UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Da programmazione logica a CLP

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione
Anno accademico 2020/2021

Prof. MARCO GAVANELLI

PROGRAMMAZIONE LOGICA A VINCOLI

- Problemi di soddisfacimento di vincoli:
  - concetti generali
- Programmazione Logica
  - vantaggi
  - limiti
- Programmazione Logica a Vincoli
  - dominio e interpretazione
  - controllo
  - modello computazionale

PROBLEMI DI SODDISFACIMENTO DI VINCOLI

- Un problema di soddisfacimento di vincoli (*Constraint Satisfaction Problem, CSP*) è definito da:
  - *un insieme di variabili* (V_1, V_2…V_n)
  - *un dominio discreto per ogni variabile* (D_1,D_2…D_n)
  - *un insieme di vincoli su queste variabili*:
    vincolo: una relazione tra variabili che definisce un sottoinsieme del prodotto cartesiano dei domini D_1xD_2x...xD_n

Soluzione di un Problema di Soddisfacimento di vincoli: un assegnamento di valori alle variabili che soddisfa i vincoli.

E. Tsang: "Foundations of Constraint Satisfaction" Academic Press, 1992.

CSP, formalmente

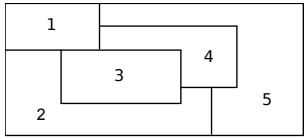
Un problema di soddisfacimento di vincoli è dato da

- Un insieme di variabili $\boldsymbol{X}=\{X_1, X_2, ...X_n\}$
- Ciascuna variabile ha associato un dominio
  $\boldsymbol{D} = \{D_1, D_2, \ldots, D_n\}$ (indicato anche con $D(X_1)$, ...$D(X_n)$)
- Un insieme di vincoli $\boldsymbol{C}=\{C_1,...,C_v\}$.
  Ciascun vincolo C_i (che coinvolge k variabili) è definito da
  - Uno scope $S_i =\{X_{i_1},X_{i_2},...,X_{i_k}\}$
  - Una relazione R_i che coinvolge le k variabili, ovvero un sottoinsieme del prodotto cartesiano dei rispettivi domini:
    $R_i \subseteq D(X_{i_1})$ **x** $D(X_{i_2})$ **x** … **x** $D(X_{i_k})$

Una soluzione è un assegnamento che soddisfa tutti i vincoli, cioè una funzione a: $X_j \rightarrow v_j$, dove

- $v_j \in D_j$
- Per ogni vincolo $C_i(X_{i_1},X_{i_2},...,X_{i_k})$, $(v_{i_1},v_{i_1},...,v_{i_1}) \in R_j$

ESEMPIO: Map Coloring



Trovare un assegnamento di colori (presi dall'insieme *red, blue, green, yellow*) alle regioni in modo che due regioni confinanti abbiano colore diverso

– *variabili* V_1, V_2, V_3, V_4, V_5: *regioni*

– *domini* D_1, D_2, D_3, D_4, D_5: *[red, blue, green, yellow]*

– *vincoli* : $V_1 \neq V_2, V_1 \neq V_3, V_1 \neq V_4, V_1 \neq V_5, V_2 \neq V_3, V_2 \neq V_4, V_2 \neq V_5, V_3 \neq V_4, V_4 \neq V_5$.

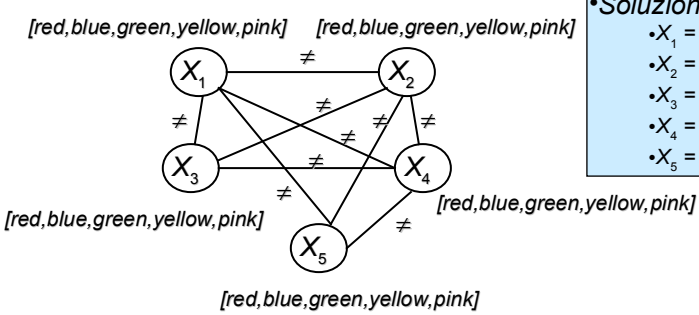
≠	red	blue	green	yellow
red	×	✓	✓	✓
blue	✓	×	✓	✓
green	✓	✓	×	✓
yellow	✓	✓	✓	×

CONSTRAINT GRAPHS

Un problema di soddisfacimento di vincoli si può rappresentare con un grafo detto constraint graph:

– *variabili* ⟺ *nodi*

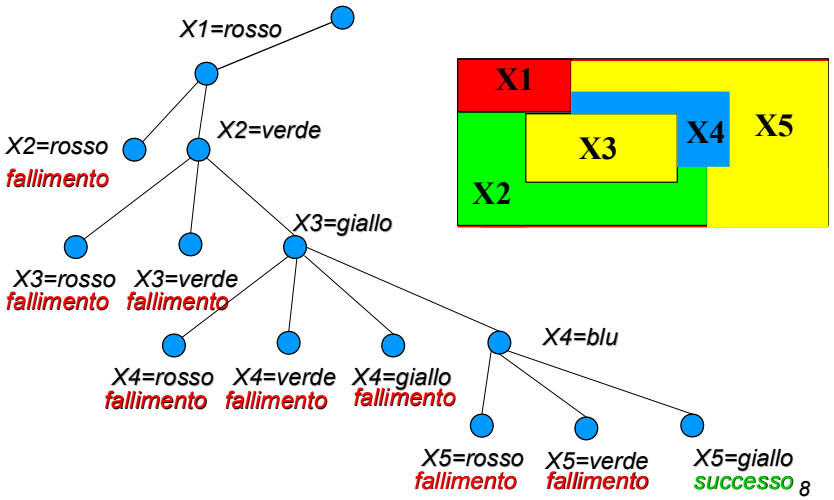
– *vincoli* ⟺ *(iper)-archi*



•*Soluzione ammissibile:*
- •X_1 = red
- •X_2 = green
- •X_3 = blue
- •X_4 = yellow
- •X_5 = pink

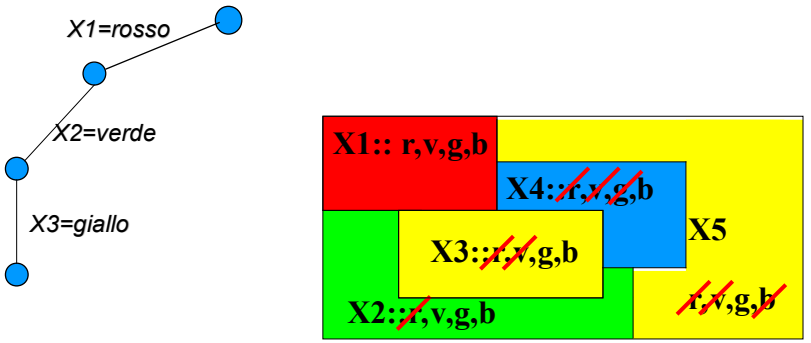
ESEMPIO: Map Coloring

- Algoritmo semplice (ma a volte inefficiente)



PROPAGAZIONE DI VINCOLI

- Eliminazione a priori dei valori incompatibili



PROBLEMI DI OTTIMIZZAZIONE

Un problema di ottimizzazione è definito da:

- *un insieme di variabili* (X_1, X_2…X_n)
- *un dominio discreto per ogni variabile* (D_1,D_2…D_n)
- *un insieme di vincoli su queste variabili*:

  vincolo: una relazione tra variabili che definisce un sottoinsieme del prodotto cartesiano dei domini D_1xD_2x...xD_n

- *una funzione obiettivo* $f(X_1, X_2…X_n)$ *da minimizzare o massimizzare*

Soluzione di un problema di minimizzazione: un assegnamento di valori alle variabili compatibile con i vincoli del problema che minimizza la funzione obiettivo

Per risolvere un problema di massimizzazione dato un solver in grando di minimizzare, si può cambiare segno alla funzione obiettivo

COP, formalmente

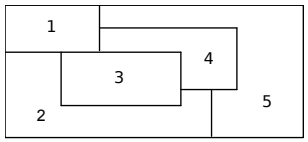
Un problema di ottimizzazione vincolata (Constraint Optimization Problem, COP) è dato da

- Un insieme di variabili $\boldsymbol{X}=\{X_1, X_2, ...X_n\}$
- Ciascuna variabile ha associato un dominio
  $\boldsymbol{D} = \{D_1, D_2, \ldots, D_n\}$ (indicato anche con $D(X_1)$, ...$D(X_n)$)
- Un insieme di vincoli $\boldsymbol{C}=\{C_1,...,C_v\}$. Ciascun vincolo C_i (che coinvolge k variabili) è definito da
  - Uno scope $S_i =\{X_{i_1},X_{i_2},...,X_{i_k}\}$
  - Una relazione R_i
    $R_i \subseteq D(X_{i_1})$ **x** $D(X_{i_2})$ **x** ... **x** $D(X_{i_k})$
- Una funzione obiettivo $f(X_{i_1},X_{i_2},..., X_{i_k})$ da minimizzare (o massimizzare)

Una soluzione è un assegnamento che soddisfa tutti i vincoli, cioè una funzione a: $X_j \rightarrow v_j$, dove

- $v_j \in D_j$
- Per ogni vincolo $C_i(X_{i_1},X_{i_2},...,X_{i_k})$, $(v_{i_1},v_{i_2},...,v_{i_k}) \in R_j$
- E tale che non esiste una soluzione migliore, cioè $\nexists\ a'$: $X_j \rightarrow v'_j$, dove
  - $v'_j \in D_j$
  - Per ogni vincolo $C_i(X_{i_1},X_{i_2},...,X_{i_k})$, $(v'_{i_1},v'_{i_2},...,v'_{i_k}) \in R_j$
  - E $f(v'_{i_1},v'_{i_2},...,v'_{i_k}) < f(v_{i_1},v_{i_2},...,v_{i_k})$

EXAMPLE: Map Coloring



• Trovare un assegnamento di colori alle zone tale che due zone adiacenti sono colorate con colori diversi, e MINIMIZZANDO il numero di colori usati

- *variabili* V_1, V_2, V_3, V_4, V_5: *zone*
- *domini* D_1, D_2, D_3, D_4, D_5: [1,2,3,4,5]
- *vincoli :* $near(V_i, V_j) \Rightarrow V_i \neq V_j$.
- *min* f = $max([V_1, V_2, V_3, V_4, V_5])$

È sempre possibile usare come dominio un sottoinsieme dei numeri interi

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Dalla programmazione logica alla CLP

Prolog per risolvere CSP

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Anno accademico 2020/2021

Prof. MARCO GAVANELLI

“Prolog è un buon linguaggio per risolvere CSP?”

Prolog per CSP

Vantaggi

- Linguaggio dichiarativo
- Variabili logiche
- Backtracking
- Reversibilità: molti predicati per generare combinazioni

Svantaggi

- Integrazione con altro software?

Es reversibilità

```
member(X,[X|_]).
member(X,[_|T]):- member(X,T).
```

- verifica se un elemento appartiene ad una lista

```
?- member(1,[4,1,2]).
yes
```

- Se il primo argomento è una variabile, in backtracking le vengono assegnati i vari elementi della lista

```
?- member(X,[4,1,2]).
yes, X=4, more?
yes, X=1, more?
yes, X=2.
```

- Utile per generare assegnamenti!

Esempio di CSP

- Variabili: X, Y - Domini: da 1 a 4 - Vincoli: X>Y

```
csp(X,Y):-
   member(X,[1,2,3,4]),
   member(Y,[1,2,3,4]),
   X>Y.
```

- Elenca, in backtracking, tutte le soluzioni del CSP
- Molto dichiarativo: dichiaro le variabili, i domini (member) e i vincoli
- Quale algoritmo viene usato?
- Che cosa devo fare se voglio cambiare euristica?
  - selezione del valore
  - selezione della variabile

Esempio Reversibilità 2

```
permutation([],[]).
permutation([Head|Tail],PermList) :-
   permutation(Tail,PermTail),
   delete(Head,PermList,PermTail).
delete(A, [A|B], B).
delete(A, [B, C|D], [B|E]) :-
   delete(A, [C|D], E).
```

Può generare le permutazioni di una lista

```
?-  permutation([a,b,c],L).
Yes, L = [a, b, c]      more? ;
Yes, L = [b, a, c]      more? ;
Yes, L = [b, c, a]      more?
```


UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

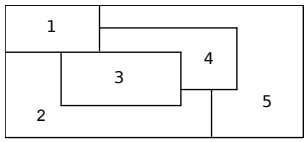
Dalla programmazione logica alla CLP

03 – Map Coloring in Prolog

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione
Anno accademico 2020/2021

Prof. MARCO GAVANELLI

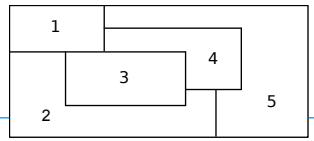
ESEMPIO: Map Coloring



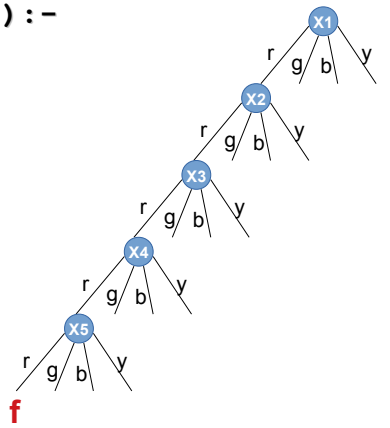
- Trovare un assegnamento di colori alle variabili che soddisfa i vincoli

– *variabili* V_1, V_2, V_3, V_4, V_5: *zone*
– *domini* D_1, D_2, D_3, D_4, D_5: *[r, g, b, y]*
– *vincoli :* $V_1 \neq V_2$, $V_1 \neq V_3$, $V_1 \neq V_4$, $V_1 \neq V_5$, $V_2 \neq V_3$, $V_2 \neq V_4$, $V_2 \neq V_5$, $V_3 \neq V_4$, $V_4 \neq V_5$.

Esempio



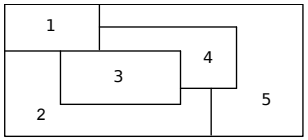

```
coloring([X1,X2,X3,X4,X5]):-
    member(X1,[r,g,b,y]),
    member(X2,[r,g,b,y]),
    member(X3,[r,g,b,y]),
    member(X4,[r,g,b,y]),
    member(X5,[r,g,b,y]),
    X2 \= X1, X3 \= X1,
    X4 \= X1, X5 \= X1,
    X3 \= X2, X4 \= X2,
    X5 \= X2, X4 \= X3,
    X4 \= X5.
```



Molto dichiarativo!
Che algoritmo viene usato?

Standard backtracking

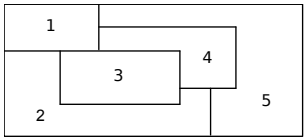
```
coloring([X1,X2,X3,X4,X5],Dom):-
    member(X1,[r,g,b,y]),
    member(X2,[r,g,b,y]),
    X2 \= X1,
    member(X3,[r,g,b,y]),
    X3 \= X1, X3 \= X2,
    member(X4,[r,g,b,y]),
    X4 \= X1, X4 \= X2, X4 \= X3,
    member(X5,[r,g,b,y]),
    X5 \= X1,
    X5 \= X2, X4 \= X5.
```



- L'ordine è importante
- un po' meno dichiarativo
- Risolve solo questa istanza

Standard backtracking: generale

```
% colori(+LStati, -Assegnam, +NodiGiaAssegnati,
  +ValoriGiaAssegnati, Dom)
% ?- colori([1,2,3,4,5], X,[],[],[r,b,g,y]).
colori([],[],_,_,_).
colori([N1|Ntail], [X|Tail], NPlaced, Placed, Values):-
  member(X,Values),
  compatible(X,N1,Placed,NPlaced),
  colori(Ntail, Tail, [N1|NPlaced], [X|Placed], Values).

compatible(_,_,[],[]).
compatible(X,N1,[P|Tail],[NP|NTail]):-
  (near(N1,NP); near(NP,N1)), !,
  X\==P,
  compatible(X,N1,Tail,NTail).
compatible(X,N1,[P|Tail],[NP|NTail]):-
  compatible(X,N1,Tail,NTail).
```




```
near(1,2).
near(1,3).
...
```


Altro esempio: N-queens

```
queens(S,N):-
% Genero la lista D=[1,2,3...,N]
   domains(1,N,D),
% Calcolo una permutazione S
   permutation(D,S),
% Verifico se rispetta i vincoli
   safe(S).

safe([]).
safe([Queen|Others]) :-
 safe(Others),
 noattack(Queen,Others,1).
```

```
domains(X,X,[X]):-!.
domains(N,Max,[N|T]):-
     N1 is N+1,
     domains(N1,Max,T).

noattack(_,[],_).
noattack(Y,[Y1|Ylist],Xdist):-
     Y-Y1=\=Xdist,
     Y1-Y=\=Xdist,
     Dist1 is Xdist +1,
     noattack(Y,Ylist,Dist1).
```

Che algoritmo è?

N-Queens Standard Backtracking

```
stdback(X,N):-
   domains(1,N,D),
   stdback(X,[],D).

stdback([], Placed, []).
stdback([X|Xs], Placed,
  Values):-
  delete(X,Values,NewValues),
  noattack(X,Placed,1),
  stdback(Xs,[X|
  Placed],NewValues).
```

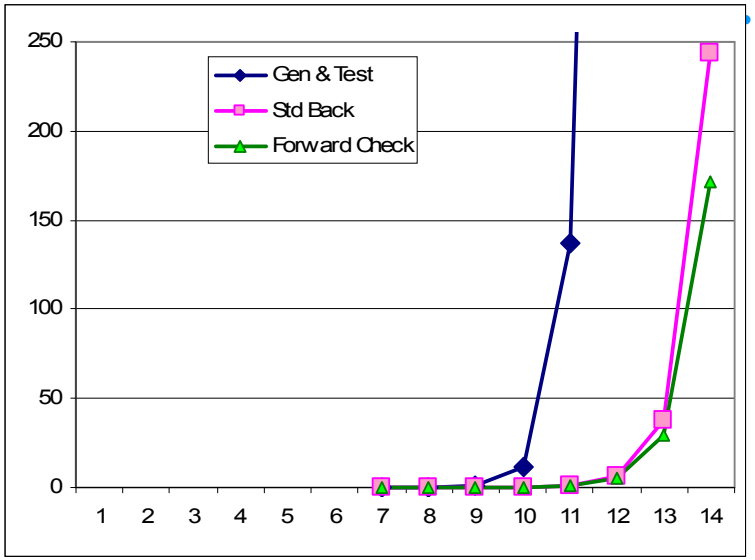
```
domains(X,X,[X]):-!.
domains(N,Max,[N|T]):-
     N1 is N+1,
     domains(N1,Max,T).

noattack(_,[],_).
noattack(Y,[Y1|
   Ylist],Xdist):-
     Y-Y1=\=Xdist,
     Y1-Y=\=Xdist,
     Dist1 is Xdist +1,
     noattack(Y,Ylist,Dist1).
```

N-Queens Forward Checking

```
fwdcheck(Var, N):-
    domains(1,N,D),
    length(Var,N),
    assegna_dom(VarDom,D,Var),
    queens_aux(VarDom).
queens_aux([]).
queens_aux([[X1,D]|Rest]):-
    member(X1,D), % istanzia X1
    forward(X1,Rest,Newrest),
    %propagazione
    queens_aux(Newrest).
forward(X,Rest,Newrest):-
    forward(X,Rest,1,Newrest).
forward(X,[],Nb,[]).
forward(X,[[Var,Dom]|Rest],Nb,[[Var,
    [F|T]]|Newrest]):-
    remove_value(X,Dom,Nb,[F|T]),
    Nb1 is  Nb +1,
    forward(X,Rest,Nb1,Newrest).
```

```
remove_value(X,[],Nb,[]).
remove_value(X,[Val|Rest],Nb,[Val|
    Newrest]):-
    compatible(X,Val,Nb), !,
    remove_value(X,Rest,Nb,Newrest).
remove_value(X,[Val|Rest],Nb,Newrest):-
    remove_value(X,Rest,Nb,Newrest).
domains(X,X,[X]):-!.
domains(N,Max,[N|T]):-
     N1 is N+1,
     domains(N1,Max,T).
compatible(Value1,Value2,Nb):-
    Value1 =\= Value2 +Nb,
    Value1 =\= Value2 - Nb,
    Value1 =\= Value2.
% Crea una lista [[Var1,Dom],
    [Var2,Dom],…]
assegna_dom([],_,[]).
assegna_dom([[V,D]|LVarDom],D,[V|
    LVar]):-
     assegna_dom(LVarDom,D,LVar).
```


N-Queens: Efficienza



secondi necessari per trovare tutte le soluzioni

Limite n. 1

Prolog spinge a scrivere programmi basati su Standard Backtracking

meno efficienti di algoritmi che applicano il pruning a priori

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Dalla programmazione logica alla CLP

04 – rappresentazione dei numeri in Prolog

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Anno accademico 2020/2021

Prof. MARCO GAVANELLI

Rappresentazione dei numeri

- Risolvere questa equazione:

$$X+\cancel{2}^{\,1} = Y+\cancel{1}$$

- *Soluzione:*

$$X+1 = Y$$

- *Possiamo ottenere questo risultato in Prolog?*

Possiamo scrivere così?

```
?- X+2 = Y+1.
```

```
p(X,2)=p(Y,1)
```


Aritmetica di Peano

Un numero naturale può essere

- 0
- Il successore `s()` di un numero naturale

`0, s(0), s(s(0)), s(s(s(0))), ...`

```
sum(X,0,X).
sum(X,s(Y),s(Z)):- sum(X,Y,Z).

?- sum(X,s(s(0)),Z), sum(Y,s(0),Z).
yes, Y=s(X).
```


E così?

```
?- Z is X+2, Z is Y+1.
```

Ordine dei goal significativo

- X is Y+1, Y=3 → errore
- Y=3, X is Y+1 → X=4

Lo stesso vale per le relazioni (**>, <, ≠, ≥, …**):

- X > 3, X = 5 → errore
- X = 5, X > 3 → yes

Limite n. 2

Prolog non interpreta i numeri

Soluzione parziale: freeze, when

`freeze(X,atomo(X))`

- Indica a Prolog che l'`atomo` deve essere selezionato (dalla risoluzione SLD) solo quando `X` non è variabile.

```
freeze(Y, X is Y+1), Y=3.
        Y/3 |
freeze(3, X is 3+1).
        X/4 |
           []
```

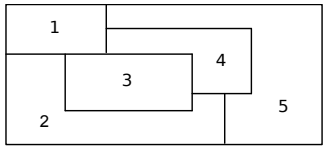

Soluzione parziale: freeze, when

`when(Goal,atomo(X))`

- Indica a Prolog che l'`atomo` deve essere selezionato (dalla risoluzione SLD) solo quando il Goal è vero.
- Es
  when(nonvar(Y), X is Y+1)
- Es
  when((nonvar(X),nonvar(Y)), X>Y)

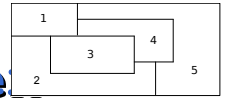
Map Coloring usando when

```
diverso(A,B):-
  when((nonvar(A),nonvar(B)),A\=B).
coloring([X1,X2,X3,X4,X5],Dom):-
  diverso(X2,X1), diverso(X3,X1),
  diverso(X4,X1), diverso(X5,X1),
  diverso(X3,X2), diverso(X4,X2),
  diverso(X5,X2), diverso(X4,X3),
  diverso(X4,X5),
  member(X1,Dom),
  member(X2,Dom),
  member(X3,Dom),
  member(X4,Dom),
  member(X5,Dom).
```



Molto dichiarativo! Che algoritmo viene usato?

Map Coloring usando whe




```
coloring([X1,X2,X3,X4,X5]):-
  when((nonvar(X2),nonvar(X1)),X2\=X1),
  when((nonvar(X3),nonvar(X1)),X3\=X1),
  …,
  member(X1,[r,g,b,y]),
  member(X2,[r,g,b,y]),
  ….
```

```
coloring(L)
when((nonvar(X2),nonvar(X1)),X2\=X1),
when((nonvar(X3),nonvar(X1)),X3\=X1),…
member(X1,[r,g,b,y]),
member(X2,[r,g,b,y]),…
X1/r
when((nonvar(X2),nonvar(r)),X2\=r),
when((nonvar(X3),nonvar(r)),X3\=r),…
member(X2,[r,g,b,y]),…
X2/r
when((nonvar(r),nonvar(r)),r\=r),
when((nonvar(X3),nonvar(r)),X3\=r),…
member(X2,[r,g,b,y]),…
```

fallimento

Ancora su freeze / when

- E se Y non diventa ground?

risposta condizionale!

```
?- freeze(Y, X is Y+1).
Delayed goals: X is Y + 1
Yes
```

- *Non semplifica le equazioni, ma è già qualcosa*
- *A carico del programmatore*
- ***Cosa ci serve ?***
  - *Che il risolutore "addormenti" i goal che non è in grado di valutare per selezionarli non appena sono disponibili nuove informazioni*

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Dalla programmazione logica alla CLP

05 – Constraint Logic Programming (CLP)

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Anno accademico 2020/2021

Prof. MARCO GAVANELLI

Esempio di CSP

- Variabili: X, Y - Domini: da 1 a 4 - Vincoli: X>Y

```
csp(X,Y):-
  when((nonvar(X),nonvar(Y)), X>Y ),
  member(X,[1,2,3,4]),
  member(Y,[1,2,3,4]).
```

- Prima fa gli assegnamenti, poi verifica i vincoli
- *Sarebbe più efficiente se, quando si invoca X>Y, questo eliminasse già a priori* dai domini gli elementi che non soddisfano i vincoli: pruning

Come fare?

- Se vogliamo che Prolog faccia anche le semplificazioni, il pruning, bisogna che conosca il tipo della variabile:
  - alcune variabili non sono normali variabili logiche (a cui può essere assegnato qualunque termine, come `p(1,a,[])` o `[[1,2],[3]]`), ma numeriche
  - Il programmatore dovrà dichiarare qual è il dominio della variabile
  - A questo punto il sistema potrà associare ad ogni variabile il suo dominio e lavorare su di esso

Programmazione Logica a Vincoli

Constraint Logic Programming (CLP)

Constraint Logic Programming

- Nuovo paradigma di programmazione, estensione della Programmazione Logica [Jaffar e Lassez, 1987]
- è uno schema per produrre nuovi linguaggi
- durante la risoluzione SLD, alcuni atomi speciali, detti vincoli (constraints), non vengono risolti, ma vengono accumulati in un'area di memoria esterna (constraint store) ed elaborati da un risolutore esterno (constraint solver).
- Ci possono essere diversi risolutori, basati su tecnologie diverse. Ciascuno di questi dà luogo ad un linguaggio della classe CLP:
  - CLP(FD): sui domini finiti basato su consistency
  - CLP(R): sui reali basato sul simplesso
  - CLP(Bool): sui booleani basato su BDD
  - …

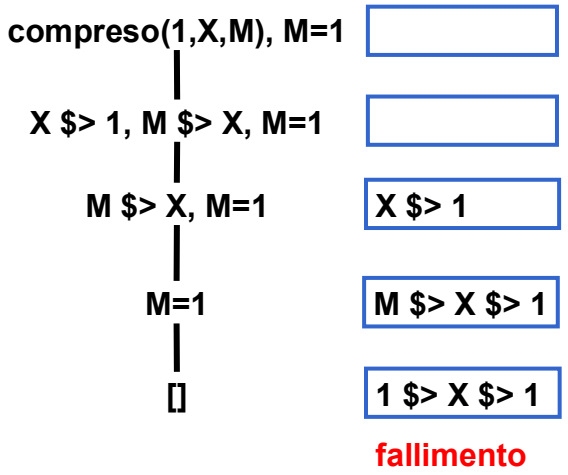
CLP: sintassi

- In CLP ci sono alcuni predicati, con una sintassi particolare, che non vengono risolti con la risoluzione SLD
- Ad esempio, supponiamo che un linguaggio CLP abbia il vincolo **$>**, che ha il significato di "maggiore"

```
compreso(Inf,X,Sup) :- X $> Inf,
  Sup $> X.
```


CLP: semantica operazionale

```
compreso(Inf,X,Sup) :- X $> Inf, Sup $> X.
```



Come fa il constraint solver?

- Da cosa si accorge il constraint solver che c'è un fallimento?
- Dipende dal tipo di solver
  - CLP(FD): ho un fallimento quando una variabile ha il dominio vuoto
  - CLP(R): ho fallimento quando l'algoritmo del simplesso rileva che non ci sono soluzioni

MACCHINA CLP: PASSI FONDAMENTALI

- ***Resolution r***
  - *selezione di un atomo dal risolvente*
  - *unificazione: aggiunta di vincoli al constraint store*
- ***Constraining c***
  - *selezione di un vincolo dal risolvente*
  - *aggiunta del vincolo al constraint store*
- ***Infer (propagazione) i***
  - *trasformazione del constraint store*
- ***Consistency s***
  - *verifica della soddisfacibilità del constraint store*

"metodi" che il risolutore deve esportare per poter essere usato in una macchina CLP

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Dalla programmazione logica alla CLP

06 – Constraint Logic Programming on Finite Domains: CLP(FD)

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione
Anno accademico 2020/2021

Prof. MARCO GAVANELLI

CLP(FD)

- ***CLP(FD): Constraint Logic Programming su domini finiti***
  - particolarmente adatta a modellare e risolvere problemi a vincoli
- *Modello del problema*
  - *Le VARIABILI rappresentano le entità del problema*
  - *Definite su DOMINI FINITI di oggetti arbitrari (normalmente interi)*
  - *Legate da VINCOLI (relazioni tra variabili)*
    - *matematici*
    - *simbolici*
  - *Nei problemi di ottimizzazione si ha una FUNZIONE OBIETTIVO*
- *Risoluzione*
  - *Algoritmi di propagazione (incompleti) incapsulati nei vincoli*
  - *Strategie di ricerca*

CLP(FD): Sintassi

- Esistono diversi linguaggi CLP(FD).
  - CHIP
  - SICStus
  - ECLiPSe
  - B-Prolog
  - SWI Prolog
  - …
- La sintassi è simile, ma non identica
- Ciascuno ha caratteristiche distintive

CLP(FD): Sintassi

In generale però si hanno

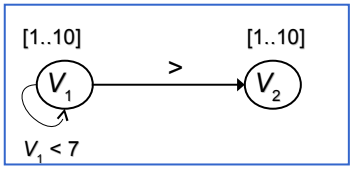
- Un predicato per definire il dominio delle variabili
  - ECLiPSe: `X :: [1..10,13..15].`
  - SICStus: `X in (1..10) \/ (13..15).`
- Una sintassi che contraddistingue i vincoli dagli altri predicati (ECLiPSe e SICStus usano il '#'):
  - `#>, #>=, #=<, #=, #\=`, … sono vincoli

CLP(FD): Semantica Operazionale

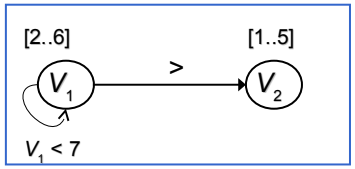
- Quando il letterale selezionato dalla risoluzione SLD è un vincolo, questo viene inserito nel constraint store
- A questo punto, si ha una fase di infer: in CLP(FD) questa fase è data da una propagazione, tipicamente Arc-Consistency (o varianti)
- Fase di Consistency: se uno dei domini risulta vuoto, si ha fallimento
- Alla fine viene fornito il constraint store come risposta, insieme al binding

CONSISTENCY

- NODE CONSISTENCY
  - una rete è *node consistent* se in ogni dominio di ogni nodo ogni valore soddisfa i vincoli unari che coinvolgono la variabile
- ARC CONSISTENCY
  - una rete è *arc consistent* se
    per ogni arco (vincolo binario) che connette due variabili V_i e V_j
    per ogni valore nel dominio di V_i
    esiste un valore nel dominio di V_j che soddisfa il vincolo
    e viceversa (da V_j a V_i)



Non Node consistent
Non Arc consistent



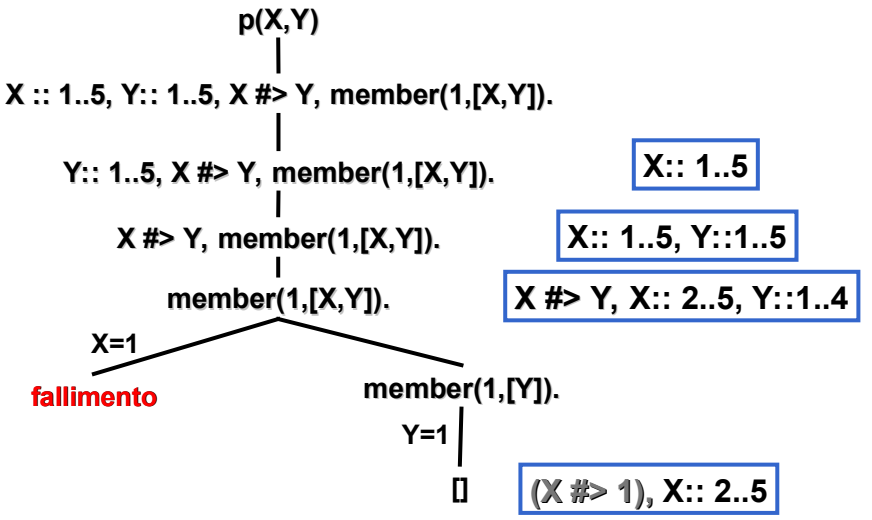
Node consistent
Arc consistent

Esempio:

```
p(X,Y):-
  X :: 1..5, Y :: 1..5,
  X #> Y, member(1,[X,Y]).
```


Esempio:

```
p(X,Y):- X::1..5, Y::1..5, X#>Y,
   member(1,[X,Y]).
member(X,[X|_]).
member(X,[_|T]):- member(X,T).
```



UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Dalla programmazione logica alla CLP

07 – ECLiPSe

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione
Anno accademico 2020/2021

Prof. MARCO GAVANELLI

ECLiPSe

- ECLiPSe è un linguaggio CLP che incorpora varie librerie, che forniscono dei risolutori specifici
  - fd sui domini finiti
  - fd_set sugli insiemi
  - eplex reali, vincoli lineari
  - …
- Inoltre, è possibile definire nuovi risolutori e vincoli
  - propia
  - CHR
  - …

ECLiPSe: CLP(FD)

- La libreria FD viene caricata col comando

  `use_module(library(fd))`

  oppure, per semplicità

  `lib(fd).`

  In alternativa, si può usare la nuova libreria `ic`

  `lib(ic).`
- A questo punto abbiamo a disposizione:
  - `::` operatore per definire il dominio di una variabile, es `A::[0,3,7,10], B::[0..15].`
    o di una lista di variabili: `[A,B,C] :: [1..10,13].`
  - Vincoli predefiniti:

    `#<, #>, #=, #\=, #<=, #>=`

Esempio

```
[eclipse 1]: lib(fd).
fd_domain.eco loaded traceable 0 bytes in 0.05 seconds
…
fd.eco     loaded traceable 0 bytes in 0.22 seconds

Yes (0.22s cpu)

[eclipse 2]: A::[0,3,7,10], B::[0..15], A#> B.

A = A{[3, 7, 10]}
B = B{[0..9]}

Delayed goals:
        A{[3, 7, 10]} - B{[0..9]}#>=1
Yes (0.00s cpu)
```

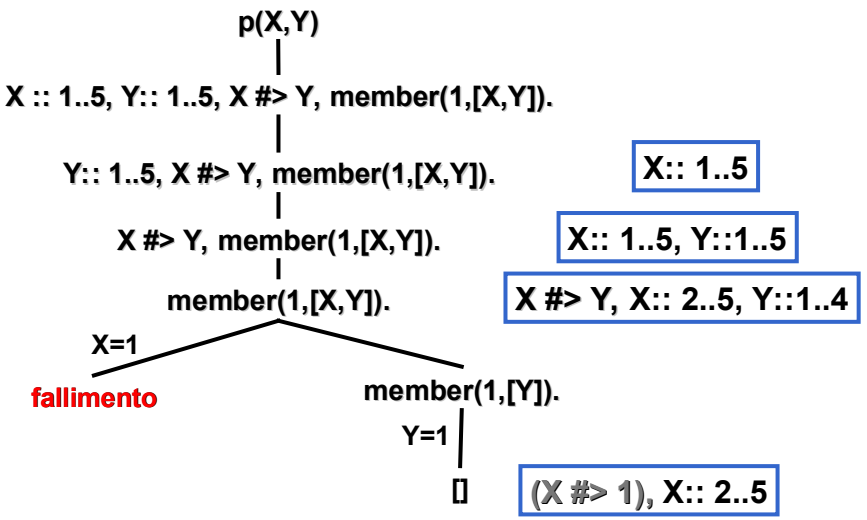
Carico la libreria

Domini arc-consistent

Vincoli da soddisfare

Esempio:

```
p(X,Y):- X::1..5, Y::1..5, X#>Y,
   member(1,[X,Y]).
member(X,[X|_]).
member(X,[_|T]):- member(X,T).
```



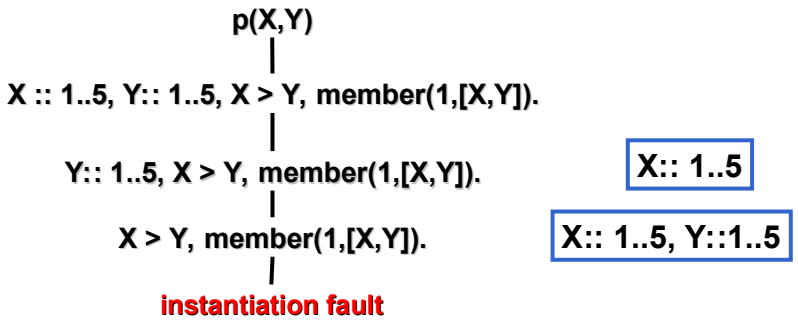
Esempio:

- Che cosa sarebbe successo se avessi scritto X>Y invece di X #> Y?

```
p(X,Y):-
  X :: 1..5, Y:: 1..5,
  X > Y, member(1,[X,Y]).
```


Esempio:

```
p(X,Y):- X::1..5, Y::1..5, X>Y,
   member(1,[X,Y]).
member(X,[X|_]).
member(X,[_|T]):- member(X,T).
```



- *Il predicato > è sempre il solito predicato di Prolog!!*
- *Quindi vuole avere entrambi gli argomenti istanziati*
- *Può essere usato solo come test, non è un vincolo!*

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Dalla programmazione logica alla CLP

08 – Labeling

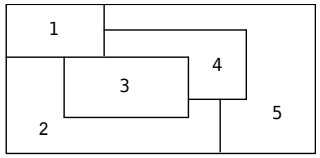
Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Anno accademico 2020/2021

Prof. MARCO GAVANELLI

ESEMPIO DI MODELLO DI UN PROBLEMA

- Map Colouring




```
:- lib(fd).
map_colouring([V1,V2,V3,V4,V5]):-
     V1::[red,green,yellow,blue],
     V2::[red,green,yellow,blue],
     V3::[red,green,yellow,blue],
     V4::[red,green,yellow,blue],
     V5::[red,green,yellow,blue],
     V1#\=V2, V1#\=V3, V1#\=V4, V1#\=V5, V2#\=V3,
     V2#\=V4, V2#\=V5, V3#\=V4, V4#\=V5,
     .....
```

variabili & domini

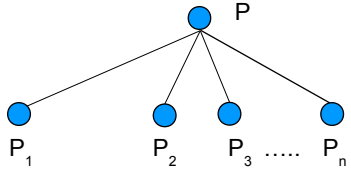
vincoli

Risolutori (in)completi

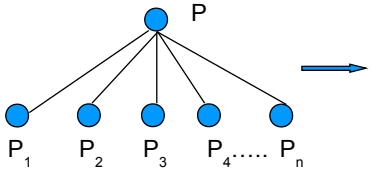
- Problema: l'Arc-Consistency da sola non è in grado di trovare una soluzione
- Il risolutore CLP(FD) è un solver incompleto:
  - Se dice 'no' non esiste soluzione
  - Se dice 'sì' potrebbe esistere o potrebbe non esistere
  - Fornisce comunque nella risposta calcolata i vincoli che devono essere soddisfatti
- Se vogliamo trovare una soluzione, dovremo fare una ricerca nello spazio degli stati

RICERCA

- Si effettua la propagazione dei vincoli
- Dopo la propagazione:
  - Tutte le variabili sono istanziate → Ho trovato una soluzione
  - Fallimento → Non c'è soluzione
  - I domini contengono alcuni valori → SEARCH
- Ricerca: idea
  - Divide il problema in sotto-problemi (BRANCHING) che possono essere risolti indipendentemente l'uno dall'altro
  - I sottoproblemi devono essere una partizione del problema originale
  - Ciascun sottoproblema viene risolto ricorsivamente



RICERCA

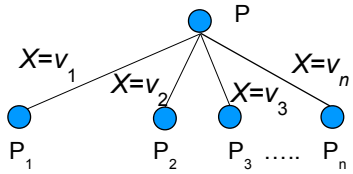


Strategia di Branching
GUESS

- Strategie di Branching definiscono il modo di partizionare il problema P in sottoproblemi più facili P_1, P_2, …, P_n.
- Spesso, per partizionare il problema, si aggiungono dei vincoli, fra loro mutuamente esclusivi
- **Per ogni sotto problema si applica di nuovo la propagazione**. Possono essere rimossi nuovi rami grazie alle nuove informazioni derivate dal branching

LABELING

- La tecnica più semplice di branching è detta *labeling*
- LABELING:
  - Seleziona una VARIABILE (ad es. X)
  - Si ha un sottoproblema per ciascun valore v_i nel dominio della variabile: il vincolo imposto è $X=v_i$.
- L'ordine in cui le variabili e i valori vengono scelti (la search strategy) non influenza la completezza dell'algoritmo ma ne influenza pesantemente l'efficienza.
- Attività di ricerca volta a trovare buone strategie.



Labeling in ECLiPSe

`indomain(X)`

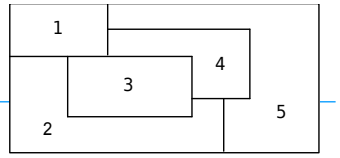
- assegna alla variabile X un valore nel dominio; in backtracking ne seleziona un altro
- Se si vuole applicare ad una lista di variabili, si può scrivere un predicato:

```
labeling([]).
labeling([H|T]):-
  indomain(H),
  labeling(T).
```

Il predicato **`labeling`** è predefinito in ECLiPSe con la definizione a fianco.

Ovviamente è una search strategy molto semplice, spesso inefficiente.

ESEMPIO COMPLETO




```
map_colouring([V1,V2,V3,V4,V5]):-
    V1::[red,green,yellow,blue],
    V2::[red,green,yellow,blue],
    V3::[red,green,yellow,blue],
    V4::[red,green,yellow,blue],
    V5::[red,green,yellow,blue],
    V1#\=V2, V1#\=V3, V1#\=V4, V1#\=V5, V2#\=V3,
    V2#\=V4, V2#\=V5, V3#\=V4, V4#\=V5,
    labeling([V1,V2,V3,V4,V5]).
```

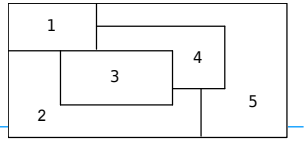
Variabili & domini

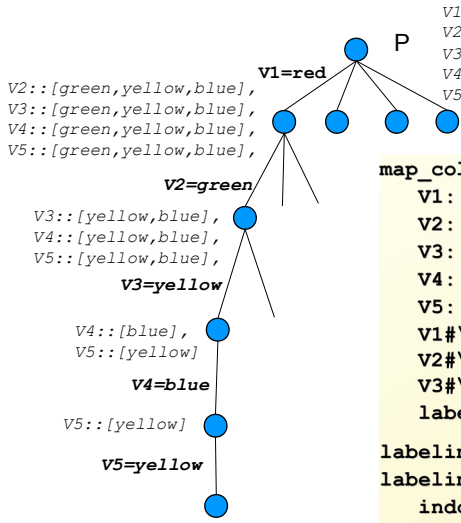
vincoli

ricerca

- Separazione fra
  - Modello del problema
  - strategia per risolverlo

SPAZIO DI RICERCA






```
map_colouring([V1,V2,V3,V4,V5]):-
   V1::[red,green,yellow,blue],
   V2::[red,green,yellow,blue],
   V3::[red,green,yellow,blue],
   V4::[red,green,yellow,blue],
   V5::[red,green,yellow,blue],
   V1#\=V2, V1#\=V3, V1#\=V4, V1#\=V5,
   V2#\=V3, V2#\=V4, V2#\=V5,
   V3#\=V4, V4#\=V5,
   labeling([V1,V2,V3,V4,V5]).

labeling([]).
labeling([H|T]):-
   indomain(H),
   labeling(T).
```

Esercizio

- Il signor Gedeone, parlando dei suoi nipoti ad un amico che gli chiede la loro età, risponde in maniera sibillina:

"Fra 11 anni, Dario avrà l'età che avevo io quando lui era 6 volte più giovane di me.

Umberto, invece, ha 3 anni più di Dario e 3 anni meno della differenza d'età che c'è tra me e Dario"

- Quanti anni ha il signor Gedeone e quanti ne hanno i suoi nipoti?

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Dalla programmazione logica alla CLP

09 – Algoritmo AC3

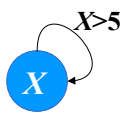
Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione
Anno accademico 2020/2021
Prof. MARCO GAVANELLI

VINCOLI UNARI E BINARI

Interpretazione dei vincoli come grafo

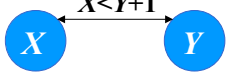
- Node Consistency: Un vincolo $c(X)$ è *node consistent* se

  $\forall d \in dom(X)$, $c(d)$ è vero (soddisfatto)



- Arc Consistency (AC): Un vincolo $c(X,Y)$ è *arc consistent* se

  $\forall d \in dom(X)$ $\exists g \in dom(Y)$ t.c. $c(d,g)$ è vero (soddisfatto) e viceversa



Arc-consistency, graficamente

dom(A) \ dom(B)	1	2	3	4	5	6	7	8	9	10
1	1	0	0	1	1	1	0	0	1	0
2	0	0	0	0	1	1	0	1	0	0
3	0	0	0	0	0	0	0	1	1	1
4	0	0	1	0	1	0	1	0	1	1
5	0	0	0	1	1	1	0	0	0	0
6	0	0	0	1	0	0	0	0	0	0
7	0	1	1	0	0	0	0	1	1	1
8	0	1	1	1	1	0	0	0	0	0
9	0	0	1	1	1	0	0	0	0	0
10	0	0	1	0	0	0	0	0	1	0

c(A,B)

Algoritmi per ottenere AC

- Vari algoritmi sono stati proposti per rendere una rete AC: (AC1) AC2, AC3, AC4, … AC7, AC2000, AC 2001, …
- Ogni algoritmo usa una lista in cui si ricorda "che cosa deve ancora valutare"
- AC3 è uno dei più usati, perché è semplice ed utilizza una lista di vincoli

AC3 (Mackworth)

La (List of active constraints) = lista di tutti i vincoli;
Ls (List of sleeping constraints) = ∅;
while *La* ≠ ∅ do
  prendi un vincolo *c*(*X*,*Y*) ∈*La* e toglilo da *La*
  se ci sono elementi non supportati in *dom*(*X*)
    allora eliminali (se *dom*(*X*) = ∅, fallisci)
  metti in *La* tutti i vincoli in *Ls* che coinvolgono *X*
  (stesso ragionamento per *Y*)
  se *c*(*X*,*Y*) non è completamente risolto
    allora mettilo in *Ls*

Esempio

- A :: 1,2,3,4,5
- B :: 1,3,4,5
- C :: 1,2,3,4
- D :: 3,4,5
- A < B
- B ≠ C
- C = 2A
- B + D < 8

Vincolo Completamente Risolto

- Un vincolo è completamente risolto (o entailed dai domini) se per ogni possibile assegnamento (dei valori presi dai domini) esso è vero
- Es `X:: 1..4, Y::6..10, X#<Y`
- Sicuramente se tutte le variabili che coinvolge sono istanziate, il vincolo è risolto (oppure è falso)

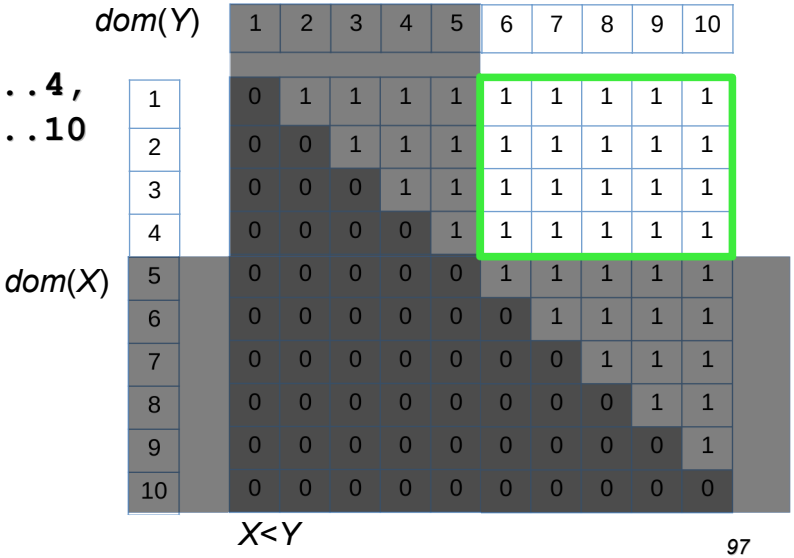
c(*A*,*B*), *A*=1, *B*=5

Domanda: è la stessa cosa dell'Arc-Consistency?

Domanda: e se solo una delle 2 variabili è istanziata, posso dire che è risolto?

Vincolo completamente risolto, graficamente

- `X::1..4,`
  `Y::6..10`
  `X#<Y`



UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Dalla programmazione logica alla CLP

10 – Bound Consistency

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione
Anno accademico 2020/2021

Prof. MARCO GAVANELLI

Complessità

- A volte l'Arc-Consistency è troppo costosa
- Quando risveglio un vincolo $c(X,Y)$, per ogni valore in $dom(X)$ cerco un valore compatibile in $dom(Y)$

  $O(d^2)$ confronti

  ogni volta che risveglio (se d è la cardinalità dei domini)!
- Mi accorgo di un fallimento quando un dominio è vuoto.
- Potrei fermarmi quando ho trovato un valore compatibile
- Ragionare per intervalli, invece di verificare tutti gli elementi del dominio

Esempio

- `A::[1,2,3,4,5,6,7,8,9,10,11,12,13,14]`
- `B::[0,1,2,3,4,5,6,7,8,9,10,11,12,13]`
- `A #< B`
- In questo caso, è inutile controllare tutti i singoli elementi nei domini: è sufficiente ragionare sugli estremi
- Non ho bisogno di tenere una lista di elementi del dominio, ma bastano gli estremi
- Risveglio il vincolo solo se elimino gli estremi del dominio di A o B

Range e Bound Consistency

- Un vincolo $c(X,Y)$ è range consistent se

  $\forall d \in \{min(X),max(X)\}\ \exists g \in dom(Y)$ t.c. $c(d,g)$ è vero (soddisfatto)

  e viceversa ($\forall d \in \{min(Y),max(Y)\}\ \exists g \in dom(X)$ t.c. $c(d,g)$ è vero)

- Un vincolo $c(X,Y)$ è bound consistent se

$\forall d \in \{min(X),max(X)\}\ \exists g \in [min(Y),max(Y)]$ t.c. $c(d,g)$ è vero e viceversa

($\forall d \in \{min(Y),max(Y)\}\ \exists g \in [min(X),max(X)]$ t.c. $c(d,g)$ è vero)

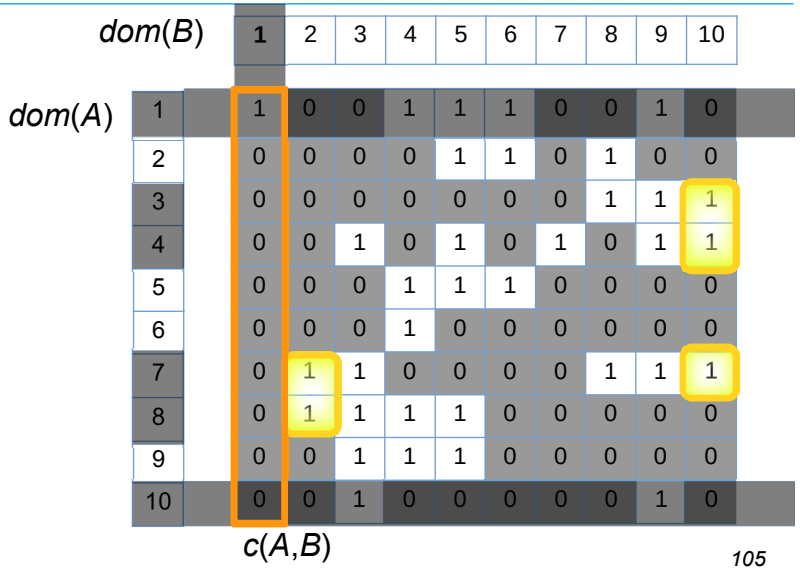
✓ Comodo per vincoli come <, >, …

✓ Non ho bisogno di tenere una lista di elementi del dominio, ma bastano gli estremi

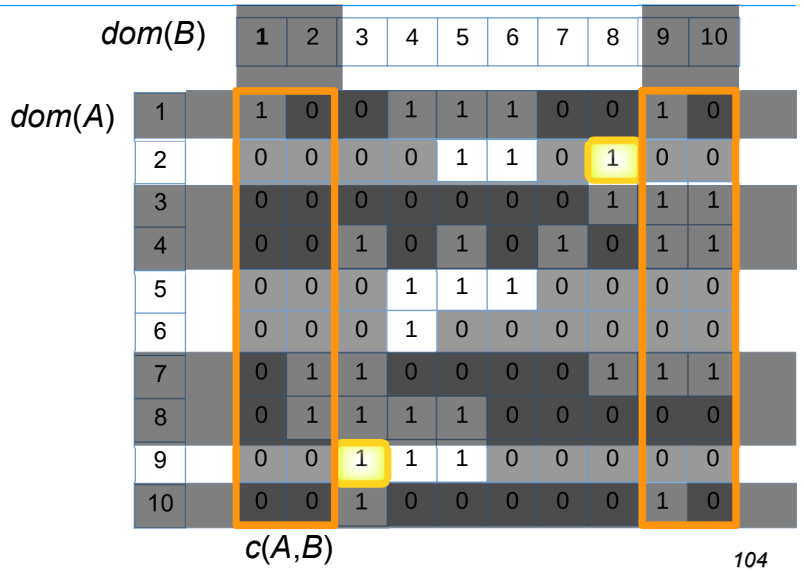
✓ Risveglio un vincolo $c(X,Y)$ solo se elimino gli estremi del dominio di X o Y

✗ Faccio meno pruning

Bound-consistency, graficamente



Range-consistency, graficamente



Esercizi

- Si scriva un predicato CLP(FD) che impone che una lista di variabili FD, date in ingresso, sia ordinata in senso decrescente stretto.
- Esempio:

```
?- [A,B,C]::0..10,  ordina([A,B,C]).
  yes, A:: 2..10
  B:: 1..9
  C:: 0..8
```

- Si scriva un predicato CLP(FD) che impone che i valori di una lista di variabili FD siano tutti diversi fra loro

UNIVERSITÀ DEGLI STUDI DI FERRARA
- EX LABORE FRUCTUS -

Dalla programmazione logica alla CLP

11 – Generalized Arc Consistency

Intelligenza Artificiale per l'Ottimizzazione Vincolata
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione
Anno accademico 2020/2021

Prof. MARCO GAVANELLI

VINCOLI N-ari

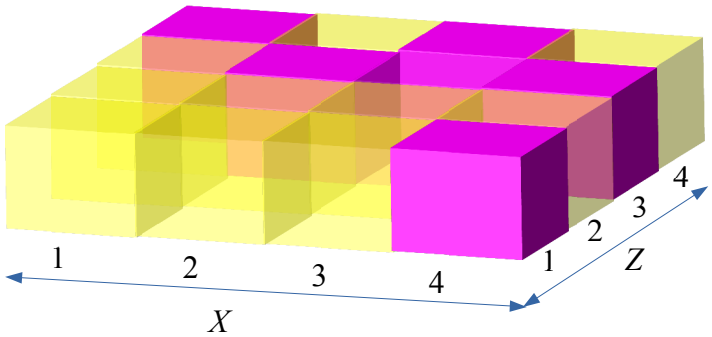
- Per vincoli *n*-ari, non c'è più l'interpretazione come grafo.
  Es: $X+Y=Z$
- *Generalized Arc Consistency (GAC)*
  (o Hyper Arc-Consistency o Domain-Consistency):
  Un vincolo $c(X_1,X_2,\ldots,X_n)$ è *arc consistent in senso generalizzato* se
  - Per ogni variabile X_i (i=1..n), per ogni valore $g \in dom(X_i)$
  - Esiste un assegnamento alle rimanenti n-1 variabili
    $$X_1 \to v_1, \ldots, X_{i-1} \to v_{i-1}, X_{i+1} \to v_{i+1}, \ldots, X_n \to v_n$$
  - *tale che* $c(v_1,\ldots,v_{i-1},g,v_{i+1},v_n)$ è vero (soddisfatto).

Domanda: Sapete definire Generalized Bound Consistency?

GAC: Graficamente

FALSO

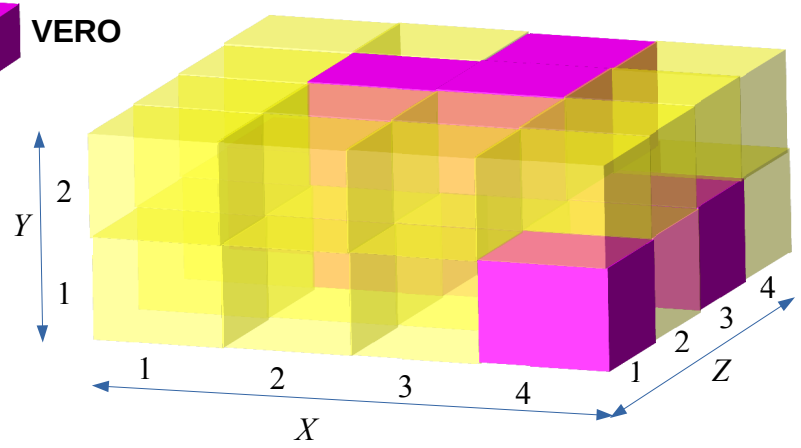
VERO



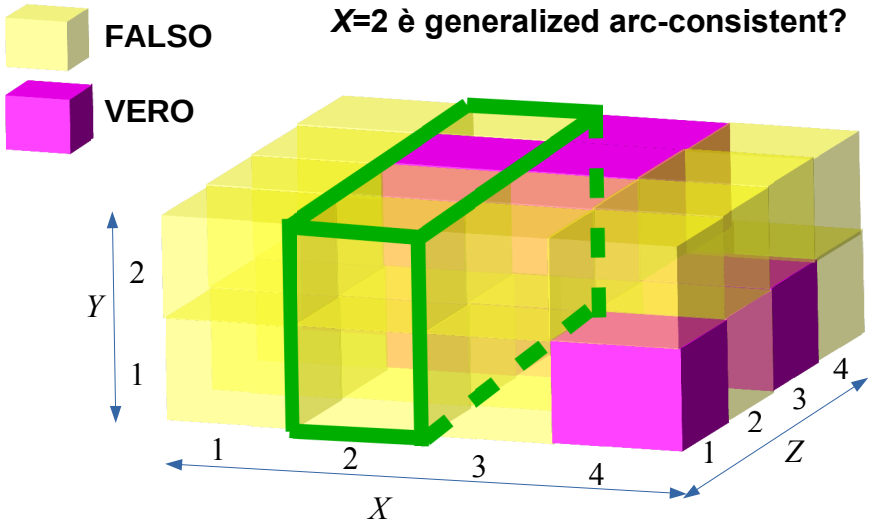
GAC: Graficamente

FALSO

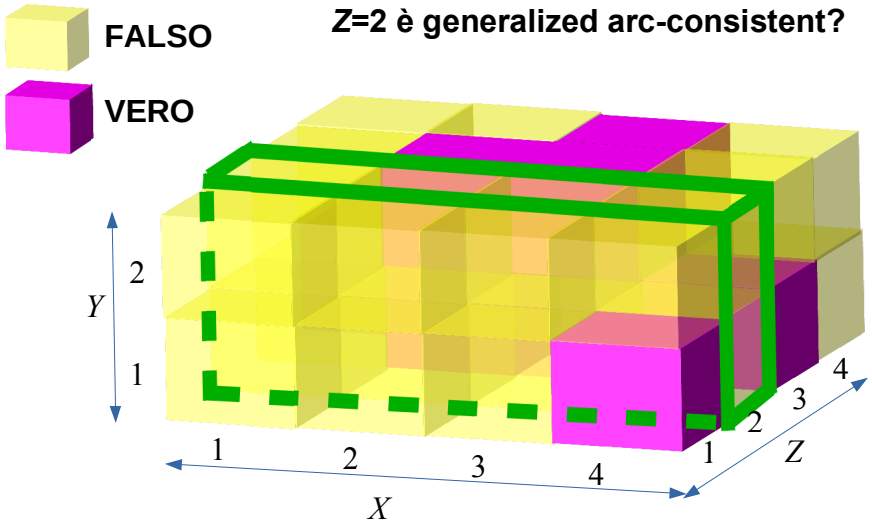
VERO



GAC: Graficamente



GAC: Graficamente



Vincoli N-ari: espressioni

- Si possono definire vincoli come

  `somma(A,B,C)`

  vero se *A*+*B*=*C*, con propagazione GAC.
- Questi vincoli sono già definiti, con zucchero sintattico. Possiamo usare direttamente:

  `A+B #= C`

  `A#< B*C+D,`

  ...

Esercizio

- Si consideri il seguente CSP:

  `A::[-2..3], B :: [-1..4],`
  `P::[2..7], A*B #= P.`
- Si mostri la propagazione nei casi Generalized Arc-Consistency e Generalized Bound-Consistency.

Esercizi

- Si scriva un predicato CLP(FD) che impone che una variabile S sia la somma di una lista di variabili con dominio
- Esempio:

```
?- [A,B,C]::0..10, S::0..100,
  sommalista([A,B,C],S).
```