Riassunto di logica computazionale

# Come si può rappresentare la conoscenza?

La conoscenza può essere rappresentata attraverso la logica, questo formalismo ha le seguenti qualità:

* è leggibile attraverso un linguaggio naturale;
* è ben definito matematicamente, ciò permette di comprendere cosa si può ottenere dalla computazione e il significato dei risultati.

Per ottenere i risultati citati prima, esistono metodi di ragionamento automatico, tutto questo fa parte della explainable AI, quella parte di intelligenza artificiale che non si comporta come una scatola nera ma che spiega il significato della computazione e dei risultati.

Un esempio di applicazione si può ritrovare nel giustizia per velocizzare i processo, tuttavia c’è il rischio di prendere una decisione senza avere competenze.

# Rappresentazione dichiarativa

La rappresentazione dichiarativa indica ciò che è vero nel dominio ed è in larga parte indipendente dalla computazione.

L’idea è quella di formalizzare diverse computazioni riutilizzando la stessa conoscenza e, quando è possibile, anche gli stessi meccanismi di inferenza.

Per fare alcuni esempi:

* A è conseguenza logica di quanto modellato?
* esiste un valore x tale che il predicato p(x) è conseguenza logica?
* dato a, esiste un valore x tale che il predicato q(a,x) rispetti una certa condizione?
* A è consistente rispetto al modello? e se è vero, cos’altro lo è?

I vari ragionamenti sulla base della conoscenza possono essere utilizzati in diversi ambiti, come ad esempio il ragionamento basato su sistemi statici o dinamici, su ontologie, eccetera.

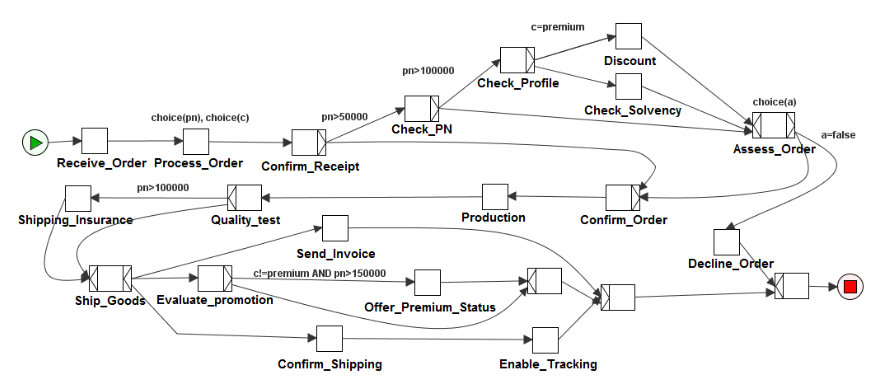
La programmazione logica può essere utilizzata sia per trovare un valore di un predicato, sia i parametri che li soddisfano.

Inoltre è anche possibile nelle conseguenze logiche trovare per quali precondizioni vale un predicato data una conseguenza e viceversa.

E’ anche possibile utilizzare la ricorsione e si possono richiedere certe condizioni sulle singole variabili, ciò permette flessibilità dal momento che si può fare con diverse combinazioni.

Tutto questo è applicabile in contesti differenti (anche dinamici), si può ad esempio cercare di prevedere il risultato di certe azioni e decidere quali fare per raggiungere un certo stato.

## Esempio: Acquisto di un prodotto online

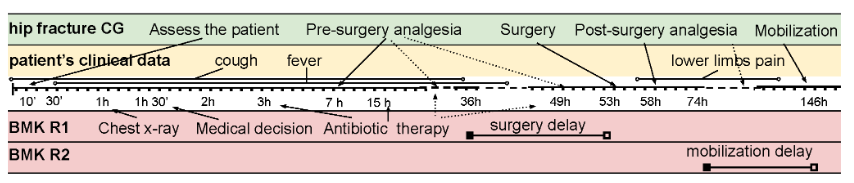


L’immagine sopra indica tutto il processo da effettuare per l’acquisto di un prodotto online. Supponendo che l’azienda in questione venda un solo tipo di prodotto, se l’utente acquista più di 50000 unità di prodotto, allora viene eseguita una approvazione prima che il cliente confermi.

Inoltre, per i clienti non premium è necessaria anche una verifica di solvibilità se il numero di unità ordinate è maggiore di 80 mila.

In generale, bisogna capire quali eventi ci sono stati e/o verificare se una sequenza di eventi/fatti è conforme a una linea guida modellata con certi meccanismi.

## Esempio: analisi delle linee guida cliniche



In questo esempio bisogna confrontare le linee guida cliniche col trattamente effettivo, come si procede?

L’idea è quella di avere delle regole che, in base alla situazione, dicono come comportarsi, ad esempio:

* Quando il paziente ha la febbre, non bisogna operarlo;
* In caso di dolore agli arti inferiori, bisogna ritardare la riabilitazione.

Quest’ultima regola si può modellare come segue:

che significa:



Tutto questo serve a dire che questi problemi sono risolvibili utilizzando delle logiche più evolute rispetto a quella classica.

## Esempio: il mondo del wumpus

Un’agente deve esplorare uno spazio 4x4 come visibile in figura, qui è presente un mostro detto wumpus e dei pozzi, l’obiettivo è trovare l’oro.

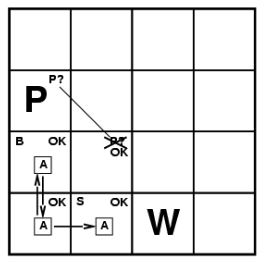
Si modella tutto con le seguenti regole:

* Il Wumpus e i pozzi non possono muoversi;
* nelle celle adiacenti al mostro, si sente cattivo odore;
* nelle celle adiacenti ai pozzi, si sente della brezza;
* L’oro luccica e si può notare ciò solo se l’agente è nella stessa cella;
* L’agente può sparare una (e una sola) freccia;
* Se l’agente è di fronte al wumpus, può sparare la freccia per ucciderlo;
* La morte del wumpus provocherà un urlo che si sentirà in ogni casella;
* Se il giocatore è nella stessa cella dell’oro, può raccoglierlo;
* Se il giocatore ha l’oro in mano, può farlo cadere nella stessa casella.

| Dalle regole si ricavano i seguenti sensori:   * Cattivo odore; * Brezza; * Luccichio; * Urlo; * Caduta nel pozzo. | Allo stesso modo, si ricavano le azioni attuabili dall’agente:   * Vai a destra/sinistra; * Vai dritto; * Raccogli; * Fai cadere; * Spara. |
| --- | --- |

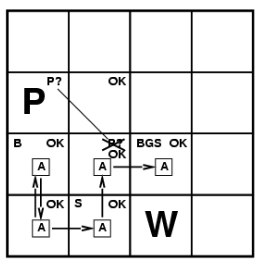
### Esplorazione: come si fa?

Partendo dalla cella (1,1), l’agente può andare in (2,1) oppure (1,2) dal momento che nella cella attuale non si sente nè puzza nè brezza.

Supponendo di andare in (2,1), si scopre che si sente brezza, quindi in una delle celle adiacenti e presente un pozzo.

Per essere sicuri, si torna indietro e si va in (1,2), qui si sente cattivo odore, il mostro è vicino!

Da ciò si può quindi concludere che la cella (2,2), adiacente a (1,2) e (2,1), è una zona sicura, altrimenti nelle celle adiacenti si sarebbe sentito cattivo odore o del vento.

Dal momento che nella cella (2,2) non si sente niente, le celle (2,3) e (3,2) sono considerate sicure.

Supponendo di andare in (2,3) è presente sia cattivo odore, sia brezza ma si vede anche il luccichio dell’oro, quindi si raccoglie. Attraverso queste regole, l’idea è quella di definire il comportamento di un agente intelligente.

## Cosa sono le logiche?

Le logiche sono linguaggi formali con cui rappresentare l’informazione e trovare da essa conclusioni tale per cui esiste una relazione ben definita.

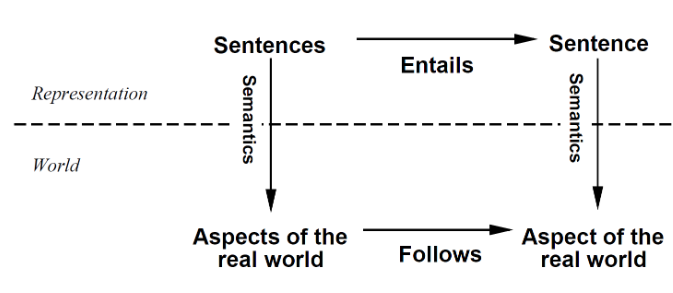
Queste logiche hanno:

* una sintassi, usata per definire le proposizioni/formule;
* una semantica, utile per dare un significato a ogni proposizione/formula.

## Entailments e conseguenza logica

Un Entailment indica che una formula segue un insieme di formule:

Nell'equazione sopra, KB è la conoscenza base, quella a disposizione in un certo istante, α invece l’insieme di formule.

Questa notazione indica la conseguenza logica, infatti α è conseguenza logica di KB se α è vero in ogni caso in cui KB è vero.

La conseguenza logica è una relazione tra formule basata sulla semantica.

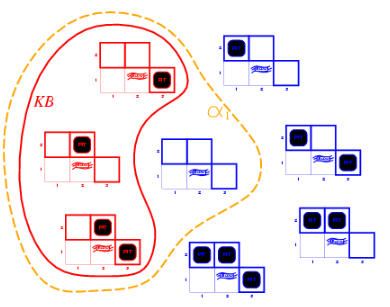
Nella rappresentazione ci sono aspetti reali rappresentati appunto con delle formule. Limitandosi alle cose vere, attraverso i ragionamenti logici ci saranno formule che saranno conseguenze logiche di altre.

Allo stesso modo si può interpretare tutto questo nel mondo reale, concludendo che si è esistono aspetti del mondo che sono conseguenza di altri.

## Cos’è un modello?

Un modello indica un mondo in cui viene valutata la verità, ad esempio: data una formula α, m è un modello di α se quest’ultima è vera per m.

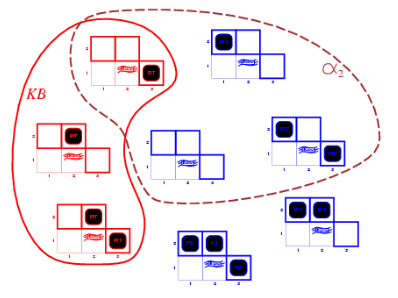
L’insieme di tutti i modelli per una formula α è M(α), da ciò si conclude che se M(KB) è contenuto in M(α), quindi se ogni interpretazione in cui KB è vera, è vera anche per α.

L’insieme M(α) contiene anche altre combinazioni rispetto a M(KB), precisamente contiene ogni combinazione in cui KB è false e α è vero.

Nel caso del wumpus, se nella casella corrente non c’è né brezza né cattivo odore, allora si può andare in una di quelle adiacenti.

Considerando solo i casi in cui si hanno pozzi nelle vicinanze, si hanno tre formule e quindi otto possibili modelli.

Ponendo KB come l’insieme delle regole più le osservazioni, si considera α1 indicante che la cella (1,2) è sicura, in questo caso α1 è conseguenza logica di KB.

Si considera ora α2, essa indica che la cella (2,2) è sicura, tuttavia ciò è vero solo in uno dei casi di α2, quindi non è conseguenza logica di KB.

Si conclude quindi che ogni combinazione di α deve soddisfare ogni caso di KB, quindi le regole e le osservazioni.

## Cos’è l’inferenza?

L’inferenza è un metodo di ragionamento automatico che permette di derivare una formula da una base di conoscenza, essa si indica nel seguente modo:

L’inferenza ha le seguenti proprietà:

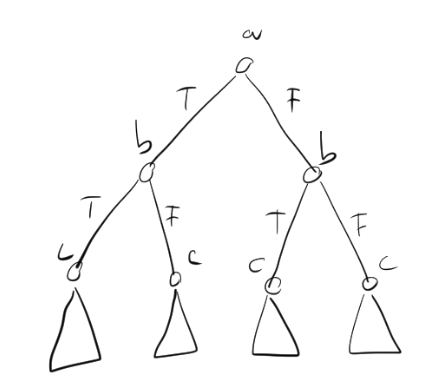
* Correttezza: Se α è derivabile da KB, allora α è anche conseguenza logica di KB;
* Completezza: Se α è conseguenza logica di KB, allora si può anche derivare da essa.

Dal punto di vista della sintassi, ogni operazione su una formula data è anch’essa una formula:

* Dato P, è una formula;
* Dati P1 e P2, è una formula;
* eccetera.

Nella semantica invece si assegnano valori di verità a ogni simbolo della formula. Considerando i predicati Pi,j e Bi,j come la presenza nella cella (i,j) rispettivamente di un pozzo o della brezza, si definiscono la seguenti regole:

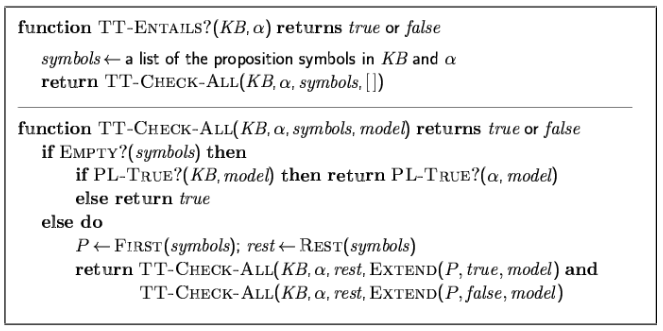
Per applicare l’inferenza si enumera ogni combinazione di valori, in alcune di queste KB sarà vera, quindi deve esserlo anche α.

Come si fa a verificare ciò?

Ebbene, si effettua una ricerca in profondità su ogni modello per vedere è corretto e/o completo, questo permette di costruire una riga della tabella di verità della formula.

L’operazione descritta sopra viene fatta per ogni riga della tabella, ciò spiega la presenza dell’AND dato che bisogna verificare ogni volta il valore di un letterale, quindi è esponenziale su n simboli.

Sulla base di quanto detto fino a ora, si costruisce un albero di computazione assegnando a ogni proposizione un valore di verità sulle differenti chiamate.

Per ogni assegnazione si va a vedere se KB viene verificato, precisamente si fa tutto nel seguente modo:

* Date le proposizioni atomiche a, b, c, …, si assegna T o F alle variabili;
* Queste assegnazioni valgono per un ramo dell’albero, per verificarli tutti si mettono in AND;
* Sulle foglie dell’albero si effettua la verifica per vedere se KB è soddisfatta.

Questo è un problema di soddisfacibilità booleana che è NP-Completo, pertanto la complessità in tempo è O(2^n) e in spazio è O(n) per l’algoritmo ricorsivo, in questo modo infatti si evita di memorizzare l’intera tabella.

## Quando una formula è valida e/o soddisfacibile?

Una formula è valida quando è vera in ogni interpretazione.

La validità di una formula è legata all’inferenza grazie al teorema di deduzione: α è conseguenza logica di KB se e solo se è una formula valida.

Una formula è soddisfacibile quando è vera per qualche modello, è invece insoddisfacibile quando non esistono modelli per cui la formula è vera.

La soddisfacibilità è legata all’inferenza nel seguente modo: α è conseguenza logica di KB se e solo se la formula è insoddisfacibile.

## Metodi di inferenza

Esistono metodi che permettono di trovare una soluzione in tempi ragionevoli:

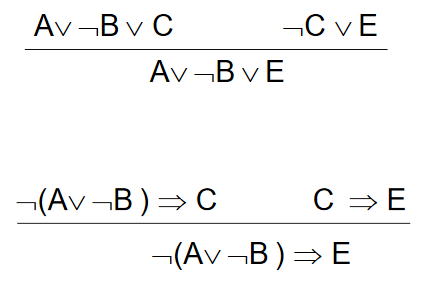
* L’applicazione di regole di inferenza opera sulle formule e ne costruisce altrettante, di solito richiede le formule in forma normale congiuntiva;
* Nel model checking si elaborano le interpretazioni delle formule, si enumerano e costruiscono modelli in modo incrementale. Esistono metodi euristici per fare tutti questi ma i risultati ottenuti, seppur corretti, sono incompleti.

Considerando il primo metodo, come si effettua la risoluzione?

Essenzialmente, data una formula, prima di tutto la si converte in forma normale congiuntiva, dopodichè si prendono in considerazione due clausole alla volta:

* Dalle due clausole prese in considerazione, si crea una nuova clausola contenente tutti i letterali a eccezione di quelli complementari tra loro.

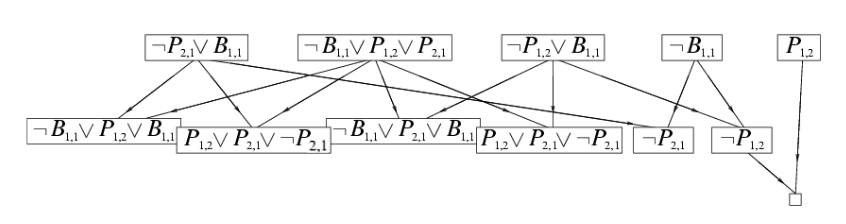
Esempio: considerando la formula , ci sono due clausole, da esse si ricava la clausola dato che e il suo complementare si annullano.

Per verificare la correttezza di questo metodo, si prende in considerazione la foto a destra: per soddisfare la disgiunzione logica, una delle due parti deve essere vera, infatti se si considera C:

* se C è falso, allora deve per forza essere vero per soddisfare la prima formula, la seconda invece è già soddisfatta;
* se C è vero, la prima formula è soddisfatta mentre nella seconda E deve essere vero per esserlo.

Quindi in ogni interpretazione la disgiunzione è vera se oppure E sono veri.

Si considera ora un insieme S di clausole, questo insieme è consistente?

Considerando S come l’insieme di clausole nella parte superiore della foto,

le formule in basso sono tutte quelle derivate da ogni coppia di clausole. Si può notare che l’ultima coppia di clausole genera la clausola vuota, corrispondente al valore falso, ciò avviene perché le clausole hanno letterali complementari.

Perché la clausola vuota è false?

Essenzialmente perchè non esiste un'interpretazione che soddisfa sia la prima clausola, sia quella complementare, di conseguenza non esistono modello per questo.

Da ciò si conclude che S è inconsistente e di conseguenza non esistono modelli per questo insieme. Se così non fosse, esisterebbe un modello per S che andrebbe bene anche per le clausole derivate e, dal momento che si deriva la clausola vuota, S non ne può avere.

Si conclude che S è inconsistente se e solo se deriva la clausola vuota, questa regola è detta anche completezza rispetto alla refutazione.

## Come si converte in forma normale congiuntiva?

Data la formula :

* si elimina la coimplicazione sostituendola con due implicazioni:
* Si eliminano le implicazioni sostituendoli con la formula equivalente:
* Si sfruttano le formule di De Morgan:
* Si applica la proprietà distributiva:

## Esempio di risoluzione

Considerando la foto a destra, per vedere se KB segue α, bisogna vedere se è soddisfacibile.

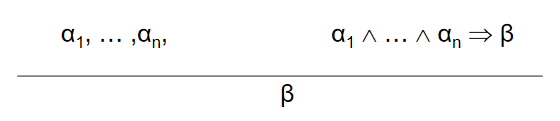
Se per una clausola C, questa è conseguenza logica di KB, allora la si può derivare dalla trasformazione in clausole da KB.

Prendendo , ciò non vale perchè non si può derivare a V b da {a} tramite la risoluzione.

Ciò vale nel senso che:

* se α è conseguenza logica di KB, allora si deriva la clausola vuota della trasformazione di . Se vale ciò, allora è inconsistente e quindi di deriva la clausola vuota per la completezza di refutazione.

# Clausole di Horn

Una clausola di Horn è una clausola avente delle limitazioni, ossia quella di avere un solo letterale positivo, tutti gli altri (se presenti) sono negativi.

Queste clausole sono un caso particolare in cui la derivazione delle formule permette di ottenere il conseguente, come visibile a sinistra (Modus Ponens).

Esistono algoritmi che permettono di gestire queste clausole formalizzando le regole in tempo lineare.

La formula deve essere corretta dato che ogni interpretazione in cui le formule iniziali sono vere, rende tali anche quelle derivate. Infatti, considerando la foto sopra, se tutti gli α tranne 1 sono veri e β è falso, si è davanti a una contraddizione.

# Forward chaining

L’idea del forward chaining è di considerare l’insieme di formule date come un grafo: ogni letterale è un nodo mentre gli archi sono le implicazioni logiche, come visibile in figura.

Ci sono inoltre gli archi multipli, questi occorrono quando l’antecedente di una regola è formato da più letterali.

Le regole che non sono implicazioni sono i fatti, cioè tutto quello che è considerato vero.

L’algoritmo del forward chaining effettua delle scansioni su tutte le clausole della conoscenza base, generando un’insieme di atomi derivati.

Inizialmente S è contenuto in KB e, se in una scansione i letterali di una clausola sono contenuti in S, vengono aggiunti i conseguenti.

Qualora siano stati trovati nuovi atomi, viene fatta partire una nuova scansione, altrimenti l’algoritmo si ferma.

Considerando ad esempio le regole della foto come conoscenza base, all’inizio l’insieme S contiene solo A e B, in seguito:

* Alla prima scansione si aggiunge L a S dato che A e B sono in questo insieme;
* Per lo stesso motivo, si aggiunge M a S nella seconda scansione;
* Nella terza scansione L e M sono in S, quindi si aggiunge P a questo insieme;
* Nella quarta scansione la regola P→ Q permette di aggiungere Q all’insieme S.
* Nella quinta e ultima scansione non si aggiunge nulla dato che tutti i possibili atomi sono stati aggiunti a S, quindi si ferma.

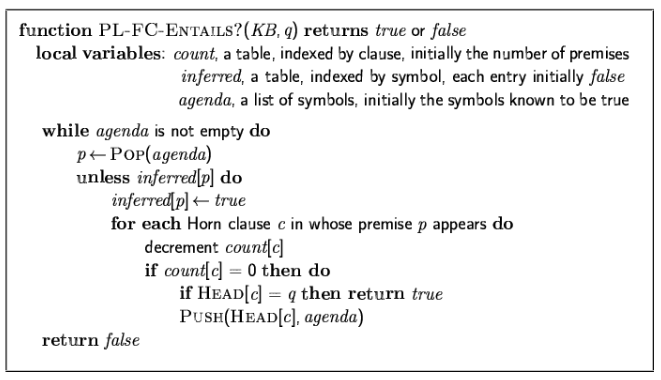
Si conclude che l’insieme S corrisponde a un modello della conoscenza base in cui A, B, L, M, P, Q sono veri mentre H e R sono falsi dato che non è possibile raggiungerli in alcun modo.

### Complessità dell’algoritmo

La conoscenza base contiene un numero n=O(|KB|) di atomi, nel caso peggiore a ogni scansione viene aggiunto un nuovo atomo.

Quindi per calcolare il costo, ogni scansione di KB richiede l’iterazione di O(|KB|) clausole e da ognuna verificare se il corpo contiene tutti gli atomi in S. Supponendo che questa verifica sia costante, il forward chaining effettua n scansioni ognuna di n passi, quindi di conclude che la complessità nel caso peggiore è .

## Algoritmo e applicazione



L’algoritmo sopra è una versione riadattata del forward chaining classico, infatti ci sono dele variabile che evitano di inferire simboli già inferiti e un array di contatori che indica il numero di premesse per una clausola. Agenda invece è una lista contenente all’inizio i letterali base.

Applicando l’algoritmo alle precedenti regole:

* Si inizia scansionando A, si decrementa il contatore per L che va a 0, quindi si inserisce nell’insieme;
* Si scansiona B, si decrementa il contatore per M che va a 1;
* Si scansiona L, si decrementa il contatore per P e per M che va a 0, quindi si inserisce nell’insieme;
* Si scansiona M, si decrementa il contatore per P che va a 0, quindi si inserisce nell’insieme;
* Si scansiona P, si decrementano i contatori per L e per Q che vanno entrambi a 0, quindi si inseriscono nell’insieme;
* Si scansiona L, dato che già inferito non si fa nulla;
* Si scansiona Q, non esistono regole in cui Q è antecedente e quindi l’algoritmo si ferma.

## Correttezza del forward chaining

Se l'algoritmo di forward chaining deriva il letterale q dato in input, allora q è conseguenza logica della conoscenza base, infatti la regola del modus ponens viene applicata a ogni passo.

A partire degli atomi, questi sono veri in ogni modello della conoscenza base dato che sono i fatti e sono già dimostrati. Il modus ponens è inoltre corretto, infatti in ogni interpretazione I che rende vere le premesse, I rende vere anche la conseguenza.

Quindi ogni passo di inferenza aggiunge a S i soli atomi appartenenti alla conseguenza, le premesse sono vere in modello di KB, quindi anche le conseguenze lo sono.

## Completezza

Il forward chaining deriva ogni formula atomica di KB perchè:

* l’algoritmo termina se non vengono aggiunte altre formule;
* Considerando il modello m ottenuto, si assegna true ai simboli inferiti;
* Ogni clausola della KB iniziale è vera in m, m non può quindi non soddisfare una clausola, rendendolo un modello di KB;
* Se q è conseguenza logica di KB, q è vero in ogni modello di KB incluso m, quindi deve essere inferito. Da ciò si conclude che il forward chaining è un metodo per indicare se q è conseguenza logica di KB.

# Backward Chaining

Il backward chaining è un metodo orientato a verificare se una formula è conseguenza logica. L’idea è che, dato q, verificare se è un fatto oppure una premessa di una regola avente q come conclusione.

Per evitare cicli, si verifica se le premesse di una regola sono già nella lista.

Inoltre per evitare di ripetere il lavoro si controlla se un simbolo è già stato considerato, è infatti utile ricordarsi se una clausola è stato dimostrata o meno, così da non riprovarci in seguito.

L'algoritmo del backward chaining funziona sfruttando la ricorsione, i casi base sono i fatti mentre quelli ricorsivi sono le regole.

Per fare un esempio, considerando le regole precedenti:

* partendo da Q si trova la regola P→ Q, quindi si fa ricorsione su P;
* Con P si trova la regola , si fa ricorsione prima su L e poi su M;
* Con L si trova la regola , si fa ricorsione su A e B;
* Non ci sono regole in cui A occorre come conclusione, quindi non ci sono chiamate ricorsive;
* La stessa cosa vale per B;
* Con M si trova la regola , si fa ricorsione su B e L;
* Come già detto prima B è un fatto e quindi non fa altre chiamate;
* Ricorrendo su L, si scopre che questo è già stato inferito, quindi non si fa nulla.

Esiste anche un altro metodo molto più affine alla programmazione e consiste nella scelta delle regole partendo dalla dimostrazione di veridicità di Q:

* Partendo da Q, la regola in cui occorre è P→ Q, quindi si va a dimostrare se P è vero;
* Per dimostrare che P è vero, bisogna dimostrare che lo sia;
* Per dimostrare che , bisogna dimostrare se è vero;
* A e B sono fatti e sono già dimostrati;
* Per dimostrare la veridicità di M, bisogna dimostrare ;
* Come già detto prima, B è già dimostrato dato che è un fatto;
* La veridicità di L ha bisogno di per essere dimostrata (che sono entrambi fatti).

Allo stesso modo, se si parte da R, bisogna dimostrare H ma quest’ultimo non è nè un fatto nè occorre come conclusione in una regola, quindi l’algoritmo fallisce.

Nel caso in cui ci siano più regole per un dato simbolo, se ne sceglie una.

## Come si individuano i cicli?

Considerando il simbolo L, è possibile notare che ha due regole, in particolare ha nella premessa il simbolo P che occorre prima nella catena di regole. Ciò significa che, analizzando le regole in cui occorre P, bisogna ricontrollare L per via di .

Per provare la veridicità di L bisogna quindi provarla per il sottogoal L, ciò si può dimostrare nel caso in cui quest’ultima L è vera per via di un caso base, ma a sto punto lo stesso caso base si può applicare anche nella prima.

Detto ciò, cosa si può dire dei due algoritmi visti?

* Il forward chaining è guidato dai dati, è automatico e inconscio, utile per applicazioni come il riconoscimento di oggetti o le decisioni di routine;
* Il backward chaining è guidato dal goal, quindi più appropriato per il problem solving, la sua complessità può inoltre essere molto meno lineare rispetto alla grandezza della conoscenza base.

# Inferenza proporzionale efficiente

Esistono due famiglie di algoritmi di questo tipo:

* Complete Backtracking Search: sono algoritmi che permettono di ottenere una soluzione completa, gli algoritmi di tipo DPLL fanno parte di questa famiglia;
* Incomplete Local Search: sono algoritmi che si basano sulla ricerca locale e restituiscono una soluzione incompleta.

## Backtracking per la soddisfacibilità booleana

L’obiettivo è trovare un modello che soddisfi tutto l’insieme delle clausole.

La fase di inizializzazione prende in input una formula, la converte in forma normale congiuntiva e da quest’ultima estrae clausole e letterali. Dopodichè:

* Verifica se ci sono ancora letterali da valutare;
* Se è così, si controlla se con il modello attuale tutte le clausole sono soddisfatte, in tal caso si restituisce true, altrimenti false;
* Se ci sono ancora simboli da valutare, si prende il primo della lista, si assegna true/false e si aggiunge al modello;
* Si effettuano due chiamate ricorsive (in OR tra loro), nella prima il simbolo estratto sarà true nel modello, nel secondo invece è false.

Da questo algoritmo si ottiene un albero di computazione binario le cui foglie indicano i test sul modello.

Se una di questa risulta true, allora non si effettua la seconda chiamata per via delle proprietà dell’or logico, quindi basta trovare un’istanza sì per validare tutto.

Questo algoritmo è esaustivo dato che prova ogni combinazione possibile, la sua complessità è quindi esponenziale rispetto all’input.

Si può fare di meglio?

## Algoritmo DPLL

L’algoritmo DPLL si basa su quello precedente aggiungendo delle migliorie grazie alle euristiche.

### Early termination

L’algoritmo termina senza controllare tutte le clausole e di conseguenza tutto l’albero, infatti basta un letterale vero per rendere vera una clausola vera e, allo stesso modo, una clausola falsa rende tale tutta la formula;

### Euristica dei letterali puri

Un letterale è puro quando appare solo in un modo, ad esempio:

Considerando queste tre clausole, i letterali puri sono A e mentre C non lo è, si possono quindi assegnare ad A e B rispettivamente true e false in modo da soddisfare più clausole possibili.

### Euristica delle clausole unitarie

Le clausole unitarie sono quelle formate da un singolo letterale, quindi si soddisfa assegnando true o false in base alla situazione.

In questo modo si lavora con meno clausole e si verifica la soddisfacibilità di queste in base ai valori assegnati.

Simboli puri e clausole unitarie vengono scoperti a ogni iterazione dell’algoritmo e quindi a ogni assegnazione.

## WalkSAT

WalkSAT è un algoritmo che non restituisce una risposta completa, l’idea è quella di non andare per tentativi ma a caso, La percentuale di possibili assegnazioni è alta se esistono assegnamenti che soddisfano la formula in input.

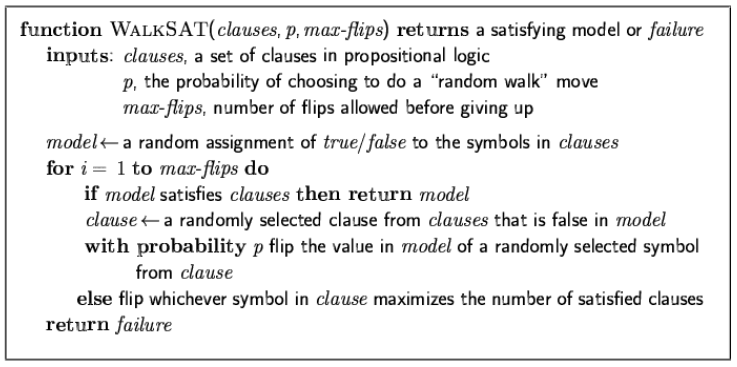
WalkSAT è un algoritmo di ricerca locale, cioè che ci si sposta di poco rispetto allo spazio di manovra.

Questo algoritmo minimizza anche i conflitti, cioè si evitano combinazione di valori che non vanno bene (cioè che non soddisfano le clausole), per fare ciò si utilizza un’euristica che riduce le clausole insoddisfacibili.

L’algoritmo è apparentemente greedy, tiene sempre in considerazione un simbolo finché non verifica la formula, in caso negativo ritorna sui suoi passi.

La casualità permette il cambiamento di valori dell'euristica per cercare di ottenere la miglior soluzione fino a quel momento.

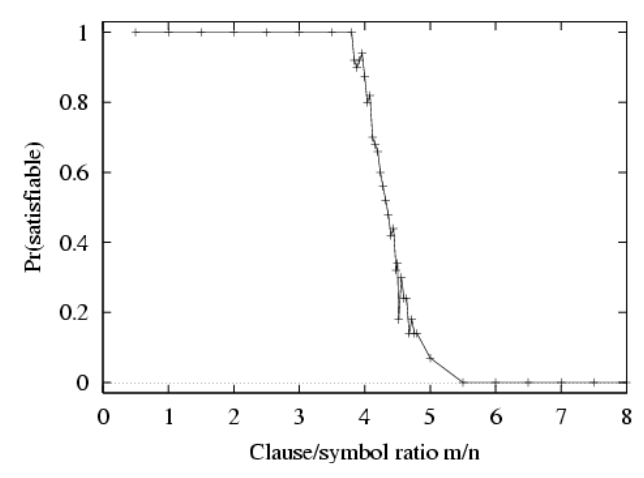
## Algoritmo



## Problemi di soddisfacibilità difficile

Ogni problema è riducibile a SAT per via del teorema di Cook, quindi si può trovare una soluzione in questo modo e poi riconvertirla.

Considerando il 3-SAT, quando m clausole sono soddisfatte con un numero n di simboli?

Ogni clausola impedisce certe combinazione di valori e, dato che ci sono 3 modi per soddisfarle (le clausole sono tutte lunghe 3), si procede assegnando valori ai simboli della prima clausola, osservare le conseguenze e così via.

Considerando il grafico a destra in cui si mette in relazione il rapporto m/n con la probabilità di soddisfare una formula. All’inizio è praticamente certo soddisfarla, tuttavia c’è un drastico calo a 4.3 circa fino a divenire impossibile a 5.5.

Ciò significa che il numero di clausole e di letterali è importante per trovare una soluzione.

Esistono risolutori che permettono di risolvere anche milioni di clausole, per farlo la conoscenza base deve contenere regole uniformi definite per ogni coppia (x,y), ad esempio:

Bisogna inoltre definire le regole per ogni tempo t:

Ciò causa una rapida proliferazione di clausole, quindi si generano tutte queste regole in automatico. A livello di rappresentazione tutto questo conviene dal momento che per ognuna di esse è possibile generare un insieme di regole.

### Dimostrazione per refutazione

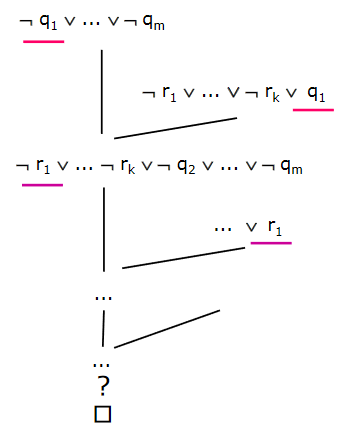
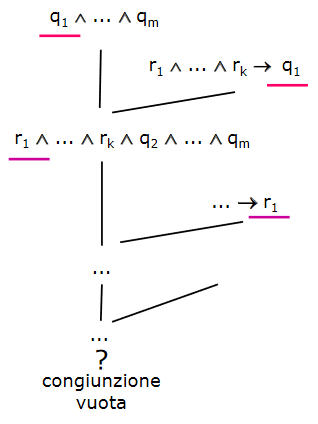
Il backward chaining può essere visto come la risoluzione di una formula F a partire da un insieme S, in tal caso F è conseguenza logica di S.

Questa dimostrazione si fa per assurdo: supponendo che la formula F sia falsa, si verifica se S U { -F } è inconsistente. Come si fa? Si utilizza la risoluzione dato che ogni formula può essere trasformata in un insieme di clausole.

L’insieme S è formato da regole nella forma , dove p è la testa e è il corpo.

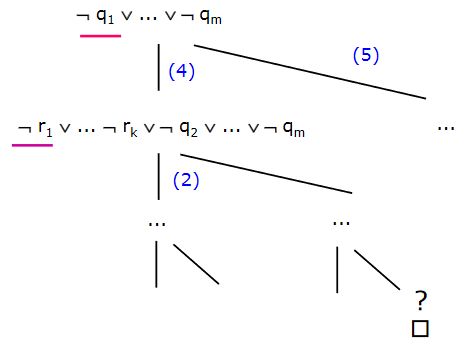
Si può notare come queste regole siano in realtà clausole di Horn per via dell’implicazione logica stessa, infatti:

Per dimostrare una formula -F nella forma , si procede nel seguente modo:

* Si sceglie un letterale seguendo un certo criterio, come ad esempio prendere il primo;
* si cerca tra le regole una clausola col letterale complementare, questo letterale è unico per via della definizione di clausola di Horn;
* si genera una nuova clausola aventi tutti e letterali di F e della regola non considerati;
* da qui, si torna al primo punto e si procede.

Questa soluzione senza refutazione ma dimostrando F da S sfruttando il backward chaining:

* Si sceglie un letterale seguendo un certo criterio, come per il precedente metodo;
* Si cerca tra le formule di S quelle con il letterale come testa;
* Si costruisce una formula F’ formata dal corpo della regola scelta più tutti i letterali di F non considerati;
* Si procede tornando la primo punto.

Questa soluzione è detta Selection Linear Definition (SLD) perchè utilizza una regola di selezione, deriva le clausole linearmente e ha in input clausole definite, ovvero quelle di Horn.

Ogni derivazione può essere illustrata numerando le clausole di S e indicando quella utilizzata a ogni passo della risoluzione.

Ogni possibile derivazione forma un albero, ogni nodo ha tanti figli quante sono le regole aventi il letterale in comune.

## Considerazioni

La risoluzione input non è in generale completa per le clausole arbitrarie non di Horn, si può comunque derivare la clausola vuota per un insieme S fatto in un certo modo ma la risoluzione SLD non deriva la clausola vuota.

Non esiste una refutazione input da S che parta da una sua clausola e risolva tutto a ogni passo di risoluzione, quello che si ottiene è una sorta di ciclo.

Quello che si può fare però è riutilizzare una delle clausole dell’asse di crescita per ottenere la clausola vuota.

Esistono quindi restrizione che funzionano per le clausole di Horn ma non per quelle normali.

# Risoluzione SLD con variabili e quantificatori

Indicando un programma come l’insieme S di predicati da uno o più argomenti, si considerano le seguenti regole:

* indica che
* indica che oppure

Come argomenti dei predicati, si possono utilizzare i seguenti termini:

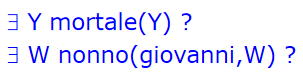
* variabili, intese come quantificate in uno scope della formula;
* costanti;
* termini complessi come altri predicati o funzioni.

Un esempio di termine complesso si può trovare nei veicoli: veicolo(Italia,aaaa) associa all’Italia una specifica targa, con veicolo(Italia,X) si intende che la targa appartiene a un qualunque mezzo ma residente in Italia.

Come si dimostra una formula date delle variabili?

In questo caso le variabili sono quantificate esistenzialmente, ciò è utile le formule complementari portano a una contraddizione.

Si considera il seguente esempio:



Negando la prima formule si ottiene che, per ogni Y, Y non è un mortale. Per vedere tutto questo si sostituisce Y con un valore specifico.

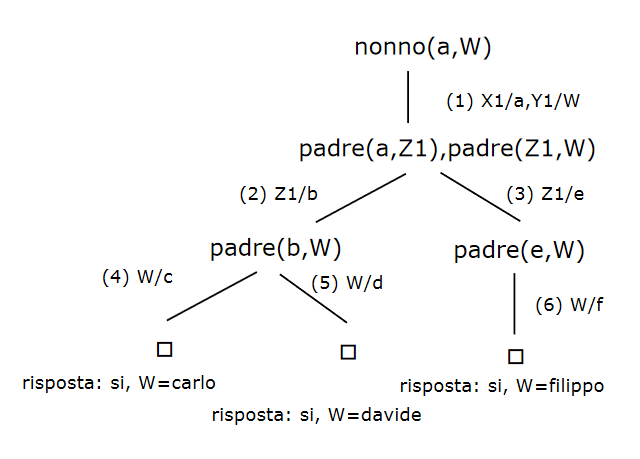
Come cambia la risoluzione?

Essenzialmente le formule si specializzano nei casi particolari. Infatti, sfruttando la regola di selezione, si sceglie un letterale A, si cerca una regola avente A come testa e si applica una sostituzione σ, ovvero si rimpiazzano le variabili a favore di termini specifici.

La sostituzione serve a rendere uguali due formule, di tutte le risoluzioni si tende a scegliere quella più generale in quanto si può reiterare il processo su di essa.

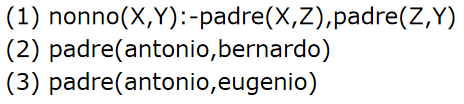
La ridenominazione delle variabile è importante al fine di evitare conflitti, l’idea è che ogni formula ricavata dalla ridenominazione è equivalente a quella di partenza.

Andando avanti nella risoluzione, se viene generata la clausola vuota combinando le formule, si ottiene una risposta al goal iniziale.

Quindi, supponendo di partire da una formula G0, ogni G0 σ è una conseguenza logica del programma, quindi sono state applicate sostituzioni in un numero finito di passi.

Combinando tutte le sostituzioni in una sequenza, si ottiene una sostituzione che, applicata alla formula iniziale che potrebbe far sparire tutte le variabili della formula.

Per fare un esempio, dato il goal nonno(antonio,W) e le seguenti regole:



Per dimostrare il goal, si individuano le clausole avente nonno(t1,t2) come testa e si rinominano le variabili, quest’ultimo passo si effettua per differenziale le variabili della formula da quelle del goal.

In questo esempio la sostituzione è la seguente:

nonno(antonio,W) nonno(X,Y)

Per rendere tutto uguale, si sostituisce antonio alla X e W alla Y.

Ora si applica la sostituzione trovata con la coda della regola, cioè padre(X,Z), padre(Z,Y), tutto diventa quindi come segue:

padre(antonio,Z), padre(Z,W)

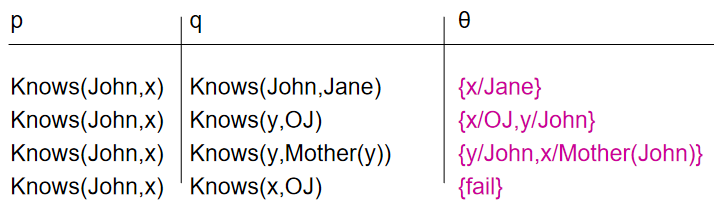
In generale si reitera tutto fino a quando non si ottiene una clausola vuota, l’equivalente di un’istanza sì.

## Unificazione

L’unificazione di due formule è una sostituzione che permette di renderle uguali, precisamente:

Data una sostituzione s e due formule a e b, Unify(a,b)=s se la sostituzione per a equivale a quella per b.

Per fare un esempio, si considera la seguente tabella:



Nel primo caso, basta sostituire x con Jane dato che x è l’unica incognita per rendere le formule uguali.

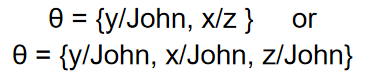
Lo stesso discorso vale anche per la seconda riga, in questo caso però le variabili sono due.

Il terzo è un caso interessante: il secondo parametro è una funzione sulla variabile y, ciò indica che y non può assumere Mother(y) come valore.

Per risolverla si va a vedere se si riesce a sostituire la y (in questo caso con John) e dopo si sostituisce la x direttamente con Mother(John) dal momento che ora si conosce il valore di y.

Il quarto e ultimo caso invece fallisce, ciò è causato dal fatto che la variabile x non può assumere sia John, sia OJ come valore.

Ovviamente esistono più sostituzioni per una data coppia di formule, per Knows(John,x) e Knows(y,z) ad esempio:



Tra tutte le possibili unificazioni, ce n’è una che è la più generale possibile ed è l’unica che permette la ridenominazione delle variabili.

Ci sono anche casi in cui l’unificazione fallisce, quali sono?

* caso in cui le funzioni sono differenti:

f(x) g(x)

* l’argomento x occorre in posizioni differenti della stessa funzione:

f(x,c) f(d,x)

* la stessa funzione prende sia x, sia una funzione di x:

f(x) f(g(x))

Quindi in generale si può dire che una funzione p(t1,...,tn) unifica con p(t1’,...,tn’) se e solo se ogni parametro di una funzione unifica col corrispondente dell’altra.

Da ciò si costruisce un sistema di equazioni convertibile in forma risolta se le formule atomiche sono unificabili.

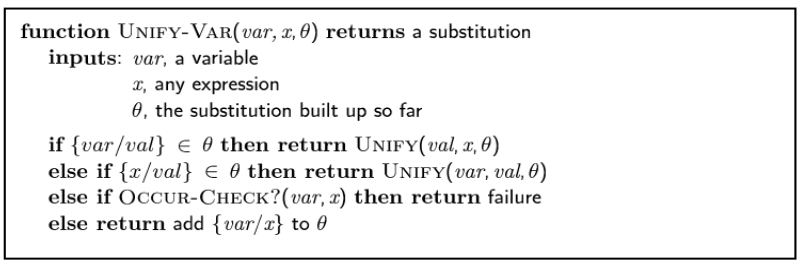
t1 =u t1’,..., tn =u tn’ → x1 =u u1, …, xk =u uk in cui ui non contiene xi.

La forma risolta permette di fornire l’unificazione più generale tra due funzioni.

Si può notare che:

* date due costanti c e d, l’unificazione di c con sè stesso è sempre vera mentre quella con d è sempre falsa;
* f(t1,...,tn) =u f(t1’,...,tn’) se e solo se t1=u t1’,...,tn=u tn’;
* x =u t se t non conviene x. Ogni occorrenza di x nelle equazioni viene sostituita con t.

## Algoritmo

L’implementazione dell’unificazione è ricorsiva:

* i primi casi base riguardano la sostituzione, si restituisce queste se x e y sono uguali oppure si propaga il fallimento;
* gli altri casi base vengono considerati se x o y sono variabili, in tal caso si effettua l’unificazione su di essi;
* se x e y sono composizioni o liste, si effettua la chiamata ricorsiva. Precisamente si ricalcola la sostituzione considerando due variabili, per tenere conto del resto si effettua una seconda chiamata.

L’unificazione delle variabili verifica se è possibile applicare una sostituzione:

* andando a controllare le occorrenze di tetha, si richiama opportunamente la funzione Unify;
* Qualora sia presente una sostituzione var/x, si propaga il fallimento;
* altrimenti si aggiunge var/x a tetha.

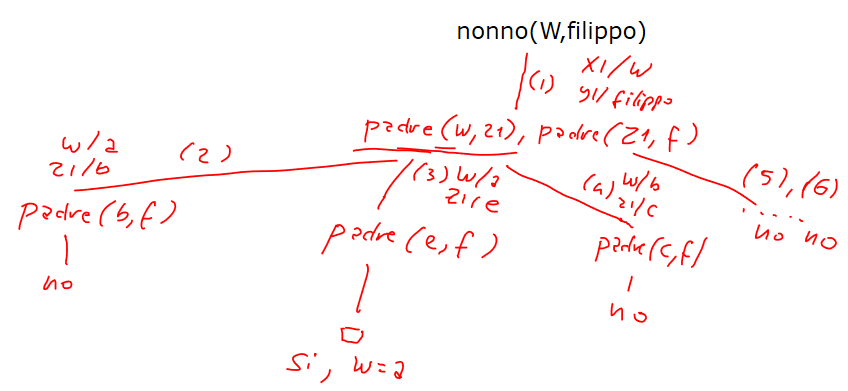
## Esercizio di risoluzione SLD con variabili

Data la conoscenza base a destra, si vuole verificare se esiste un nonno di Filippo:

nonno(W,filippo)

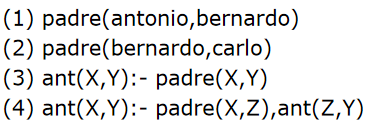
Come si fa a verificarlo?

Essenzialmente si controlla se ci sono occorrenze nella conoscenza base, in questo caso la prima regola permette di avanzare e di ottenere X1/W e Y1/filippo come prima sostituzione (come visibile nell'albero).

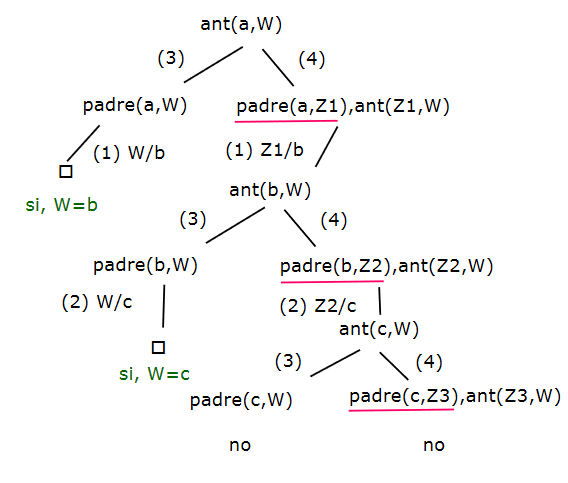
Ora bisogna andare a gestire la coda della prima regola, cioè padre(W, Z1), padre(Z1, filippo), per farlo si procede allo stesso modo.

La differenza rispetto a prima è che qui sono presenti più cammini.

Scegliendo la seconda regola, si ottiene padre(bernardo, filippo), ciò però non permette di ottenere la clausola vuota dal momento che questo fatto non è nella conoscenza base, di conseguenza questa formula non dà soluzioni.

Si prova ora ad applicare la terza regola: in questo caso si ottiene padre(eugenio, filippo) e, a differenza di prima, questa formula permette di ottenere la clausola vuota (dato che è nella base di conoscenza). Si può concludere che W è uguale ad antonio.

L’applicazione delle regole 4, 5 e 6 non permette di ottenere altri risultati in questo caso.

Una seconda ipotesi è quella di cambiare la regola di selezione gestendo prima padre(Z1, filippo), l’idea è quella di trovare prima il padre di filippo e poi il padre di quest’ultimo, permettendo in alcuni casi di trovare la soluzione più velocemente.

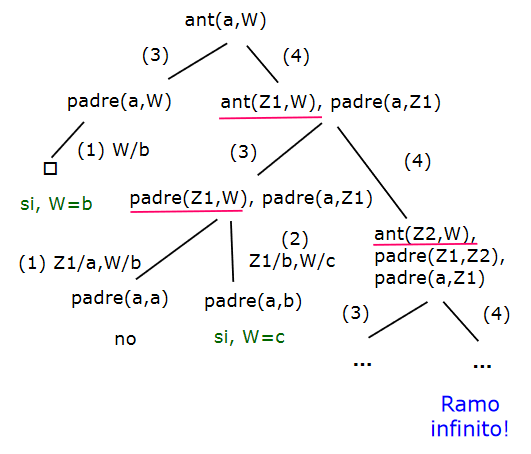
Si considera ora la base di conoscenza a destra: un antenato è una definizione ricorsiva in cui nel suo caso base è un padre e in quello ricorsivo è un padre di un altro antenato.

Considerando antenato(antonio,W), come si risolve?

La risoluzione rispetto a prima non cambia, in ogni possibile caso la sequenza di regola da applicare è sempre finita, come si può vedere dall’albero a destra.

## Cosa succede se si invertono padre e antenato nella quarta regola?

Ebbene, viene gestito prima il caso ricorsivo rispetto a quello base e ciò può portare a rami infiniti se si vogliono ottenere tutte le possibili soluzioni.

In questi casi anche la strategia di ricerca è importante:

* la strategia di default è quella leftmost, cioè va in profondità seguendo prima il ramo di sinistra;
* la strategia rightmost è la stessa cosa ma andando in profondità sul ramo di destra.

In questo esempio, l’utilizzo di queste regole con la strategia rightmost rischia una crescita infinita del ramo prima di trovare soluzioni, ciò si può vedere chiaramente nel secondo albero a destra.

Lo stesso risultato si ottiene con la ricerca rightmost cambiando la regola di selezione.

# Interpretazione procedurale del modello

Con la regola di selezione e la ricerca di default si ottiene l’interpretazione procedurale di un programma logico, nel senso che:

* l’insieme delle clausole di uno stesso predicato è simile alla definizione delle funzioni;
* gli argomenti sono comparabili ai parametri;
* il goal singolo può essere considerato come una chiamata a una procedura coi parametri attuali;
* unificazione è il passaggio parametri;
* I goal in congiunzioni sono paragonabili alle sequenze, l’ordine quindi può essere rilevante.

Ci sono anche delle differenze da un programma logico a uno normale, cioè la possibilità di definire relazioni. Data la definizione di una relazione binaria r(X,Y):

* questa formula può essere utilizzata con nessun argomento, uno tra X e Y oppure entrambi, questa proprietà è detta reversibilità ma ciò non è sempre possibile;
* per una X ci possono essere zero o più Y tali che soddisfino r(X,Y), può essere interessante ottenere uno oppure tutti i possibili punti.

Le varie clausole di una definizione permettono di trovare una soluzione dato che, quando interpretate come formule logiche:

* da una parte si descrive come funziona il mondo;
* dall’altra a sapere quali delle cose scritte sono effettivamente vere.

L’utilizzo delle clausole permette anche di trattare i casi mutuamente esclusivi, per farlo si fissa un sottoinsieme di variabili come input, per ognuno di essi si utilizza una sola clausola.

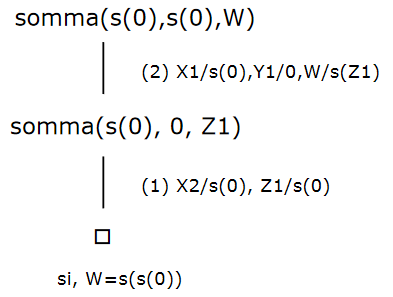
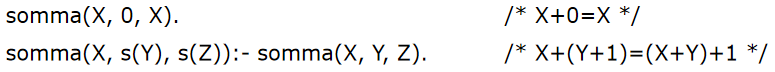
L’unificazione è più generale dato che serve per ottenere reversibilità in base alle variabili in input, specializzandosi su un caso particolare di una clausola, un goal o entrambi.

## Numeri naturali in logica

Per rappresentare i numeri naturali in logica, si possono utilizzare i numerali di Church: un numero n è la successione di n incrementi a partire da 0.

Con questa notazione si può definire la relazione ternaria somma(X,Y,Z) avente il seguente significato: Z è la somma di X e Y.

La definizione di somma ha un caso base che è somma(X,0,X), cioè la somma di X con 0 equivale a X stesso.

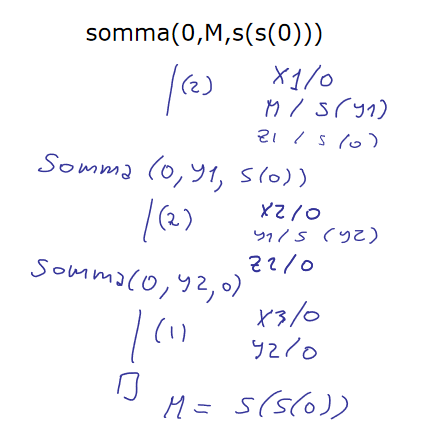
Il caso ricorsivo invece “srotola” il secondo argomento per ricorsione fino a quando non diventa 0, rientrando in questo modo nel caso base.

Utilizzando queste formule come base di conoscenza, come viene gestito somma(s(0), s(0), W)?

Essenzialmente si applica prima la seconda regola, ottenendo {X1/s(0), Y1/0 e W/s(Z1)} come sostituzione.

Dopo si applica la prima regola, ciò permette di ottenere la clausola vuota, quindi dalla sostituzione {X2/s(0), Z1/s(0)} ottenuta in questo momento più quella precedente, si ottiene che W è uguale a s(s(0)).

E come si fa a gestire somma(0,M,s(s(0)))?

La gestione non è tanto differente, quello che si fa infatti è applicare tante volte la seconda regola fino a quando non si ottiene 0 sul terzo parametro, quindi applicare la prima regola.

Nell’esempio si applica la seconda regola due volte (ricavando {X1/0, M/s(Y1), Z1/s(0)} e {X2/0, Y1/s(Y2), Z1/0} come sostituzioni) e si ottiene la clausola vuota applicando infine la prima, quindi M equivale a s(s(0)).

# Come si gestiscono le liste usando le regole?

Le liste sono un esempio di termini complessi , esse possono essere costruite con una qualunque costante e un qualunque simbolo a due argomenti.

La notazione per le liste è la seguente:

* vuota: indica la lista vuota;
* lista(Testa,Resto) è una lista in cui Testa è il primo elemento e Resto è la sottolista rimanente;

Ciò vuol dire che lista(4, lista(2, lista(7,vuota))) è una lista contenente 4, 2 e 7, esiste però una notazione più convenzionale più leggera e flessibile e con predicati built-in:

* [] indica la lista vuota;
* [Testa|Resto] indica una lista il cui primo elemento è Testo e Resto è la sottolista rimanente.

Da ciò si può quindi ridurre lista(4, lista(2, lista(7,vuota))) in [4|[2|7|[]]]] oppure più semplicemente [4, 2, 7].

Quest’ultima notazione è più comoda dato che con essa si può scrivere una lista con un numero di elementi dati oppure per metterne in evidenza un certo numero.

In generale, rinunciando all’invertibilità degli argomenti si possono gestire varie strutture e casi.

## Unificazione di liste

Considerando p([a,b,c,d] e p(X|Y), l’unica possibile unificazione è X=a e Y=[b,c,d].

Con p([a,b,c,d]) e p(X,Y|Z), l’unificazione è X=a, Y=b e Z=[c,d].

L’unificazione per p([a]) e p(X|Y) è X=a e Y=[] dato che la lista non contiene altri elementi.

Per p([]) e p([X|Y]) invece non esiste un’unificazione dal momento che si riescono a trovare valori di X e Y adeguati.

## Operazioni su liste

### Predicato is\_list

Con [|] è possibile costruire anche strutture che non sono liste, ciò è dovuto alla mancanza di un controllo sintattico.

Come si fa quindi a capire se una struttura è una lista o meno?

Ebbene, si utilizza il predicato is\_list(L) il quale verifica che L è una lista, la sua definizione è la seguente:

* is\_list([]): la lista vuota è a tutti gli effetti una lista;
* is\_list([X|L]):- is\_list(L): una lista è tale se lo è anche la sua sottostruttura L.

Per fare degli esempi, considerando is\_list([a,b]), questa chiama is\_list([b]) che a sua volta chiama is\_list([]), portando tutto a termine.

Considerando invece is\_list([a|b]), questa chiama is\_list(b) e ciò porta a un fallimento.

### Predicato member

Il predicato member(T,L) indica se T è un elemento della lista L, con la seguente definizione:

* member(T,[T,\_]): T è il primo elemento;
* member(T,[\_,|L]):-member(T,L): T si trova nella sottolista.

“\_” indica una variabile diversa da tutte le altre di cui non interessa il nome. Questo predicato può essere utilizzato per verificare se un elemento è dentro una data lista oppure per trovare tutti gli elementi, ad esempio: Considerando member(X,[5,3,8]), la risoluzione porta alla costruzione dell’albero a destra.

### Predicato append

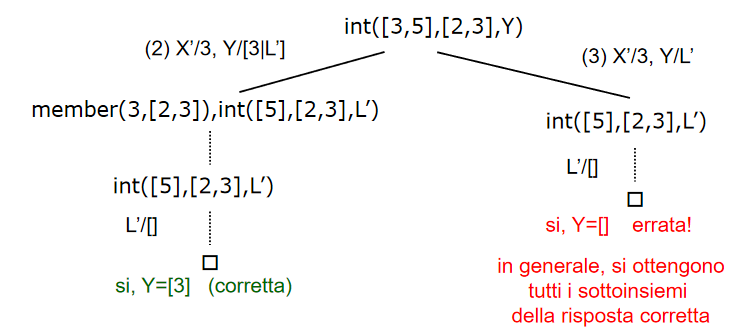
Il predicato append(L1,L2,L3) indica che L3 è la concatenazione di L1 e L2, per dirlo utilizza la seguente definizione:

* append([],L,L): la concatenazione di una lista con la lista vuota dà la lista stessa;
* append([H|L1],L2,[H|L3]):-append(L1,L2,L3): se le teste della prima e della terza lista sono uguali, si controllano le sottoliste.

L’idea è quella di effettuare la ricorsione sul primo argomento fino a quando non è una lista vuota, permettendo di applicare in questo modo la prima regola nella ricorsione.

### Predicato int

int(L1,L2,L3) è un predicato duale alla concatenazione, questo indica che L3 è l’intersezione delle liste L1 e L2, definita nel seguente modo:

* int([],L2,[]): l’intersezione con una lista vuota genera la lista vuota;
* int([X|L1],L2,[X|L]):- member(X,L2), int(L1,L2,L): se la prima e la terza lista hanno la testa uguale a X, si verifica se X è in L2 e si reitera sulle sottoliste;
* int([X|L1],L2,L):- int(L1,L2,L): se condizione del precedente punto è falsa, si reitera sulle sottoliste senza controllare se X è in L2.

Questo approccio funziona se la risoluzione si ferma alla prima soluzione trovata, senza questo limite ci sono anche casi indesiderati (corrispondenti a sottoinsiemi della soluzione corretta).

### Costrutti per il controllo

Cosa si può fare per risolvere il problema dell’intersezione?

Un prima soluzione è quella di aggiungere un predicato per verificare se la soluzione è davvero corretta, un’altra invece è quella di provare a unificare le regole ma ciò non è possibile.

Per casi come questi è stato introdotto il taglio col carattere “!” per eliminare le alternative per il goal genitore. Un effetto collaterale del tagli è il fatto che, oltre a tagliare i “fratelli” dallo spazio di ricerca, vengono anche tagliate le soluzioni alternative.

Un controllo di questo tipo può avere la seguente definizione:

* a(X):-b(X),!,c(X).
* a(X):-d(X)

Ciò si può leggere come un’if-then-else e si può ripetere a cascata, il risultato dipende dall’ordine delle clausole, inoltre il significato delle clausole stesse dipende dalle precedenti.