Università di Perugia
Dipartimento di Matematica e Informatica

Appunti *Knowledge Representation and Automated Reasoning*

Riassunto per Esame Orale

Autori: *Chiara Luchini, Cristian Cosci, Fabrizio Fagiolo, Nicolò Posta, Tommaso Romani, Nicolò Vescera*

Anno Accademico 2022-2023

Last Update: March 8, 2023

Contents

*"E voi siete capaci a
risolvere i vari conflitti che si creano?
Perché io queste tecnologie come
Gtab (GitHub), non le ho mai usate."*

Stefano Bistarelli

Chapter 1

Constraint Satisfaction Problems

1.1 Definizione di CSP

Un vincolo è semplicemente una relazione logica tra diverse incognite (o variabili), ognuna delle quali assume un valore in un dato dominio. Un vincolo quindi restringe i possibili valori che le variabili possono assumere e ne rappresenta alcune informazioni parziali sulle variabili di interesse.
Formalmente, **un constraint satisfaction problem (o CSP)** è definito da:

- Un insieme di variabili $X_1, X_2, ..., X_n$;
- Una funzione che mappa ogni variabile a un dominio finito;
- Un insieme di vincoli $C_1, C_2, ..., C_m$;
- Un insieme D_i non vuoto di possibili valori per ogni variabile X_i .

Ogni vincolo C_i coinvolge alcuni sottoinsiemi di variabili e specifica le combinazioni di valori consentite per quel sottoinsieme. Uno **stato** del problema è definito da un'**assegnazione di valori** ad alcune o a tutte le variabili $\{X_i = v_i, X_j = v_j, ...\}$. Un'assegnazione che non viola alcun vincolo è chiamata **assegnazione coerente o legale**. Un'**assegnazione completa** è quella in cui viene menzionata ogni variabile, e una **soluzione** a un CSP è un'assegnazione completa che soddisfa tutti i vincoli. Alcuni CSP richiedono anche una soluzione che **massimizzi una funzione obiettivo**. Ciascun vincolo limita la combinazione di valori che un insieme di variabili può assumere contemporaneamente. Una soluzione di un CSP è l'assegnazione a

ciascuna variabile di un valore dal suo dominio che soddisfi tutti i vincoli. Il compito è trovare una soluzione o tutte le soluzioni.
Pertanto, il CSP è un problema combinatorio che può essere risolto mediante la ricerca.
Per comprendere meglio un problema di soddisfacimento di vincoli è opportuno evidenziare i concetti di stato e di assegnamento:

- **Stato:** Rappresenta ogni combinazione di valori assunti dalle variabili $\{x_1, \ldots, x_n\}$.
- **Assegnamento:** Rappresenta la soluzione del problema, cioè assegnare dei valori a tutte le variabili (assegnamento completo) in modo tale da soddisfare tutti i vincoli del problema ("assegnamento consistente"). Il compito è trovare una soluzione o tutte le soluzioni. Pertanto, il CSP è un problema combinatorio che può essere risolto mediante la ricerca.

1. Un assegnamento è **inconsistente** se viola un vincolo.
2. Un CSP è **soddisfacibile** se esiste almeno una soluzione.
3. Un assegnamento a un sottoinsieme S delle variabili è localmente consistente se soddisfa i vincoli esistenti tra le variabili in S.

1.1.1 Esempi

1.1.1.1 Esempio 1

Problema: Assegnazione delle ore di lezione dei professori (Time Tabling).

Dato: "Il prof A non può fare lezione dalle 11 alle 13", " Il prof B non può fare lezione nell'Aula B" che sarebbero i vincoli, l'obiettivo è avere un assegnamento per tutte le variabili del problema (che saranno i professori e le aule).

Il CSP quindi è un modo di rappresentare i problemi attraverso una serie di relazioni.

1.1.1.2 Esempio 2

Lavoriamo in una mappa dell'Australia che mostra ciascuno dei suoi stati e territori e il compito ci viene affidato è di colorare ogni regione di rosso, verde o blu in modo tale che le regioni vicine non hanno lo stesso colore. Per formulare questo come un CSP, definiamo le variabili come le regioni: WA, NT, Q, NSW , V , SA e T. Il dominio di ciascuna variabile è l'insieme

$\{rosso, verde, blu\}$. I vincoli richiedono che le regioni vicine abbiano colori distinti; ad esempio, le combinazioni consentite per WA e NT sono le coppie

$$\{(rosso, verde), (rosso, blu), (verde, rosso),\\ (verde, blu), (blu, rosso), (blu, verde)\}$$

Ci sono molte soluzioni possibili, come $\{$WA = rosso, NT = verde, Q = rosso, SA = blue, NSW = verde, V = rosso, T = verde$\}$.
È utile visualizzare un CSP come grafico di vincoli. I nodi del grafico corrispondono a variabili del problema e gli archi corrispondono a vincoli

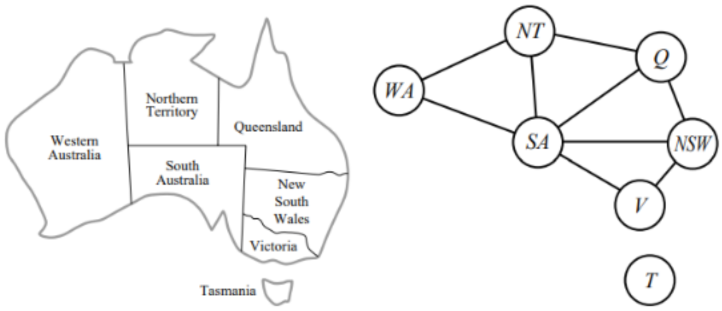


Figure 1.1: Map-Coloring

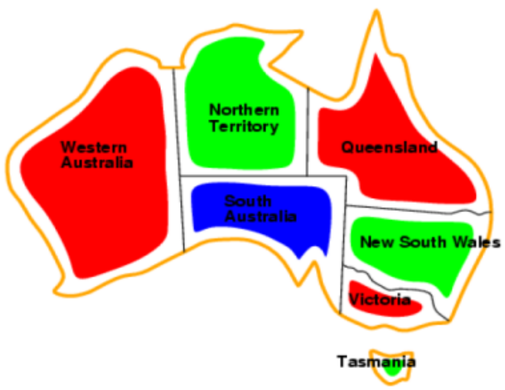


Figure 1.2: Soluzione Map-Coloring

In un CSP binario ogni vincolo è in relazione con due variabili. Il tipo più semplice di CSP coinvolge variabili che sono discrete e hanno domini finiti, i problemi di colorazione della mappa sono di questo tipo.

1.1.2 Tipi di Variabili

Le variabili **discrete** possono avere domini:

- **Finiti:** grandezza dell'assegnamento completo $d \to O(d^n)$;
- **Infiniti:**
  - hanno bisogno di un linguaggio di vincoli
  - i vincoli lineari sono risolvibili, i non lineari sono indecidibili;

Le variabili **continue**:

- Tempo di inizio e fine per specifici problemi (Hubble telescope)

Le variabili continue con dei vincoli lineari sono risolvibili in un tempo polinomiale dai metodi LP.

1.1.3 Tipi di vincoli

- **Unari:** I vincoli coinvolgono una singola variabile. $SA \neq green$
- **Binari:** I vincoli coinvolgono coppie di variabili. $\text{SA} \neq \text{W A}$
- **High order:** I vincoli coinvolgono tre o più variabili.
- **Preferences:** Ho una preferenza su un valore piuttosto che un altro per una certa variabile. Questi sono anche chiamati vincoli Soft, che rappresentano un costo per ogni *assegnamento* di variabile.

Inoltre i vincoli possono essere espressi in maniera:

- **Implicita:** non viene direttamente indicata la relazione fra gli elementi del dominio che sono permessi. Un esempio può essere $x < y$, dove non si elencano tutti i possibili assegnamenti delle variabili che non violano quel vincolo ma si possono calcolare;
- **Esplicita:** si elencano tutti i valori ammessi per le variabili coinvolte nel vincolo.

Nell'esempio precedente si avranno tutte le coppie di valori ammessi in base a quel vincolo.

1.1.4 Proprietà dei CSP

1. **Commutatività:** Si considera l'assegnamento di una singola variabile per volta.
2. **Monotonicità:** Appena un vincolo viene violato posso interrompere la ricerca perché sono sicuro che non esisterà soluzione.
3. **L'ordine** con il quale seleziono le variabili e assegno i valori è molto importante, abbiamo bisogno di euristiche intelligenti.
4. **Indipendenza:** quando le variabili non hanno vincoli tra di loro (come ad esempio sulla mappa dell'Australia c'era la T che non era collegata agli altri nodi), il problema si può decomporre in sotto problemi che possono essere risolti in modo indipendente.
5. É applicabile un controllo per **consistenza:** Invece di fare un controllo di consistenza alla fine, lo faccio ad ogni passo.
6. Possono essere visti come un problema di ricerca.

1.1.5 Standard search formulation

Gli stati sono definiti dal valore assegnato finora:

- **Stato iniziale:** assegnamento vuoto $\{\}$;
- **Funzione successore:** assegna un valore a una variabile non assegnata che non va in conflitto con l'assegnamento corrente;
- **Goal test:** l'assegnamento corrente è completo. Questa formula viene usata per tutti i CSP e ogni soluzione appare a profondità n con n variabili. Il cammino è irrilevante, così che può usare la formulazione complete-state.

1.2 Metodi di ricerca sistematici

I CSP sono problemi combinatori, quindi si deve trovare una soluzione che appartiene all'insieme dato da tutte le possibili combinazioni di assegnamenti. I vari metodi per fare questo sono:

1. **Generate and Test:** Probabilmente il metodo di risoluzione dei problemi più generale. L'algoritmo consiste nei seguenti due passaggi che si ripetono:

(a) si generano le etichette

(b) si controlla se vanno bene gli assegnamenti.

Alcuni possibili miglioramenti sono ad esempio uno smart generator, ovvero si assegnano i valori alle variabili e se non vanno bene si effettuano dei cambiamenti sugli assegnamenti errati (**ricerca locale**). Un altro miglioramento consiste nel fare il test sugli assegnamenti e poi fare backtracking.

2. **Backtracking:** Assegno uno alla volta i valori alle variabili finché non violo un vincolo; appena lo violo, torno indietro nell'assegnamento. Si estende quindi una soluzione parziale verso una soluzione completa.

1.2.1 Backtracking nel dettaglio

Supponiamo di avere un problema CSP con 4 variabili (A, B, C, D), supponiamo che i vincoli siano

$$A = D, B \neq D, A + C < 4$$

L'algoritmo consiste in:

1. Assegna il valore alla variabile;
2. Si controlla la consistenza;
3. Finché tutte le variabili sono etichettate;

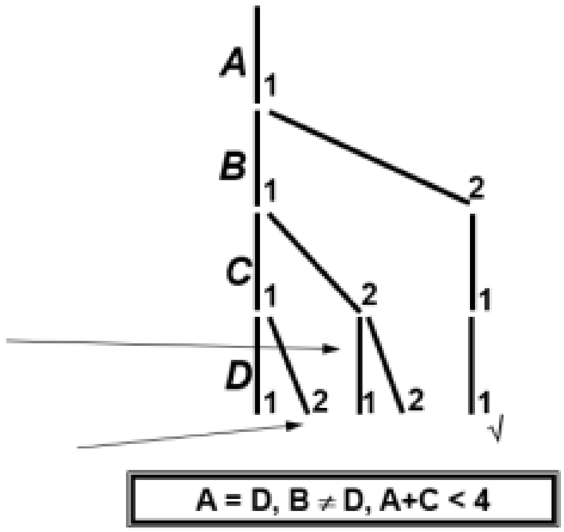


Figure 1.3: Esempio backtracking

Se tutte le variabili non sono etichettate si torna al passo 1. L'assegnamento delle variabili è commutativo, ad esempio

$$[WA = red \text{ allora } NT = green] \text{ come } [NT = green \text{ allora } WA = red]$$

Si devono solo considerare gli assegnamenti alle singole variabili per ogni nodo → b = d e ci sono d^n foglie.
La **DFS** per i CSP son gli assegnamenti a variabili singole è chiamata ricerca **backtracking**, essa è l'algoritmo base uniformed per i CSP.

- Problem:
  X::{1,2}, Y::{1,2}, Z::{1,2}
  X = Y, X ≠ Z, Y > Z

generate & test

X	Y	Z	test
1	1	1	fail
1	1	2	fail
1	2	1	fail
1	2	2	fail
2	1	1	fail
2	1	2	fail
2	2	1	passed

backtracking

X	Y	Z	test
1	1	1	fail
		2	fail
	2		fail
2	1		fail
	2	1	passed

Figure 1.4: Esempio generate and test vs backtracking

1.2.1.1 Ricerca backtracking: incrementale

La strategia di ricerca è la **DFS (Depth First Search):**

1. scegli una variabile non istanziata, scegli un valore dal suo dominio, controlla se qualche vincolo è violato;
2. se nessun vincolo viene violato, continua la ricerca in modo ricorsivo;
3. altrimenti, torna indietro: torna alla decisione precedente e fai un'altra scelta.

In termini di grandezza dell'albero di ricerca il numero di foglie è pari a d^n, dove n è il numero di variabili e $d = max|D_i|$.

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING([], csp)

function RECURSIVE-BACKTRACKING(assigned, csp) returns solution/failure
   if assigned is complete then return assigned
   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assigned, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assigned, csp) do
      if value is consistent with assigned according to CONSTRAINTS[csp] then
         result ← RECURSIVE-BACKTRACKING([var = value|assigned], csp)
         if result ≠ failure then return result
   end
   return failure
```

Figure 1.5: Algoritmo di backtracking.

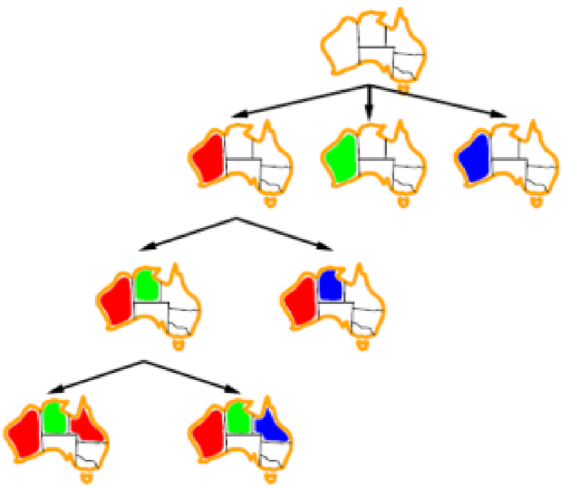


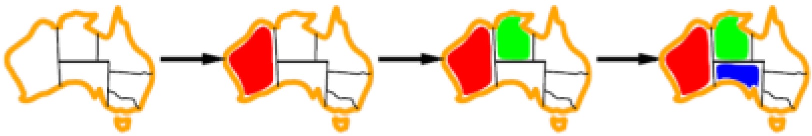
Figure 1.6: Esempio backtracking.

1.2.2 Miglioramenti dell'efficienza del backtracking

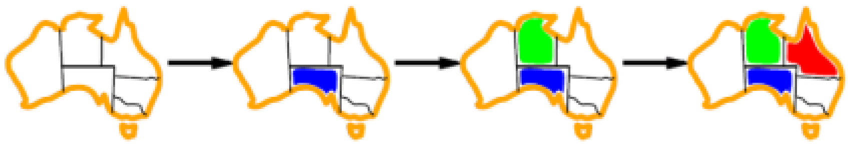
Per impostazione predefinita, SELECT-UNASSIGNED-VARIABLE seleziona semplicemente la successiva variabile non assegnata nell'ordine dato dalla lista VARIABLES[csp].
Questo ordinamento di variabili statiche raramente si traduce nella ricerca più efficiente. Ad esempio, dopo le assegnazioni per WA = rosso e NT = verde, c'è un solo valore possibile per SA, quindi ha senso assegnare SA = blue next piuttosto che assegnare Q. Infatti, dopo l'assegnazione di SA, le scelte per Q, NSW e V sono tutte forzate.

Variabile più vincolata. Con questo approccio si va a scegliere, fra le variabili disponibili per l'assegnamento, quella più vincolata in base ad alcune caratteristiche:

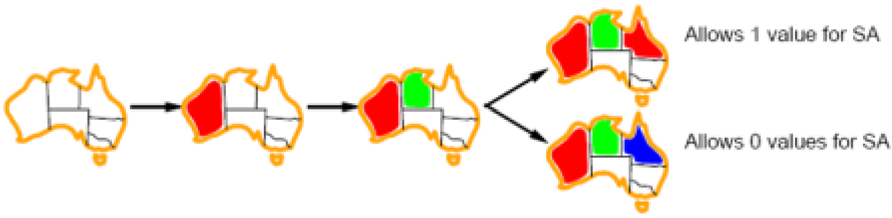
- si sceglie la variabile con il **minor numero di valori legali** (*minimum remaining values-MRV*). È stata anche chiamata la "variabile più vincolata" o euristica "fail-first", quest'ultima perché seleziona una variabile che ha maggiori probabilità di causare un errore presto, potando così l'albero di ricerca.



- si sceglie la variabile con **più vincoli possibili** (*degree heuristic*) sulle variabili rimanenti.



- data una variabile, si sceglie **il valore meno vincolante** (*least-constraining-value*), quello che esclude il minor numero di valori nelle restanti variabili.

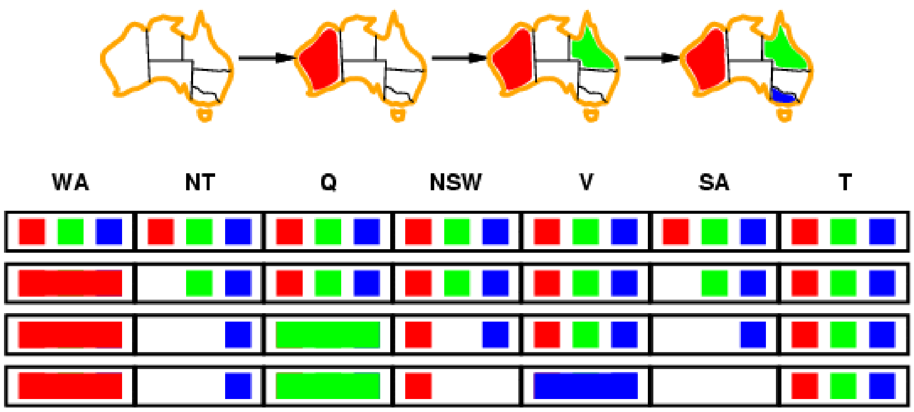


1.2.3 Propagazione delle informazioni attraverso vincoli

Finora il nostro algoritmo di ricerca considera i vincoli su una variabile solo nel momento in cui la variabile viene scelta da SELECT-UNASSIGNED-VARIABLE. Ma guardando alcuni dei vincoli all'inizio della ricerca, o anche prima dell'inizio della ricerca, possiamo drasticamente ridurre lo spazio di ricerca.

1.2.3.1 Forward Checking

Prima di assegnare una variabile controlla che questo assegnamento mi renda possibile futuri assegnamenti ad altri nodi, poiché se dopo l'assegnamento, un dominio per una variabile diventasse vuoto, non si avrebbe soluzione e quindi l'assegnamento non andrebbe nemmeno provato.
Euristica utilizzata: elimina i valori dal dominio quando viene fatto un assegnamento ad una variabile, **se un dominio diventa vuoto, interrompi la ricerca poiché la soluzione non esiste con gli assegnamenti fatti**. Questo metodo offre spesso una soluzione senza avere backtrack.



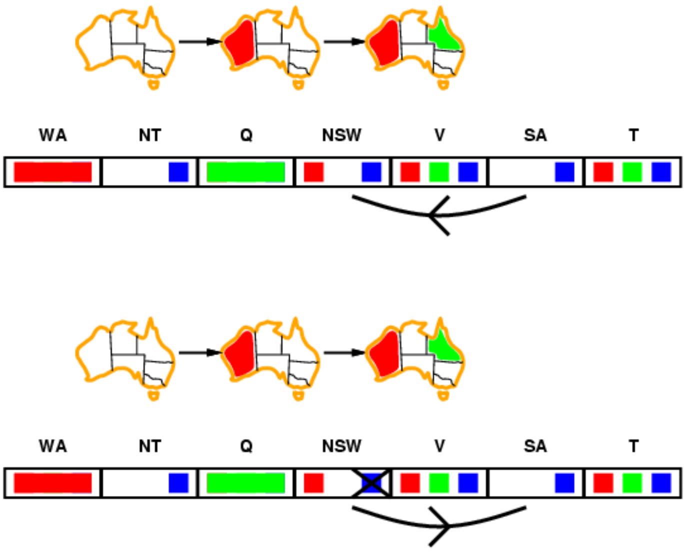
	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	R	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	R	B	G	R B	R G B	B	R G B
After *V=blue*	R	B	G	R	B		R G B

Figure 1.7: Forward checking applicata a Map-coloring problem.

In questo caso assegnare blu a V rende possibili zero colori per SA, quindi questo tipo di assegnamento non può andare bene.

1.2.3.2 Constraint propagation

Sebbene il forward checking rilevi molte incoerenze, non le rileva tutte. Per esempio, consideriamo la terza riga della Figura 2.5. Essa mostra che quando WA è rosso e Q è verde, sia NT che SA sono costretti a essere blu ma questo non è possibile perché sono due zone vicine. Il forward checking non rileva questo come un'incoerenza, perché non guarda abbastanza avanti. La propagazione del vincolo (constraint propagation) è il termine generale per propagare le implicazioni di un vincolo su una variabile su altre variabili; in questo caso dobbiamo propagare da WA e Q su NT e SA, e quindi sul vincolo tra NT e SA per rilevare l'incoerenza.

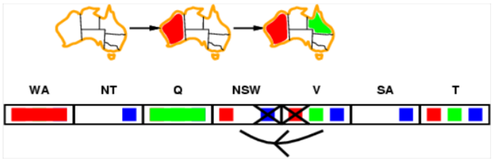


Simplest form of propagation makes each arc consistent

***X* → *Y* is consistent iff**

for every value *x* of *X* there is some allowed *y*

If *X* loses a value, neighbors of *X* need to be rechecked



Simplest form of propagation makes each arc consistent

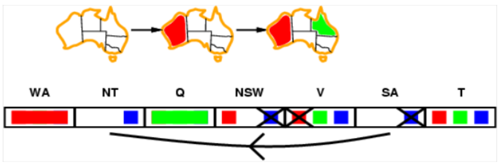
***X* → *Y* is consistent iff**

for every value *x* of *X* there is some allowed *y*

If *X* loses a value, neighbors of *X* need to be rechecked

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment



1.3 Local Consistency nei CSP

La consistenza locale è la condizione che si verifica quando ogni nodo di una rete soddisfa tutti i vincoli che lo coinvolgono. Quest'ultima nei CSP consente di **ridurre lo spazio di ricerca** di un algoritmo eliminando dal dominio delle variabili i valori che non rispettano i vincoli, cioè che non hanno **supporto**; può essere definita in rapporto alla consistenza di nodo, arco o cammino.

- **Consistenza di Nodo:** Consiste nella verifica dei valori del dominio di un nodo rispetto ai vincoli del nodo stesso. Ad esempio, se il dominio del nodo x_1 è pari a $\{1, 2, 3\}$ e il vincolo unario di x_1 è $x_1 > 1$, la consistenza di nodo consente di ridurre il dominio di x_1 a $\{2, 3\}$ riducendo il processo di ricerca dell'algoritmo.

- **Consistenza di Arco:** Consiste nella verifica dei valori di un dominio di un nodo rispetto ai vincoli binari con un altro nodo. Ad esempio, dati due nodi x_1 e x_2 con rispettivi domini $\{1, 2, 3\}$ e $\{1, 4, 5, 9\}$, il vincolo $x_2 = x_1^2$ consente di ridurre il dominio del nodo x_2 a $\{1, 4, 9\}$ poiché non sussiste alcun nesso tra il valore 5 dell'insieme dominio di x_2 e qualsiasi altro elementi dell'insieme dominio di x_1.

- **Consistenza di Cammino:** Consiste nella verifica dei valori di un nodo rispetto ai vincoli n-ari con altri n nodi. La consistenza di cammino è un estensione della consistenza di arco ed è detta anche k-consistenza. Si distingue dalla consistenza di arco che, invece, è dedicata ai vincoli binari tra due nodi.

1.3.1 Node Consistency

In questo caso tutti i vincoli sono unari, le variabili sono rappresentate da dei **vertici**. Il vertice è **node consistent** se ogni valore nel dominio delle variabili soddisfa tutti i vincoli unari imposti sulla variabile X. Spesso vengono rappresentati come un cappio o arco che ritorna sullo stesso nodo.

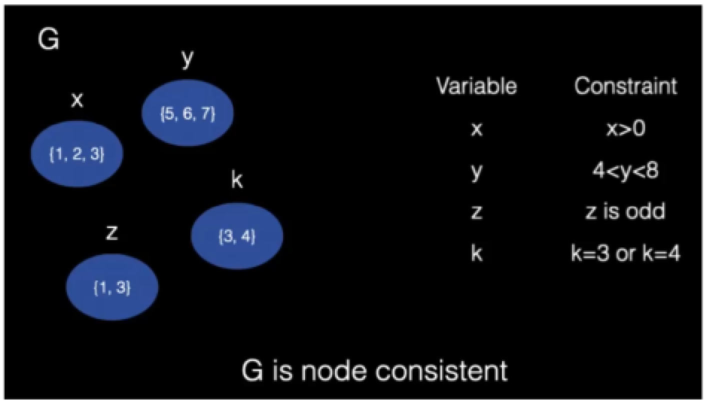


Figure 1.8: Esempio grafo node consistent

1.3.2 Arc Consistency

Quando tutti i vincoli sono binari si parla di arc consistency. Un arco (u, v) è consistente se per ogni valore x del dominio $dom(u)$ esiste un valore y nel $dom(v)$ tale che un assegnamento $u = x$ e $v = y$ soddisfa tutti i vincoli binari che coinvolgono sia u che v.

Punto fisso e supporto: Tutto il processo di consistenza locale viene applicato fin quando non si riesce a trovare un punto fisso, cioè fino a quando il dominio delle variabili resta invariato.

Problema arc-consistente: Un problema nel quale applicando local consistency, nessun dominio di nessuna variabile viene modificato.

Supporto: Per la riduzione dobbiamo vedere quali elementi hanno supporto in Y, ovvero quali hanno un assegnamento in Y tale per cui il vincolo viene soddisfatto.

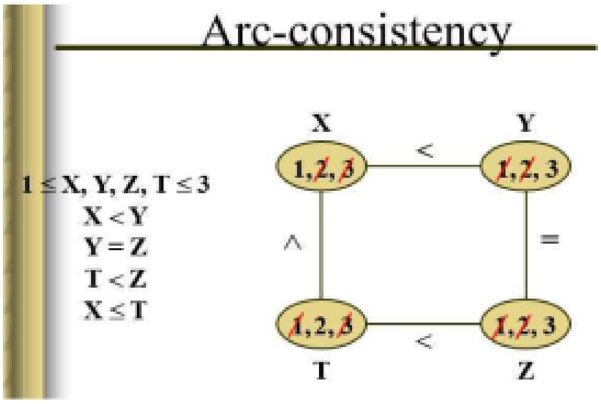


Figure 1.9: Esempio arc-consistency

Quando $X = 1$ esiste un elemento in Y tale per cui $X < Y$? Si, 2,3
Quando $X = 2$ esiste un elemento in Y tale per cui $X < Y$? Si, 3
Quando $X = 3$ esiste un elemento in Y tale per cui $X < Y$?
No, quindi posso eliminare 3 il dominio di X

L'operazione che ho fatto si chiama Local Consistency, ma dato che ho utilizzato il vincolo binario si chiama Arc Consistency.

Conclusione: In questo caso sono stato molto fortunato perché è rimasto 1 solo valore possibile per ogni variabile. Quindi la soluzione è unica e sarà:

$$X = 1, Y = 3, Z = 3, T = 2$$

Se fossi stato meno fortunato comunque avrei ridotto gli elementi del dominio e quindi quando sarei andato a fare la ricerca con backtrack comunque lo spazio di ricerca si sarebbe di molto diminuito (avrei controllano meno assegnamenti).

Vantaggio dell'applicazione della Local Consistency prima della Backtracking: Se faccio backtrack il dominio degli stati che vado ad esplorare è esponenziale perché ad X dovrei moltiplicare tutti i possibili valori di Y, a cui dovrei moltiplicare tutti i possibili valori di Z che moltiplico per tutti i valori possibili per T. All'inizio quindi in quel problema si avevano $3^4 = 81$ possibili assegnamenti, mentre se applico prima Local Consistency il problema diventa polinomiale (facile) rispetto al numero di variabili che coinvolgo (in questo caso lo spazio di ricerca mi si riduce ad 1).

1.3.3 Algoritmi per Consistenza Locale

1.3.3.1 Procedura Revise

Per fare diventare un arco (u, v) consistente si cancellano tutti i valori x dal $dom(u)$ che sono inconsistenti con tutti i valori in $dom(v)$.
Restituisce true se è stata fatta una modifica al dominio di u.

```
procedure REVISE((u,v))
  DELETED <-  false
    for each x in dom(u) do
      if there is no such y in dom(v) such that (x, y) is consistent then
        delete X from dom(u)
        DELETED <- true
      end if
    end for
  return DELETED
end REVISE
```

Figure 1.10: Procedura REVISE

1.3.3.2 Algoritmo AC-1

Un singolo passo dell'algoritmo REVISE non è sufficiente. L'algoritmo base per arc consistency è AC-1, il quale esegue l'algoritmo REVISE finché il dominio delle variabili cambia. In questo si ripete la procedura REVISE ogni volta che viene modificato un valore in un dominio.

```
procedure AC-1(G)
  repeat
    CHANGED  false
      for each arc (u,v) in G do
        CHANGED <- REVISE((u,v)) or CHANGED
      end for
  until not(CHANGED)
end AC-1
```

Figure 1.11: Algoritmo AC-1

Una sola revisione riuscita di un arco su una particolare iterazione causa la revisione di tutti gli archi nella prossima iterazione anche se gli archi non sono influenzati dal cambiamento.
L'inefficienza sta nel fatto che se anche una singola chiamata alla procedura di REVISE su una particolare iterazione risultasse vera, tutti gli archi verrebbero scansionati nuovamente nella successiva iterazione. Questo però non è sempre necessario, perché la modifica del dominio di una variabile influenza solamente le variabili che sono collegate a X da un vincolo e non le restanti.
La complessità temporale è $O(e \cdot n \cdot d^3)$ e quella spaziale è $O(e \cdot n \cdot d)$.

1.3.3.3 Algoritmo AC-2

AC-2 è un algoritmo che può fare arc consistency in un solo passo attraverso i nodi. Il risultato è ottenuto passando per i nodi in un ordine numerico:

- Allegare ad un nodo tutti i valori che non sono in conflitto con i nodi precedentemente assegnati;
- Guarda i vicini di questo nodo che sono stati già valutati; se un valore non ha un'assegnazione corrispondente per lo stesso arco, eliminalo;
- Ogni volta che qualsiasi valore è cancellato da un arco, guarda ai suoi vicini a sua volta, e controlla se un loro valore può essere eliminato. Se può essere eliminato, si continua il processo iterativamente finché non ci sono più cambiamenti che possono essere fatti. Poi si prosegue con gli altri archi.

In sostanza si sceglie un ordine fra i nodi, prendiamo ad esempio y come primo e controlliamo tutti i vincoli fra y e k se c'è qualche valore di $dom(y)$ che va in conflitto allora si elimina. Stessa cosa si per l'arco (x, y) se si fa qualche modifica si rimette in coda l'arco in modo da controllare se ci sono altre modifiche da fare. Se un valore b del nodo i è rimosso allora si aggiunge tutti (k, i) alla coda Q, per il controllo degli archi.

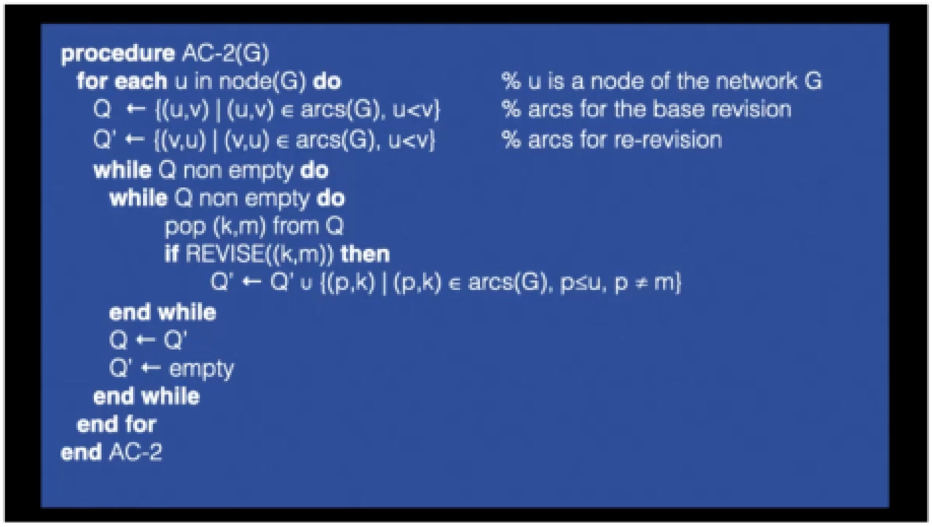


Figure 1.12: Algoritmo AC-2

La complessità temporale è $O(n^2 \cdot d^3)$ e quella spaziale è $O(n^2)$.

1.3.3.4 Algoritmo AC-3

AC-3 è un miglioramento di AC-2, alcuni di essi possono essere già nella coda Q. Se è così allora non dovrebbero essere inseriti di nuovo.

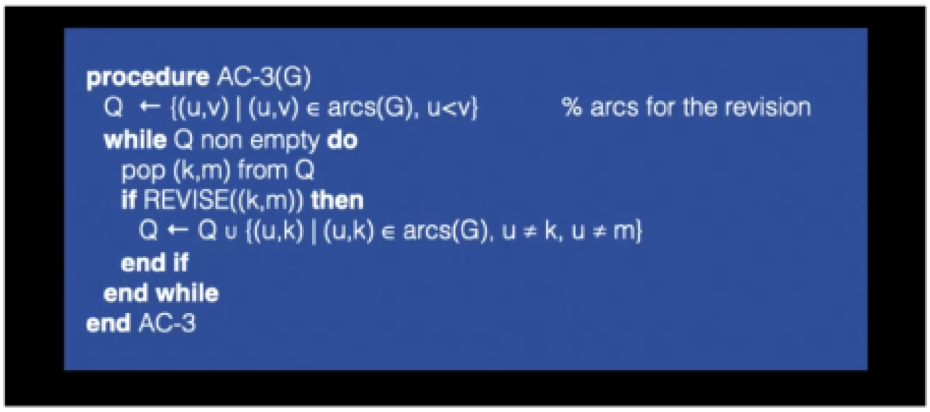


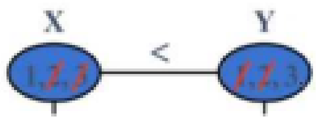
Figure 1.13: Algoritmo AC-3

In sostanza si prende un arco dalla coda Q, si fa la REVISE, se faccio una modifica allora si aggiunge alla coda Q tutti gli archi (u, k) che non sono stati già controllati ovvero $u \neq k$ e $u \neq m$.
La complessità temporale è $O(n^2 \cdot d^2)$ e quella spaziale è $O(n^2)$.

1.4 Ricapitolando

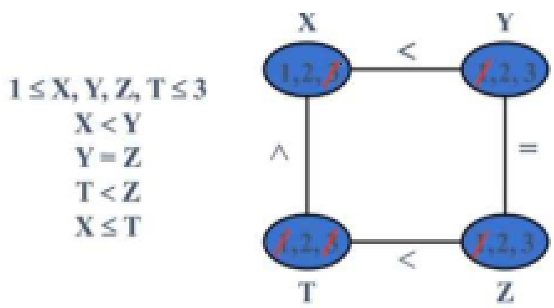
1.4.1 Arc-consistency

Algoritmo che ha la caratteristica di lavorare a livello locale sul CSP. L'obiettivo è quello di semplificarlo in maniera tale da ridurre la quantità di elementi possibili nel dominio e quindi ridurre il tempo necessario quando faccio la ricerca con backtrack.
Supporto: un elemento del dominio di X può essere buttato via se non ha supporto in y.



Ad esempio, 1 ha supporto in X perché in Y c'è 2 e 3. Se un elemento non ha supporto lo posso escludere dai valori assegnabili alla variabile. Un problema si dice arc-consistente se ho buttato via tutti gli elementi che NON hanno supporto.

Questo è un problema Arc-Consistente?



No, perché il 2 andrebbe eliminato da Z. Se riesco a buttare via tutti gli elementi che non hanno supporto allora il problema diventa arc-consistente.

1.4.2 Soluzione di un CSP

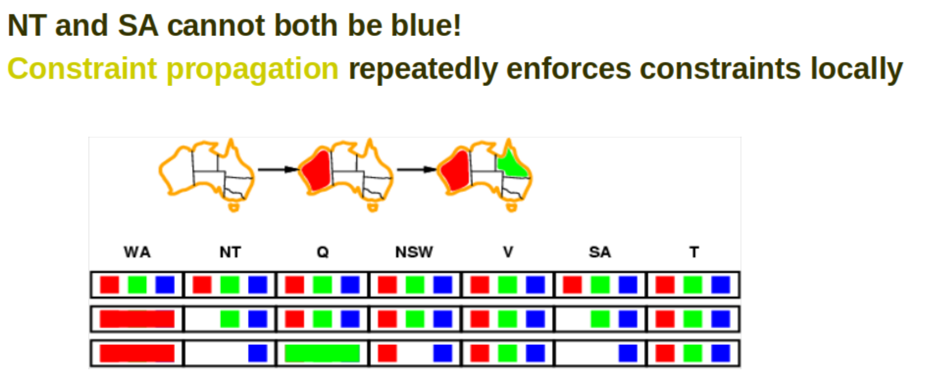
La soluzione ad un problema è un assegnamento per tutte le variabili.
Nel caso in cui fossi interessato solo ad un sottoinsieme di variabili parliamo di **Vincolo Distinto.** Le operazioni sono:

- **Combinazione** tra i vincoli.
- **Proiezione** nel caso in cui esista questo vincolo distinto.

Queste operazioni sono di tipo insiemistico nel caso dei CSP classici (Crisp), nel caso in cui invece passiamo a dei problemi di vincoli Soft, e quindi è presente una nozione di priorità tra le soluzioni (il rosso mi costa più del giallo) parliamo di SCSP.

1.4.3 Applicare Local Consistency durante il Backtrack

L'esempio fatto adesso applica la Local Consistency prima di fare Backtrack. Tuttavia è possibile applicarla anche durante la ricerca con backtrack ogni volta che si modifica il dominio di una variabile, vediamo come: (credo che serva solamente sapere che si può fare anche durante e non l'esempio vero e proprio).



1.4.3.1 L'Arc Consistency è sufficiente ?

Se ci dice bene si, come visto sopra (tolgo talmente tante cose che la soluzione me la trova da solo), ma la maggior parte delle volte permette solamente di restringere lo spazio di ricerca, quindi si può rendere un CSP arc-consistente e poi cercare una soluzione con backtracking; oppure assegnare iterativamente un valore e rendere le variabili restanti arco consistenti. Inoltre, se il dominio di una variabile diventa vuoto, sappiamo che non esisterà soluzione.

1.4.4 Vincolo distinto (Proiezione)

Alcune volte non ci interessa sapere il valore che possono ottenere tutte le variabili, ma solamente un sottoinsieme di esse. Definiamo quindi il:
Vincolo distinto (proiezione): Quali sono gli elementi del problema di cui vogliamo conoscere la soluzione.

Chapter 2

Soft Constraint Satisfaction Problems

2.1 Definizione di Soft CSP

Un Soft constraint satisfaction Problem (SCSP) è una coppia $< C, con >$ dove:

- C è un insieme di Vincoli Soft;
- con è l'insieme delle variabili sulle quali tali vincoli valgono.

2.1.1 Differenza tra vincoli

- **Un Vincolo Crisp** delimita l'insieme dei valori ammissibili per una variabile per la soluzione al problema (è quindi una funzione caratteristica che associa ad ogni variabile un assegnamento di 0 o 1 in base a se può assumere o no un certo valore).
- **Un Vincolo Soft** è una funzione che associa ad ogni assegnamento di variabile un valore parzialmente o totalmente ordinato da un insieme di pesi A.
  Formalmente è una funzione definita da C $\rightarrow < con, def >$ dove:
  - **con** $\subseteq$ V indica per ogni vincolo a quale variabile esso è associato (ad esempio x_2, x_3).
  - **def** è una funzione (è il peso associato ad ogni assegnamento di variabile):

$$\text{def: } D^k \rightarrow \text{A}$$

Che associa ad ogni elemento del dominio un valore del semiring. k rappresenta il numero di variabili.

La struttura utilizzata per descrivere problemi di Soft CSP è chiamata semiring.

2.1.2 Esempio

I vincoli che abbiamo visto in precedenza sono dei vincoli assoluti, la cui violazione esclude una possibile soluzione. Molti dei problemi CSP reali includono i vincoli preference i quali indicano quali soluzioni sono preferite. Per esempio in un problema di timetable in università ci potrebbe essere il professore X che preferisce insegnare la mattina mentre il professore Y preferisce insegnare il pomeriggio. Un timetable dove il prof X insegna alle 14 e il prof Y alle 9 potrebbe essere una soluzione, ma non è quella ottimale viste le preferenze. I vincoli sulle preferenze possono essere codificati spesso come dei costi applicati sugli assegnamenti individuali delle variabili. Riprendendo l'esempio di prima possiamo dare all'assegnamento prof X = (lezione alle 14) un costo di 2, mentre all'assegnamento prof X = (lezione alle 9) un costo di 1. In questo modo si cerca fra le possibili soluzioni quella ottimale andando a minimizzare (o massimizzare...) il costo della soluzione.
Supponiamo di avere un problema di colorazione del grafo e cerchiamo di trovare una soluzione ottimale.
Partiamo riprendendo la definizione di un CSP, esso è definito come $P = \{V, D, C, PC, con, def, a\}$ i quali indicano:

- V = insieme delle variabili;
- D = insieme dei domini associati alle variabili;
- C = insieme di vincoli, definito come l'associazione variabile-vincolo ovvero quali variabili sono coinvolte in quale vincolo;
- PC = sono i vincoli primitivi e variano in base al vincolo e al dominio. A seconda del tipo di vincolo che possiamo avere i primitivi possono essere diversi, esempio se lavoro sugli interi esso conterrà vincoli con gli operatori $<, >, =$.
  Sostanzialmente esso indica i tipi di vincoli che posso usare se lavoro su un dominio specifico. Inoltre indicano se il vincolo è implicito (tutti i colori diversi) o esplicito (valgono le coppie $< r, g, b >, < g, r, b >, \ldots$).

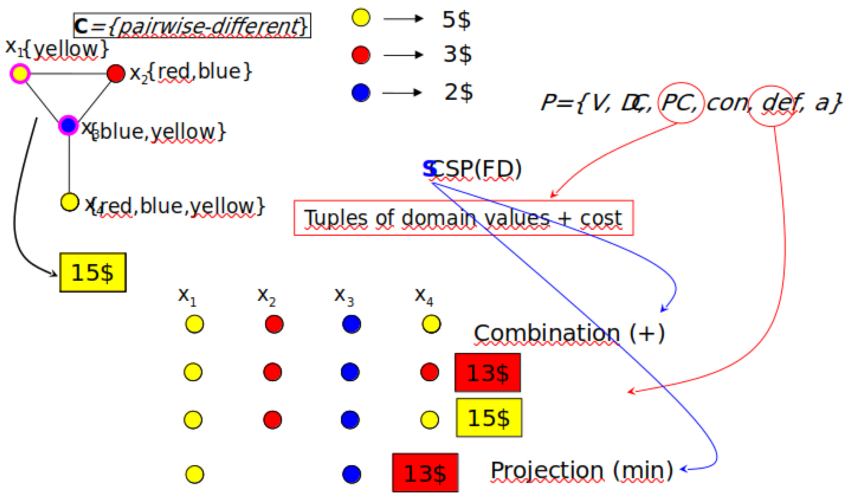


Figure 2.1: Esempio di SCSP su grafo.

- con = funzione che definisce quali variabili sono coinvolte in quale vincolo, essa dato un vincolo restituisce le variabili che sono connesse a questo;

- def = funzione che indica quali sono i valori del dominio possibili per una specifica variabile. In un SCSP questa funzione oltre a dire i valori possibili per una variabile deve indicare anche il costo associato a quei valori, nell'esempio in Figura 4.1 se passiamo in input alla funzione def la variabile x_1 essa ci ritornerà come valori possibile il colore giallo con costo 5. Questa è la differenza con la funzione def di un problema CSP.

- a = sottoinsieme di V contenente tutte le variabili interessate nella soluzione del SCSP, ad esempio nel caso del grafo abbiamo che le variabili che ci interessano per la soluzione ottimale sono solo x_1 e x_3, non tutte quante.

Nell'esempio in Figura 4.1 i vincoli binari sono hard, perché la soluzione richiede per forza che i nodi collegati abbiamo colori diversi sennò si violano i vincoli, mentre i vincoli unari (quelli del costo sul colore) sono soft. Nel caso di un CSP quando utilizziamo un algoritmo di ricerca per trovare una soluzione esso si può fermare alla prima soluzione che trova oppure, se richiesto, le cerca tutte quante. Nel caso di SCSP siamo costretti a trovare tutte le possibili soluzioni in modo da scegliere la migliore. Normalmente si utilizzano due operazioni per trovare la soluzione ottimale in un SCSP:

- **Operatore di Combinazione($\times$):** dove metto insieme tutti gli assegnamenti, combinando i vincoli, quindi si calcola anche il costo totale ad esempio con l'operazione di somma;
- **Operatore di Proiezione ($+$):** operazione di scelta della soluzione migliore data dalla combinazione in base al minimo o al massimo del costo.

2.2 Semiring

Un semiring è una quintupla:

$$< A, +, \times, 0, 1 >$$

- **A:** *l'insieme degli elementi* che mi rappresentano i costi (dominio dei costi). Esso può essere l'insieme dei reali oppure un intervallo specifico (ad esempio $[0, 1]$)
- $+$: *operatore di proiezione*, usato per fare la scelta fra le soluzioni trovate dalla combinazione, può essere il minimo o massimo. Possiamo definire alcune proprietà:
  - **idempotente:** se faccio $a + a$ dove l'operatore $+$ è il minimo il risultato è sempre a, quindi quando un operatore è idempotente è possibile definire un ordinamento ovvero

    $$a \leq b \text{ dove b è meglio di a} \iff a + b = b$$
- $\times$: *operatore di combinazione*, utilizzato per combinare i vincoli, può essere somma,moltiplicazione etc... dipende dal problema. Possiamo definire alcune proprietà:
  - **commutativa:** si considera il set di vincoli invece delle tuple.
- **0:** rappresenta il valore minimo (peggiore) dell'insieme A, ovvero il bottom sotto il quale non si può andare, per l'intervallo $[0, 1]$ è 0;
- **1:** rappresenta il valore massimo (migliore) di A, il top, per l'intervallo $[0, 1]$ è 1.

Tutti questi che abbiamo visto sono simboli quindi quando si utilizza l'operatore $+$ non ci si riferisce alla somma ma ad un operatore per la proiezione che potrebbe essere min, max, OR, ...

2.2.1 Tipi di Semiring

- **Probabilistic:** $<[0,1], \max, \times, 0, 1>$ si massimizza la probabilità,
- **Weighted:** $<R^+, \min, +, +\infty, 0>$ si minimizza la somma dei costi,
- **Fuzzy:** $<[0,1], \max, \min, 0, 1>$,
- **Classical:** $<\{false, true\},\ \vee(\text{OR}),\ \wedge(\text{AND}),\ false,\ true>$.

2.2.2 Esempi pratici SCSP: Domanda esame

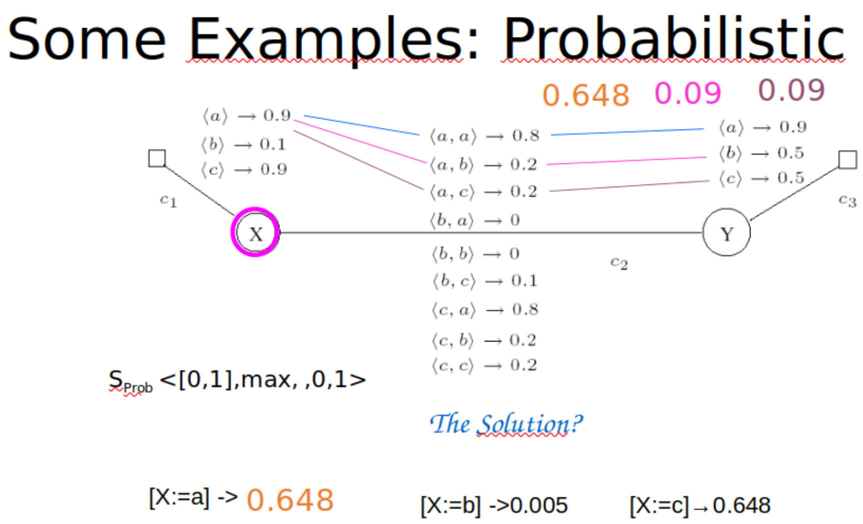
Il vincolo in mezzo è binario, se a è compreso tra x ed y.

$$x \leq a \leq y$$

A può assumere sia il valore di

1. $x \rightarrow x = a$
2. $y \rightarrow y = a$

2.2.2.1 Esempio con probabilistic



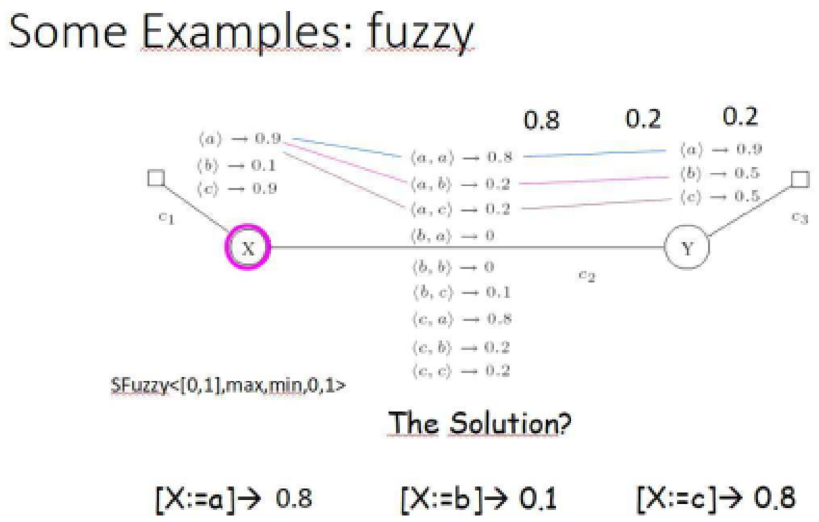
Descrizione del semiring: I valori possibili sono tra 0 ed 1, l'operazione di proiezione è il Massimo; quella di combinazione è la **moltiplicazione**; il minimo valore è 0; il massimo valore è 1.
Ci domandiamo: Quanto vale l'assegnamento X = a?

1. Dato che l'operazione $\times$ di combinazione è la moltiplicazione faccio:
   (a) $0.9 * 0.8 * 0.9 = 0.648$
   (b) $0.9 * 0.2 * 0.5 = 0.09$
   (c) $0.9 * 0.2 * 0.5 = 0.09$

   Per poter inserire 0.648 in $x = a$ dovrei dividere per 0.8 i 3 valori su c_2 e 0.9 i tre valori su c_3 (secondo me).

2. Dato che l'operazione $+$ di proiezione in questo caso è max prendo il massimo di tutti gli elementi trovati e quella sarà la soluzione di quando ad X assegno a. Facendo lo stesso ragionamento si calcolano tutte le restanti soluzioni.

2.2.2.2 Esempio con Fuzzy



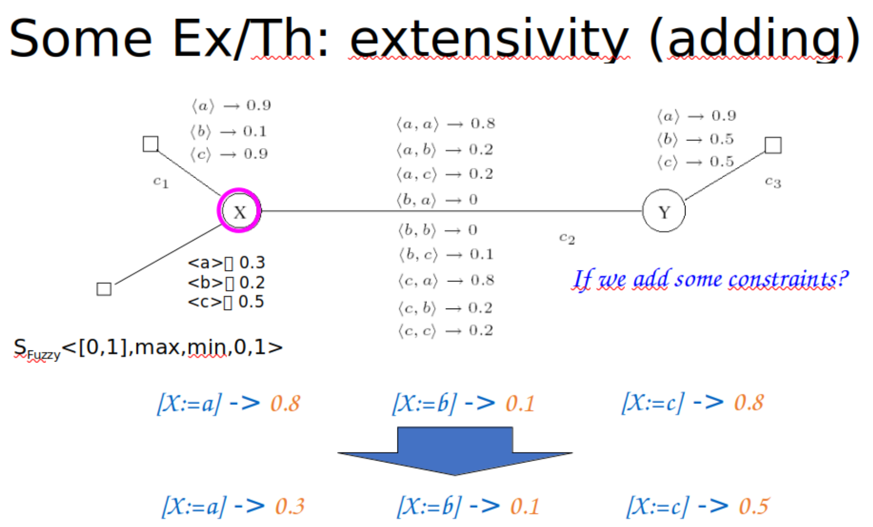
For instance, for the tuple $\langle a, a\rangle$ (that is, $X = Y = a$), we have to compute the minimum between 0.9 (which is the value assigned to $X = a$ in constraint c_1), 0.8 (which is the value assigned to $\langle X = a, Y = a\rangle$ in c_2) and 0.9 (which is the value for $Y = a$ in c_3). Hence, the resulting value for this tuple is 0.8. We can do the same work for tuple $\langle a, b\rangle \to 0.2$, $\langle a, c\rangle \to 0.2$, $\langle b, a\rangle \to 0$, $\langle b, b\rangle \to 0$, $\langle b, c\rangle \to 0.1$, $\langle c, a\rangle \to 0.8$, $\langle c, b\rangle \to 0.2$ and $\langle c, c\rangle \to 0.2$. The obtained tuples are then projected over variable X, obtaining the solution $\langle a\rangle \to 0.8$, $\langle b\rangle \to 0.1$ and $\langle c\rangle \to 0.8$. $\triangle$

Teorema dell'Estensività:

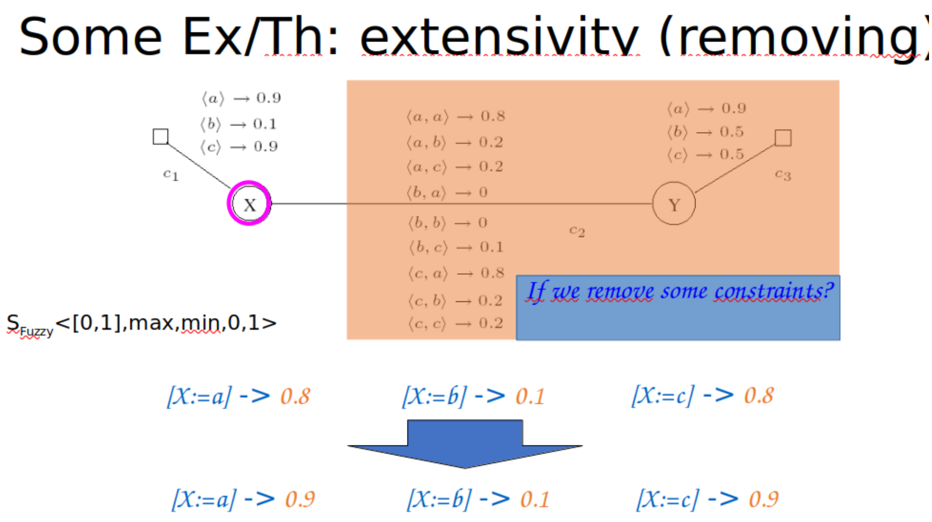
L'aggiunta di vincoli è monotona, peggiora la soluzione

Questo significa che più aggiungo vincoli e più la soluzione peggiora, più tolgo vincoli più la soluzione mi migliora. Questo perché più vincoliamo il problema e meno possibilità abbiamo, meno lo vincoliamo e più sono le possibilità.

2.2.2.3 Esempio di aggiunta vincolo (quindi peggioro)



2.2.2.4 Esempio di rimozione vincolo (quindi miglioro)



2.3 Local Consistency nei Soft CSP

Fare Local consistency nel caso di CSP Soft non significa eliminare dei valori dal dominio, ma significa ridurre (ottimizzare) il valore del semiring associato a quella variabile in maniera tale da fare in modo che questo valore si avvicini il più possibile a quello della soluzione finale.

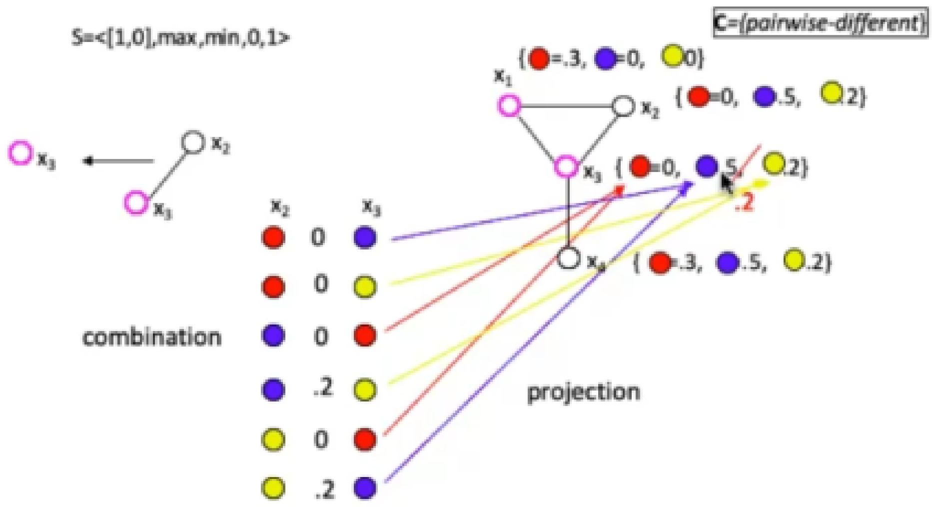
*Posso applicare Arc Consistency in un SCSP **solo** se l'operatore × è **idempotente**.*

Per esempio, non potremmo applicare Arc Consistency se come operatore × utilizzassimo la somma, ma potremmo farlo se usassimo min.

$$1 \times 1 = 2 \text{ se } \times = somma$$

$$1 \times 1 = 1 \text{ se } \times = minimo$$

2.3.1 Esempio con CSP Soft:



In questo esempio sono presenti solo vincoli unari, ciò mi dice che quando ad x_1 assegno rosso ha valore 0.3, quando blu o giallo ha valore 0; cosi per tutte le variabili. Siccome il valore 0 è il bottom del semiring, dare 0 significa che non sono assegnamenti permessi, quindi li possiamo direttamente eliminare.
Ottimizzazione della soluzione su CSP Soft: Voglio ridurre (ottimizzare) il dominio della variabile x_3 , vediamo come fare con tutti i passaggi:

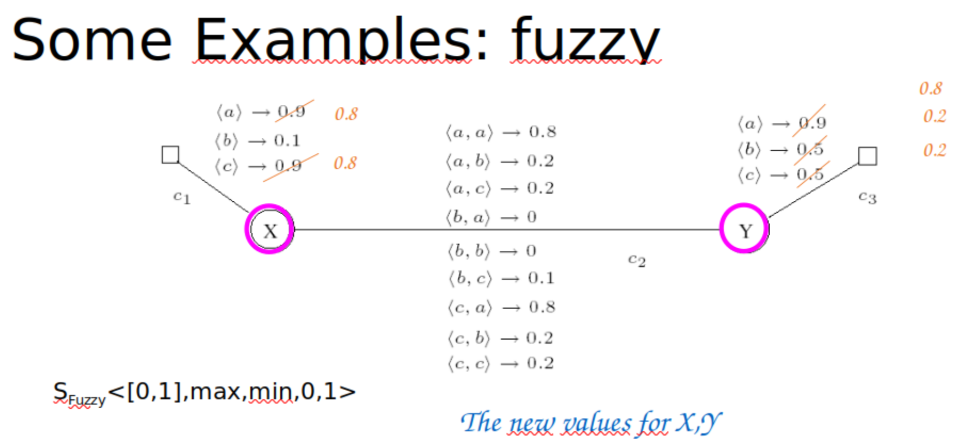
1. **Combinazione:** Prendo tutte le possibili tuple, gli assegno un valore e poi in base all'operatore di combinazione (che in questo caso è min) faccio la proiezione.
   Quindi parto cosi: Il minimo se x_2 è rosso e x_3 è blu è 0, perché è il minimo tra 0 e 0.5.
   Il minimo se x_2 è rosso e x_3 è giallo è 0, perché è il minimo tra 0 e 0.2.
   Continuo cosi per tutte le tuple.
   Non ho considerato le coppie di colore uguale perché c'era scritto $pairwise - different$.

2. **Proiezione:** Tra tutti i risultati che genera la combinazione faccio la proiezione sulla variabile in questione, in questo caso x_3. Adesso: che valore ha x_3 quando ha valore rosso? Devo fare il massimo tra 0 e 0 che è 0 e tutto rimane invariato perché x_3 sul rosso ha comunque 0. Per x_3 = blu il massimo è 0.2. Per x_3 = giallo il massimo è comunque

0.2. L'unica modifica da apportare quindi è al blu di x_3 che al posto di 0.5 ha valore 0.2.

Nel caso **Crisp CSP** i valori vengono eliminati completamente dal dominio. Nel caso **Soft CSP** ottimizzo il valore del semiring associato a quell'assegnamento. Una volta ottimizzati tutti i valori si utilizzano tecniche di **Branch And Bound**, che può essere applicato o all'inizio o durante la ricerca, per trovare la soluzione.

2.3.2 Esempio con Fuzzy



Consideriamo il vincolo unario c_1 sulla variabile X ed il vincolo binario c_2 sulle variabili X ed Y. Faccio c_1 (x) c_2 proiettato $x(a)$:

1. Combinazione (Min): c_1 combinato c_2 (c_3 non lo prendo in considerazione perché faccio (x):

    - Quando vale $< a, a >$? Faccio 0.9 min 0.8 = 0.8
    - Quando vale $< a, b >$? Faccio 0.9 min 0.2 = 0.2
    - Quando vale $< a, c >$? Faccio 0.9 min 0.2 = 0.2

    Facendo questo ho combinato i vincoli c_1 e c_2 . Per sapere poi quanto vale c_1 (x) c_2 devo andare a proiettare su X.

2. Proiezione (Max): Prendo il massimo tra 0.8, 0.2, 0.2. In conclusione, al posto di 0.9 su (a) posso scriverci 0.8.

Altra domanda sul fuzzy (assegnamento): Quanto vale l'assegnamento $x = a$, $y = b$?
devo fare il minimo tra 0.9, 0.2 e 0.5 che sarebbe 0.2.
(Non applico la proiezione quindi non uso il massimo).
Altra domanda: Applichiamo local consistency per X = a.
Devo fare min(0.9, 0.8, 0.9) = 0.8.
Devo fare min(0.9, 0.2, 0.5) = 0.2.
Devo fare min(0.9, 0.2, 0.5) = 0.2. Se vado a proiettare fa 0.8 perché prendo il massimo. Se io vado a calcolare la soluzione, adesso vado a calcolare la soluzione di "quanto vale $x = a$ $y = b$"?
min tra 0.8, 0.2 e 0.2 = 0.2 che è come prima, la soluzione non è cambiata.
Ho migliorato il bound ma non ho modificato la soluzione, è una cosa buona.
Nell' esempio successivo (prossima sezione, sarebbe quello probabilistico) la soluzione **viene modificata** a 0.72 e non più a 0.9 quindi non posso accettarla, è proprio sbagliata.

2.3.3 Idempotenza degli operatori

Un operatore si dice idempotente se facendo:

$$a \ operazione \ a \text{ mi restituisce sempre } a.$$

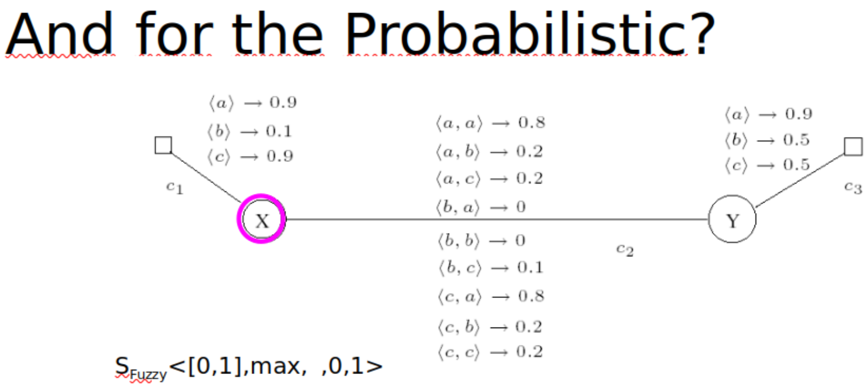
L'idempotenza si può analizzare sia per gli operatori che si usano per la Proiezione sia per quelli di Combinazione.

- Idempotenza su **Combinazione**: Dipende strettamente dal semiring utilizzato:
  - **Fuzzy:** Idempotente, perché il *minimo* tra a ed a è sempre a (il minimo tra due valori uguali da sempre lo stesso valore).
  - **Probabilistic:** Non idempotente, perché $a*a$ non fa esattamente a (ad esempio $0.1*0.1 = 0.01$, la moltiplicazione infatti non ha la proprietà di idempotenza).
  - **Weighted:** Non idempotente, perché $a+a$ non fa esattamente a.
  - **Classic:** Idempotente, infatti a AND $a = a$

- Idempotenza su **Proiezione:** Tutti gli operatori sono idempotenti e questo segue dalla definizione stessa di c-semiring (definisco l'ordinamento quando effettuo la proiezione).

La distinzione tra operatore idempotente e non idempotente è importante perché gli algoritmi di consistenza locale solo nel caso in cui la combinazione fosse idempotente **mantengono le soluzioni del problema**. Qualora facessi arc-consistency su semiring che non ammettono operazioni idempotenti la soluzione che si genera è proprio **sbagliata**, e non può essere accettata.

Vediamo un esempio in cui applicare arc-consistency va a modificare la soluzione finale del problema (e quindi non è accettabile):



Domanda: Quanto vale l'assegnamento $x = a$, $y = b$ su questo problema? devo moltiplicare 0.9, che sarebbe $x = a$, per $< x = a, y = b >$ che sarebbe 0.2 per $y = b$ che sarebbe 0.5. Il risultato è 0.09.
Domanda: Quanto vale c_1 c_2 proiettato x (su a)? devo fare la combinazione:

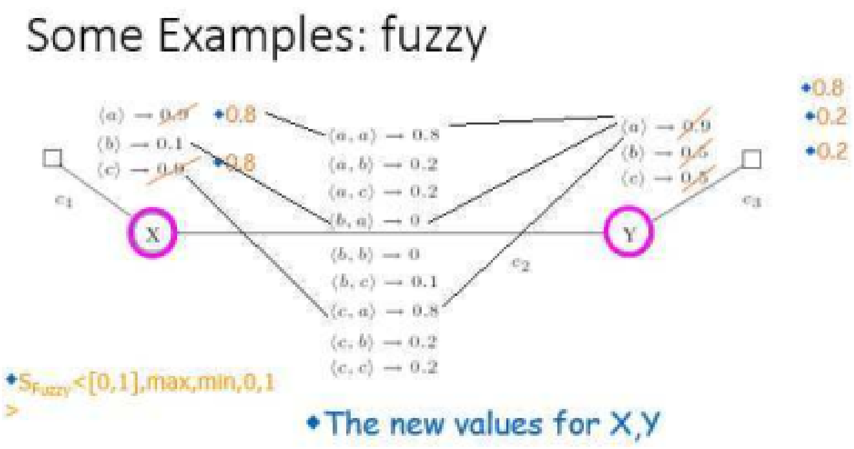
- $0.9 * 0.8 = 0.72$
- $0.9 * 0.2 = 0.18$
- $0.9 * 0.2 = 0.18$

Poi prendo il Massimo (proiezione) che sarebbe 0.72 e questo valore mi va al posto di 0.9. Il problema però è che ho cambiato la soluzione finale, perché la proiezione mi ritorna proprio il 0.72 che è un valore errato data la non idempotenza della moltiplicazione (è proprio una soluzione sbagliata).

L'operazione di local consistency nei CSP Crisp quindi classici è importante perché mantiene la soluzione del problema riducendo gli elementi del dominio e non modificando le soluzioni.

Nel caso Soft invece, la soluzione non viene modificata solamente se l'operazione di combinazione è idempotente. Nel caso in cui non lo fosse si andrebbero a modificare le soluzioni stesse del problema, e quest'ultime non sarebbero accettabili perché sbagliate.

2.3.4 Altro esempio con vincolo c_3



I nuovi valori per il vincolo c_3 sono ottenuti analizzando prima $y = a$ e quindi poi si analizza $< a, a >$ e poi $x = a$. Poi $y = b$ e quindi poi $< b, b >$ e poi $x = b$. Utilizzando la combinazione e poi la proiezione i valori tornano quelli.

2.3.5 Semiring con operazioni non idempotenti

Vogliamo ottimizzare i CSP Soft anche se la loro operazione di combinazione non è idempotente.
L'operatore che utilizzeremo è l'opposto dell'operatore di combinazione ed è chiamato "*Diviso*": $\div$. Analizziamo la riga del Fuzzy:

- Prima versione: L'operazione inversa del minimo, indicata con l'operatore $\div$ è max. Supponiamo che l'elemento x sia tale che $b * x = a$, la domanda è: Quanto fa $a \div b$? Fa quell'elemento x tale che il minimo tra b ed x è minore uguale di a. Devo quindi in qualche modo andarmi a trovare gli inversi degli operatori visti prima.

Division in the soft CSPs istances

- Classical CSPs

$$a \div b = max\{x \mid b \wedge x \leq a\} = (b \implies a)$$

- Fuzzy CSPs

$$a \div b = max\{x \mid min\{b, x\} \leq a\} = \begin{cases} 1 & \text{if } b \leq a \\ a & \text{if } a < b \end{cases}$$

- Weighted CSPs

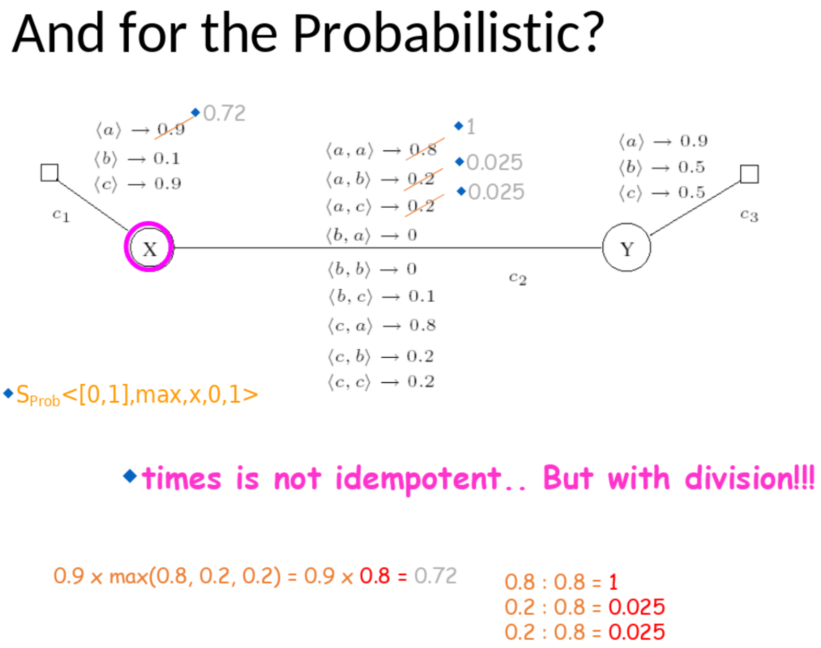
$$a \div b = min\{x \mid b \hat{+} x \geq a\} = \begin{cases} 0 & \text{if } b \geq a \\ a \hat{-} b & \text{if } a > b \end{cases}$$

- Probabilistic CSPs
  - … guess (inverse of x … division! ⏪

- Seconda versione: Quando fa $a \div b$? (si legge "*a Diviso b*"). Fa il numero più grande x tale che il minimo tra b ed x è più piccolo di a

Analizziamo la riga del **Weighted:**
Quanto fa $a \div b$? Fa il più piccolo (min) elemento x tale per cui $b + x$ fa a. Questo però lo possiamo fare solo se a è maggiore di b (altrimenti si prenderebbero in considerazioni valori non consoni tipo negativi) e se b è maggiore uguale di a si da come risultato 0.

Esempio probabilistico:



Come faccio ad ottimizzare il CSP in maniera tale da mantenere le soluzioni? Inizio in maniera identica al precedente metodo:

- Combinazione
  1. $0.9 * 0.8 = 0.72$
  2. $0.9 * 0.2 = 0.18$
  3. $0.9 * 0.2 = 0.18$
- Proiezione: $\text{Max} = 0.72$

Fatto questo devo **dividere per il massimo valore tra tutti quelli utilizzati:**

$$max(0.8, 0.2, 0.2) = 0.8$$

Ora divido per 0.8 tutti i valori nel vincolo c_2 cosi da ottenere i valori in figura. Dato che ho ottimizzato il valore di (a) e cambiato i valori interessati, la soluzione rimane invariata, quindi anche su semiring non idempotenti si può fare arc-consistency grazie all'operatore di *"Divisione"* ($\div$).

Importante: in questo caso abbiamo utilizzato la *divisione* perché era l'operazione inversa della *moltiplicazione* (che è l'operatore di combinazione utilizzato nei CSP probabilistici). Se però l'operazione di combinazione fosse stata la *somma* (e quindi CSP Weighted) l'operazione $\div$ sarebbe stata la *sottrazione*.

Chapter 3

Argumentation Framework

3.1 Definizione di Argumentation Framework

Come i CSP l'argumentation è un altro metodo che offre l'intelligenza artificiale per rappresentare la conoscenza e risolvere i problemi. Quello che andremo a rappresentare sono delle situazioni/dialoghi che vogliamo studiare dal punto di vista logico.
Un argumentation framework (AF) è una coppia (A,R) dove

- A è un set di argomentazioni
- R $\subseteq$ A $\times$ A è una relazione rappresentante gli "attacchi" ("sconfitte")

$$F(\{a, b, c, d, e\}, \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\})$$

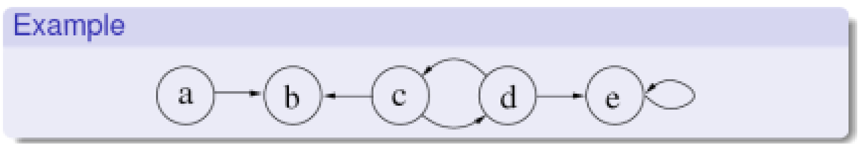


Figure 3.1: Argumentation framework

Si ha un attacco quando si ha un'espressione logica (frase, dato...) che è in contraddizione con un'altra. Gli attacchi possono essere anche pesati, essi possono dipendere anche da chi ha detto quella frase.

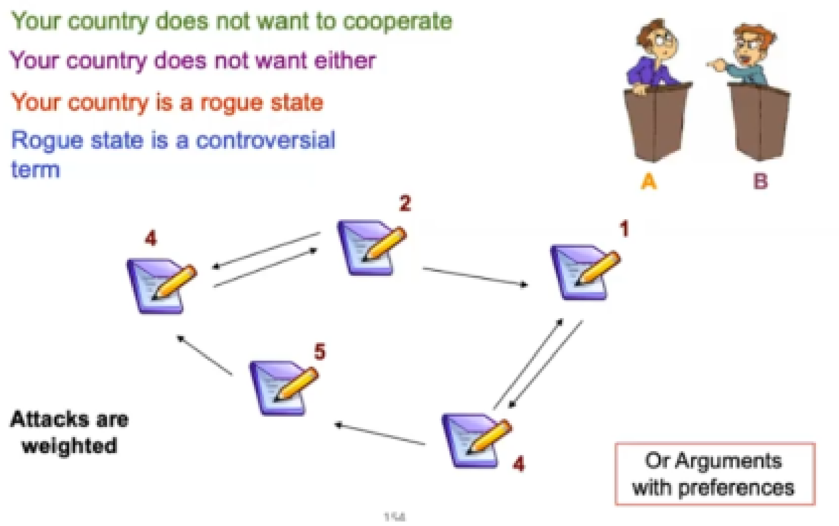


Figure 3.2: Esempio di argumentation framework

Ci possono essere anche casi in cui è noto chi dice l'argomento e altri in cui non lo è. Per quest'ultima vogliamo selezionare gli argomenti che sono più validi rispetto agli altri, ad esempio in Figura 6.2 il 4 e il 2 sembrano buoni argomenti perché attaccano gli altri e contrattaccano nel caso siano attaccati. Lo scopo è di definire dei criteri per trovare gli argomenti più forti, validi (che stanno "in piedi da soli"), in modo da selezione i conflitti che riescono a sopportare gli attacchi dall'esterno.

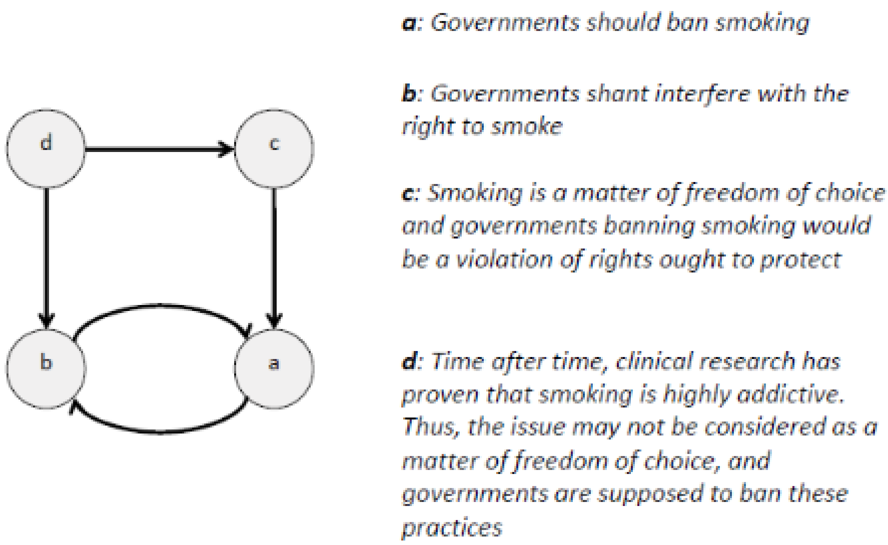


Figure 3.3: Altro esempio di argumentation framework

Dobbiamo trovare gli argomenti che ”stanno bene insieme”, la prima nozione di questo tipo è un insieme di argomenti senza conflitti.

3.2 Tipologie di semantiche

Esistono principalmente tre tipologie di semantiche basate sulla metodologia con la quale vengono calcolate.

1. **Estensione:** Andiamo a guardare dei sottoinsieme di argomenti (tutte quelle viste precedentemente)
2. **Labeling:** Il labeling è una funzione che assegna delle etichette (colori) ad ogni argomento in modo tale da distinguere gli argomenti accettati dai restanti.
3. **Ranking:** Restituisce un ordimento sugli argomenti. Dice quindi ”l'argomento a è migliore di b”. Questo sistema è più raffinato.

3.3 Extension-Based Semantics

Il task principale che viene svolto sugli argumentation framework è il computo della semantica, cioè si selezionano dei criteri con i quali si vanno a scegliere dei sottoinsiemi di argomenti che condividono una qualche proprietà particolare. Di seguito una breve descrizione:

- **Conflict Free**: Argomenti che non si attaccano.
- **Admissible**: Richiede la nozione di difesa, cioè un contrattacco che un argomento fa ad un altro argomento che è attaccato.
- **Complete**: Deve essere admissible, se un argomento è difeso questo è per forza dentro l'estensione. Nell'esempio sopra a era l'unico argomento che era sempre difeso (non veniva attaccato da nessuno) quindi l'insieme complete sono tutti gli argomenti che hanno dentro a.
- **Preferred**: Si ottiene per inclusione insiemistica, cerco le admissible più grandi. tra {a},{c},{d},{a, c},{a, d} dato che voglio la più grande, la singola a non potrà essere preferred, perché si trova dentro {a, c},{a, d} che sono più grandi.
- **Grounded**: Set più piccolo tra tutte le complete.

- **Stable**: Seleziono un insieme di argomenti che è conflict free e che attacca tutti gli altri

La prima che vediamo è quella degli insiemi Conflict Free, cioè quegli insiemi che tra loro non hanno conflitti.

3.3.1 Estensioni Conflict-Free

Dato un Augmentation Framework F = (A, R), l'insieme S ⊆ A è conflict-free se, per ogni (a, b) ∈ S, si ha che (a, b) ∉ R. (Per ogni coppia di elementi in S non è presente una relazione d'attacco tra questi elementi).

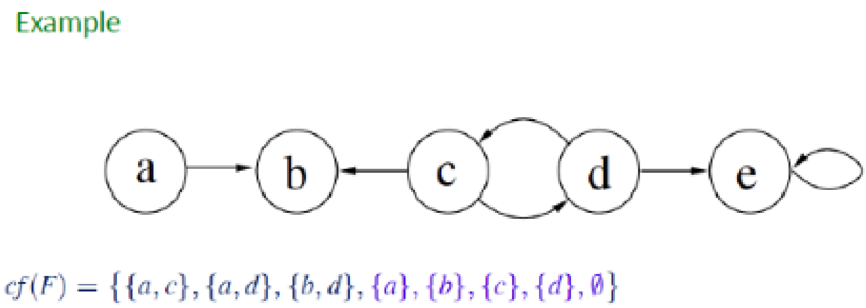


Figure 3.4: Esempio insieme conflict-free.

In questo caso andiamo a scegliere come coppie gli argomenti che non sono in conflitto (quindi che non si attaccano) fra di loro, $(\{a,c\},\{a,d\}$ ma non $\{a,b\})$, e anche i singoli argomenti tranne e poiché esso si contraddice da solo visto che ha un cappio.

Importante: per il calcolo: inizio con inserire **l'insieme vuoto**, poi i **singoli** argomenti che non si auto-attaccano, poi **le coppie, terne, quadruple...**

3.3.2 Estensioni Admissible

Dato un Augmentation Framework F = (A, R), l'insieme S ⊆ A è ammissibile se:

- S è **conflict free;**

- Ogni a ∈ S è **difeso** da S (cioè ogni elemento che appartiene all'insieme è difeso dagli elementi dell'insieme stesso). Un elemento a ∈ A è difeso da S se, per ogni b ∈ A con (b, a) ∈ R, esiste un c ∈ S tale per cui (c, b) ∈ R (a è difeso da S se per ogni b che attacca quell'elemento a esiste un altro elemento sempre dentro S che contrattacca l'attacco di b verso a).

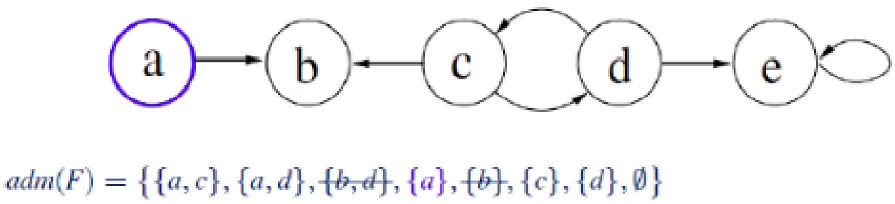


Figure 3.5: Esempio insieme Ammissibile.

Non è necessario che sia lo stesso argomento a difendersi da altri attacchi, ad esempio in questo caso, supponendo che non ci sia a, b è attaccato da c ma se prendo d lui oltre che difendere se stesso da c (poiché attaccato) difende anche b. Guardando la definizione di difesa, un argomento a è difeso da un argomento b, (b, a) ∈ R, nel momento in cui esiste un argomento c tale che c attacca b, (c, b) ∈ R. Il sottoinsieme {b,d} non viene scelto poiché d è attaccato da c ma a si difende a sua volta contrattaccando, mentre b è attaccato sia da a che c, nel primo caso nessuno lo difende nel secondo d difende b perché attacca c.

- l'insieme ∅ (vuoto) è **ammissibile**? Si, nessuno lo attacca.
- l'insieme ∅ (vuoto) è **Conflict Free**? Si.

3.3.3 Estensioni Complete (Tutti Difesi)

Dato un Augmentation Framework F = (A, R), l'insieme S ⊆ A è completo se:

- S è ammissibile;
- Ogni a ∈ A admissible difeso da S è contenuto in S. Un elemento a ∈ A è difeso da S se, per ogni b ∈ A con (b, a) ∈ R, esiste un c ∈ S tale per cui (c, b) ∈ R.

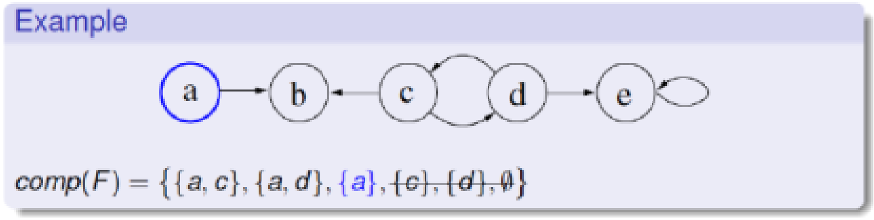


Figure 3.6: Esempio insieme Completo.

Quindi un insieme complete contiene tutti gli insiemi ammissibili e anche tutti gli argomenti che sono difesi.
L'insieme $\emptyset$ (vuoto) è **Completo**? Va messo soltanto quando tutti gli argomenti sono attaccati e quindi quando **non** c'è un qualche argomento che è sempre difeso.
Infatti sopra non va messo perché a è sempre difeso. a è sempre difeso da tutti perché non è attaccato da nessuno, quindi nel calcolo dei difensori va sempre messo dentro.

1. {a, c} è **completo?** Si!

    - a chi difende? attacca b quindi difenderebbe tutti quelli attaccati da b, ma b non attacca nessuno quindi a difende solo se stesso.
    - c chi difende? tutti quelli attaccati da b quindi nessuno e tutti quelli attaccati da d quindi c.

    L'insieme dei difensori è a, c, che sta dentro S={a, c}, quindi è completo.

2. {a, d} è **completo?**

    - a chi difende? attacca b quindi difenderebbe tutti quelli attaccati da b, ma b non attacca nessuno quindi a difende solo se stesso.
    - d chi difende? se stesso, perché viene attaccato da c ma si difende da solo con un contrattacco.

    L'insieme dei difensori è {a, d} che sta dentro S quindi anche questo è completo.
    {a} è **completo?** Può stare da solo perché non viene attaccato da nessuno, quindi è **come se venisse difeso da tutti quanti**, e lui non difende nessuno, poiché b non attacca nessuno.
    Quindi se un argomento non attaccato da nessuno ne attacca un altro

che a sua volta non attacca nessuno allora quell'elemento è complete e può stare da solo.
{c} è **completo?** Difensori: {a, c} che non è incluso in c quindi No.
{d} è **completo?** Difensori: {a, d} quindi No. L'idea degli insiemi complete è che se un insieme difende qualcosa, quel qualcosa "deve essere messo dentro" e **deve rimanere ammissibile.**

La differenza quindi è che nell'insieme ammissibile vuol dire che mi difendo, complete vuol dire che dentro ci sono tutti quelli difesi.

3.3.4 Estensioni Grounded (Minimale)

F = (A, R), l'insieme S $\subseteq$ A è grounded se:

- S è **completo**;
- Per ogni sotto insieme T $\subseteq$ A completo in F si ha che T $\subsetneq$ S

Quindi un insieme completo è grounded se è il **più piccolo dei complete**, ovvero se **non** esiste un sottoinsieme T complete che è più piccolo di lui. Si calcola tramite **l'intersezione** delle complete. **N.B** L'insieme grounded è sempre **UNICO**, cioè composto da un solo elemento.

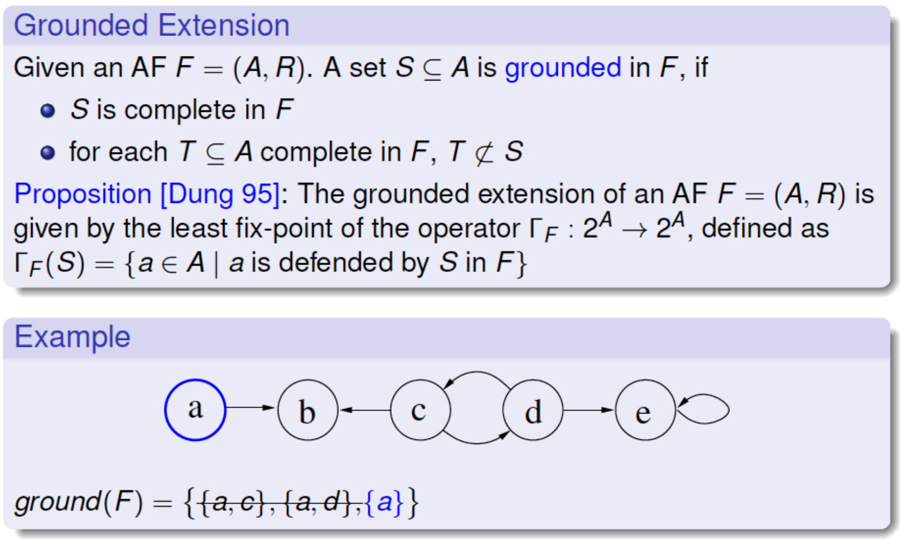


Figure 3.7: Esempio Grounded

a, c è **grounded**? i suoi sottoinsiemi singoli sono a e c, questi sono completi?

no, perché a lo è ma non c quindi non è grounded.
a, d è **grounded**? i suoi sottoinsiemi singoli sono a e d, questi sono completi? no, perché a lo è ma non d quindi non è grounded.
a è **grounded**? Si, i suoi sottoinsiemi singoli sono a ed è completo.

N.B L'insieme grounded è dato anche dall'intersezione di tutti gli insiemi completi, infatti sopra se intersechiamo quei 3 insiemi completi l'unica cosa che viene fuori era a che infatti è l'unico insieme grounded.

3.3.5 Estensioni Preferred (Massimale)

Dato un Augmentation Framework F = (A, R), l'insieme S $\subseteq$ A è preferred se:

- S è ammissibile;
- Per ogni sotto insieme T $\subseteq$ A ammissibile in F , si ha che S $\subsetneq$ T (cioè se nessuno degli ammissibili S è più grande di T).

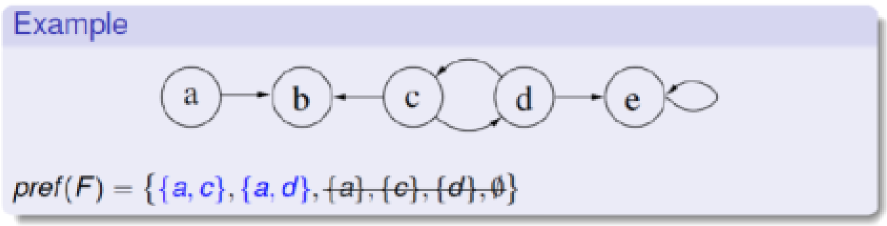


Figure 3.8: Esempio Preferred

Si ottiene per inclusione insiemistica, **Le più grandi delle admissible**. (a) da solo è contenuto in (a,c) che è più grande quindi sicuramente non sarà preferred, stessa cosa per (d). (c) stessa cosa per (a,c).
Al contrario di grounded dove si andava a scegliere l'insieme con l'elemento in comune con gli altri insieme, in questo caso si va a scegliere tra gli insiemi ammissibili quelli che sono più grandi. Da notare che si sceglie tra gli insiemi ammissibili ma si può dimostrare che si può scegliere da quelli complete.

3.3.6 Estensioni Stable

Dato un AF, F=(A,R). Un insieme S $\subseteq$ A è stabile in F, se

- S è **conflict-free** in F

- per ogni a ∈ A / S, esiste una b ∈ S, tale che (b, a) ∈ R.

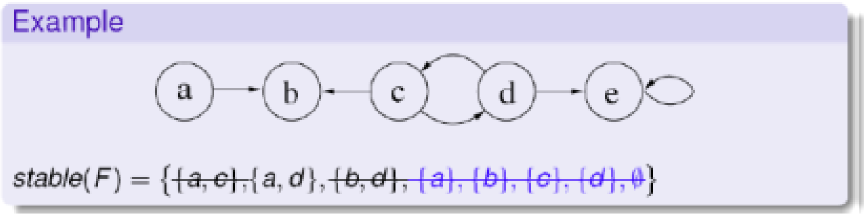


Figure 3.9: Esempio Stable

Per ogni elemento fuori da S esiste un elemento di S che lo attacca. Questo significa che gli insiemi stable sono gli insiemi conflict-free che attaccano tutti gli altri, ovvero che tutti gli elementi che stanno fuori dall'insieme esaminato sono attaccati.
In questo caso si sceglie l'insieme i quali elementi attaccano tutti gli elementi fuori dall'insieme conflict-free, ad esempio {a,c} non si prende perché a attacca b e c attacca d e b ma nessuno dei due attacca e. Mentre nell'insieme {a,d} a attacca b e d attacca sia c che e. Esiste anche una semantica semi-stabile la quale nel caso cui esista un insieme stabile allora essa coincide con quest'ultimo ma quando non c'è la stabile allora sceglie tra gli insieme preferred quelli che ne attaccano di più tra gli insiemi fuori a quest'ultimo. L'obiettivo della semantica stabile è di avere cardinalità più grande possibile e di attaccare tutti gli insiemi fuori.

a, c **non è stabile** perché è si ammissibile (b che sta fuori è attaccato e ok), d sta fuori ed è attaccato, ma e non lo attacca ne a ne c quindi questo insieme non può essere stabile.
a, d **è stabile** perché a attacca b e d attacca c, e quindi tutti gli elementi fuori dall'insieme sono attaccati.
La semantica stabile è la più forte di tutte le Estensioni, perché sono le posizioni più forti in un dialogo, sono la scelta migliore quando utilizzo AF come decision making. Il problema delle Estensioni stabili è che non sempre esistono.

3.4 Labeling-Based Semantics

Il Labeling è una funzione che assegna delle etichette rappresentate come colori ad ogni argomento. Le etichette sono di tre tipi:

1. **Verdi:** Corrispondono ad argomenti IN, cioè sono "dentro" l'insieme di argomenti accettati.

2. **Rosse:** Corrispondono ad argomenti OUT, cioè sono "fuori" dall'insieme di argomenti accettati.

3. **Gialle:** "Undecided" e sono fuori dagli insiemi direttamente accettati(verdi) ma non sono direttamente Rejected (rossi), ovvero non ho una motivazione esplicita per non accettare quel nodo, quindi sono una via di mezzo.

Qualsiasi assegnamento di colori è un labeling (posso colorare tutti i nodi di verde e apposto), però chiaramente non rispetterà le semantiche che adesso introdurremo. Le semantiche basate su labeling corrispondono esattamente alle semantiche basate su Estensioni, ed infatti possiamo calcolare le stesse cose.

3.4.1 Labeling Conflict-Free

Per ogni argomento $a \in A$ si ha che:

- a è **IN** se non ha altri attaccanti IN.
- a è **OUT** se è attaccato da almeno un argomento IN.

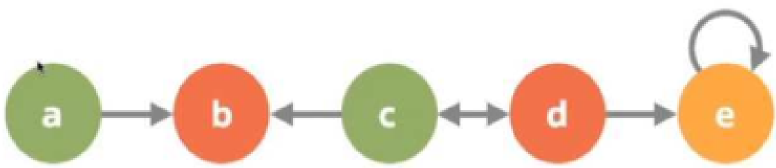


Figure 3.10: Esempio di Conflict-Free Labeling.

- a è verde perché non ha nessun altro argomento verde che lo attacca.
- c è verde per lo stesso motivo
- b e d sono rossi perché c'è almeno un argomento verde che li attacca.
- e è attaccato da un argomento rosso, ma le regole non specificano il colore in questo caso, quindi l'argomento resta fuori sia dalla colorazione IN che OUT. Per esprimere questa incertezza utilizzo la label gialla.

Domanda: Questo labeling sotto è conflict free? (cioè soddisfa le regole scritte sopra?)

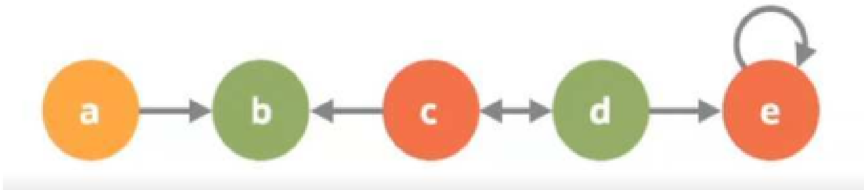


Figure 3.11: Conflict free label

Si, nessuna regola viene violata.

Domanda: Questo labeling sotto è conflict free?

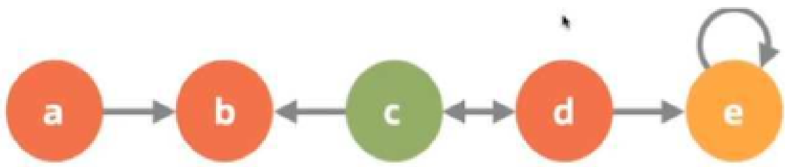


Figure 3.12: No conflict free label

No! perché a è rosso ma non ha un argomento verde che lo attacca.

3.4.2 Labeling Admissible

Per ogni argomento $a \in A$ si ha che:

- a è **IN** se tutti i suoi attaccanti sono OUT.
- a è **OUT** se è attaccato da almeno un argomento IN.

In questo caso c'è la nozione di Difesa, ovvero che un argomento è accettato (è IN) solamente se **tutti** i suoi attaccanti sono sconfitti (OUT)

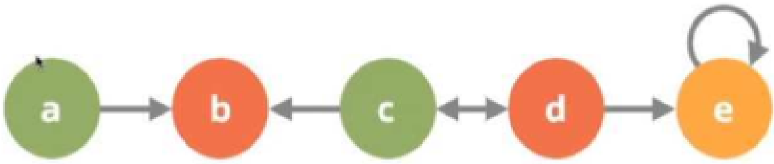


Figure 3.13: Esempio Labeling Admissible

Questo labeling è admissible, ad esempio c è accettato perché d che lo attacca è sconfitto, cioè OUT.

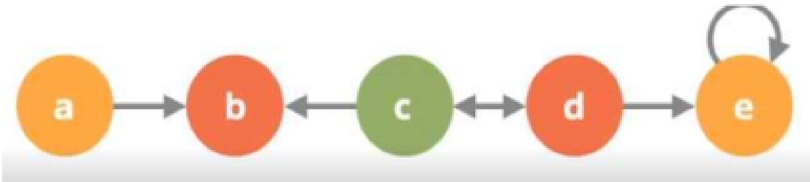


Figure 3.14: Esempio2 Labeling Admissible

Questo esempio è comunque admissible, perché non ho obblighi sull'argomento a, posso etichettarlo di Verde e la regola sarebbe soddisfatta, ma non avendo attaccanti posso etichettarlo sia giallo che verde e non violo nessuna regola.

Differenza: la differenza tra Admissible e Complete è che A in complete deve essere per forza verde mentre qui può essere anche giallo.

3.4.3 Labeling Complete

Per ogni argomento $a \in A$ si ha che:

- a è **IN** *se e solo se* tutti i suoi attaccanti sono etichettati come OUT o quell'argomento non ha attaccanti.
- a è **OUT** *se e solo se* è attaccato da almeno un argomento IN.

Domanda: Questo labeling è complete?

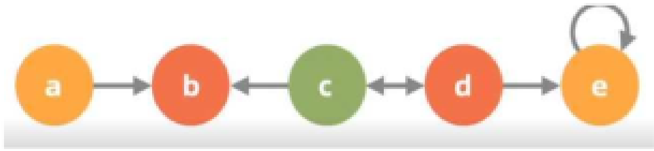


Figure 3.15: Esempio Labeling NON COMPLETE

No, perché l'argomento a è etichettato di giallo, ma la prima regola mi dice che un argomento è etichettato di verde quando tutti i suoi attaccanti sono etichettati di rosso, ma ho un se e solo se, devo leggerlo anche dalla parte opposta, ovvero:

É IN se ha tutti attaccanti etichettati OUT, ma dato che a non ha attaccanti deve per forza essere verde, cosi come d.

La differenza con l'admissible è proprio quel "se e solo se". L'admissible ci permette di ignorare degli elementi che potrebbero essere accettati (infatti posso sia etichettarlo verde o giallo a prima), ma questo non accade nella complete, cioè se tutti gli argomenti che mi attaccano sono out (e questo accade anche quando nessun argomento mi attacca) allora devo essere per forza verde.
Il labeling complete dell'esempio sarebbe:

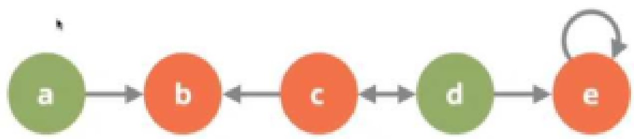


Figure 3.16: Esempio Labeling COMPLETE

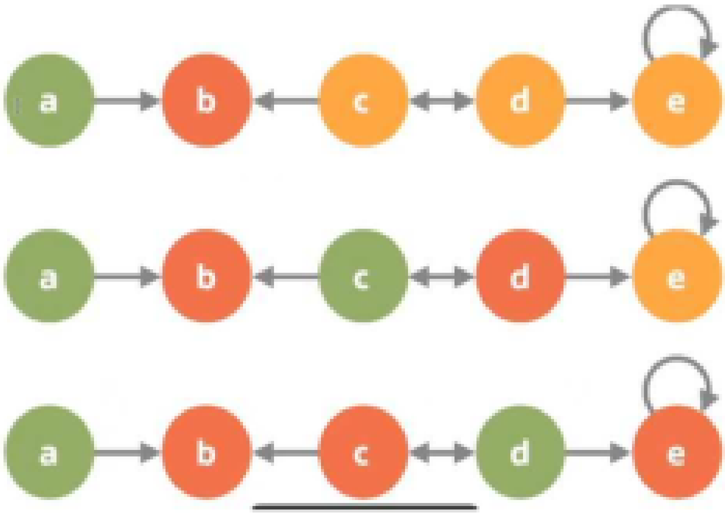
3.4.4 Labeling Grounded (Minimale)

Il labeling deve essere:

- **Completo**
- L'insieme degli argomenti **IN** deve essere **Minimale** tra tutte le labeling complete.

3.4.5 Labeling Preferred (Massimale)

- **Completo**
- L'insieme degli argomenti **IN** deve essere **Massimale** tra tutte le labeling complete.



Di queste:

1. **Grounded** perché solo l'argomento a è IN. L'argomento a dovrà essere in qualsiasi estensione IN, proprio perché non essendo attaccato la regola dice che deve per forza essere IN.
2. **Preferred** questa è sicuramente Admissible.
3. **Preferred**: a, d è massima rispetto l'inclusione.

3.5 Ranking-Based Semantics

Un ranking è un ordinamento parziale o totale di un insieme di argomenti. Quello che ottengo da una semantica di ranking è un vero e proprio ordinamento, cioè una cosa del tipo:

$$a > d > c > e > b$$

e non un insieme come lo era fino ad adesso. Quindi potremo dire quando un argomento è "migliore" rispetto ad un altro. Possiamo avere:

- Ordinamento **quantitativo:** Consiste nell'assegnare prima dei punteggi agli argomenti come "a vale 5, b vale 7" quindi concludo che b è migliore.
- Ordinamento **qualitativo**: "Seguendo questo ordinamento l'argomento a è migliore dell'argomento d.

Metodi visti

1. Categorizer
2. Graded Defense: $a_1 \in d_1^1(X_1)$
3. Shapely Value

3.5.1 Ordinamento quantitativo

3.5.1.1 Categorizer

Ordina gli argomenti utilizzando la seguente funzione:

1. Per ogni argomento guarda il numero degli attacchi che riceve.
2. Utilizza questo valore per assegnare un ranking a tale argomento.

In altre parole abbiamo: Se sono attaccato da argomenti deboli allora sono un argomento forte; se sono attaccato da argomenti forti sono un argomento debole. Se un argomento non ha attaccanti $R^{-}1\ (x) = 0$ allora il

$$Cat(x) = \begin{cases} 1 & \text{if } R_1^-(x) = 0 \\ \frac{1}{1 + \sum_{y \in R_1^-(x)} Cat(y)} & \text{otherwise} \end{cases}$$

Figure 3.17: Funzione categorizer.

valore è 1 sennò è dato 1 dalla formula $\frac{1}{1+\sum_{y\in R^-1_(x)} Cat(y)}$, questo significa che se ho attaccanti forti allora l'argomento x è debole
Spiegazione: Si parte sempre dagli iniziatori del grafo, cioè gli argomenti non attaccati.

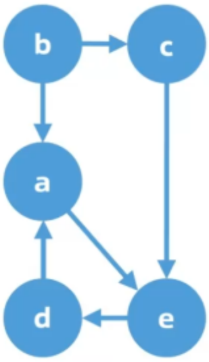


Figure 3.18: Esempio categorizer.

- b : Ha valore 1 perché la frase: "non ha attaccanti" si traduce in $R_1^-(x) = 0$ quindi va nella prima opzione della funzione, si arresta subito e torna 1.

- c : Non vale la prima opzione perché è attaccato da b, quindi seconda opzione e calcolo 1 + la sommatoria dei valori dei suoi attaccanti cioè 1 solamente b. Quindi verrebbe $\frac{1}{1+1} = 0.5$

In questo caso Cat(b)=1 perché non ha attaccanti e Cat(c)=0.5 perché è attaccato da b con punteggio 1. Mentre Cat(a)=0.38, Cat(d) =0.65 e Cat(e) = 0.53.

Si continua cosi per tutti gli argomenti. Infine si ordinano i risultati e si ottiene un ordinamento sui rispettivi argomenti:

$$b > d > e > c > a$$

Quindi b è preferito tramite la funzione *cat* a d e cosi via.

3.5.2 Ordinamento Qualitativo

I principi sono simili a quello precedente:

- più è grande il numero degli attaccanti su un argomento b, più è debole il livello di giustificazione di b

- più è grande il numero di argomenti che difendono a, più è forte il livello di giustificazione di a.

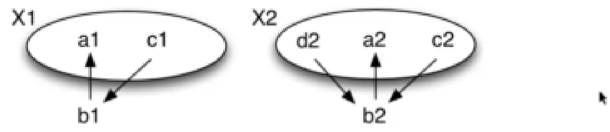
3.5.2.1 Graded Defense, Dung's Theory

In questo caso andiamo a definire una relazione di preferenza tra le coppie dei possibili argomenti **(non si assegnano punteggi)**. I principi che utilizzeremo sono due:

1. Più attaccanti un argomento ha, peggiore è quell'argomento.
2. Più argomenti sono in mia difesa, più un argomento è forte.

Suddivido gli argomenti con la **funzione**: d_n^m (X) che rappresenta tutti quegli argomenti che **non** hanno almeno m attaccanti che a loro volta **non** sono contrattaccati da almeno n argomenti.

Esempio



$$a_1 \in d_1^1(X1) \quad a_2 \in d_1^1(X2) \quad a_1 \notin d_2^1(X1) \quad a_2 \in d_2^1(X2)$$

Figure 3.19: Esempio graded defense.

a_1 ha un attaccante ed un difensore, quindi appartiene. a1 ha si un attaccante, ma non ha due difensori, quindi non appartiene.

- $a_1 \in d_1^1$ (X1): non c'è almeno 1 attaccante che non sia attaccato a sua volta da almeno 1 argomento. Questo vuol dire che c'è almeno un argomento che è contrattaccato da almeno un altro argomento. Infatti a1 è attaccato da b1 che è a sua volta attaccato da c1 .
- $a_2 \in d_1^1$ (X2): non c'è un argomento che attacca a2 che a sua volta non sia contrattaccato da almeno un altro argomento, ma addirittura c_i sono due argomenti (che sarebbero d_2 , c_2) che contrattaccano l'attaccante di a_2 (che sarebbe b_2), quindi diciamo anche che $a_2 \in d_1^2$ (X2). (Esiste un attaccante di a_1 ma non esistono due difensori di a_1).

$$a_2 \in d_2^1 \text{ (X2)}$$

Va letto come: Non esiste un argomento (quindi prima l'esponente) che non sia contrattaccato da almeno 2 argomenti (quindi poi il pedice). Gli attaccanti li leggo ad apice, i difensori (cioè chi attacca il mio attaccante) li leggo a pedice. Il dilemma che ci troviamo davanti è:

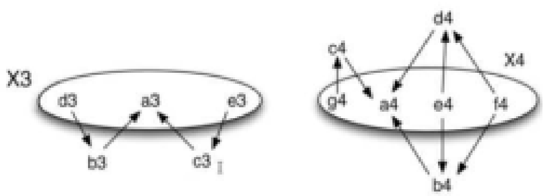
É meglio essere poco attaccati (cioè apice basso) o è meglio avere tanti difensori (pedice alto?)

In formule sarebbe: appartenere a d_1^3 è meglio o peggio di appartenere a d_3^1?

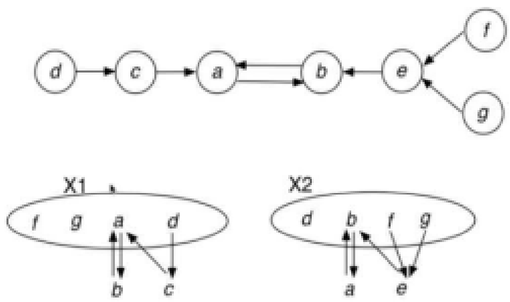
Formula

$$d_n^m \text{ è meglio di } d_t^s \iff \text{m <= s AND t <= n}$$

Cioè un argomento non attaccato da almeno m argomenti che non siano contrattaccati da almeno n difensori è meglio della stessa cosa con s e t se e solo se il primo argomento ha sia **meno attaccanti**, cioè m <= s che **più difensori** t <= n.
Altri esempi: leggere solamente l'appartenenza.



$a_3 \notin d_2^2(X3)$ $a_3 \in d_3^3(X3)$ $a_4 \in d_2^2(X4)$ $a_4 \notin d_3^3(X4)$



$a \notin d_2^1(X1)$ $b \in d_2^1(X2)$ $b >^{pref} a$

Applicazione della Graded Semantics ad un grafo **Ci domandiamo:** è vero che a $\in d_2^1$ (X1)?
Cioè è vero che a non è attaccato da almeno un argomento che non sia a sua volta attaccato da almeno due argomenti?
NO, perché a è attaccato da almeno 1 argomento, vero, ma questo argomento è attaccato da un solo argomento, che sarebbe d per c e a stesso per b. Il controllo sarebbe risultato vero che entrambi gli attaccanti venivano contrattaccati da almeno 2 argomenti.

Ci domandiamo: è vero che b $\in d_2^1$ (X2)?, cioè è vero che b non è attaccato da un singolo argomento che a sua volta non sia contrattaccato da almeno due argomenti?

Importante: Si, perché b è attaccato da almeno un argomento (e) che è a sua volta attaccato da almeno 2 argomenti, f e g.

Si vede per prima l'apice, e la domanda è: **quell'argomento è attaccato da almeno il numero che c'è scritto?**

Se la risposta è no allora sicuramente non appartiene, altrimenti si vede il numero in basso, cioè i difensori. Se ad apice c'era 1 e a pedice 2 ci

chiediamo:

C'è almeno 1 argomento che attacca a che è attaccato a sua volta da 2 argomenti? Se la risposta è si allora appartiene, no altrimenti.

Ora, la regola diceva che devo avere un minor numero di attaccanti e un maggior numero di difensori per essere un argomento migliore, quindi a ha lo stesso numero di attaccanti di b che sarebbe 2, il numero di difensori invece è pari a massimo 1 per a, mentre per b troviamo un contrattacco di 2 argomenti, quindi proprio perché b ha più difensori di a, lo preferisco. quindi:

$$b >^{pref} a$$

3.5.2.2 Shapely Value

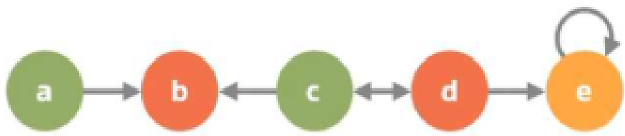
Negli AF va interpretato come "il valore che un argomento apporta dentro una certa estensione".

Assegnamento dei valori agli argomenti
Devo calcolare lo Shapely value per ogni argomento data una certa semantica. Si utilizzano due funzioni, per gli IN e per gli OUT:

RANKING FUNCTIONS

Depends on the semantics σ

$$v^{I}_{\sigma,F}(S) = \begin{cases} 1, & \text{if } S \in in(L_\sigma) \\ 0, & \text{if } otherwise \end{cases} \qquad v^{O}_{\sigma,F}(S) = \begin{cases} 1, & \text{if } S \in out(L_\sigma) \\ 0, & \text{if } otherwise \end{cases}$$



- La funzione verde assegna agli argomenti il valore 1 se sono argomenti IN che sono accettati da qualche semantica σ, oppure 0. Se calcolo il valore per l'insieme a, c questo darà 1 perché l'insieme è IN.
- La funzione rossa invece di guardare gli argomenti che sono IN e dargli punteggio 1, guarda quelli OUT e gli da punteggio 1 in maniera negativa.

Quindi se la funzione V^I da punteggio 1 significa che sono una buona estensione, se la funzione V^O da punteggio 1 allora sono una cattiva estensione. Adesso quindi siamo arrivati al punto di avere per ogni insieme di argomenti un valore che è 0 o 1 in base alla funzione per gli IN o per gli OUT e quindi possiamo generare un ordinamento.

$\forall a, b \in A, a \succ b$ *iff*

- $\phi_a(v^I_{\sigma,F}) > \phi_b(v^I_{\sigma,F})$, *or*
- $\phi_a(v^I_{\sigma,F}) = \phi_b(v^I_{\sigma,F})$ *and* $\phi_a(v^O_{\sigma,F}) < \phi_b(v^O_{\sigma,F})$

$\forall a, b \in A, a \simeq b$ *iff*

- $\phi_a(v^I_{\sigma,F}) = \phi_b(v^I_{\sigma,F})$ *and* $\phi_a(v^O_{\sigma,F}) = \phi_b(v^O_{\sigma,F})$

Ordinamento degli argomenti in base al valore **Il primo punto si traduce in:**
L'argomento a è **migliore** dell' argomento b se e soltanto se:

- Lo Shapely value di a rispetto agli argomenti IN è maggiore dello Shapely value di b sempre rispetto agli argomenti IN.
- Oppure se il valore è uguale (quindi non mi basta andare a controllare solamente gli IN), a deve avere anche uno Shapely value per gli OUT minore di quello di b.

Il secondo punto è l'indifferenza

Scegliere prima a o b è indifferente se e soltanto se:

- Lo Shapely value per gli IN di a è esattamente uguale allo Shapely value per gli IN di b e stessa cosa per il valore di OUT.

Chapter 4

Soft Argumentation Framework (Weighted AF)

4.1 Definizione di Soft Argumentation Framework

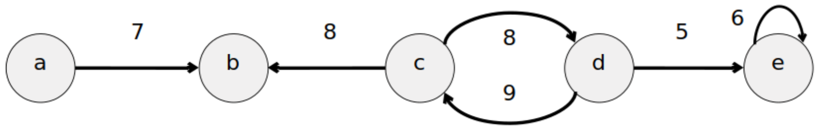
Un Soft Argumentation Framework è una quadrupla:

$$(A_{rgs}, R, W, S)$$

dove:

- A_{rgs} è un insieme di Argomenti.
- R è una relazione di attacco sugli argomenti in A_{rgs}.
- W : A_{rgs} x A_{rgs} $\rightarrow$ S è una funzione binaria che rappresenta il peso associato ad ogni arco.
- S è un semiring $< A, +, x, bottom, top >$ Dati a, b $\in$ Args , $\forall$(a, b) $\in$ R, W (a, b) = s significa che a attacca b con un peso di s $\in$ S.

Esempio:



$\mathcal{A}_{rgs} = \{a, b, c, d, e\}$, $R = \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\}$,
with $W(a, b) = 7, W(c, b) = 8, W(c, d) = 9, W(d, c) = 8$,
$W(d, e) = 5, W(e, e) = 6$, and $\mathbb{S} = \langle \mathbb{R}^{+} \cup \{\infty\}, \min, \hat{+}, \infty, 0\rangle$

$$\langle \mathbb{R}^{+} \cup \{+\infty\}, min, \hat{+}, \infty, 0\rangle$$

Cambia la nozione di attacco: Prima l'attacco era una funzione booleana (a attacca b). Adesso invece quando a attacca b gli viene associato un valore (comunque appartenente al semiring).
Cambia la nozione di difesa: Nell'esempio sopra, nel caso in cui fossimo negli AF tradizionali C si difenderebbe dall'attacco di D (perché ricordiamo era una funzione booleana). In questo caso però, essendo che d attacca c con 9 e c risponde con 8, c non potrà difendersi dall'attacco, poiché la difesa non è sufficiente a contrastare quest'ultimo. Questa nozione dipende strettamente dal semiring utilizzato, poiché per ogni semiring (i cui tipi sono gli stessi introdotti precedentemente) si avranno relazioni diverse.

4.2 w-difesa (Dw)

Dato un Soft Argumentation Framework (A_{rgs}, R, W, S), un sottoinsieme di argomenti B ⊆ Args w-difende un argomento b ∈ Args se e soltanto se, dato a ∈ Args tale per cui R(a, b), allora:

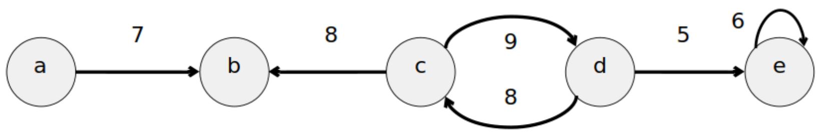
$$W(a, B \cup \{b\}) >=_s sW(B, a)$$

L'insieme B w-difende l'elemento b se e soltanto se difende b da tutti gli attacchi che arrivano a b cioè da tutti gli R(a, b).
$>=_S$ è da intendere come elemento migliore o peggiore all'interno del semiring.
In altre parole, devo verificare che il peso degli attacchi che ricevo sia inferiore al peso degli attacchi che invio.

- Con W (a, B ∪ {b}) si intende il costo degli attacchi che vanno da a all'insieme B ∪ {b} (cioè tutti gli attacchi che apporta quell'elemento a all'insieme B "dall'esterno verso l'interno") sommati con l'operatore di combinazione del semiring. Per sapere quindi quanto vale l'attacco di a verso l'insieme B ∪ {b} nel caso di Semiring Weighted ad esempio devo fare la somma di tutti gli attacchi (proprio perché l'operatore di combinazione è la somma).

- W (B, a) sarebbe "con quanto l'insieme B attacca a, cioè tutti gli attacchi dall'interno di B all'esterno". Anche questo dipende strettamente dal semiring, nel Weighted vanno tutti sommati.

Esempio



$\{c\}$ defends c from d because $W(d, \{c\}) \geq_S W(\{c\}, d)$, i.e., $(8 \leq 9)$.
On the other hand, $\{d\}$ does not defend d because
$W(c, \{d\}) \not\geq_S W(\{d\}, c)$

Per il calcolo degli insiemi ammissibili esistono vari metodi, che variano in base a regole imposte da autori di articoli scientifici.

4.3 Estensioni nei Weighted AF

4.3.1 w-Conflict Free

Dato un W F = (A_{rgs}, R, W, S), un sottoinsieme di argomenti B ⊆ Args è w-conflict free se e soltanto se W(B, B) = top (top del semiring). Questo significa che nessuno degli elementi dentro l'insieme attacca un altro elemento sempre dentro l'insieme. Dire che il peso è uguale al top del semiring (o al bottom) significa che quel peso non è presente, e quindi non è presente una relazione di attacco.

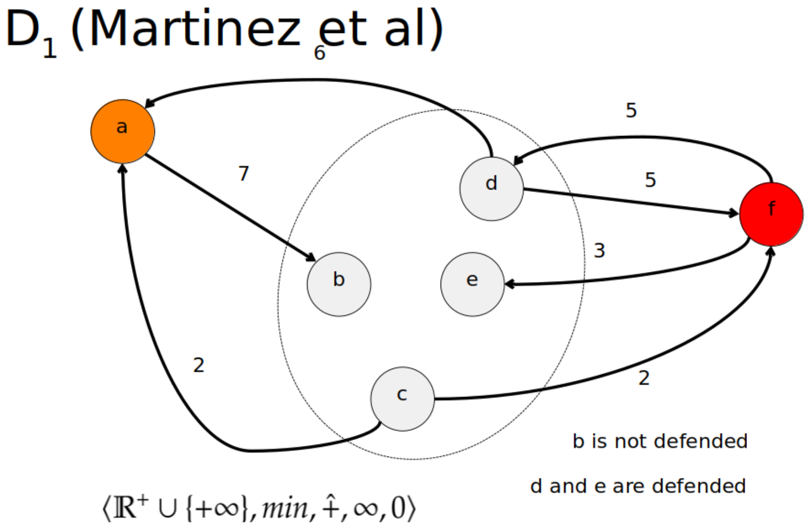
4.3.2 w-Admissible

Dato un W F = (A_{rgs}, R, W, S), un insieme di argomenti B ⊆ Args w-conflict-free è w-admissible se e soltanto se tutti gli argomenti di B sono w-difesi da B.

4.4 Distinzione tra gli insiemi Admissible

4.4.1 Martinez e Simari (D1)

L'obiettivo è capire se l'insieme B = {b, c, d, e} riesce a difendersi dall'attacco di a e poi dall'attacco di f , cioè se quell'insieme è ammissibile. Prendiamo in considerazione per tutti il semiring Weighted. Secondo Martinez e Simari non si aggregano ne le difese ne gli attacchi, quindi prendo il massimo degli attacchi ed il massimo delle difese.



Esempio: Devo verificare che il **massimo** valore degli **attacchi** sia maggiore del **massimo** valore delle **difese**.
Attacco di a **verso** b:

- Attaccanti (a): Max(7) = 7
- Difensori (d,c): Max(6,2) = 6
- Conclusione: 6 < 7, b non è difeso e quindi non ammissibile

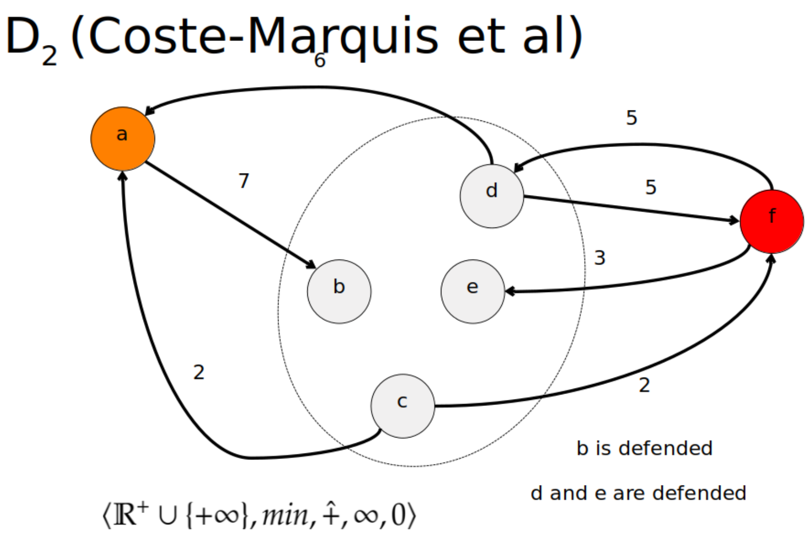
Attacco di f **verso** d, e:

- Attaccanti (f): Max(5,3) = 5
- Difensori (d,e): Max(5,2) = 5 (il 2 viene dall'attacco di c verso f)
- Conclusione: 5 = 5, (d), (e) sono difesi e quindi ammissibili.

La nozione formale di difesa in Simari-Martinez è: Dato W F = (A_{rgs}, R, W, S), a, b, c $\in A_{rgs}$, B $\subseteq A_{rgs}$, allora b è difeso da B se per ogni R(a, b), esiste almeno uno c $\in$ B tale per cui W (a, b) $>=_s$ W (c, a).

4.4.2 Coste-Marquis (D2)

Questo approccio aggrega solamente le **difese.**



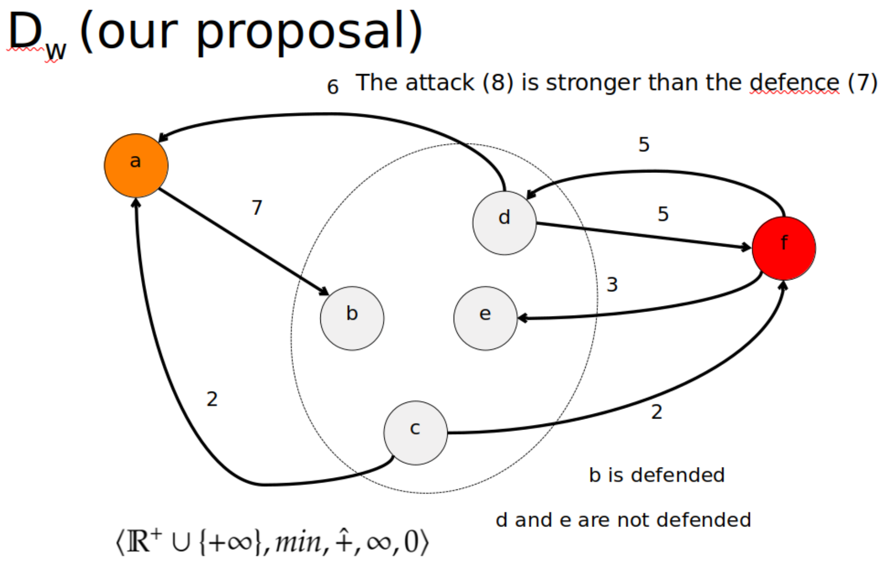
Attacco di a **verso** b:

- Attaccanti (a): Max(7) = 7
- Difensori (d,c): Difesa(6)+Difesa(2) = 6+2 = 8
- Conclusione: 8 > 7, b è difeso e quindi ammissibile

4.4.3 Santini e Bistarelli (Dw)

Questo approccio aggrega **sia gli attacchi che le difese, ciò vuol dire che b è difeso ma d ed e no.**
Attacco di a **verso b:**

- Attaccanti (a): Attacco(7) = 7



- Difensori (d,c): Difesa(6)+Difesa(2) = 6+2 = 8
- Conclusione: 8 > 7, b è difeso e quindi ammissibile

Attacco di f **verso d, e:**

- Attaccanti (f): Attacco(5)+Attacco(3) = 5+3 = 8
- Difensori (d,e): Difesa(5) + Difesa(2) = 7
- Conclusione: 8 > 7, d, e **non** sono difesi e quindi non ammissibili.

4.4.4 Teoremi di implicazione tra le nozioni

N.B1 (da sapere): La relazione di w-difesa implica la relazione di difesa:

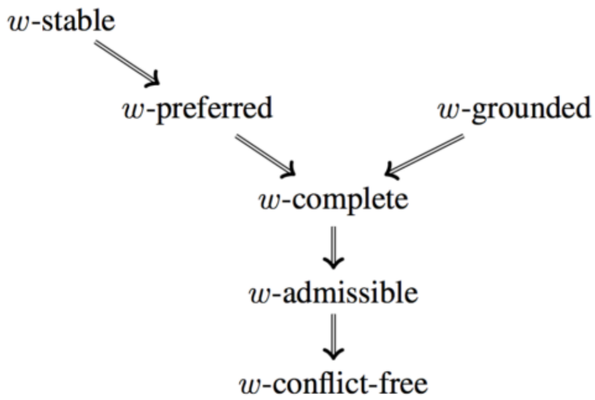
$$\text{B w-difende b} \rightarrow \text{B difende b}$$

N.B2: Nel semiring Classic (booleano) si ha che:

$$\text{B w-difende a} \iff \text{B difende a}$$

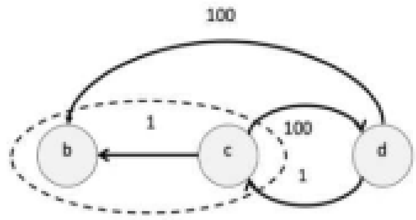
- $D_W \Rightarrow D_2$, Bista implica Costa-Marquis
- $D_1 \Rightarrow D_2$, Martinez implica Costa-Marquis
- Quindi se è ammissibile per Martinez e Bista allora è ammissibile anche per Costa-Marquiz

- Se S $=<[0,1], max, min, 0, 1>$ cioè semiring Fuzzy, allora D1 $\Longleftrightarrow$ D2
- Se $S =< [0,1], max, min, 0, 1 >$ cioè semiring Fuzzy, allora Dw $\Longleftrightarrow$ D1
- Se $S =< [true, false], or, and, false, true >$ cioè semiring Classic, allora $D_w \Longleftrightarrow D_0 \Longleftrightarrow D1 \Longleftrightarrow D_2$ (ma cosa è D_0)??



L'implicazione delle semantiche è la stessa sia nel caso classico che in quello pesato.

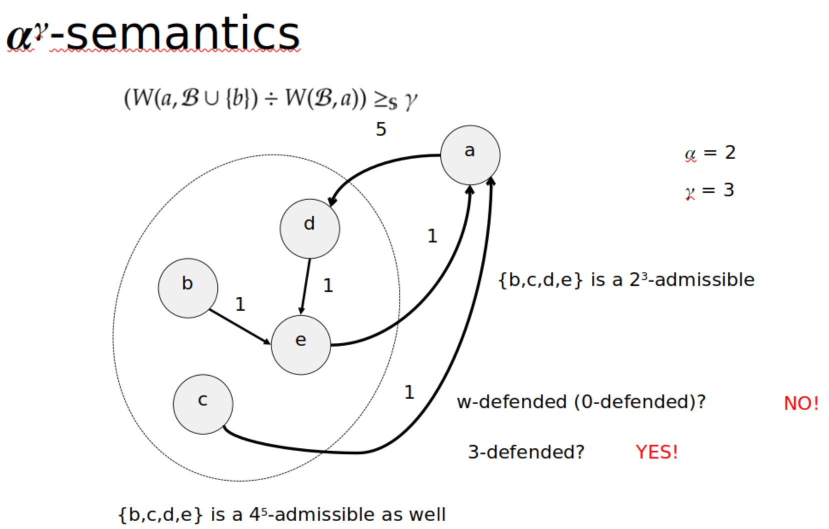
4.5 Orthogonal Relaxations



In questo esempio notiamo che (d, c) non stanno bene insieme, perché si attaccato 100. (b, c) sono abbastanza compatibili, perché la relazione di attacco è presente ma con peso molto basso. Abbiamo poi che b non è ammissibile (non riesce a difendersi da c), c è ammissibile, d è ammissibile e nessuna coppia di argomenti è ammissibile. Questo però non è una cosa positiva, perché si vorrebbero cercare delle coalizioni superiori al singleton. Se però riuscissi ad accettare un po' di conflitti interni potrei comunque creare una coalizione. Infatti notiamo che potremmo mettere insieme (b, c) dato che la relazione di attacco è si presente ma con peso molto basso. Introduciamo quindi l'Alpha-gamma consistenza.
Questi rilassamenti sono strettamente correlati e influenzano l'un l'altro: permettere un piccolo conflitto può portare ad avere più argomentazioni in un'estensione, e di conseguenza si ha una difesa più forte o più debole. Un esempio reale potrebbe essere in campo politico, partiti con piccole divergenze interne si uniscono in modo da raggiungere una percentuale sufficiente di voti per governare.

4.6 Alpha-Gamma consistenza

Che cosa è l'**alpha-gamma consistenza?**
Questa consistenza definisce il quantitativo di attacchi "interni" cioè alpha ed "esterni" cioè gamma che si ammettono.
In questo esempio, l'insieme $B = \{b, c, d, e\}$ non è conflict-free (b e d attaccano e), però se volessimo misurare il valore di questo conflitto considerando il semiring Weighted (operazione somma), l'insieme è 2-conflict-free (infatti 2 elementi in questo insieme attaccano un altro elemento sempre appartenente all'insieme). Se quindi la soglia α fosse 2 riuscirei a sopportare l'attacco tramite il rilassamento dei conflitti interni. Guardiamo l'ammissibilità



dell'insieme: a attacca l'insieme B con 5 e B è difeso da 1 ed 1. In realtà l'argomento d andrebbe escluso dall'insieme perché porta un attacco con peso molto alto all'insieme stesso. Se però devo per forza costruire un organo di 4 elementi devo accettare sia i conflitti interni sia l'attacco verso d (potrei anche portare dentro a).

4.6.1 Unità di sopportazione γ

N.B secondo Bistarelli l'insieme B non è ammissibile, perché l'attacco di 5 non è difeso dalla somma delle difese 1+1=2. Facendo il calcolo: (attacco-difesa) 5-2=3 si calcola l'unità di **sopportazione** γ che in questo caso è 3. Quindi:

- α Peso del **conflitto interno** (in questo caso 2, dall'attacco di b pari ad 1 e d sempre 1 verso e)
- γ Peso del **conflitto esterno** (in questo caso 3 (5 - 2))

L'insieme dell'esempio sopra è 2^3 **admissible** perché $\alpha = 2$ e $\gamma = 3$. Inoltre B è anche stabile, perché tutti gli elementi esterni (in questo caso solo a) sono attaccati. Nel caso in cui portassimo d fuori si avrebbe un valore più basso di γ ma B non sarebbe stato stabile, perché nessuno attaccava d (a rimaneva comunque fuori).
La semantica Alpha Gamma può essere descritta come:

$$W(a, B \cup \{b\}) \div W(B, a)) >=_s \gamma$$

dove:

- $W(a, B \cup \{b\})$: Il peso dell'attacco dall'esterno a verso l'interno B
- $\div$: opposto della combinazione (x), abbiamo fatto esempi sempre con il Weighted, cioè la somma, quindi assumiamo sia la **sottrazione**
- W (B, a) Il peso dell'attacco dall'interno B verso l'esterno a
- Il risultato deve essere $>=_s \gamma$ (migliore all'interno del semiring s).

L'insieme $B = \{b, c, d, e\}$ è anche 4^5 **admissible**, e per dimostrare questo facciamo riferimento alle inclusioni.

4.6.2 Inclusioni in Alpha-Gamma consistenza

Teorema: Dato un W F=(A_{rgs}, R, W, S) con S=$(A, (+), (x), bottom, top)$ e $\alpha, \gamma \in$ A:

$$\alpha^\gamma - stable \rightarrow \alpha^\gamma - prefered \rightarrow \alpha^\gamma - complete \rightarrow \alpha^\gamma - admissible \rightarrow \alpha - conflict - free$$

Formal results

Theorem 5.5. *Given $\langle \mathcal{A}_{rgs}, R, W, S = \langle A, \oplus, \otimes, \bot, \top \rangle \rangle$, and $\alpha_1, \alpha_2, \gamma_1, \gamma_2 \in A$ s.t. $\alpha_1 \geqslant_S \alpha_2$ and $\gamma_1 \geqslant_S \gamma_2$, then*

1. *the set of α_1-conflict-free extensions is a subset of the set of α_2-conflict-free extensions.*
2. *the set of $\alpha_1^{\gamma_1}$-admissible extensions is a subset of the set of $\alpha_2^{\gamma_2}$-admissible extensions.*
3. *for each $\alpha_1^{\gamma_1}$-complete extension $\mathcal{B}_1$, there exists an $\alpha_2^{\gamma_2}$-complete extension $\mathcal{B}_2$, such that $\mathcal{B}_1 \subseteq \mathcal{B}_2$.*
4. *for each $\alpha_1^{\gamma_1}$-preferred extension $\mathcal{B}_1$, there exists an $\alpha_2^{\gamma_2}$-preferred extension $\mathcal{B}_2$, such that $\mathcal{B}_1 \subseteq \mathcal{B}_2$.*
5. *for each $\alpha_1^{\gamma_1}$-stable extension $\mathcal{B}_1$, there exists an $\alpha_2^{\gamma_2}$-stable extension $\mathcal{B}_2$, such that $\mathcal{B}_1 \subseteq \mathcal{B}_2$.*

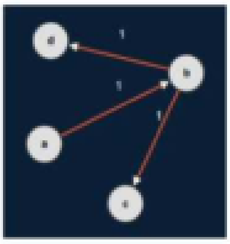
Le uniche cose spiegate dal prof nella figura sopra sono:
Se un estensione è α_1 ammissibile e $\alpha_1 >=_s \alpha_2$ allora quell'estensione è anche α_2 ammissibile. Stessa cosa per γ.
Questo significa che nell'esempio sopra che era 2^3 admissible sarà anche 4^5 ammissibile (ecco spiegata la domanda della sezione sopra).

4.7 w-Grounded

Gli insiemi grounded sono gli insiemi complete più piccoli.
Def di Grounded (Minimale) nel caso normale (Crisp AF):

E $\in$ gr(F) if f E $\in$ co(F) e **non esiste** E' $\in$ co(f) tale per cui E' **non è contenuto** in E.

Un estensione E è grounded se e solo se è una complete e nessun altra estensione dentro le complete è più piccola di lei (quindi è la più piccola delle complete).



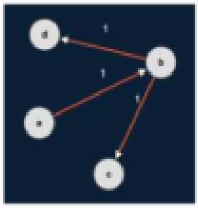
- Co(F) = {{a,d},{a,c}}
- Gr(F) = {{a,d},{a,c}}

Calcolo delle complete se non ci fossero i pesi: L'insieme complete era (a, d, c) perché a difendeva sia d che c insieme.
Calcolo delle complete con pesi (usiamo Bistarelli): Le complete in questo caso sono a perché non viene attaccata da nessuno e più tutte quelle difese da a cioè d e c. **Non le difende tutte e due insieme** perché il costo 1 non è sufficiente e difenderle tutte e due, deve per forza farlo uno alla volta (cosi ha detto il bista). Quindi le complete sono (a, d) e (a, c). Applicando la definizione di Grounded all'esempio otteniamo che l'insieme è sempre lo stesso delle complete, ma questo è un problema, perché le grounded hanno la proprietà di essere **uniche**.
Tutto quello sopra è per dimostrare che la definizione di Grounded con pesi negli AF va cambiata a:
Un elemento E è Grounded se e soltanto se:

1. **É ammissibile** (prima si richiedeva che era complete);
2. É incluso nell'intersezione delle **complete**;
3. Non esiste nessun elemento **ammissibile** più grande di E.

Definition of w-grounded (unique!)

$E \in gr(F)$ *iff* $E \in co(F)$ *and there is no* $E' \in co(F)$ *s.t.* $E' \subsetneq E$



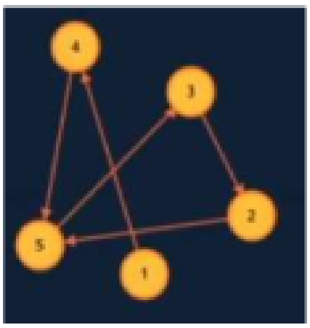
- Co(F) = {{a,d},{a,c}}
- Gr(F) = {{a}}

$E \in gr(F)$ *iff* $E \in ad(F)$, $E \subseteq \bigcap co(F)$ *and there is no* $E' \in ad(F)$ *satisfying* $E' \subseteq \bigcap co(F)$ *s.t.* $E \subsetneq E'$,

L'insieme degli ammissibili in questo caso era: (a), (a, c), (a, d) quindi l'intersezione viene a che è appunto l'unica grounded (e non è attaccata da nessuno).
Negli Weighted AF la definizione di Grounded va cambiata perché altrimenti si perderebbe la proprietà di essere **uniche**.

4.8 Argomento Scetticamente/Credulosamente accettato

Trattiamo adesso un problema decisionale (quindi chiedere se un argomento è SI/NO), introducendo il significato di un argomento Credulosamente o **Skep/(Scet)ticamente** accettato su questo esempio.



- **Cred** (Almeno in un insieme): Ci domandiamo: esiste **almeno** una volta che l'argomento 2 compare in output? La domanda va posta in base a quello che calcoliamo, ad esempio, nel caso in cui calcolassimo le conflict-free e il 2 non compare tra le coppie che lo sono, la risposta sarà NO, mentre nel caso in cui comparisse in almeno una coppia (o anche da solo) la risposta sarebbe SI.

- **Skept** (In tutti gli insiemi): Se un elemento è **sempre** dentro un estensione. In questo caso l'output sarà NO perché non c'è un elemento che è comune a tutti gli insiemi. É possibile selezionare l'elemento, quindi magari per il valore 4 da NO, ma per il valore 3 da SI.

Chapter 5

Domande

5.1 Domande d'esame

1. Cosa è un CSP?
2. Cosa è un Soft CSP?
3. Fatemi vedere come funziona Local Consistency nel Fuzzy, nel caso classico
4. Calcola la soluzione di un CSP
5. Cosa è la combinazione e proiezione?
6. Che differenza c'è tra un SCSP classico e un SCSP fuzzy
7. Cosa è arc consistency
8. Come si fa nel caso classico e nel caso soft
9. Quando un problema è arc-consistente (punto fisso).
10. Ci scrive un problema Crisp e noi dobbiamo applicare arc consistenza.
11. Quali sono gli algoritmi per ottenere arc-consistenza? (AC1,2,3,4)
12. Cosa è un Argumentation Framework?
13. Cosa è un Soft Argumentation Framework? Come cambia la nozione di difesa e attacco?
14. Ti mostro un esempio, dimmi quali sono gli insiemi Conflict Free

15. Disegna un AF a caso, dimmi quali sono le Estensioni conflict-free, quelle admissible e tutte le altre.
16. Inserisci un peso sugli archi nell'AF che hai disegnato e ricalcolami tutte le semantiche
17. Dammi la definizione delle semantiche richieste
18. Differenze tra Martinez-Simari, Coste-Marquiz e Bistarelli.
19. Nel caso Soft AF quale è la definizione di grounded?