at\_leaf(L, LT).$
    - $at\_leaf(L, node(N, LT, RT)) \leftarrow at\_leaf(L, RT).$
      - ***ricorsive***, coprono tutti i casi di albero
  - $in\_tree(L, T)$ 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
  - ci si ragiona usando *solo* funzioni e costanti
  - spesso *predefinita* (come in PROLOG, LISP,...)
- definizione ricorsiva:
  - come sequenza vuota — denotabile con una costante ad hoc, ad es. nil
  - come elemento seguito da una lista — tramite la funzione $cons(Hd, Tl)$
    - Hd primo elemento (testa) e Tl resto della lista (coda)
      - ad es. lista con a, b, c: $cons(a, cons(b, cons(c, nil)))$
- *uso*: attraverso predicati come
  - $append(X, Y, Z)$ vera 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

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
  - sotto-linguaggio utile in termini *pratici*
- operatori aggiuntivi: disgiunzione e quantificazione esplicite
- *primo ordine*: quantificazione degli individui del dominio

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

Decidibilità

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)

Domanda — E il DATALOG?

Procedure di Dimostrazione con Funzioni

Estensioni delle procedure su DATALOG per gestire termini con funzioni

⚠ numero di termini possibili *infinito*

Procedura BU: grounding delle clausole + procedura BU proposizionale

- una clausola potrebbe generare un'*infinita* sequenza di conseguenze:

Proprietà

- **fairness**: assicura un criterio di selezione delle clausole *equo* (*fair*)
  - ogni clausola della KB prima o poi dev'essere selezionata
- procedura *completa* con un criterio di selezione fair:
  ogni conseguenza sarà generata

Esempio — KB 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.

- per derivare una nuova conseguenza ad ogni iterata
- non selezionando 3 e 4, non si potrà mai derivare a o b

Problema di **starvation**: clausole sistematicamente trascurate

Procedura TD: si comporta come in DATALOG

- l'*unificazione* segue ricorsivamente la struttura dei termini in profondità
  - adesso anche applicazioni di funzioni a più livelli
- modifica all'algoritmo:
  una variabile X non si unifica con termini t in cui già occorra (se $t \neq X$)
- serve il **controllo di occorrenza** (*occurs-check*)
  - altrimenti, procedura non più sound/corretta

Esempio — KB con una sola clausola: $\{lt(X, s(X)).\}$

Interpretazione intesa:

- predicato lt “minore di” per il dominio degli interi
  - funzione *successore* $s(X)$: per denotare l'intero $X + 1$

La query ask $lt(Y, Y)$. dovrebbe fallire

- falsa nell'interpretazione intesa
  - nessun numero è minore di se stesso

Invece, ammettendo l'unificabilità di X e $s(X)$, nessun fallimento!

Algoritmo di unificazione con simboli funzione e *occurs-check*:

- avendo selezionato $\alpha = \beta$, restituisce $\perp$
  se α variabile e β termine che la contiene, diverso da α stesso
  (o viceversa)
- omettendo il controllo (possibile in PROLOG) per aumentarne l'efficienza, procedura non sound!

Procedura non sound

potrebbe derivare g anche quando questo sia falso in un modello di KB

Esempio — Considerate le clausole

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

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

- $yes(F, L) \leftarrow append(F, c(L, nil), c(l, c(i, c(s, c(t, nil)))))$
  - risolvendo con una copia della 1. con variabili ridenominate con 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 una copia della 1. con variabili ridenominate con indice 3
  - 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 una copia della 1. con variabili ridenominate con indice 3
  - sostituzione: $\{X_2/c(s, X_3), Y_3/c(L, nil), A_3/s, Z_3/c(t, nil)\}$
- $yes(c(l, c(i, c(s, X_3))), L) \leftarrow append(X_3, c(L, nil), c(t, nil))$

Ora applicabili entrambe le clausole di $append$:

- 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: $\{X_3/c(t, X_4), Y_4/c(L, nil), A_4/t, Z_4/nil\}$
- $yes(c(l, c(i, c(s, X_3))), L) \leftarrow append(X_4, c(L, nil), nil)$

La dimostrazione *fallisce*:
le teste delle clausole non si unificano con l'atomo nel corpo

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

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

Liste: Notazione Prolog

- $[\,]$ lista *vuota*, o nil
- $[E \mid R]$ lista con primo elemento E e resto della lista R, o $cons(E, R)$
- *semplificazione*: $[X \mid [Y]]$ si può scrivere $[X, Y]$, con Y sequenza di valori
  - ad es.: $[a \mid [\,]]$ → $[a]$; $[b \mid [a \mid [\,]]]$ → $[b, a]$; $[a \mid [b \mid C]]$ → $[a, b \mid C]$

Esempio — casi precedenti nella notazione:

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

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

risposta: $F = [l, i, s], L = t$

Dimostrazione identica: rinominate una funzione e una costante

Uguaglianza

A volte utile usare più termini per denotare uno *stesso* individuo:

- ad es. 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 es. nome distinto per insegnamenti differenti

Spesso non si sa se due nomi denotino lo stesso individuo:

- ad es. se la `persona del turno delle 8` sia la stessa `persona del turno delle 12`
- nelle procedure di ragionamento viste, tutte le risposte valide a prescindere dal fatto che i termini denotino gli stessi individui

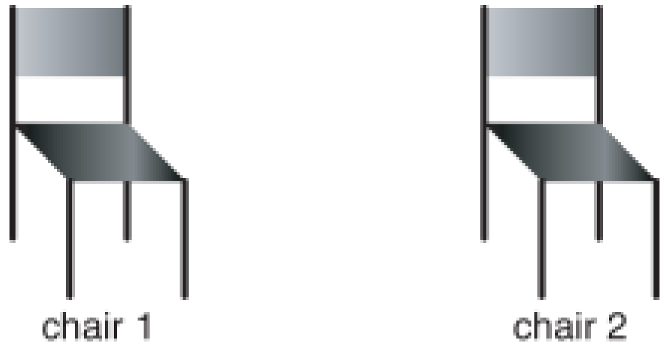
Uguaglianza con Predicato =

Predicato speciale "$=$" (*sintassi infissa*)
con interpretazione intesa standard indipendente dal dominio

$$t_1 = t_2$$

- termine t_1 **uguale** al termine t_2:
  - atomo vero in un'interpretazione I sse t_1 e t_2 mappati sullo stesso individuo
- il predicato $=$ <u>non</u> indica semplice similarità o identicità ma ***identità***:
  - $a = b$ indica che ci sono due nomi (costanti) per un solo individuo,
    <u>non</u> due individui simili (o indistinguibili) del dominio

Esempio — Si consideri il caso in figura:



- non è vero che $chair1 = chair2$
  - pur risultando identiche sotto tutti gli aspetti
- se non si rappresenta la precisa ***posizione***, non sono distinguibili
  - può essere vero che $chairOnRight = chair2$
    ma <u>non</u> che la sedia a destra sia *simile* a $chair2$: essa è $chair2$!

Ammettere Asserzioni d'Uguaglianza

Se non si ammettono atomi con $=$ (uguaglianze) nelle teste delle clausole, in una interpretazione un termine può essere uguale solo a se stesso

Utile poter asserire che termini distinti denotino lo *stesso individuo*

- ad es. $chairOnRight = chair2$

Per poter di derivare uguaglianze → clausole con tali atomi nella testa

1. *assiomatizzare* l'uguaglianza come un qualunque altro predicato
2. definire *procedure speciali* d'inferenza per l'uguaglianza

Assiomatizzare l'Uguaglianza

Assiomatizzazione di base per le variabili

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

Per funzioni e predicati → **schema di assiomi** da istanziare:

- sostituendo un termine con uno uguale,
  il valore di una funzione/predicato <u>non cambia</u>:
  - *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:

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

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

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

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

Problemi con tali assiomi

- ragionamento *meno efficiente* se inclusi esplicitamente nella KB
- *terminazione non garantita* di interpreti TD + ricerca in profondità
  - ad es. l'assioma di *simmetria* può causare un ciclo infinito
    a meno di controlli sulla ripetizione di sotto-goal

Procedure Speciali di Ragionamento con l'Uguaglianza

Paramodulazione — implementa l'uguaglianza estendendo la procedura

Idea: se $t_1 = t_2$, ogni occorrenza di t_1 può essere sostituita con t_2

- uguaglianza trattata come *regola di riscrittura*
  - sostituisce eguali con eguali
- per ogni individuo conviene fissare una *rappresentazione canonica*:
  - termine sul quale vengono mappate tutte le altre sue rappresentazioni

Esempio — rappresentazione dei *numeri*:

- molti termini per rappresentare lo stesso numero:
  - ad es. `4*4`, `13+3`, `273-257`, `2^4`, `4^2`, `16`
- *rappresentazione canonica*: sequenza di cifre in base 10 (tipicamente)

Esempio — *matricole*: rappresentazione canonica per ogni studente

- distingue studenti diversi con lo stesso nome
- forme diverse per il nome di una stessa persona
  possono essere mappate sulla sua matricola
  - `"Cristiano Ronaldo dos Santos Aveiro"`, `"Cristiano Ronaldo"`, `"Ronaldo, Cristiano"`, `"C. Ronaldo"`, `"Ronaldo, C."`, `"Ronaldo"` … → matr. `"CR0007"`

Convenzione alternativa all'assiomatizzazione dell'uguaglianza:

- nella semantica data, ϕ non necessariamente iniettiva:
  termini *diversi* possono denotare lo stesso individuo *o* individui distinti

Assunzione di Unicità dei Nomi (*Unique Names Assumption*, UNA):
termini ground diversi denotano individui distinti (tipica del contesto DB)

- per ogni coppia di termini ground distinti t_1 e t_2, si assume che

$$t_1 \neq t_2$$

  dove $\neq$ significa “non uguale a”
- sotto UNA, un atomo con $\neq$ può occorrere nel *corpo* delle clausole:
  - la si può imporre aggiungendo agli assiomi la clausola $X = X.$
    - si può essere uguali solo a se stessi

Esempio — Si consideri un DB di n studenti in cui ognuno ha due insegnamenti a scelta nel piano di studi:

- si sa che uno studente ha superato $math302$ e $psyc303$
  - avrà superato i 2 esami richiesti solo se $math302 \neq psyc303$ ossia se le costanti denotano insegnamenti differenti
  - occorre conoscere quali codici denotino insegnamenti diversi
- invece di $n(n-1)/2$ assiomi di disuguaglianza, si adotta la convenzione: ***codici distinti denotano corsi distinti***

Assiomatizzazione della UNA

Si aggiunge a quello per l'uguaglianza il seguente *schema di assiomi per 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

Osservazioni

- termini ground *identici* sse si unificano
  - non vale per termini non ground
    - ad es. $a \neq X$ ha istanze vere nelle quali X ha un certo valore b, e un'istanza falsa in cui X ha il valore a
- UNA utile a integrare DB, senza dover esplicitare distinzioni
  - come $kim \neq sam$, $kim \neq chris$, $chris \neq sam$, ...
- UNA *inappropriata* in certi casi
  - ad es., $2 + 2 \neq 4$ oppure $clark\_kent \neq superman$

Procedura Top-Down con UNA

Per incorporare la UNA: disuguaglianza come predicato speciale

⚠ *problema*: troppe diversità tra individui da specificare

Casi possibili nel dimostrare $t_1 \neq t_2$

1. t_1 e t_2 non si unificano → $t_1 \neq t_2$ ha ***successo***
    - ad es. per $f(X, a, g(X)) \neq f(t(X), X, b)$
2. t_1 e t_2 identici → $t_1 \neq t_2$ ***fallisce***
    - ad es. per $f(X, a, g(X)) \neq f(X, a, g(X))$
3. successo per alcune istanze, ma fallimento per altre
    - ad es. $f(W, a, g(Z)) \neq f(t(X), X, Y)$
        - $\{X/a, W/t(a), Y/g(Z)\}$ loro MGU
        - istanze ground compatibili con l'MGU → ***fallimento***
        - istanze non compatibili con l'MGU → ***successo***
    - evitare di enumerare tutte le istanze di successo: troppe!
4. ***NB*** per coppie di termini ground, unici casi possibili: 1 e 2

TD estesa per incorporare la UNA
per $t_1 \neq t_2$ si considerano i casi:

1. → *successo*
2. → *fallimento*
3. → da *posticipare* in attesa di altri atomi-goal 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 query ha *successo* se non ci sono altri atomi da selezionare
     e nessuno dei casi precedenti è applicabile
     - c'è sempre un'istanza che ha successo,
       quella che assegna a ogni variabile una costante distinta, non già usata
     - vanno interpretate con cautela le variabili libere nella *risposta*:
       sarà vera *solo* per alcune loro istanze, non per tutte

Esempio — Definizione di studente che abbia superato due esami:

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

Query: ask $passed\_two\_courses(sam)$

- la verità di $C_1 \neq C_2$ non può essere determinata → ***posticipato***
- selezionando $passed(sam, C_1)$, che lega $engl101$ 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 $C_1 \neq C_2$ sono legate allo stesso valore → ***fallimento***
- scegliendo l'altra clausola per $grade(sam, C_2, M)$,
  sostituzione $\{C_2/phys101, M/89\}$
  - variabili in $C_1 \neq C_2$ legate a cost. distinte → ***successo***
- resta solo $89 > 50$ (vera) → la query ha ***successo***

⚠ test di disuguaglianza sempre come ultima chiamata (ultimo sotto-goal) ?

Differimento dei goal

1. spesso *più efficiente*
   - es. $C_1 \neq C_2$ differito può essere testato prima di controllare se $87 > 50$
   - sebbene questo test possa essere veloce, spesso altri test evitabili anticipando i test di disuguaglianze che saranno violate
2. 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 fallire

Assunzione di Conoscenza Completa

Assunzione di Conoscenza Completa

L'assunzione di conoscenza completa vista in precedenza assume che un enunciato che *non segua* logicamente dalla KB sia *falso*

Per estenderla ai programmi logici con variabili e funzioni serve considerare:

- gli *assiomi dell'uguaglianza*
- la proprietà di *chiusura del dominio* (*domain closure*)
- una nozione più sofisticata del *completamento*

Ciò definisce una forma di **negazione per fallimento** [NAF]

Esempio — Relazione $student$ definita:

- $student(mary).$
- $student(john).$
- $student(ying).$

Per l'assunzione di conoscenza completa sarebbero gli ***unici*** studenti:

$$student(X) \leftrightarrow X = mary \vee X = john \vee X = ying.$$

ossia

- se X è $mary$, $john$ o $ying$, allora è uno studente ***e***
- se X è studente allora dev'essere uno dei tre
  - kim non può essere uno studente
  - $\neg student(kim)$ richiede di dimostrare
    $kim \neq mary \wedge kim \neq john \wedge kim \neq ying$ → serve la UNA

L'assunzione di conoscenza completa include la UNA:

- perciò si devono includere gli schemi di *assiomi* per *uguaglianza* e *disuguaglianza*

Forma normale di Clark della clausola

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

è la clausola

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

dove:

- se clausola atomica allora B è $true$
- $W_1, \ldots, W_m$ variabili originarie della clausola
- $V_1, \ldots, V_k$ nuove variabili per la clausola

Si pongano tutte le clausole per un dato p in forma normale, con lo stesso insieme di *nuove variabili*:

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

equivalenti a

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

- enunciato equivalente all'insieme di clausole originario

Completamento di Clark *del predicato* p:

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

dove nei corpi la **negazione per fallimento** ($\sim$)
è sostituita dalla negazione logica standard ($\neg$)

Semantica:

- $p(V_1, \ldots, V_k)$ vero sse vero almeno uno dei corpi B_i

Completamento di Clark *di una KB* — comprende i completamenti di ogni predicato e degli assiomi di uguaglianza e disuguaglianza

Esempio — Per le clausole:

- $student(mary).$
- $student(john).$
- $student(ying).$

forma normale:

- $student(V) \leftarrow V = mary.$
- $student(V) \leftarrow V = john.$
- $student(V) \leftarrow V = ying.$

equivalente a

$$student(V) \leftarrow V = mary \vee V = john \vee V = ying.$$

completamento di $student$:

$$\forall V \; student(V) \leftrightarrow V = mary \vee V = john \vee V = ying.$$

Esempio — Definizione ricorsiva:

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

In forma normale:

- $passed\_each(L, S, M) \leftarrow L = [\,].$
- $passed\_each(L, S, M) \leftarrow \exists C \exists R\, L = [C \mid R] \wedge passed(S, C, M) \wedge$
  $passed\_each(R, S, M).$
    - rimosse le uguaglianze dei renaming delle variabili
    - e rinominate opportunamente le variabili

Completamento:

$$\begin{aligned} \forall L \forall S \forall M\, passed\_each(L, S, M) \leftrightarrow{} & L = [\,] \vee \\ & \exists C \exists R\, (L = [C \mid R] \wedge \\ & passed(S, C, M) \wedge \\ & passed\_each(R, S, M)). \end{aligned}$$

Esempio — Si consideri una KB con:
$course(C)$ vero se C è un corso; $enrolled(S,C)$ vero se S è iscritto a C

- <u>senza assunzione di conoscenza completa</u>, non si può definire $empty\_course(C)$: vero se non ci sono iscritti al corso C
  - c'è sempre un modello della KB in cui ogni corso abbia qualche iscritto
- con la NAF, la definizione sarebbe:
  - $empty\_course(C) \leftarrow course(C) \ \wedge \sim has\_enrollment(C).$
  - $has\_enrollment(C) \leftarrow enrolled(S,C).$
- completamento:
  - $\forall C \ empty\_course(C) \leftrightarrow course(C) \ \wedge \neg has\_enrollment(C).$
  - $\forall C \ has\_enrollment(C) \leftrightarrow \exists S \ enrolled(S,C).$
- occorre cautela nell'inclusione di variabili libere in atomi con la NAF:
  - in genere cambiano il significato inteso
- $has\_enrollment$ serve ad evitare di avere una variabile libera in una negazione tipo $\sim enrolled(S,C)$

Esempio — Supponendo di definire $empty\_course$ con:
$empty\_course(C) \leftarrow course(C) \wedge \sim enrolled(S, C).$

- completamento: $\forall C \; empty\_course(C) \leftrightarrow \exists S \; course(C) \wedge \neg enrolled(S, C).$
  errato: infatti data una KB con
  - $course(cs422).$
  - $course(cs486).$
  - $enrolled(mary, cs422).$
  - $enrolled(sally, cs486).$

    falsa l'istanza della clausola precedente:
    $empty\_course(cs422) \leftarrow course(cs422) \wedge \sim enrolled(sally, cs422)$
    corpo vero e testa falsa ← $cs422$ ha iscrizioni
    - conflitto con la verità delle clausole istanziate dalla KB
- difatti il completamento (errato) equivale a
  $\forall C \; empty\_course(C) \leftrightarrow course(C) \wedge \neg \exists S \; enrolled(S, C).$ ovvero
  $\forall C \; empty\_course(C) \leftrightarrow course(C) \wedge \forall S \; \neg enrolled(S, C).$

TD con variabili e funzioni + NAF: problematica

Esempio — Si consideri la KB con le clausole:

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

e la query ask $p(X)$

- Secondo la semantica, una sola risposta con $X = d$
  - $r(d)$ e $\sim q(d)$ vere, quindi $p(d)$ segue logicamente dalla KB
- Ma la dimostrazione di $p(X)$ con TD *fallisce*:
  - selezionando $\sim q(X)$ riesce a dimostrare $q(X)$ con $\{X/a\}$
  - mentre $p(X)$ vero se $\{X/d\}$, essendo $\sim q(d)$ e $r(d)$
- La procedura così risulta *incompleta* e anche *non corretta*:
  - se si aggiungesse $s(X) \leftarrow \sim q(X).$, dal fallimento di $q(X)$ si arriverebbe a poter derivare *erroneamente* $s(X)$

NAF in Presenza di Variabili Libere

Problemi potrebbero sorgere a causa delle *variabili libere nei goal negati*

- la procedura dovrebbe *posticipare* il sotto-obiettivo *negato* 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 blocca* (**flounders**)
  - non si potrà concludere *nulla* su tale goal

Esempio — Si considerino la KB seguente e la query: ask $p(X)$.:

KB:

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

Completamento:

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

Sostituendo $X = a$ ad $r(X)$:

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

quindi:

- $p(X) \leftrightarrow X = a$

Risposta: $X = a$

- ma il differimento del goal non aiuterebbe a trovarla
- la procedura dovrebbe *analizzare i casi* nei quali il goal è fallito

argomento avanzato

Riferimenti

[1] D. Poole, A. Mackworth: *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press [Ch.13]
[2] D. Poole, A. Mackworth, R. Goebel: *Computational Intelligence: A Logical Approach*. Oxford University Press
[3] S. J. Russell, P. Norvig: *Artificial Intelligence* Pearson. 4th Ed. - cfr. anche ed. Italiana
[4] J. Sowa: *Knowledge Representation: Logical, Philosophical, and Computational Foundations* Brooks Cole/Cengage
[5] R.A. Kowalski: *Logic for problem solving, revisited*. BoD (2014) [online]
[6] L.S. Sterling and E.Y. Shapiro: *The art of Prolog: advanced programming techniques*. 2nd edition, MIT Press (1994)
[7] J.W. Lloyd: *Foundations of logic programming*. 2nd edition, Symbolic Computation Series, Springer-Verlag. (1987)
[8] K.L. Clark: *Negation as failure*. In Logic and Databases, H. Gallaire and J. Minker (Eds.), pp. 293-322. (1978)
[9] I. Bratko: *Prolog programming for artificial intelligence*. Pearson (2001)
[10] C. Date: *Database in Depth: Relational Theory for Practitioners*. O'Reilly Media (2005)
[11] P.Blackburn, J. Bos & K. Striegnitz: *Learn Prolog Now!*. College Publications (2012) [online]

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

[∢] consigliata la lettura
[versione] 3/11/2022, 18:08:15

Figure tratte da [1] salvo diversa indicazione