**RAPPRESENTAIONE E RAGIONAMENTO RELAZIONALE**

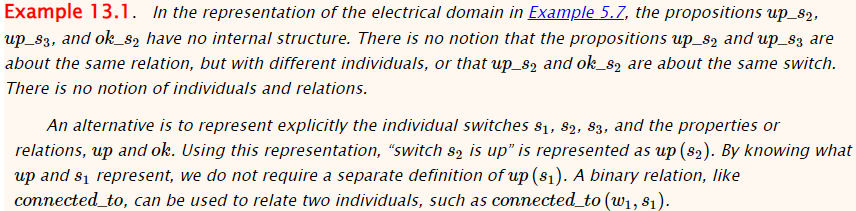
Focalizzeremo la nostra attenzione a come rappresentare **individui**(cose, oggetti, etc) e le **relazioni**  che sussistono tra tali individui. Le rappresentazioni che utilizzeremo sono forme compatte che ci permettono di descrivere il mondo con maggiore efficienza rispetto alle singole caratteristiche ”features”.

# **Sfruttare la struttura relazionale:**

La rappresentazione del dominio ci permette di sfruttare le caratteristiche della rappresentazione stessa per migliorare e rendere più efficiente il ragionamento in tale mondo rappresentato, di fatti abbiamo visto che il passaggio da una rappresentazione a stati a una con features ha permesso una maggiore efficienza sul ragionamento, ora cercheremo di sfruttare un ‘altra rappresentazione ovvero quella tra individui e relazioni per trarne giovamento nel ragionamento.

* **Individui:** sono oggetti, entità e cose presenti nel dominio
* **Relazioni:** specificano quello che è vero a riguardo di tali individui e riguardano  **proprietà, proposizioni e relazioni**

Nel ragionamento proposizionale non avendo la nozione di individuo non possiamo essere certi che una proposizione riguardi uno stesso individuo ovvero quello che è rappresentato nel mondo reale come non possiamo essere certi che sia diverso se ad esempio vogliamo modellare tale diversità sulle proposizioni. Il ragionamento relazionale ci permette di tenere gli individui separati dalle relazioni nelle quali essi sono coinvolti.



Modellare il dominio tramite individui e relazioni rispetto alle caratteristiche che gli elementi hanno nel mondo reale ci permette:

* Una rappresentazione naturale, mantenendo la struttura dell’individuo separata dalle proprietà che esso ha
* Un agente intelligente potrà modellare il dominio senza conoscere a priori le caratteristiche dell’individuo o quanti individui ci sono
* Il ragionamento può essere condotto senza tenere in considerazione individui in particolare (ovvero l’agente può sapere che ci sono individui che hanno una proprietà e la verificano senza sapere chi essi siano).
* Un agente potrebbe agire senza conoscere quali siano o saranno le caratteristiche
* Permette di ragionare su infiniti individui

**CONCETTUALIZZAZIONE:**  è il processo tramite il quale il progettista di una KB mappa i simboli che ha in mente con gli individui e le relazioni che tali simboli rappresentano.

Con tale rappresentazione la correttezza di una KB è definita dalla sua semantica e no da un particolare algoritmo che andrà a dimostrare se una data query è conseguenza logica della KB.

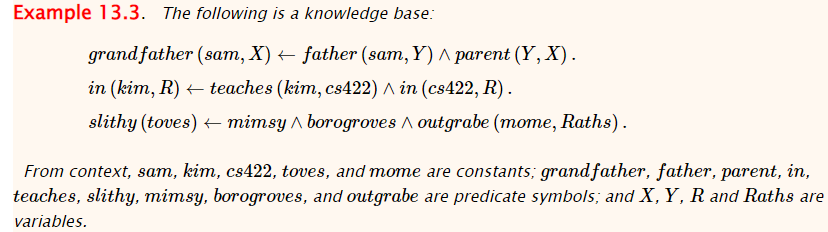
**DATALOG**

**[un linguaggio per regole relazionali]**

La sintassi di Datalog vede una parola come una sequenza di letterali, cifre o underscores, in particolare:

* Una **variabile** è una parola che inizia con la maiuscola
* Una **costante** è una parola che inizia con minuscola o un numero
* Un **simbolo di predicato** è una parola che inizia per minuscola ed è distinta dalla constante grazie al contesto
* Un **termine**  può essere una **variabile** oppure una **costante**
* Un **simbolo atomico** o semplicemente un **atomo** è un **predicato n-ario** nella seguente forma: p or p(t1, ….., tn) dove ogni ti è un **termine** e prende il nome di **argomento del predicato.**
* Resta identico il significato e la nozione già vista di clausola definita, regola, clausola atomica o fatto, query e di KB.
* Una **espressione**  è o un termine, un atomo, una clausola definita oppure una query.

Ex di una KB in datalog:



In fine, un espressione è detta **ground** se non contiene variabili.

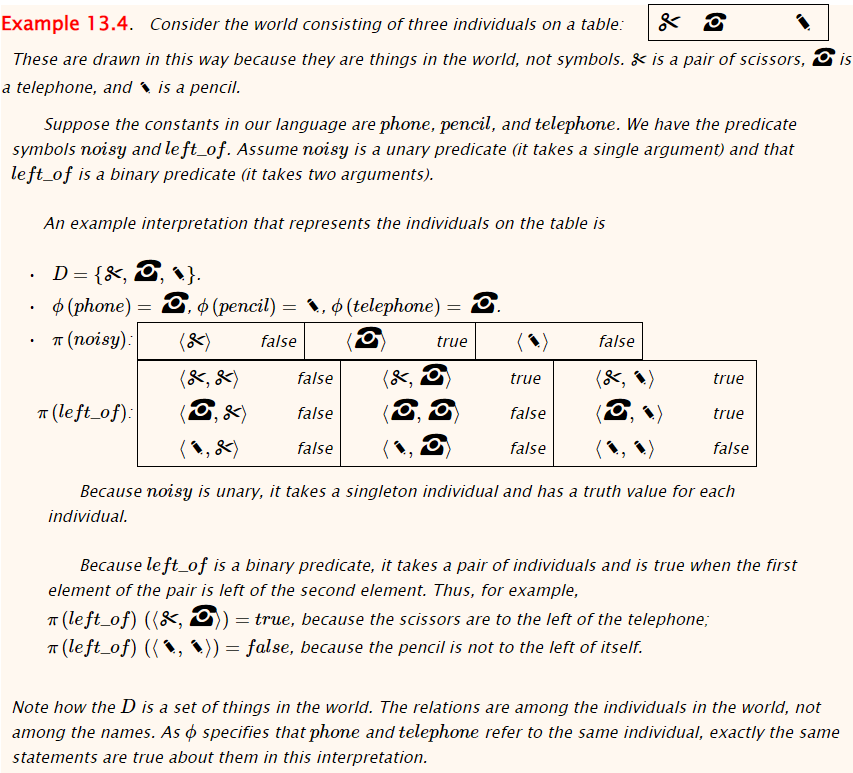
# **Semantica di ground DataLog:**

Un interpretazione è una tripla del tipo: I = ⟨ D, ϕ, π⟩ dove abbiamo:

* D è un insieme non vuoto chiamato dominio e gli elementi del dominio sono gli individui che vogliamo rappresentare nel mondo.
* ϕ è un mapping che assegna ad ogni costante un elemento specifico del dominio
* π è un mapping che assegna ad ogni predicato n-ario una funzione con dominio Dn e codominio {True, False}

per cui con ϕ (c) si intende la costante c che denota l’individuo del dominio D che risulta proprio individuato come ϕ (c) .

π (p) denota una relazione espressa con il simbolo di predicato p che risulterà vera o false sulla tupla di individui considerati.



Un atomo può essere vero o falso in una Interpretazione e in particolare l’atomo p(t1, ….., tn ) è vero in I se π (p)(<t’1, ….., t’n >) = True dove ogni  è un individuo denotato dal termine ti , altrimenti è falso in I.

Il concetto di  **modello** e **conseguenza logica**  rimangono gli stessi visti per il calcolo proposizionale.

# **Interpretazione delle variabili in DataLog:**

Quando una variabile compare in una clausola, la clausola sarà vera nell’interpretazione solo se la clausola risulterà vera per ogni possibile valore che tale variabile può assumere. Per definire formalmente una variabile, definiamo un **assegnazione di variabile**  come ρ di modo che ad ogni termine nell’interpretazione sarà denotato da un individuo del dominio ove se il temine è una cosante allora utilizzeremo ϕ (termine) per ottenere l’individuo del dominio denotato se invece il termine risulta una variabile allora useremo ρ(termine) per recuperare l’individuo del dominio cercato.

Per cui una clausola in un interpretazione risulta vera se lo è per tutti i possibili assegnamenti che possono essere fatti sulle variabili di tale clausola e per questo si dice che le variabili sono **quantificate universalmente.**

Le clausole definite rendono la quantificazione universale delle variabili implicita; tuttavia, in alcuni contesti potrebbe risultare comodo rendere la quantificazione universale esplicita e per farlo si utilizzano i 2 quantificatori logici universale ∀ ed esistenziale ∃.

* ∀ X p(X) ∀, read “**for all** X, p(X)” means p(X) is true for every variable assignment for X. X is said to be a **universally quantified**.
* ∃X p(X), read “**there exists** an X such that p(X)” means p(X) is true for some variable assignment for X. X is said to be **existentially quantified**.

Esempi :

la clausola P(X) ← Q(X,Y) significa in forma esplicita :

∀X ∀Y (P(X) ← Q(X,Y)) la quale risulta equivalente a:

∀X (P(X) ← ∃Y Q(X,Y))

# **La visione umana della semantica:**

Una metodologia di progettazione di una KB prevede che il progettista compia i seguenti 5 step:

**step1:** scelta del dominio che si vuole rappresentare, una volta scelto si dovranno specificare gli individui appartenenti al dominio D.

**step2:** associare le costanti del linguaggio agli individui del dominio.

**Step3:** per ogni relazione che si vuole rappresentare, bisogna associare un simbolo di predicato nel linguaggio del ragionamento. L’associazione dei simboli con il loro significato forma l’interpretazione intesa.

**Step4:** si scrivono le clausole che sono vere nell’interpretazione intesa ovvero effettuiamo l’assiomatizzazione del dominio e quindi scriviamo gli assiomi che andranno a costituire la KB.

**Step5:** Infine è possibile interrogare la KB ponendo queries e controllando che il sistema risponda correttamente.

In questo processo il progettista non comunica nessuna semantica e nessuna interpretazione alla macchina sino a quando non raggiunge lo **step4** ovvero l’assiomatizzazione.

# **Queries con variabili:**

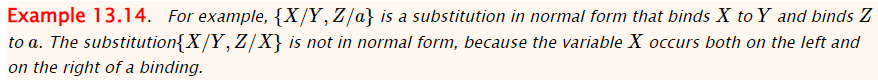
Le queries che presentano variabili permettono di interrogare la KB per scoprire quali individui rendono la query vera. Un **istanza di query** è ottenuta sostituendo i termini alle variabili presenti nella query, in particolare ogni occorrenza della variabile (ovvero tutte le volte che nella query compare la stessa variabile) essa sarà sostituita dal termine. Una risposta ad una query, quindi, risulta o un istanza che risulti conseguenza logica della KB oppure “no” che sta a significare che non ci sono istanze della query che seguono logicamente dalla KB.

**DIMOSTRAZIONI E SOSTITUZIONI**

Sia la BU che TD come procedure di dimostrazioni possono essere estese nel linguaggio Datalog, l’estensione comporta che durante il processo di dimostrazione per provare una clausola essa sia provata più volte ovvero ogni volta per ogni istanza data dalla variabile libera.

# **Istanze e sostituzione:**

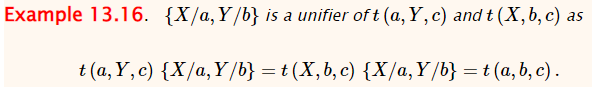
Un istanza di una clausola è data dalla sostituzione delle variabili con lo stesso termine. Una sostituzione è un insieme del tipo: { V1/t1 , … , Vn/tn } dove ogni Vi è una variabile distinta e ogni ti è un termine e la coppia Vi/ti è detta **binding** (combinazione). Diciamo che una sostituzione è in **forma normale**  se nessuna delle Vi appartenenti alla sostituzione ricompare in uno dei termini delle combinazioni ovvero in un tj.



L’applicazione di una sostituzione σ = { V1/t1 , … , Vn/tn } ad una espressione e si scrive nella seguente forma eσ, ed essa risulta essere la stessa espressione con la differenza che ogni occorrenza delle variabili Vi è sostituita con il corrispettivo termine di binding ti , tale espressione è chiamata  **istanza di** e, se essa non contiene nessuna variabile è detta **ground istanza di** e**.**

p(a, X) {X/c} = p(a, c) questo è un esempio di istanza ground dopo la sostituzione.

Una sostituzione è detta **unificatore** delle espressioni e1 ed e2se **e1σ** è identica a **e2σ**. Ovvero se quando una volta applicata la stessa sostituzione **σ** si ha che le due espressioni risultanti siano identiche.

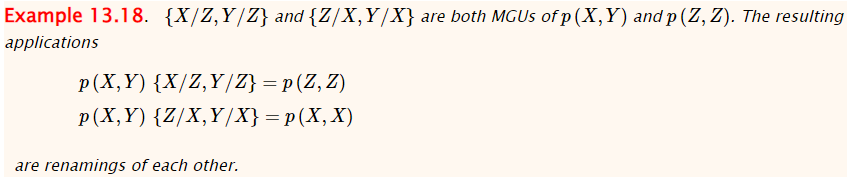


Una sostituzione **σ** è il  **most general unifier MGU**  delle espressione **e1** e **e2** se:

* **σ** è un  **unificatore**  delle 2 espressioni
* data una sostituzione **σ’** è anche questa  **unificatore** per le 2 espressioni e in particolare **eσ’** risulta che è un istanza **eσ** per ogni espressione **e.**

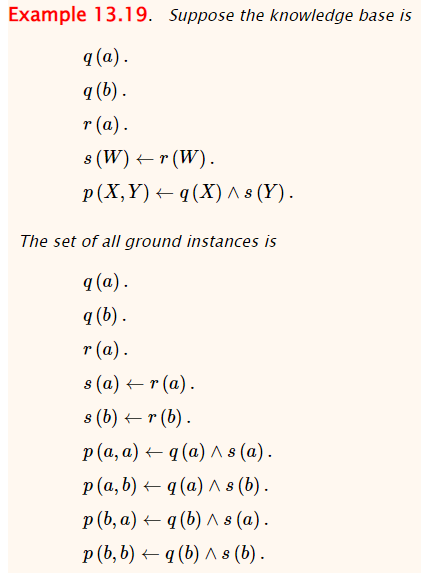
Una espressione **e1** è detta **ridenominazione**  di **e2** se differiscono solo per il nome delle variabili e in tal caso sono entrambe istanze l’una dell’altra.

Se due espressioni hanno un **unificatore** allora esse hanno almeno un **MGU,**  le espressioni risultanti dall'applicazione delle **MGU** alle espressioni sono tutte ridenominazioni reciproche ovvero se **σ** e **σ’** sono sostituzioni risultanti dello stesso **MGU** alle 2 espressioni **e1** e **e2** allora **e1σ** è una ridenominazione di **e2σ**.



# **Procedura di dimostrazione Bottom-up con variabili:**

L’algoritmo di dimostrazione BU può essere applicato a clausole Datalog in particolare a istanze ground che saranno ottenute tramite la sostituzione di costanti alle variabili presenti nella clausola. Nel caso in cui non siano state definite costanti nella KB o nella query occorrerà inventarne una.



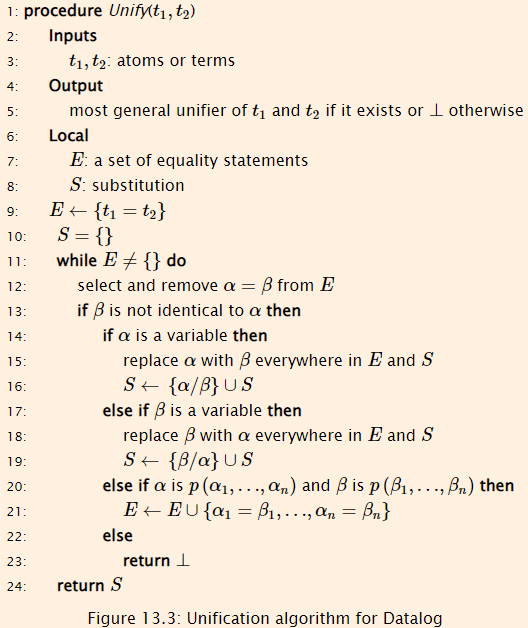
Le costanti presenti nella KB (oppure nella query) sono usate per effettuare le sostituzione al fine di ottenere istanze ground. La procedura risulta completa per gli **atomi ground**.

L’interpretazione di **Herbrand** è una interpretazione nella quale il dominio è simbolico ed è costituito da tutte le costanti del linguaggio, in questa interpretazione ogni costante denota se stessa (come individuo del dominio), questo significa che per un dato programma avremo D e ϕ fissati e l’unico da essere definito sarà π.

Se consideriamo l’interpretazione di **Herbrand** nella quale gli atomi veri sono le istanze ground delle relazioni che eventualmente sono derivate dall’algoritmo di dimostrazione BU, allora possiamo constatare che l’interpretazione di **Herbrand** risulta essere un modello e in particolare come nel caso delle variabili libere, l’interpretazione di **Herbrand**  risulta essere il  **minimo modello**  ovvero il modello con il minor numero di atomi veri. Questo comporta che se KB ⊧ g allora g è vera nel modello minimale ed è eventualmente derivata dalla procedura.

# **Unificazione:**

Il problema della **unificazione** consiste nel trovare il  **MGU** per 2 atomi o termini e l’algoritmo che permette di trovarlo è il seguente:



Nell’algoritmo sottolineamo la linea 20 nella quale quando α e β sono simboli di predicato se essi non hanno lo stesso numero di argomenti ovvero di termini porteranno al fallimento dell’algoritmo ovvero non avranno un MGU.

**RISOLUZIONE DEFINITA CON VARIABILI:**

La procedura di dimostrazione TD può essere estesa per gestire le variabili permettendo che le istanze delle regole siano usate nel processo di derivazione.

Una **clausola generalizzata di risposta**  è della forma :

yes( t1, …, tk ) ← a1 ∧ a2 ∧ … ∧am

dove t1, …, tk sono termini e a1 ∧ a2 ∧ … ∧am sono atomi e l’uso dell’atomo speciale yes permette l’estrazione della risposta andando a determinare quali istanze della query con variabili sono conseguenze logiche della KB.

L’algoritmo TD lavorerà con una clausola generalizzata di risposta che inizialmente per la query q sarà del tipo:

yes( V1, …, Vk ) ← q

ad ogni passo dell’algoritmo si seleziona un atomo presente nel corpo della clausola generalizzata di risposta e una volta selezionato si andrà a scegliere una clausola presente nella KB la cui testa possa essere unificata con l’atomo precedentemente selezionato.

Quindi la **risoluzione SLD** per la clausola generalizzata di risposta sarà processata nel seguente modo:

yes( t1, …, tk ) ← a1 ∧ a2 ∧ … ∧am

sarà selezionato l’atomo a1 e dopo di ché sarà scelta una clausola presente nella KB la cui testa sia unificabile con l’atomo selezionato, supponiamo che tale clausola sia:

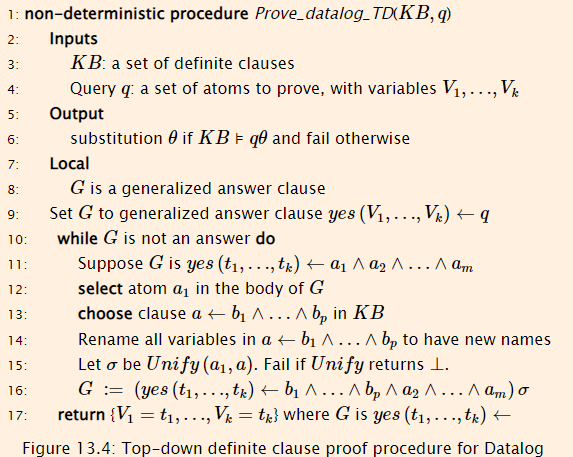
a ← b1 ∧ … ∧ bp

e che a1 e a abbiano come **MGU σ** allora la clausola di risposta sarà:

( yes (t1 , … , tk ) ← b1 ∧ … ∧ bp ∧ a2 ∧ … ∧ am ) σ

Dove il corpo della clausola scelta dalla KB va a sostituire l’atomo selezionato. Ne consegue che una **derivazione SLD** è una sequenza di clausole generalizzate di risposta γ0, γ1, ……, γn dove abbiamo che:

* γ0 è una clausola di risposta corrispondente alla query
* γi è ottenuta selezionando una atomo del corpo nella clausola generalizzata di risposta γi -1 , e andando a scegliere una clausola presente nella KB la cui testa si unifichi con l’atomo selezionato precedentemente, infine si andrà a sostituire nel corpo di γi -1 l’atomo selezionato con il corpo della clausola scelta per l’unificazione ottenendo γi.
* Alla fine, γn sarà una risposta se sarà della forma: yes( t1, …, tk ) ← e l’algoritmo TD ritornerà come soluzione la sostituzione utilizzata per derivare la risposta ovvero V1 = t1 , …, Vk = tk.



L’algoritmo è non deterministico poiché le scelte effettuate per scegliere chi sia la clausola della KB con la quale verificare l’unificazione possono portare a fallimento oppure a successo (questo significa che a volte l’algoritmo funzionerà altre volte fallirà).

**SIMBOLI DI FUNZIONE:**

Se si usano costanti per nominare ogni individuo del dominio questo limiterà la KB ad essere finita, ovvero il sistema ragionerà su un numero finito di individui. Tuttavia, vorremo poter ragionare su di un numero potenzialmente infinito. Per risolvere questo problema vengono usati i **simboli di funzione** i quali permettono di descrivere individui indirettamente, ovvero anziché utilizzare costanti per descrivere un individuo esso viene descritto in termini di altri individui.

Sintatticamente un **simbolo di funzione** è una parola che inizia con una lettera minuscola, alla luce di ciò estendiamo la definizione di **termine** il quale può essere o una **variabile** o una **costante** oppure della forma f( t1, …, tn) dove f è un **simbolo di funzione** e ogni ti risulta essere un termine.

Per poter utilizzare questa nuova sintassi anche la semantica del Datalog deve essere espansa, in particolare la definizione di ϕ deve essere espansa di modo tale che esso assegni ad ogni simbolo di funzione n-ario una funzione di dominio Dn e codominio D. in questa espansione una costante può essere vista come un simbolo di funzione 0-ario ovvero senza parametri.

Una KB costituita da clausole con simboli di funzione può calcolare qualsiasi funzione calcolabile, per cui una KB cosi fatta può essere interpretata come un programma che viene chiamato **programma logico.** Un **programma logico**  è Turing completo.

**LOGICA DEL PRIMO ORDINE:**

**First-order predicate calculus**  è una logica che estende il calcolo proposizionale affinché possa includere atomi con simboli di funzione e variabili logiche. In tale linguaggio tutte le variabili logiche devono essere esplicitamente quantificate con i quantificatori universale ed esistenziale. La semantica del **calcolo dei predicati del primo ordine** è la stessa dei programmi logici ma con un insieme di operatori più ricco.

La logica del primo ordine è *di* ***primo ordine***perché consente la quantificazione degli individui nel dominio, ma non consente né predicati come variabili né quantificazione sui predicati.

**LOGICA DEL SECONDO ORDINE:**

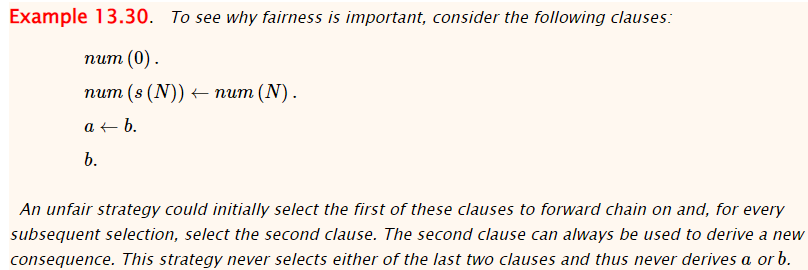
Essa consente la quantificazione sulle relazioni e sui predicati di primo ordine i cui argomenti sono relazioni del primo ordine. La logica del secondo ordine permette di definire la **chiusura transitiva** cosa che non può essere definita con la logica del primo ordine.

La logica del primo ordine è **SEMI-DECIDIBILE** e questo significa che una procedura di dimostrazione sound & complete esiste per ogni conseguenza logica che possa essere derivata dalla KB.

La logica del secondo ordine è **INDECIDIBILE** per cui nessuna procedura di dimostrazione che sia sound & complete può essere implementata in una macchina di Turing.

# **Procedura di dimostrazione con simboli di funzione:**

Le procedure di dimostrazione con variabili vengono utilizzate per il caso in cui abbiamo i simboli di funzione. La differenza principale è che la classe di termini viene espansa per includere simboli di funzione. L'uso di simboli di funzione coinvolge infiniti termini. Ciò significa che, durante il **forward chaining delle clausole**, dobbiamo assicurarci che il criterio di selezione per selezionare le clausole sia[**equo**](https://artint.info/2e/html/ArtInt2e.Ch3.S4.html) (fair).



Questo problema che si verifica ovvero quello di ignorare alcune clausole per sempre è conosciuto come **starvation.** Un criterio di scelta delle clausole risulta **equo “fair”** se tra le possibili scelte ogni clausola a pari possibilità delle altre di essere scelta. La procedura **BU** può generare infinite sequenze di conseguenze e se il criterio di scelta è equo ogni conseguenza logica della KB sarà derivata né segue che la procedura sarà **completa.**

Nella procedura **TD**  non ci sono cambi, ma con l’**unificazione** la procedura diventa più complicata poiché l’algoritmo dovrà ricorsivamente discendere nella struttura dei termini. L’unico cambio che c’è avviene nella procedura di **unificazione** dove una variabile X che non si unifica con un termine t nel X compare. Il controllo di questa condizione è detto  **occurs check**, se questo controllo non fosse fatto la procedura non sarebbe **sound.** La modifica all’'algoritmo di unificazione consiste nel ritornare ⊥ se seleziona un'uguaglianza α = β, Dove α è una variabile e β è un termine che non è α, ma contiene α(o viceversa). Questo passaggio è il **controllo delle occorrenze** “occurs check”.

**UGUAGLIANZA**

A volte risulta comodo associare più di un termine per nominare un singolo individuo del dominio, mentre in altri casi vogliamo che ogni nome si riferisca a un differente individuo. Attraverso il significato di **uguaglianza**  possiamo capire quando 2 termini rappresentano o meno lo stesso individuo [questo nuovo concetto è estremamente potente, infatti quando abbiamo ragionato con il calcolo dei predicati del primo ordine o con il calcolo proposizionale non avevamo questa certezza che ci viene data dall’**uguaglianza** ovvero non sapevamo se i termini ai quali ci riferivamo erano per certezza lo stesso individuo oppure no].

L’uguaglianza risulta essere un predicato speciale e diremo che t1 **equals** t2 , scritto t1 = t2 è vero nell’interpretazione intesa se t1 e t2 denotano lo stesso individuo del dominio.

L’uguaglianza **non significa somiglianza** infatti se a e b sono costanti e a = b, allora questo non significa che essi si somigliano, ma più che altro che lo stesso individuo ha due nomi diversi.

# **Ammettere asserzioni di uguaglianza:**

Senza permettere il concetto di uguaglianza applicato alle teste delle clausole l’unica cosa che può risultare uguale ad un dato termine è il termine stesso(per riflessione dell’ uguaglianza). Per permettere al sistema intelligente di lavorare con KB che ammettono l’uguaglianza abbiamo 2 modi:

* **ASSIOMATIZZAZIONE DELL’UGUALGLIANZA:**

L’uguaglianza può essere assiomatizzata nella seguente maniera, ovvero aggiungendo le clausole alla KB che andranno a definire le **proprietà** dell’uguaglianza che sono **riflessività**, **simmetricità** e **transitività:**

A picture containing text

Description automatically generated

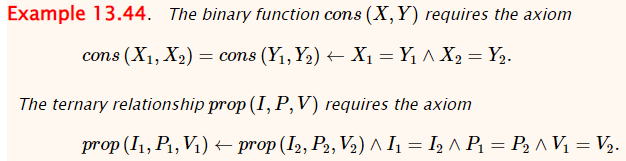
Gli altri assiomi che bisognerà aggiunger dipendono dalle funzioni e dalle relazioni definite, ovvero le 3 proprietà che abbiamo espresso per semplici variabili devono potersi adattare alle funzioni e alle relazioni nella seguente forma:

* **Schema di assiomatizzazione per un simbolo di funzione n-ario:**

f(X1, …, Xn) = f( Y1, …, Yn ) ← X1 = Y1 ∧ ⋯ ∧ Xn = Yn

|  |  |
| --- | --- |
|  |  |

* **Schema di assiomatizzazione per un simbolo di predicato n-ario:** p(X1, …, Xn) ← p(Y1, …, Yn) ∧ X1 = Y1 ∧ ⋯ ∧ Xn = Yn



* **PROCEDURE SPECIALI DI RAGIONAMENTO CON L’UGUAGLIANZA:**

La **paramodulazione** è un modo per aumentare una procedura di dimostrazione affinché implementi l’uguaglianza. L’idea di base è che se t1 = t2 risultano uguali allora t1 può essere rimpiazzato da t2, questo significa che l’uguaglianza sarà trattata come una **riscrittura delle regole**, andando a sostituire ciò che è uguale a quello che è uguale. Questo approccio per funzionare al meglio necessità di una rappresentazione **canonica** per ogni individuo, la quale specifica un termine che può essere usato da altre rappresentazioni affinché esse(queste rappresentazioni differenti) mappino l’individuo rappresentato dal termine della rappresentazione canonica con altri termini.

# **Unique name assumption UNA:**

**[assunzione univoca dei nomi]**

Invece di utilizzare il processo dell’assiomatizzazione che abbiamo visto essere abbastanza complicato per specificare le 3 proprietà dell’uguaglianza, si preferisce assumere l’ipotesi che ogni termine ground “ovvero ogni costante” denoti un individuo differente. Quindi sotto l’ipotesi di **UNA** ogni termine ground denota un individuo differente e questo vale per ogni coppia di termini ground ovvero se t1 e t2 sono termini ground allora t1 ≠ t2 dove ≠ significa “non uguale a”.

La **UNA** può essere assiomatizzata con il seguente schema di assiomatizzazione della disuguaglianza:

* c ≠ c′ per ogni costante distinta presente nella KB
* f(X1, …, Xn) ≠ g(Y1, …, Ym) per ogni simbolo di funzione distinto nella KB
* f(X1, …, Xn) ≠ f(Y1, …, Ym) **←** Xi ≠ Yi per ogni simbolo di funzione, questo significa che si saranno ne schemi differenti per ogni simbolo di funzione n-ario.
* f(X1, …, Xn) ≠ c per ogni simbolo di funzione distinto dalla costante

Con questo tipo di assiomatizzazione 2 termini ground sono identici se e solo se essi sono **unificabili**.

L'assunzione di nomi univoci è molto utile per le applicazioni di database, in cui non si desidera esplicitare che kim ≠ sam and kim ≠ chris and chris ≠ sam.

**PROCEDURA DI DIMOSTRAZIONE TOP-DOWN PER UNA:**

La procedura non tratta la disuguaglianza come un predicato e questo perché bisognerebbe specificare troppi predicati di disuguaglianza, per cui se un sotto goal è dato da t1 ≠ t2 allora abbiamo i seguenti 3 casi:

1. t1 e t2 non sono unificabili e in tal caso t1 ≠ t2 ha successo.
2. t1 e t2 sono identici e questo include avere le stesse variabili nelle medesime posizioni allora in tal t1 ≠ t2 caso fallisce.
3. Altrimenti ci saranno istanze di t1 ≠ t2 che avranno successo e altre che falliranno.

Nella TD avremo che la disuguaglianza del primo tipo avrà successo, quella del secondo tipo fallirà e quella di terzo tipo sarà ritardata, affinché siano provati dei sotto goal che permettano di far ricadere il terzo tipo in uno dei precedenti primo o secondo. In questo caso una risposta non significherà che la query sarà vera per ogni istanza delle variabili libere , ma piuttosto che sarò vera per alcune istanze.

**ASSUNZIONE DI CONOSCENZA COMPLETA**   
L’assunzione di conoscenza completa detta  **closed world assumption** ci permette di dire che se una affermazione non segue come conseguenza logica della KB allora essa sarà falsa e questo ci permette una dimostrazione con **NAF.** L’assunzione di conoscenza completa include in sé la **UNA**.