La Programmazione con Vincoli nel contesto del corso IA II

Che cos'è?

- Per molti problemi (ad esempio problemi di ottimizzazione) l'obiettivo è quello di potere modellare le relazioni esistenti tra gli oggetti e trovare gli oggetti che le soddisfano

- Un vincolo rappresenta una relazione tra un insieme di oggetti, implicitamente definiti dal vincolo nel quale compaiono

  Esempio: F = 1.8 * C + 32

  dove C ed F sono rispettivamente le temperature Celsius e Fahrenheit

  Il vincolo definisce la relazione esistente tra F e C (*definizione implicita degli oggetti*)

- Più in generale:
  - **Constraint primitivo** (vincolo primitivo): formula atomica di una teoria logica decidibile
  - **Constraint** (vincolo): congiunzione di vincoli primitivi

- La programmazione con vincoli introduce il concetto di vincolo all'interno di linguaggi di programmazione

Che cos'è?

- *Pregio principale*: dichiaratività Si specificano le proprietà degli oggetti del discorso senza specificare direttamente gli stessi
- La programmazione con vincoli rappresenta un settore multidisciplinare. Combina tecniche:
  - matematiche
  - informatiche tradizionali
  - tecniche di AI
  - ricerca operativa
  - calcolo numerico
  - ...

  e più recentemente tecniche tipiche delle basi di dati
- Nel contesto di questi seminari analizzeremo due aspetti relativi alla programmazione con vincoli:
  - *tecniche di propagazione dei vincoli*
    $\Rightarrow$ sviluppate principalmente nel contesto AI, relativamente al contesto CSP (constraint satisfaction problems)
  - *applicazioni dei vincoli nel contesto basi di dati*
    $\Rightarrow$ direzione di ricerca molto recente

La Propagazione dei Vincoli in CLP

Sommario

Introduzione al CLP

- Il vantaggio principale della programmazione logica risiede nella **dichiaratività**
- La programmazione logica presenta due problemi di base:
  - gli oggetti manipolati non sono interpretati e l'uguaglianza coincide con l'uguaglianza sintattica
  - le computazioni corrispondono ad una visita depth-first dell'albero di derivazione (procedura di generate and test)

    $\Rightarrow$ problemi di efficienza
- **Constraint Logic Programming** (CLP) cerca di risolvere i problemi precedenti introducendo constraint (vincoli) nel contesto delle regole logiche
  - **Constraint primitivo** (vincolo primitivo): formula atomica di una teoria logica decidibile
  - **Constraint** (vincolo): congiunzione di vincoli primitivi
- CLP permette di superare i problemi sopra delineati:
  - introduce oggetti interpretati nelle regole logiche
  - rimpiazza l'unificazione con l'operazione di constraint solving
  - usa i vincoli **passivamente** (verifica soddisfacibilià) e **attivamente** (generazione nuovi valori e relativo cut dell'albero di derivazione)

CLP: sintassi

Dominio dei vincoli

Ogni dominio è caratterizzato da:

- una segnatura Σ_C
- una struttura su Σ_C che rappresenta l'interpretazione intesa dei vincoli
- una teoria su Σ_C che rappresenta le proprietà del dominio
- una funzione, chiamata *solver* che mappa ogni elemento di $\mathcal{L}_\mathcal{C}$ in *true*, *false* or *unknown*, in relazione alla sua soddisfacibilità

Programma CLP

Programma logico in cui le regole possono contenere vincoli espressi secondo la teoria prescelta

Regola CLP

$$H \leftarrow c_1, ..., c_n \diamond B_1, ..., B_n$$

nella regola precedente c_i, $1 \leq i \leq n$ rappresenta un vincolo primitivo su un certo dominio C

CLP: sintassi

Esempio

Dominio dei numeri reali:

- La segnatura è composta da $\leq$,$\geq$,$<$, $>$, $=$ come simboli di predicati, le quattro operazioni +,-,*,/ come simboli di funzione, e sequenze di cifre intere e decimali come simboli di costante

- L'interpretazione intesa è data da $\mathbb{R}$ con le tipiche operazioni sui reali

- La teoria dei campi reali chiusi è una teoria per $\mathbb{R}$

- Un solver per $\mathbb{R}$ è stato implementato nel sistema $CLP(\mathbb{R})$

- Esempi di vincoli primitivi:
  $-23X * 2Y \geq Z$
  $X - 10Y \leq 2$

CLP: sintassi

Esempio

Data la definizione di un pasto come composto da un antipasto, un piatto principale e un dessert ed un database di cibi, contenente informazioni sul loro valore calorico, il seguente programma CLP determina i pranzi leggeri come quelli la cui somma delle calorie non eccede 10

```
pranziLeggeri(A,M,D) ← I > 0,J > 0,K < 0,I + J + K <= 10 ◇
   antipasti(A,I),piattiPrincipali(M.J), dessert(D,K)

piattiPrincipali(M,I) ← carne(M,I).
piattiPrincipali(M,I) ← pesce(M,I).

antipasti(insalata,1).
antipasti(formaggi,6).

carne(manzo,5).
carne(maiale,7).

pesce(sogliola,2).
pesce(tonno,4).

dessert(frutta,2).
dessert(gelato,6).
```

CLP: semantica

In analogia con la programmazione logica, sono state proposte tre semantiche per il CLP:

- top-down
- fixpoint
- model-theoretic

Senza entrare nei dettagli:

- i risultati intermedi di una computazione CLP sono composti da un *constraint store* e dai goal rimanenti che devono essere risolti
- Goal generico:

  $$\leftarrow c_1, ..., c_n \diamond B_1, ..., B_m$$

  $c_1, ..., c_n$ constraint store
  $B_1, ..., B_m$ vincoli rimanenti

- in ogni stato, il constraint store deve essere **consistente**, cioè deve ammettere almeno una soluzione

  $\Rightarrow$ viene utilizzato il **constraint solver** corrispondente al dominio utilizzato

CLP: semantica

Esempio

Si consideri il goal

$\leftarrow \diamond$ `pranziLeggeri(A,M,D)`

Si ottiene

```
← I > 0, J > 0, K < 0, I + J + K <= 10 ◇
  antipasti(A,I), piattiPrincipali(M,J), dessert(D,K)
```

- La computazione continua riducendo tutti gli atomi presenti nel goal
- ogni alternativa = una derivazione
- la derivazione termina quando non ci sono più atomi da ridurre e il constraint store è consistente

CLP: semantica

Esempio di computazione consistente

```
← A = insalata, I = 1,1 + J + K <= 10,1 > 0,J < 0,K > 0 ◇
  piattiPrincipali(M,J), dessert(D,K)
```

```
← A = insalata,
  I = 1,M = M1,J = I1,1 + J + K <= 10,1 > 0,J < 0,K > 0 ◇
  piattiPrincipali(M1,I1), dessert(D,K)
```

```
← A = insalata, M = manzo, J = 5, M1 = manzo,
  I1 = 5,1 + 5 + K <= 10,1 > 0, 5 < 0,K > 0 ◇
  piattiPrincipali(M,J), dessert(D,K)
```

```
← A = insalata, M = manzo, J = 5, M1 = manzo, D=frutta,
   I1 = 5, , K = 2, 1 + 5 + 2 <= 10, 1 > 0, 5 < 0, 2 > 0 ◇
```

Esempio di computazione inconsistente

```
← A = formaggi,
  I = 6, 6 + J + K <= 10, 6 > 0, J < 0, K > 0 ◇
  piatti_principali(M,J), dessert(D,K)
```

```
← A = formaggi,
  I = 6,M = M1,J = I1,6 + J + K <= 10,6 > 0,J < 0,K > 0, ◇
  piatti_principali(M1,I1), dessert(D,K)
```

```
← A = formaggi, M = manzo, J = 5, M1 = manzo,
  I1 = 5, 6 + 5 + K <= 10, 5 > 0, 6 < 0, K > 0, ◇
  dessert(D,K)
```

Il Constraint Solver

- L'esempio precedente mette in evidenza che il meccanismo di valutazione di base del CLP si preoccupa solo di verificare la consistenza di un insieme di vincoli (deve esistere almeno una soluzione)

- Quindi i vincoli sono utilizzati **passivamente**, non generano valori ma devono essere soddisfatti dai valori generati

- Le soluzioni dell'esempio precedente sono definite, cioè in ogni soluzione ad ogni variabile presente nel goal viene assegnato un valore

- In altri casi, le soluzioni possono essere indefinite, cioè alcune variabili possono essere definite da vincoli non di uguaglianza

  In questo caso l'insieme dei valori assegnati a questa variabile è indefinito e, in dipendenza del dominio, può anche essere infinito

  `:= X+Y >= 5` ◇

Il Constraint Solver

- Non sempre è possibile verificare la soddisfacibilità di un vincolo

- I constraint solver che possono restituire il valore *unknown* sono detti *incompleti*

- I solver incompleti vengono utilizzati quando un solver completo non esiste oppure esiste ma la complessità non è trattabile

- **Esempio di solver completo**: algoritmo di Gauss-Jordan per i vincoli lineari

- **Esempio di solver incompleto**: non esiste un constraint solver completo per la teoria dei vincoli non lineari sugli interi

- **Esempio di solver completo non trattabile**: vincoli non lineari sui reali (altamente esponenziale)

Un Constraint Solver completo per vincoli lineari reali

Algoritmo di Gauss-Jordan

- Data una congiunzione di vincoli lineari reali, si sceglie un vincolo primitivo v
- si sceglie una variabile x contenuta in v e si riscrive il vincolo nella forma $x = e$, dove e non contiene x
- si sostituisce e ad x nel vincolo e si aggiunge $x = e$ al vincolo
- si ripete fino a che si arriva alla forma risolta:

  $x_1 = e_1 \wedge ... \wedge x_n = e_n$

  dove $x_1, ..., x_n$ sono le variabili di interesse
- la soddisfacibilità di un vincolo di questo tipo è immediata

Esempio

$1 + X = 2Y + Z$
$Z - X = 3$

$X = 2Y + Z - 1$
$Z - 2Y - Z + 1 = 3$

$X = Z - 3$
$Y = -1$

$\Rightarrow$ *il vincolo è soddisfacibile*

Un Constraint Solver completo per domini finiti

- Si supponga di sapere che i valori ammessi per ogni variabile in un vincolo C appartengono ad un insieme finito D tale che $D(X)$ è l'insieme di valori ammessi per la variabile X

  $(C, D) \Rightarrow$ *constraint satisfaction problem (CSP)*

- in questi casi esiste un algoritmo molto semplice per verificare la soddisfacibilità di un vincolo C:
  - si sceglie una variabile x contenuta in C
  - per ogni valore contenuto nel dominio associato ad x:
    * si rimpiazza x nel vincolo con il valore prescelto; sia C_1 il vincolo risultante
    * se C_1 è soddisfacibile allora C è soddisfacibile
    * altrimenti, si analizzano gli altri valori
    * se per nessun valore di x, C_1 è soddisfacibile, C non è soddisfacibile
- $\Rightarrow$ *algoritmo di backtracking*

Un Constraint Solver per domini finiti

Esempio

Un Constraint Solver per domini finiti

Esempio di problema con dominio finito

Si intende colorare le differenti regioni all'interno di una mappa utilizzando un numero limitato di colori in modo tale che due regioni adiacenti non abbiano lo stesso colore

Il problema si può modellare supponendo di avere una variabile per ogni regione e supponendo che i valori delle variabili siano i colori disponibili

Se le regioni sono cinque: A, B, C, D, E ed i colori sono tre: red, $yellow$ e $green$, il seguente vincolo esplicita il problema:

$$
\begin{array}{l}
A \neq B \wedge A \neq C \wedge A \neq D \\
\wedge A \neq E \wedge B \neq C \wedge B \neq D \wedge B \neq E \\
\wedge C \neq D \wedge C \neq E \wedge D \neq E \wedge
\end{array}
$$

$$
\begin{array}{l}
(A = red \vee A = yellow \vee A = blue) \wedge \\
(B = red \vee B = yellow \vee B = blue) \wedge \\
(C = red \vee C = yellow \vee C = blue) \wedge \\
(D = red \vee D = yellow \vee D = blue) \wedge \\
(E = red \vee E = yellow \vee E = blue)
\end{array}
$$

La propagazione dei vincoli

- Il constraint solver utilizza i vincoli in modo passivo
- I vincoli possono però anche essere utilizzati attivamente, cioè per generare valori
- Questo approccio permette di ridurre l'inefficienza dell'approccio generate and test, tipico della programmazione logica
- L'operazione con la quale i vincoli vengono utilizzati attivamente è chiamata *propagazione*
- Le tecniche di propagazione sono state principalmente definite in AI per domini finiti
- La propagazione può essere utilizzata in due modi:
  - per definire solver, spesso incompleti ma efficienti (polinomiali nel numero delle variabili e dei vincoli primitivi)
  - per rendere più efficienti solver completi, riducendo il numero di derivazioni da analizzare

La propagazione dei vincoli

Sono state proposte molteplici tecniche di propagazione di vincoli:

- *Tecniche di propagazione locale.* Sono applicabili a tutti i domini ma hanno efficacia limitata

- *Tecniche di propagazione per domini finiti.* Sono applicabili quando il dominio dei vincoli è un insieme finito e noto a priori

- *Tecniche di propagazione generalizzata.* Non propagano valori ma ulteriori vincoli
  $\Rightarrow$ non li vedremo nel contesto del seminario

Propagazione locale

- **Idea**: determinare le variabili il cui valore è fissato dal vincolo (*variabili determinate dal vincolo*) ed eliminare tali variabili dal vincolo stesso, sotituendole con il valore corrispondente

- Sia e un'espressione che non contiene variabili. Una variabile x è *determinata* da un vincolo C ad avere valore e se ogni soluzione di C è anche una soluzione di $x = e$

- **Esempio**: Sia $C \equiv X = Y + 2Z - 2 \wedge Y = 1 - 2Z$.
  X è determinata da C ad avere valore -1

- In base a questo approccio, i vincoli sono visti come un mezzo di propagazione data-flow dei valori delle variabili

Propagazione locale

Esempio

$V = V1 \wedge V = V2 \wedge V1 = I1 * R1 \wedge V2 = I2 * R2 \wedge I = I1 + I2$

può essere rappresentato come:

Si consideri il vincolo addizionale: $V = 10 \wedge R1 = 5 \wedge R2 = 10$
Il vincolo può essere propagato come segue:

La propagazione genera il vincolo:

$V = 10, R1 = 5, R2 = 10, V1 = 10, I1 = 2, V2 = 10, I2 = 1, I = 3$

Questo significa che il vincolo di partenza, congiunto con il nuovo vincolo, è soddisfacibile

$\Rightarrow$ *propagazione verifica anche soddisfacibilità*

Propagazione locale

Esempio

Si consideri il vincolo addizionale:

$I1 = 2 \wedge V1 = 10$

Il vincolo può essere propagato come segue:

La propagazione non assegna un valore ad ogni variabile, quindi non si può sapere se il vincolo di partenza congiunto con $I1 = 2 \wedge V1 = 10$ è soddisfacibile oppure no

$\Rightarrow$ la propagazione non permette di ottenere un solver completo ma può comunque essere utilizzata per rendere più efficiente il solver (riduce il dominio delle variabili)

Propagazione locale

- La propagazione su un vincolo C manipola sempre due oggetti:
  - un insieme di variabili determinate S, rappresentate come $S \equiv x_1 = e_1 \wedge x_2 = e_2 \wedge \ldots \wedge x_n = e_n$
    x_i variabili distinte e_i espressioni senza variabili
  - un vincolo C

- La propagazione locale agisce nel modo seguente:
  - sceglie un *vincolo primitivo* c in C ed inizializza S a vero
  - se c è insoddisfacibile, C è insoddisfacibile
  - se c è soddisfacibile e non contiene variabili lo elimino da C
  - altrimenti
    * si determinano tutte le variabili il cui valore è determinato da c
    * si aggiorna S
    * si rimpiazzano queste variabili con il valore corrispondente in C
  - si procede fino a che C ed S non cambiano
  - se al termine C corrisponde alla congiunzione vuota
    $\Rightarrow$ il vincolo di partenza è soddisfacibile

Propagazione locale

Vantaggi e svantaggi

- L'algoritmo è facile da implementare
- I solver basati sulla propagazione locale in genere non sono completi in quanto generano le variabili determinate considerando un vincolo primitivo alla volta
- **Esempio**:

  $X = Y + Z \wedge X = T - Y \wedge T = 5 \wedge Z = 7$

  L'algoritmo determina $T = 5$ e $Z = 7$. Infatti:

  da $T = 5 \wedge Z = 7 \Rightarrow X = Y + 7 \wedge X = 5 - Y$

  Risolvendo il sistema precedente:

  $X = 6, Y = 6$

  ma questa soluzione non è determinata dalla propagazione locale
- Considerano solo vincoli di uguaglianza

  Per vincolo di disuguaglianza c, l'insieme delle variabili determinate da c è spesso l'insieme vuoto
- A causa delle limitazioni precedenti, viene spesso associata ad un solver completo per renderlo più efficiente

Propagazione per domini finiti

- Sono alla base di solver con complessità polinomiale ma incompleti per vincoli su domini finiti
- **Idea di base**: se l'insieme dei valori associati ad una certa variabile è vuoto, allora il vincolo non è soddisfacibile

  Quindi:

  – Partono dal vincolo e dal dominio di partenza
  – Considerano un vincolo primitivo alla volta e usando informazioni relative al dominio di ciascuna variabile eliminano valori dai domini delle altre variabili
  – Nel momento in cui il dominio associato ad una variabile diventa vuoto
    $\Rightarrow$ vincolo insoddisfacibile
- Possono essere combinati con tecniche di backtracking:
  – gli algoritmi vengono usati per ridurre i domini di partenza
  – i domini così calcolati vengono utilizzati per restringere il backtracking (vengono tagliati tutti i rami corrispondenti a valori non propagati)
- Le tecniche proposte si differenziano per:
  – il tipo di vincoli utilizzato per la riduzione dei domini
  – la conseguente riduzione applicata

Propagazione per domini finiti

Alcuni tipi di propagazione

Node consistency
Un vincolo primitivo c è *node consistent* con dominio D se una delle seguenti condizioni è vera:

- $|vars(c)| \neq 1$
- $vars(c) = \{x\}$ e per ogni $d \in D(x)$, $x = d$ è una soluzione per c

Un vincolo $c_1 \wedge ... \wedge c_n$ è *node consistent* con dominio D se ogni vincolo primitivo c_i è node consistent con D, $1 \leq i \leq n$

$\Rightarrow$ *considera solo vincoli con una variabile*

Arc consistency
Un vincolo primitivo c è *arc consistent* con dominio D se una delle seguenti condizioni è vera:

- $|vars(c)| \neq 2$
- $vars(c) = \{x, y\}$ e per ogni $d_x \in D(x)$, esiste $d_y \in D(y)$ tale che $x = d_x \wedge y = d_y$ è una soluzione per c, e viceversa

Un vincolo $c_1 \wedge ... \wedge c_n$ è *arc consistent* con dominio D se ogni vincolo primitivo c_i è arc consistent con D, $1 \leq i \leq n$

$\Rightarrow$ *considera solo vincoli con due variabili*

Propagazione per domini finiti

- Il problema della colorazione di una mappa è un classico problema node e arc consistent
  - è *node consistent* perchè tutti i vincoli primitivi contengono due variabili
  - è *arc consistent* perchè per ogni vincolo primitivo, è sempre possibile fissare un valore per una variabile e trovare un valore per l'altras in modo che il vincolo sia soddisfacibile
  - Se il dominio dei colori è composto solo da *due* elementi, ad esempio rosso e giallo, il problema diventa insoddisfacibile poichè non è possibile colorare due regioni adiacenti con soli due colori

    Rimane tuttavia arc consistent

- Quindi:

arc consistent $\Rightarrow$ soddisfacibile

soddisfacibile $\not\Rightarrow$ arc consistent

Propagazione per domini finiti

È sempre possibile trasformare un problema a dominio finito in un altro problema che sia node o arc consistent

Node consistency
Input un vincolo $C = c_1 \wedge ... \wedge c_n$
il dominio D
Output un nuovo dominio D per C
for $i := 1$ to n **do**
if $|vars(c)| = 1$ **then**
let $\{x\} = vars(c)$
$D_1 := \{d \in D(X) | \{x = d\}$ è una soluzione per $c\}$
$D = D_1$
endif
endfor
return D

Arc consistency
Input un vincolo $C = c_1 \wedge ... \wedge c_n$
il dominio D
Output un nuovo dominio D per C
repeat
$W := D$
for $i := 1$ to n **do**
if $|vars(c)| = 2$ **then**
let $\{x, y\} = vars(c)$
$D_1(x) := \{d_x \in D(x)|$ esiste $d_y \in D(y)$ tale che $\{x = d_x, y = d_y\}$ è una soluzione per $c\}$
$D_1(y) := \{d_y \in D(y)|$ esiste $d_x \in D(x)$ tale che $\{x = d_x, y = d_y\}$ è una soluzione per $c\}$
$D = D_1$
endif
endfor
until $W = D$ /* finchè il nuovo dominio non varia */
return D

Propagazione per domini finiti

Solver basato sulla arc consistency

Input un vincolo $C = c_1 \wedge ... \wedge c_n$
il dominio D
Output $true, false$, or $unknown$
Method
$D = node\_consistent(C, D)$
D = arc_consistent(C,D)
if for some variable the domain is empty **then**
return $false$
else
if the domain of each variable is a singleton set **then**
return $satisfiable(C, D)$
else return $unknown$
endif
endif

Propagazione per domini finiti

Esempio

Si consideri il vincolo:

$$X < Y \wedge Y < Z \wedge Z \leq 2$$

ed il seguente dominio:

$D(X) = D(Y) = D(Z) = \{1, 2, 3\}$.

- Applicando la node consistency si ottiene $D(X) = D(Y) = \{1, 2, 3\}, D(Z) = \{1, 2\}$
- Applicando la arc consistency:
  - si sceglie $X < Y$
  - il nuovo dominio diventa $D(X) = \{1, 2\}, D(Y) = \{2, 3\}, D(Z) = \{1, 2\}$
  - si sceglie $Y < Z$
  - il nuovo dominio diventa $D(X) = \{1, 2\}, D(Y) = \{\}, D(Z) = \{\}$
  - si sceglie nuovamente $X < Y$
  - il nuovo dominio diventa $D(X) = \{\}, D(Y) = \{\}, D(Z) = \{\}$
  - si sceglie nuovamente $Y < Z$
  - il nuovo dominio diventa $D(X) = \{\}, D(Y) = \{\}, D(Z) = \{\}$ e l'algoritmo di arc consistency termina
- il solver restituisce $false$

Propagazione per domini finiti

- La nozione di node e arc consistency non considera (ignora) i vincoli contenenti più di due variabili

- Una semplice generalizzazione del concetto di arc consistency a vincoli con un numero arbitrario di variabili (*hyper-arc consistency*) porta ad algoritmi NP-hard in generale

- Si adotta una semplificazione
  $\Rightarrow$ *bound consistency*

- Si può applicare quando i domini delle variabili sono intervalli

Propagazione per domini finiti

Bound consistency

Un vincolo primitivo aritmetico c è *bound consistent* con dominio D se per ogni variabile $x \in vars(c)$ esiste:

- un assegnamento di numeri *reali*, $d_1, ...d_k$, alle variabili rimanenti in c, $x_1, ..., x_k$, tale che $min_D(x_j) \leq d_j \leq max_D(x_j)$, per ogni d_j, e $\{x = min_D(x), x_1 = d_1, ..., x_k = d_k\}$ è una soluzione per c
- un assegnamento di numeri *reali*, $d'_1, ...d'_k$, alle variabili rimanenti in c, $x_1, ..., x_k$, tale che $min_D(x_j) \leq d_j \leq max_D(x_j)$, per ogni d_j, e $\{x = max_D(x), x_1 = d_1, ..., x_k = d_k\}$ è una soluzione per c

Un vincolo $c_1 \wedge ... \wedge c_n$ è *bound consistent* con dominio D se ogni vincolo primitivo c_i è bound consistent con D, $1 \leq i \leq n$

Osservazione

- In base a questa definizione, i domini devono assegnare **intervalli** alle variabili
- la nozione di consistenza si basa su numeri reali

Propagazione per domini finiti

Esempio

Vincolo: $X = 3Y + 5Z$

Dominio: $D(X) = [2..7]$, $D(Y) = [0..2]$, $D(Z) = [-1, 2]$

$\Rightarrow$ non è bound consistent con D

Riscrivendo il vincolo come $5Z = X - 3Y$, assegnando 2 a Z si ottiene 10 per il lato sinistro ma il valore massimo del lato destro è 7!

Per ottenere bound consistency:

$D(X) = [2..7]$, $D(Y) = [0..2]$, $D(Z) = [0, 1]$

Propagazione per domini finiti

- È sempre possibile trasformare un problema a dominio finito in un altro problema che sia bound consistent (non vediamo l'algoritmo)

- È però necessario definire le *regole di propagazione* per la tipologia di vincoli prescelta

- tali regole permettono di calcolare nuovi intervalli per ogni variabile

- in genere, la definizione delle regole di propagazione per un certo vincolo è molto complessa

- per questo motivo i solver basati sulla bound consistency in genere si applicano ad una classe ristretta di vincoli
  - equazioni o disuguaglianze aritmetiche lineari
  - vincoli non lineari in forme specifiche come $t_1 = t_2$x t_3, $t_1 \neq t_2$, $t_1 = minimum\{t_2, t_3\}$, dove ogni t_i è una variabile o una costante e nessuna variabile compare più di una volta nel vincolo primitivo

Propagazione per domini finiti

Regole di propagazione per il vincolo $X = Y + Z$

Partendo da un dominio D, il nuovo dominio può essere calcolato come segue:

$X_{min} := maximum\{min_D(X), min_D(Y) + min_D(Z)\}$
$X_{max} := minimum\{max_D(X), max_D(Y) + max_D(Z)\}$
$D(X) := \{d_x \in D(X) | X_{min} \leq d_x \leq X_{max}\}$

$Y_{min} := maximum\{min_D(Y), min_D(X) - min_D(Z)\}$
$Y_{max} := minimum\{max_D(X), max_D(Y) - min_D(Z)\}$
$D(Y) := \{d_y \in D(Y) | Y_{min} \leq d_y \leq Y_{max}\}$

$Z_{min} := maximum\{min_D(Z), min_D(X) - max_D(Y)\}$
$Z_{max} := minimum\{max_D(Z), max_D(X) - min_D(Z)\}$
$D(Z) := \{d_z \in D(Z) | Z_{min} \leq d_z \leq Z_{max}\}$

Esempio

$D(X) = [4..8], D(Y) = [0..3], D(Z) = [2..2]$
si ottiene
$D(X) = [4..5], D(Y) = [2..3], D(Z) = [2..2]$
Il vincolo è bound consistent rispetto a questo dominio

Constraint solver incrementali

Nel contesto della valutazione di un programma CLP è necessario verificare la consistenza di vincoli generati in modo incrementale

Soluzione Si mantiene il vincolo in una forma *parzialmente risolta* e si estende tale forma durante la derivazione

Constraint solver incrementali

- Vincoli lineari $\Rightarrow$ algorimo di Gauss-Jordan per solving

- Per verificare la consistenza
  $\Rightarrow$ si genera una forma normale $x_1 = e_1 \wedge ... \wedge x_n = e_n$
  dove $x_1, ..., x_n$ non compaiono in $e_1, ..., e_n$

- Nel solving incrementale:
  - la forma risolta viene associata ad ogni nodo dell'albero di derivazione
  - ad ogni derivazione la forma risolta viene estesa

- Questo approccio può essere applicato a tutti i solver che generano una forma risolta per determinare la soddisfacibilità di un vincolo

- **Requisito**: il backtracking deve potere riabilitare la forma risolta associata al nodo a cui si ritorna

Constraint solver incrementali

Esempio

- Se $h = 1$:
  - applico S a c'_1
  - se c'_1 non contiene variabili se ne verifica immediatamente la soddisfacibilità
  - altrimenti:
    - si riscrive c'_1 nella forma $x = e$
    - si applica $x = e$ ad S
    - $S_1 = S \wedge x = e$
  - se $h > 1$: i passi precedenti vengono ripetuti per $c'_1, ..., c'_h$

Altre operazioni sui vincoli

Oltre alle tecniche di verifica della soddisfacibilità e di propagazione, esistono altre tecniche che un sistema CLP in genere supporta:

- *Proiezione.* Permette di restringere un vincolo ad un insieme di variabili

  Utile per restringere le soluzioni alle variabili presenti nel goal

  Esempio: la proiezione del vincolo $X \geq Y \wedge Y \geq Z \wedge Z \geq T \wedge T \geq 0$ su X corrisponde al vincolo $X \geq 0$

  Per i vincoli lineari, viene utilizzato l'algoritmo di Fourier

- *Semplificazione.* Permette di rimpiazzare un vincolo con un vincolo equivalente ma con una forma più semplice

  Utile quando per la forma più semplice esistono algoritmi di soddisfacibilità e propagazione più semplici

  Esempio: tecniche per eliminare la ridondanza

- *Ottimizzazione.* Permettono di determinare la soluzione migliore per un vincolo

  Esempio: algoritmo del simplesso per vincoli lineari reali