

Programmazione Logica a Vincoli

Andrea Omicini

LIA - DEIS - Università di Bologna

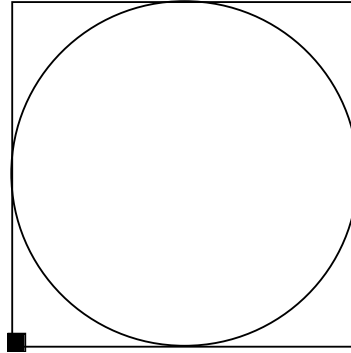
Sommario

- **Programmazione non-algoritmica: cenni**
- **Problemi a vincoli: concetti generali**
- **I paradigmi di programmazione**
- **La programmazione logica a vincoli**
- **Esempi e applicazioni**

Programmazione non-algoritmica

- **Algoritmo**
 - *Successione* (finita) di *passi* (azioni) che determinano la soluzione di un problema (*F. Luccio*)
 - Specifica di azioni e della loro successione $\Rightarrow$ controllo
 - **La programmazione imperativa**
 - Programmare con l'assegnamento
 - Strutture dati e tipi di dato
 - Costrutti di controllo
 - Thread di controllo
 - es.: *la programmazione strutturata*
- ? Si può programmare senza algoritmi = senza controllo ?**

Programmazione non-algoritmica (2)



- **Quadrato circoscritto nel cerchio**
 - Come lo rappresento?
 - *una unica struttura dati ad hoc*
 - *più strutture correlate*
 - Come gestisco la modifica di uno dei due componenti?
 - *gestire la correlazione*
 - Come definisco una *relazione invariante*?

Programmazione non-algoritmica (3)

- **Relazione**

`circoscritto(quadrato, cerchio)`

- **Dichiaratività**

- “Dichiarativi sono ... [i discorsi] in cui sussiste un'enunciazione vera oppure falsa” (Aristotele)

? È possibile programmare per dichiarazioni ?

- **Dichiarazione ≠ Azione**

- Chi definisce l'azione? NON il programmatore!

? È possibile programmare per dichiarazioni = senza controllo?



? È possibile programmare senza algoritmi ?

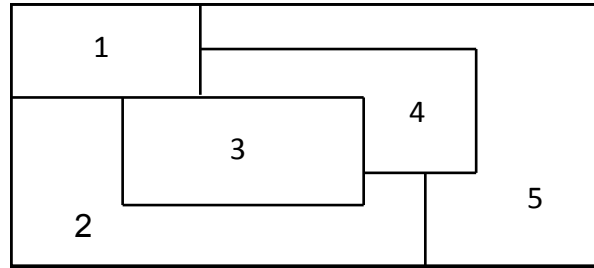
Problemi a Vincoli

- **Variabili $X_1, X_2, \dots, X_n$**
 - rappresentano le grandezze del problema
 - definite su un dominio di valori $D_1, D_2, \dots, D_n$
- **Vincoli**
 - rappresentano le relazioni tra le grandezze
 - definiscono un sottoinsieme del prodotto cartesiano dei domini $D_1 \times D_2 \times \dots \times D_k$;

$$v(X_1, X_2, \dots, X_k)$$

- ogni vincolo tra k variabili è sempre riconducibile a vincoli binari.

Esempio



Grandezze: zone della mappa X_1, X_2, X_3, X_4, X_5

Dominio: insieme di colori $\{r, g, b, y\}$

Vincoli: due zone adiacenti devono essere colorate con colori diversi.

if near(i, j) then $X_i \neq X_j$

Concetti generali

- **Constraint store**
 - collezione dei vincoli del problema
- **Soddisfacibilità**
 - una collezione di vincoli è soddisfacibile (consistente) se esiste (almeno) una soluzione
- **Propagazione di vincoli**
 - meccanismo di inferenza di nuovi vincoli a partire da quelli dati

Esempio

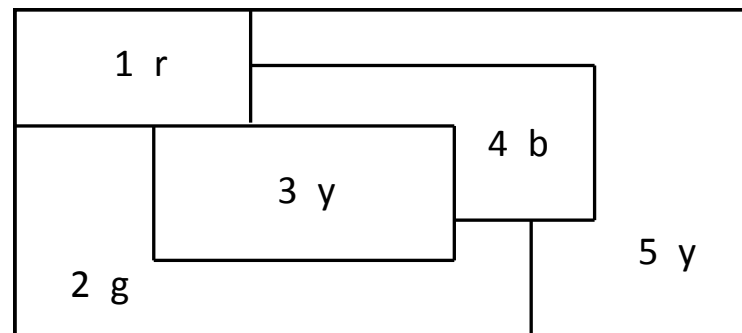
- **Constraint store**

vincoli unari: $\forall i \in [1,5], X_i \in \{r,g,b,y\}$

vincoli binari: $\forall i,j \in [1,5], \text{if near}(i, j) \text{ then } X_i \neq X_j$

- **Soddisfacibilità**

- Una possibile soluzione



Esempio (2)

- **Propagazione dei vincoli**

- Modifica del constraint store (nuovo vincolo)

$$(\text{if near}(1,2) \text{ then } X_1 \neq X_2) \wedge \text{near}(1,2)$$

$\Downarrow$

$$X_1 \neq X_2$$

- Modifica del constraint store (riduzione di un dominio)

$$(X_1 = r) \wedge (X_1 \neq X_2) \wedge (X_2 \in \{r, g, b, y\})$$

$\Downarrow$

$$X_2 \in \{g, b, y\}$$

Quale strumento

- **Cosa ci serve**
 - capacità di esprimere relazioni
 - test di consistenza
 - capacità inferenziali
- **Cosa abbiamo**
 - linguaggi logici (relazionali)
 - linguaggi imperativi (basati sull'assegnamento)

Linguaggi Imperativi

- **Esprimere relazioni**
 - domini predefiniti: tipi primitivi del linguaggio
 - relazioni: espressioni booleane, strutture dati
- **Test di consistenza**
 - assegnamento distruttivo
 - ⇒ non incrementalità del test
 - attivazione esplicita del test di consistenza
- **Capacità inferenziali**
 - attivazione esplicita del passo di inferenza

Linguaggi Logici

- **Esprimere relazioni**

- basati su relazioni
- dominio non interpretato: Universo di Herbrand
- unificazione interpretabile come vincolo (= sintattico)
- sostituzione come store dei vincoli

Universo di Herbrand di un programma P :

insieme di tutti i termini "ground" che possono essere costruiti a partire dai simboli di costante e di funzione di P

es. $p(X) :- q(f(X)).$

$p(a).$

$p(b).$

$q(f(c)).$

$H = \{a, f(a), f(f(a)), f(f(...f(a)))..., b, f(b), f(f(b)), ..., c, f(c), f(f(c))\}$

Linguaggi Logici (2)

- **Test di consistenza**
 - nozione e algoritmo di consistenza sul dominio di Herbrand
 - variabili write-once $\Rightarrow$ incrementalità del test
- **Capacità inferenziali**
 - passo di risoluzione
 - algoritmo di unificazione
 - clausole come invarianti: verità universali ed eterne
 - logica (programmatore) + controllo (linguaggio)

Esempio

- **$p(X, X)$.**
 - clausola (fatto) come invariante
- **$?- p(Z, Y), \dots$**
 - unificazione del goal con la testa della clausola
 - vincoli risultanti: $Z = X', Y = X'$, consistenti
 - propagazione: $Z = Y$
- **$?- \dots, Z = 3, \dots$**
 - nuovo vincolo $Z = 3$
 - test di consistenza ok
 - propagazione: $X' = Y = 3$
- **$?- \dots, Y = 4.$**
 - fallimento test di consistenza

Limiti dei Linguaggi Logici

- **Dominio non interpretato**
 - Universo di Herbrand: dominio “sintattico”
 - ⇒ unificazione sintattica
 - ⇒ assenza di tipi (domini) predefiniti

- **Esempio**

$$X = 2 + 5$$

- x istanziato alla struttura $2+5 \Rightarrow x = 7$ fallisce
- Soluzione: predicato *is*
 - $X \text{ is } 2 + 5 \Rightarrow X = 7$ ha successo
 - Problema: $X \text{ is } Y + 5$. Se Y è libera fallisce
 - Direzionalità: *is* introduce una valutazione

Limiti dei Linguaggi Logici (2)

- **Dominio non interpretato**

$$X > Y$$

- $>$ visto come test e non come vincolo
- valutazione espressioni (come `is`)

- **Cosa vorremmo?**

- $X > Y$, $X = Y+5$: vincoli collezionati nel constraint store verificati ad ogni passo della computazione

Limiti dei Linguaggi Logici (3)

- **Ordine testuale dei goal significativo**
 - $x > 5, x = 7$ fallisce
 - $x = 7, x > 5$ ha successo
- **Necessità di intervenire sull'ordine di valutazione dei goal (secondo lo stato delle variabili)**
 - `delay`, `freeze` + metapredicati (`var`, `ground`, etc.)
 - a carico del programmatore !!!

Limiti dei Linguaggi Logici (4)

- **Inefficienza dell'esplorazione dello spazio delle soluzioni**
 - relazioni come test
 - *generate and test*
 - *standard backtracking*
 - relazioni come vincoli
 - *forward checking*
 - *look ahead*

Limiti dei Linguaggi Logici (5)

- **Inefficienza dell'esplorazione dello spazio delle soluzioni: un esempio**
 - LP: generazione di una soluzione + test
$$X_1 = r \wedge X_1 \neq X_2 \wedge X_2 \in \{r, g, b, y\}$$
$$X_2 = r \quad \text{fail (backtracking)}$$
$$X_2 = g \quad \text{consistente (proseguo)}$$
- **Cosa vorremmo?**
 - Propagazione dei vincoli *a priori*
$$X_1 = r \wedge X_1 \neq X_2 \wedge X_2 \in \{r, g, b, y\}$$
$$X_2 \in \{g, b, y\}$$
 - *Pruning* dello spazio delle soluzioni

Limiti dei Linguaggi Logici (6)

- **Vincoli come risultato di una computazione**

- LP: risposta calcolata come insieme di vincoli d'uscita
 - *funziona con il predicato = (unificazione)*

```
p(X,X).                % DB Prolog
?- p(Z,Y).             % Goal
Z = Y                  % Risposta calcolata
```

- *non funziona con un predicato come >*

```
p(X,Y) :- X > Y.       % DB Prolog
?- p(Z,Y).             % Goal
fail
```

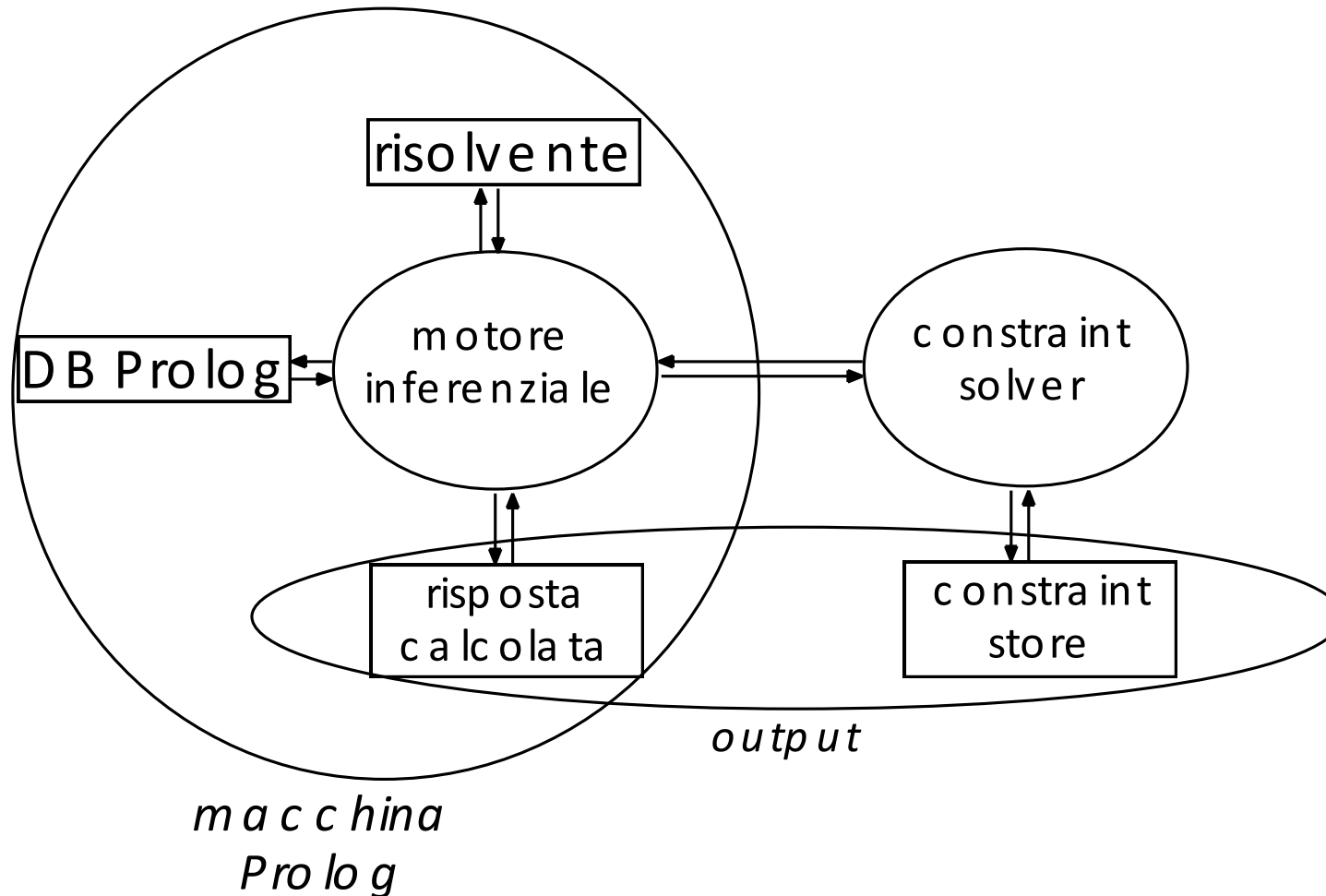
- **Cosa vorrei ?**

- Risposta: vincoli rimasti al termine della computazione
yes if Z > Y
- freeze “congela” un predicato (vincolo)
ed eventualmente lo fornisce in uscita

Estendere la macchina logica

- **Eventi rilevanti**
 - vincolo dal risolvente
 - binding di una variabile sul dominio
- **Estensioni**
 - constraint solver
 - constraint store
 - definizione di predicati ad-hoc (vincoli)
 - estensione del meccanismo di unificazione

La macchina logica estesa



Constraint Logic Programming

- **Schema CLP(X): Jaffar & Lassez, 1987**
- **Domini pre-interpretati nel linguaggio CLP**
 - CLP(R): dominio dei numeri reali $\Rightarrow X > 4.3$
 - 4.3 è il numero reale, non un generico simbolo
 - $>$ è la relazione d'ordine sui reali, non un generico predicato o un “test”
 - valgono tutte le proprietà del campo reale
- **Capacità di esprimere e trattare relazioni sul dominio come vincoli**
 - CLP(R): $X > Y + 5$

Constraint Logic Programming (2)

- **Nozione di consistenza e di propagazione embedded nel linguaggio**

- CLP(R): $X > Y + 5, Y = 3$
 - *passo di inferenza: $x > 8$*
 - *nozione di consistenza: specializzata su R*

- **Vincoli come ingressi e risultati**

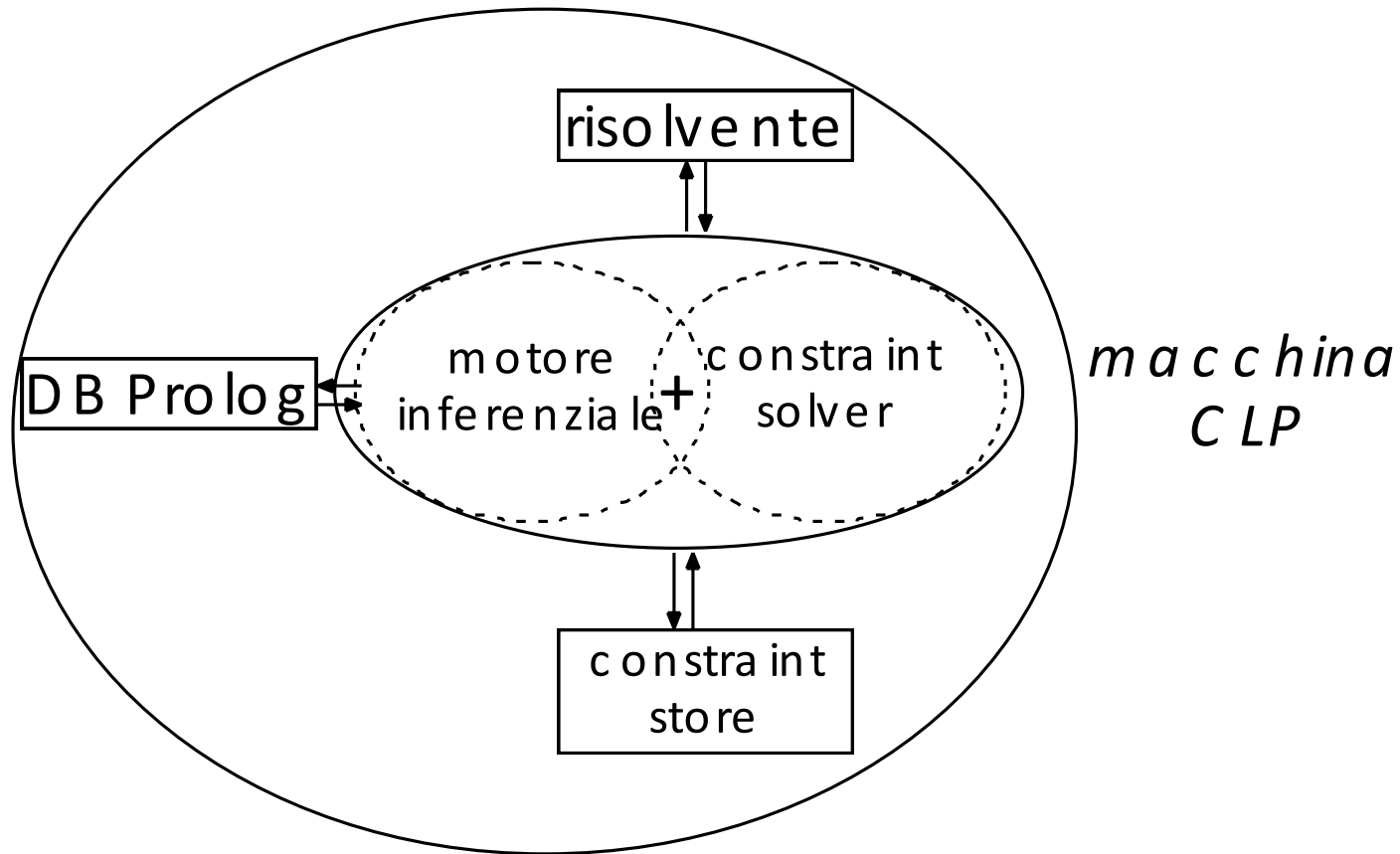
- CLP(R):

$p(A, B) :- A > B + 5.$

$?- p(X, Y), Y = 3.$

$X > 8, Y = 3$

La macchina CLP



Passi fondamentali

- **Risoluzione**
 - selezione di un atomo dal risolvante
 - unificazione: aggiunta di vincoli al constraint store
- **Constraining**
 - selezione di un vincolo dal risolvante
 - aggiunta del vincolo al constraint store
- **Infer (propagazione)**
 - vincoli *entailed*
 - trasformazione del constraint store
- **Consistenza**
 - verifica della soddisfacibilità del constraint store

Esempio CLP(*R*): codice e goal

```
r(X,Y):- p(X), X > Y, q(Y).
```

```
p(Z):- Z = 3.
```

```
q(K):- K > 5.
```

```
q(K):- K > 2.
```

```
?- r(A,B).
```

Esempio CLP(R): i passi (1)

passo	risolvente	constraint store
risoluzione	$r(A, B)$	$\emptyset$
(unif.)	$p(X'), X' > Y', q(Y')$	$A = X', B = Y'$
inferenza	$(idem)$	$A = X', B = Y'$
consistenza	$(idem)$	$(idem)$
risoluzione	$p(X'), X' > Y', q(Y')$	$A = X', B = Y'$
(unif.)	$Z' = 3, X' > Y', q(Y')$	$A = X', B = Y', X' = Z'$
inferenza	$(idem)$	$A = X' = Z', B = Y'$
consistenza	$(idem)$	$(idem)$
constraint	$Z' = 3, X' > Y', q(Y')$	$A = X' = Z', B = Y'$
	$X' > Y', q(Y')$	$A = X' = Z', B = Y', Z' = 3$
inferenza	$(idem)$	$A = X' = Z' = 3, B = Y'$
consistenza	$(idem)$	$(idem)$
constraint	$X' > Y', q(Y')$	$A = X' = Z' = 3, B = Y'$
	$q(Y')$	$A = X' = Z' = 3, B = Y', X' > Y'$
inferenza	$(idem)$	$A = X' = Z' = 3 > Y' = B$
consistenza	$(idem)$	$(idem)$

Esempio CLP(R): i passi (2)

risoluzione	$q(Y')$	$A=X'=Z'=3>Y'=B$
(unif.)	$K'>5$	$A=X'=Z'=3>Y'=B, Y'=K'$
inferenza	<i>(idem)</i>	$A=X'=Z'=3>Y'=B=K'$
consistenza	<i>(idem)</i>	<i>(idem)</i>
constraint	$K'>5$	$A=X'=Z'=3>Y'=B=K'$
	$\emptyset$	$A=X'=Z'=3>Y'=B=K', K'>5$
inferenza	<i>(idem)</i>	$A=X'=Z'=3>Y'=B=K'>5$
consistenza	fallimento	<i>(inconsistente)</i>
risoluzione	$q(Y')$	$A=X'=Z'=3>Y'=B$
(unif.)	$K'>2$	$A=X'=Z'=3>Y'=B, Y'=K'$
inferenza	<i>(idem)</i>	$A=X'=Z'=3>Y'=B=K'$
consistenza	<i>(idem)</i>	<i>(idem)</i>
constraint	$K'>2$	$A=X'=Z'=3>Y'=B=K'$
	$\emptyset$	$A=X'=Z'=3>Y'=B=K', K'>2$
inferenza	<i>(idem)</i>	$A=X'=Z'=3>Y'=B=K'>2$
consistenza	<i>(idem)</i>	<i>(idem)</i>
	successo!	$A=3, 2<B<3$

Un metainterprete

```
solve(true, C, C) :- !.
solve((A,B), C, NewC) :-
    solve(A, C, TempC), solve(B, TempC, NewC).
solve(A, C, NewC) :-
    clause(H, Body),           % atomi
    unify(A, H, UC),           % vincoli unif.
    infer(C, UC, TempC),       % propagazione
    consistent(TempC), !,      % consistenza
    solve(Body, TempC, NewC).
solve(V, C, NewC) :-
    constraint(V),              % vincoli
    infer(C, V, NewC),          % propagazione
    consistent(NewC).           % consistenza

?- solve(+Goal, +InC, -OutC).
```

La semantica operativa

- **Definizione formale di una macchina astratta**
 - i componenti
 - lo stato della macchina
 - i “passi elementari”: transizioni di stato
- **Computazioni come sequenze di passi della macchina astratta**
- **CLP(X): uno schema**
 - componenti:
 - *motore + solver*
 - *constraint store*
 - *DB Prolog*
 - *risolvente*
 - stato come tripla $\langle A, C, S \rangle$
 - 4 transizioni: *risoluzione, constraint, infer, consistenza*

La semantica operativa (2)

- **Stato della macchina astratta $\langle A, C, S \rangle$**
 - A (multi)set di atomi e vincoli
 - $C + S$ constraint store
 - C insieme dei vincoli attivi
 - S insieme dei vincoli passivi
- **Vincoli attivi:**
 - vincoli che vengono modificati nel corso della computazione
 - unici che vengono testati in fase di consistenza
- **Vincoli passivi:**
 - vincoli che non vengono mai modificati nel corso della computazione ma permettono di modificare altri vincoli
 - non vengono testati in fase di consistenza

La semantica operativa (3)

- **Transizioni**

- risoluzione: a atomo logico, $h :- B$ clausola (rinominata)
nel DB Prolog, $\{a=h\}$ tutti i vincoli di unificazione
$$\langle A \cup a, C, S \rangle \rightarrow_r \langle A \cup B, C, S \cup \{a = h\} \rangle$$
- constraint: c vincolo
$$\langle A \cup c, C, S \rangle \rightarrow_c \langle A, C, S \cup c \rangle$$
- inferenza: funzione infer : $(C', S') = \text{infer}(C, S)$
$$\langle A, C, S \rangle \rightarrow_i \langle A, C', S' \rangle$$
- consistenza: predicato consistent : $\text{consistent}(C)$
$$\langle A, C, S \rangle \rightarrow_s \langle A, C, S \rangle$$

- **Specializzazione dello schema sul dominio X**

- $C, S, \text{infer}, \text{consistent}$

Prolog come CLP(H)

- **Dominio**
 - Universo di Herbrand H
- **Vincoli di uguaglianza**
 - predicato predefinito = sintattico
 - sostituzione come constraint store
- **infer, consistent**
 - proprietà dell'uguaglianza
 - *riflessiva*
 - *simmetrica*
 - *transitiva*

Esempio

- **$p(X, X)$.**
 - DB Prolog
 - H è il dominio di x
- **$?- p(f(c), Y), \dots$**
 - unificazione del goal con la testa della clausola
 - vincoli risultanti: $f(c) = x', Y = x'$, consistenti
 - propagazione: $f(c) = Y$
- **$?- \dots, Y = a.$**
 - nuovo vincolo $Y = a$
 - fallimento test di consistenza
- **fail**

CLP(*FD*)

- **Domini Finiti**

- Dominio predefinito (es: interi) $\Rightarrow$ proprietà, vincoli ecc..
- Ogni variabile (grandezza) spazia su un sottoinsieme **finito** del dominio: *range* (es: $X::[4..10]$, $Y::[3,24,7]$)
- Istanziamento della variabile ad uno dei valori contenuti nel range di definizione

$X::[4..10]$, $Y::[3,24,7]$, $X > Y + 5$, $Z::[5..20]$, $Z < X$, $Z > Y$

- **Semantica Operazionale**

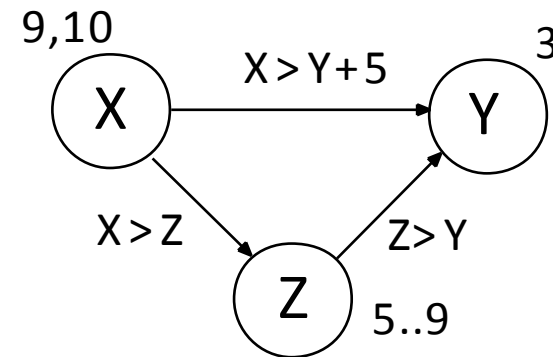
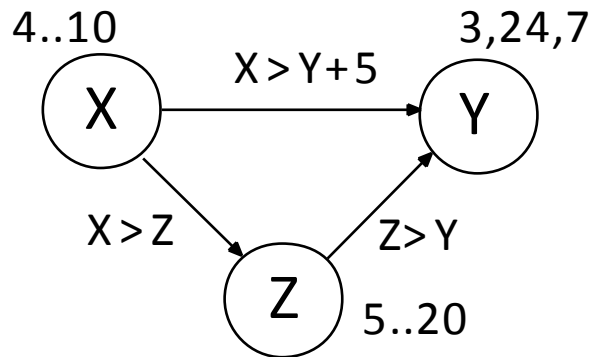
- Store attivo: vincoli unari di range su dominio finito
- Store passivo: vincoli binari, altri vincoli unari
- Inferenza: arc-consistency (*compromesso!*)
- Consistenza: un vincolo è consistente se ogni variabile coinvolta ha un dominio non vuoto

Arc-Consistency

- **Rete (grafo) di vincoli:**
 - Problema a vincoli rappresentabile con un grafo
nodi: variabili (grandezze)
archi: vincoli
- **Rete arc-consistente**
 - Se ogni arco della rete è arc-consistente
- **Arco arc-consistente**
 - Un arco V_{xy} che collega le variabili X e Y è arc-consistente se e solo se per ogni valore $x \in D_X$ (dominio di X) esiste almeno un valore $y \in D_Y$ tale che $V_{xy}(x,y)$ è vero

Arc-Consistency (2)

$X::[4..10]$, $Y::[3,24,7]$, $X > Y + 5$, $Z::[5..20]$, $Z < X$, $Z > Y$



- **Arc-consistenza**
 - $X::[9,10]$, $Y=3$, $Z::[5..9]$
- **Constraint Store**
 - Modifica e consistenza su vincoli attivi
 - I vincoli passivi necessari solo per la propagazione

Esempio

```
figlio(X,Y) :- genitore(Y,X).
genitore(X,Y) :- X > Y + 15.
coetaneo(Z,Z).
piuanziano(A,B,D) :- D > 0, A = B + D.
giovane(W) :- W < 30.

?- Giovanna::[0..50],
   Franco >= 0,
   figlio(Franco, Giovanna),
   giovane(Giovanna),
   coetaneo(Giovanna, Beppe),
   piuanziano(Beppe, Franco, Diff), Diff > 28.

Beppe=29, Diff=29, Giovanna=29, Franco=0.
```

Esempio: il programma

```
figlio(X,Y) :- genitore(Y,X).  
genitore(X,Y) :- X > Y + 15.  
coetaneo(Z,Z).  
piuanziano(A,B,D) :- D > 0, A = B + D.  
giovane(W) :- W < 30.
```

- **Modello del programma**
 - descrizione delle relazioni vere nel dominio (come LP)
 - grandezze come variabili
 - domini come interi
 - `genitore(X,Y) :- X > Y + 15.`
 - *difficile da capire in LP!!*
 - *regola di inferenza di nuovi vincoli*

Esempio: il goal

```
?- Giovanna::[0..50],  
   Franco >= 0,  
   figlio(Franco, Giovanna),  
   giovane(Giovanna),  
   coetaneo(Giovanna, Beppe),  
   piuanziano(Beppe, Franco, Diff), Diff > 28.
```

- **Modello della computazione**

- il goal
 - *insieme di atomi logici da dimostrare?*
 - *insieme di relazioni che definiscono il particolare problema nel dominio descritto dal programma*
 - *restrizione tramite vincoli dello spazio delle soluzioni*
- la computazione
 - *non specifico nessuna azione, solo vincoli e regole di inferenza*
 - *programmazione non-algoritmica!*

Esempio: stato iniziale

risolvente

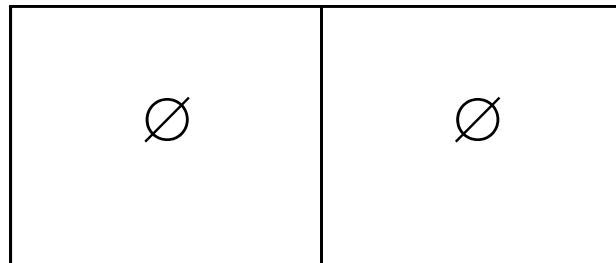
```
Giovanna::[0..50], Franco >= 0,  
figlio(Franco, Giovanna),  
giovane(Giovanna),  
coetaneo(Giovanna, Beppe),  
piuanziano(Beppe, Franco, Diff),  
Diff > 28.
```

```
figlio(X,Y) :-  
    genitore(Y,X).  
genitore(X,Y) :-  
    X > Y + 15.  
coetaneo(Z,Z).  
piuanziano(A,B,D) :-  
    D > 0, A = B + D.  
giovane(W) :- W < 30.
```

DB Prolog

C
(Vincoli attivi)

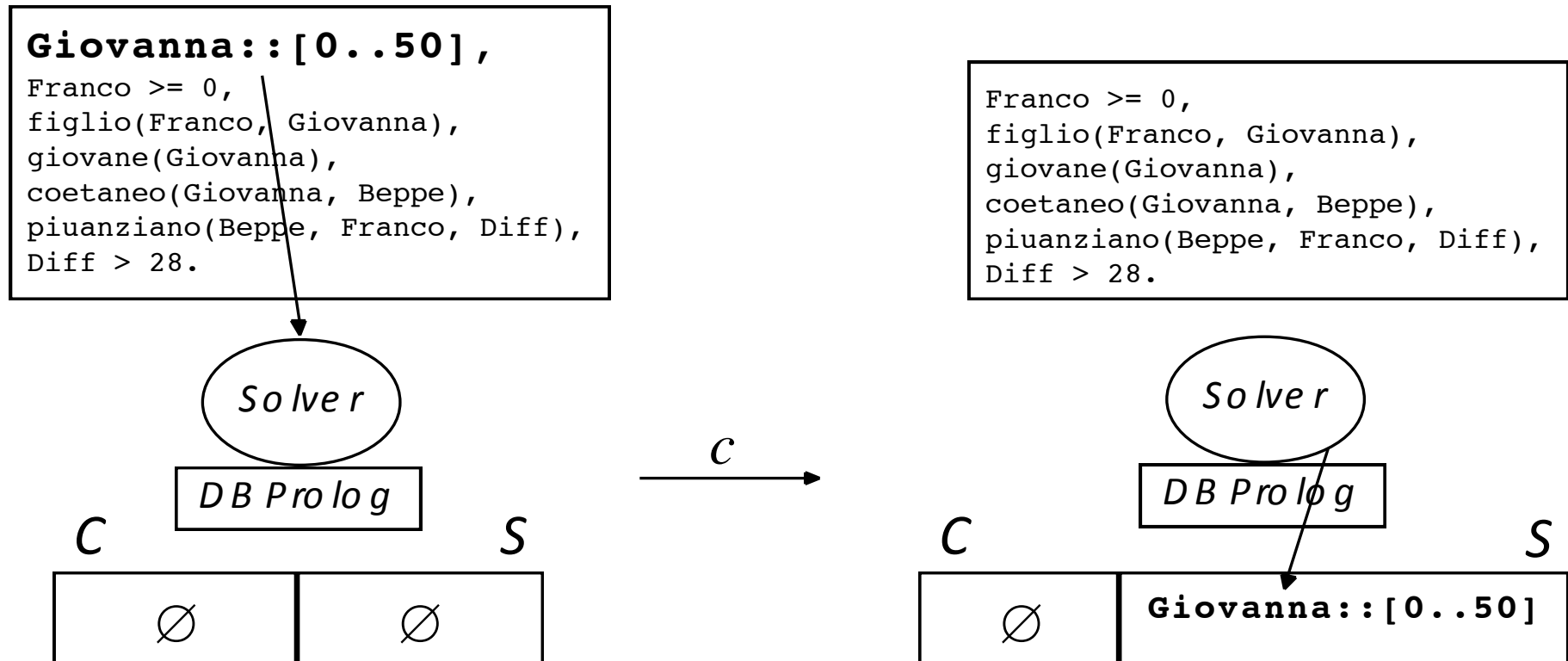
Constraint Solver



Constraint Store

S
(Vincoli passivi)

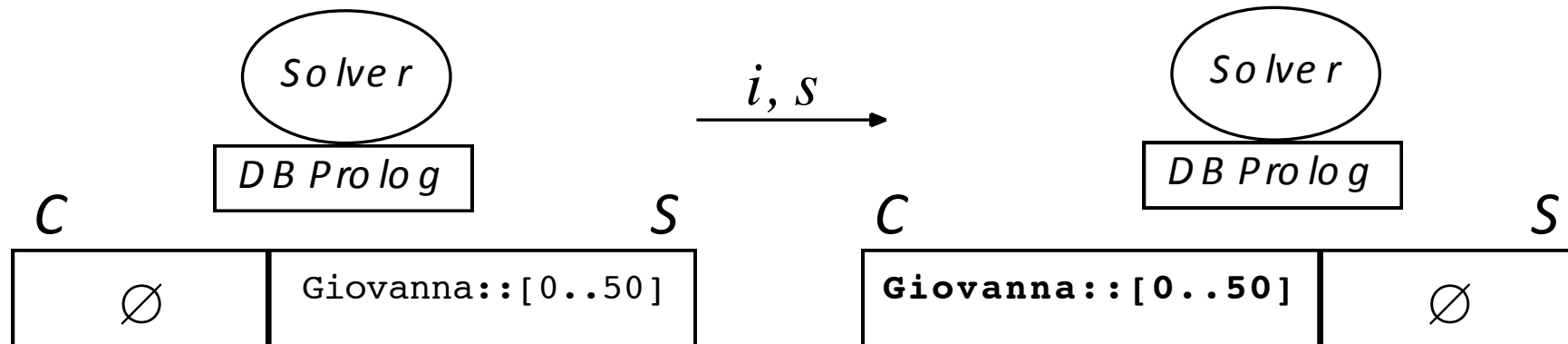
Esempio: i passi computazionali (1)



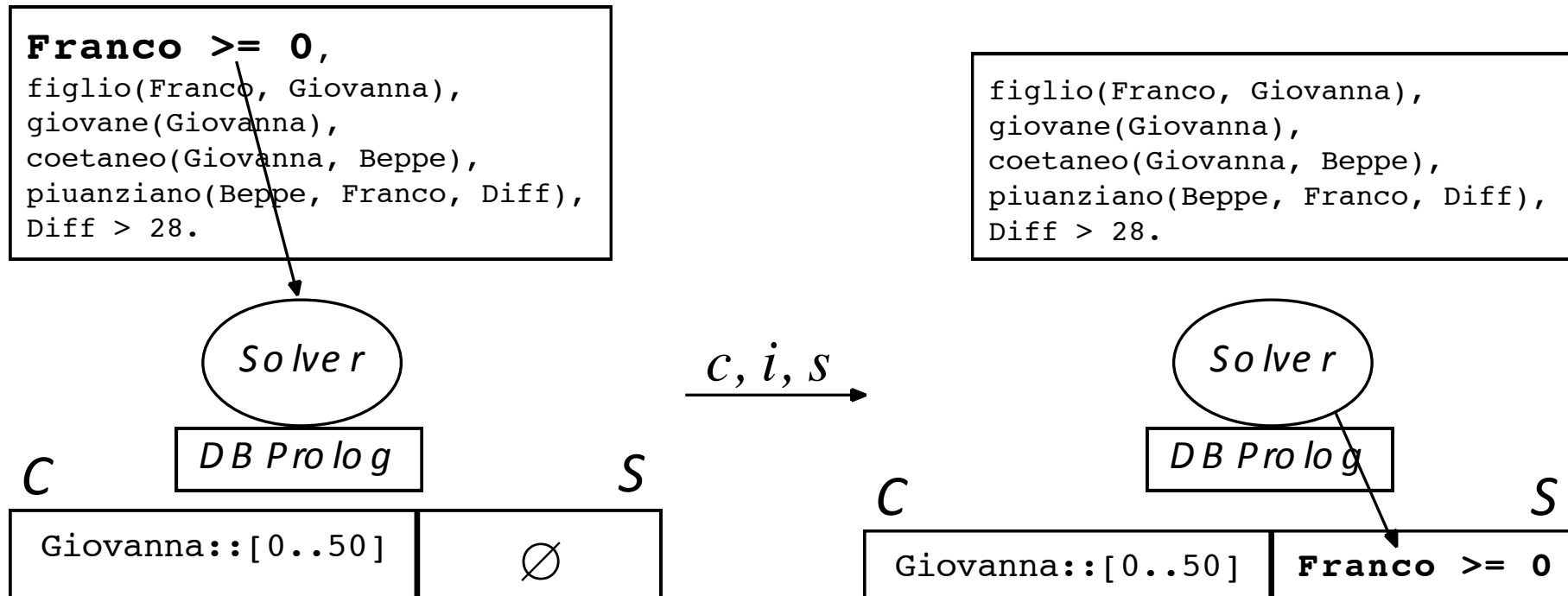
Esempio: i passi computazionali (2)

```
Franco >= 0,  
figlio(Franco, Giovanna),  
giovane(Giovanna),  
coetaneo(Giovanna, Beppe),  
piuanziano(Beppe, Franco, Diff),  
Diff > 28.
```

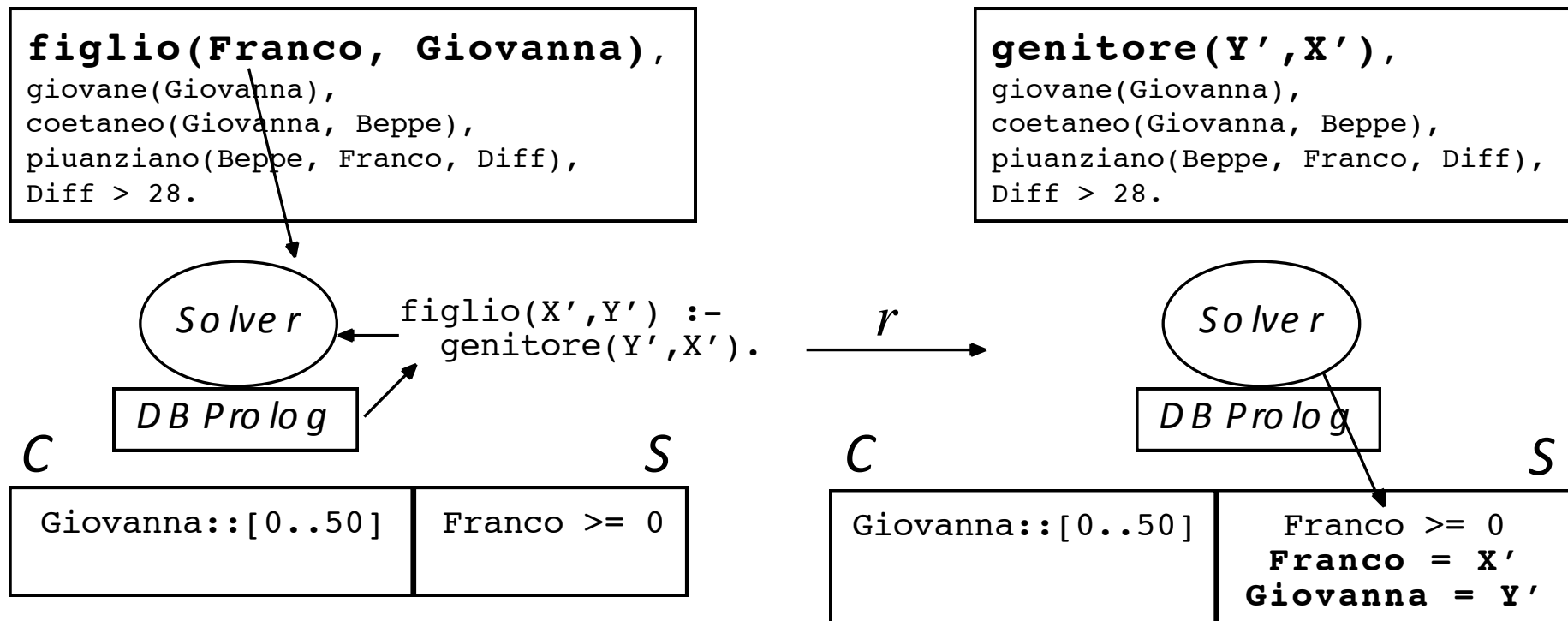
```
Franco >= 0,  
figlio(Franco, Giovanna),  
giovane(Giovanna),  
coetaneo(Giovanna, Beppe),  
piuanziano(Beppe, Franco, Diff),  
Diff > 28.
```



Esempio: i passi computazionali (3)



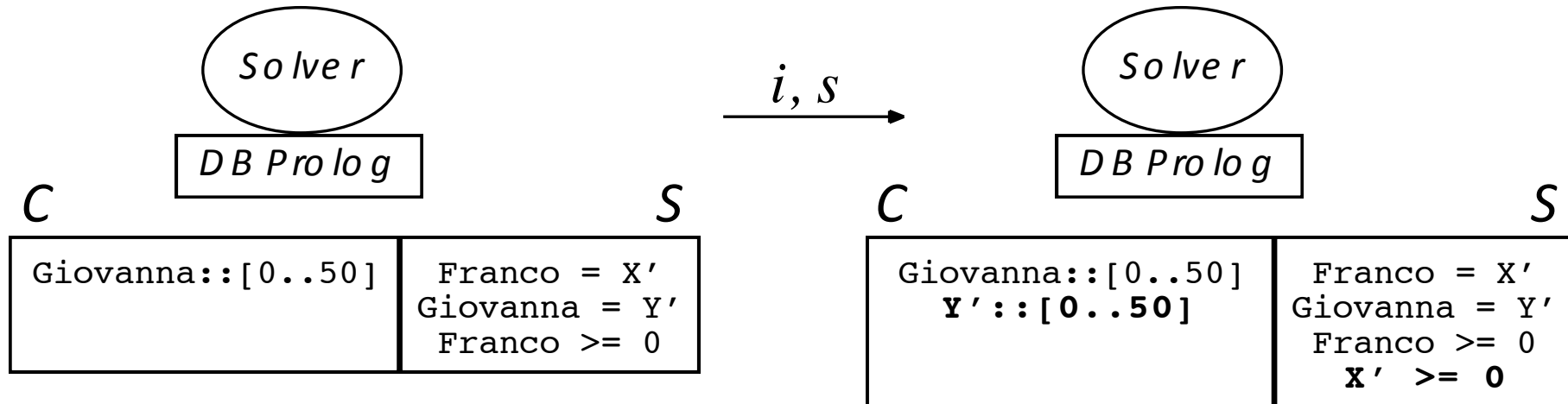
Esempio: i passi computazionali (4)



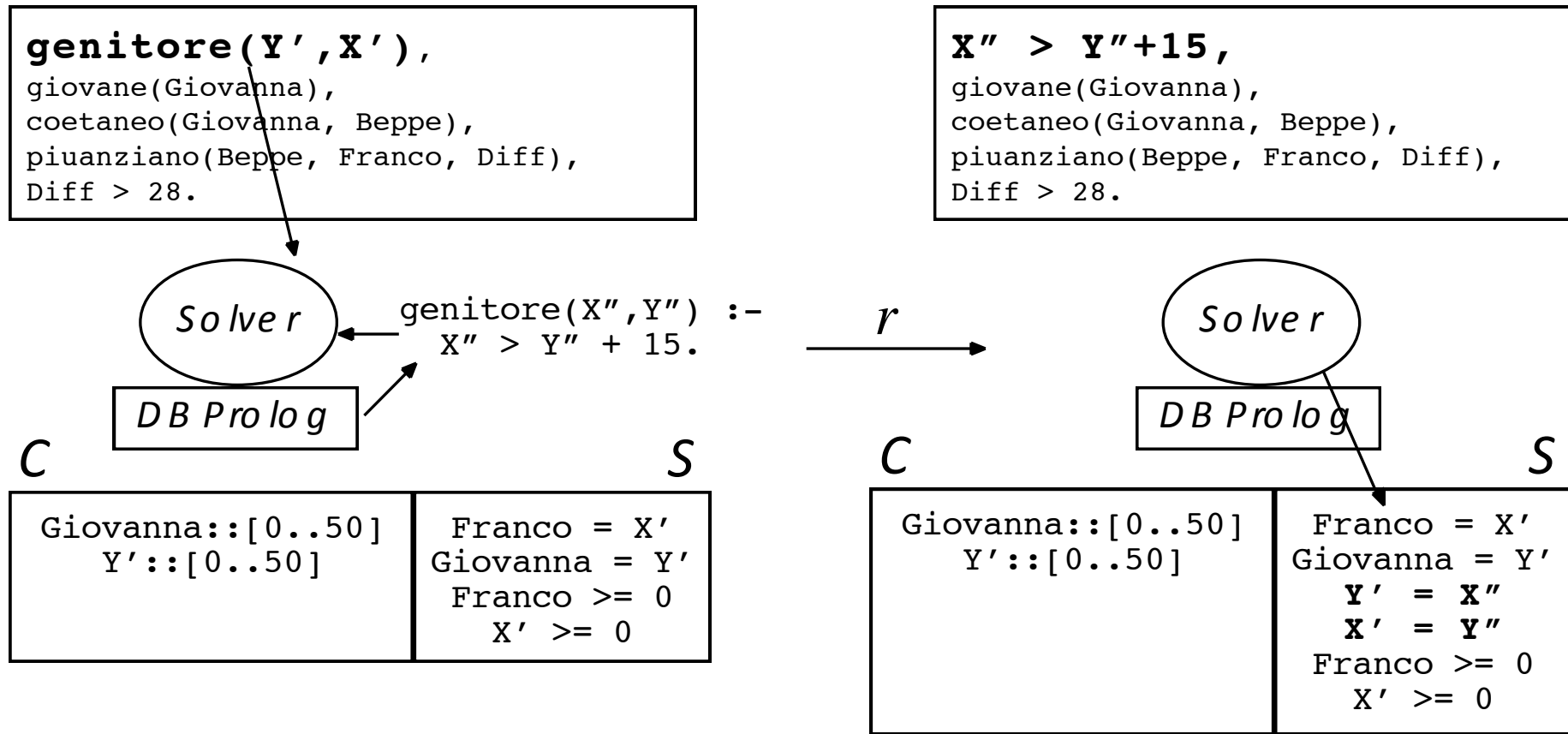
Esempio: i passi computazionali (5)

```
genitore(Y', X'),
giovane(Giovanna),
coetaneo(Giovanna, Beppe),
piuanziano(Beppe, Franco, Diff),
Diff > 28.
```

```
genitore(Y', X'),
giovane(Giovanna),
coetaneo(Giovanna, Beppe),
piuanziano(Beppe, Franco, Diff),
Diff > 28.
```



Esempio: i passi computazionali (6)



Esempio: i passi computazionali (7)

```
X'' > Y'' + 15,
giovane(Giovanna),
coetaneo(Giovanna, Beppe),
piuanziano(Beppe, Franco, Diff),
Diff > 28.
```

Solver

DB Prolog

C

S

```
Giovanna::[0..50]
Y'::[0..50]
```

```
Franco = X'
Giovanna = Y'
Y' = X''
X' = Y''
Franco >= 0
X' >= 0
```

i, S

```
X'' > Y'' + 15,
giovane(Giovanna),
coetaneo(Giovanna, Beppe),
piuanziano(Beppe, Franco, Diff),
Diff > 28.
```

Solver

DB Prolog

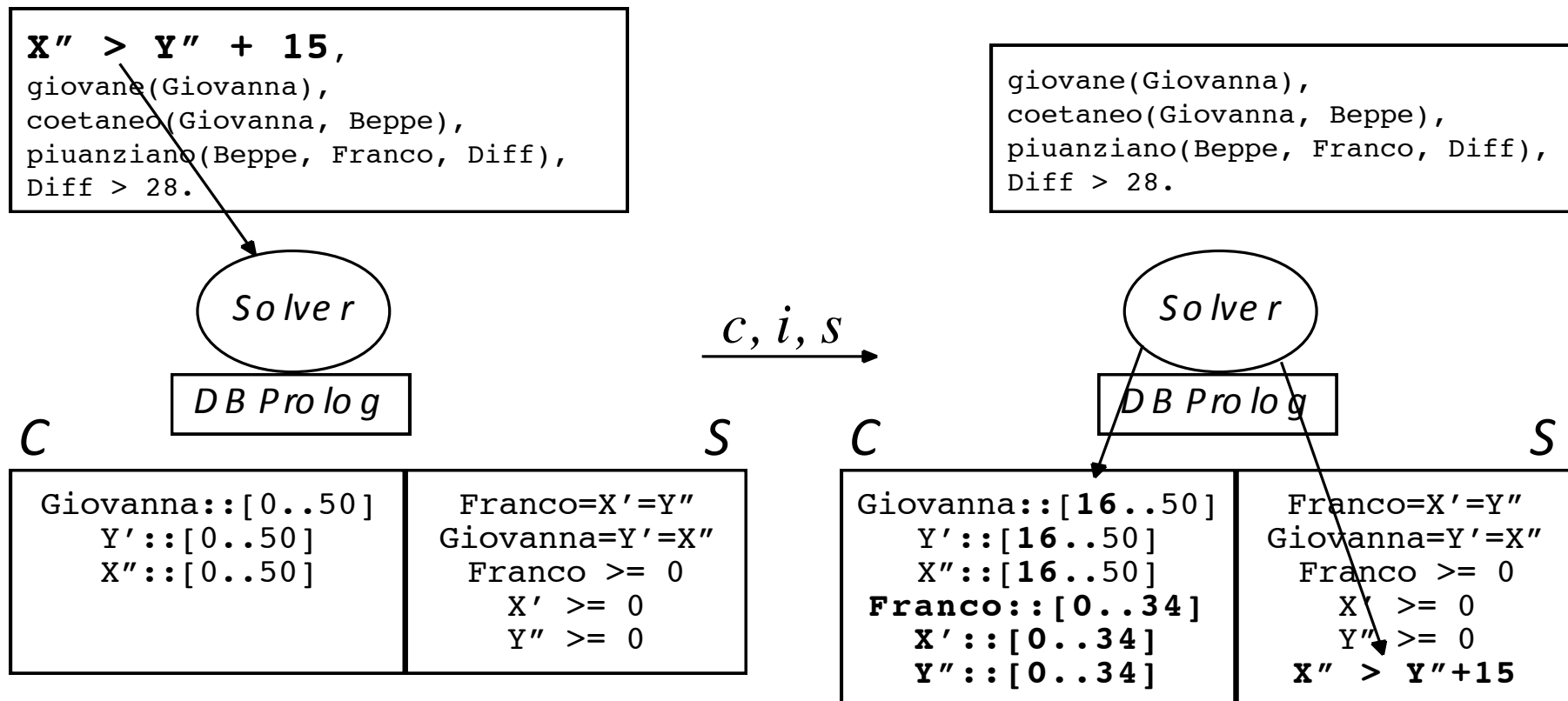
C

S

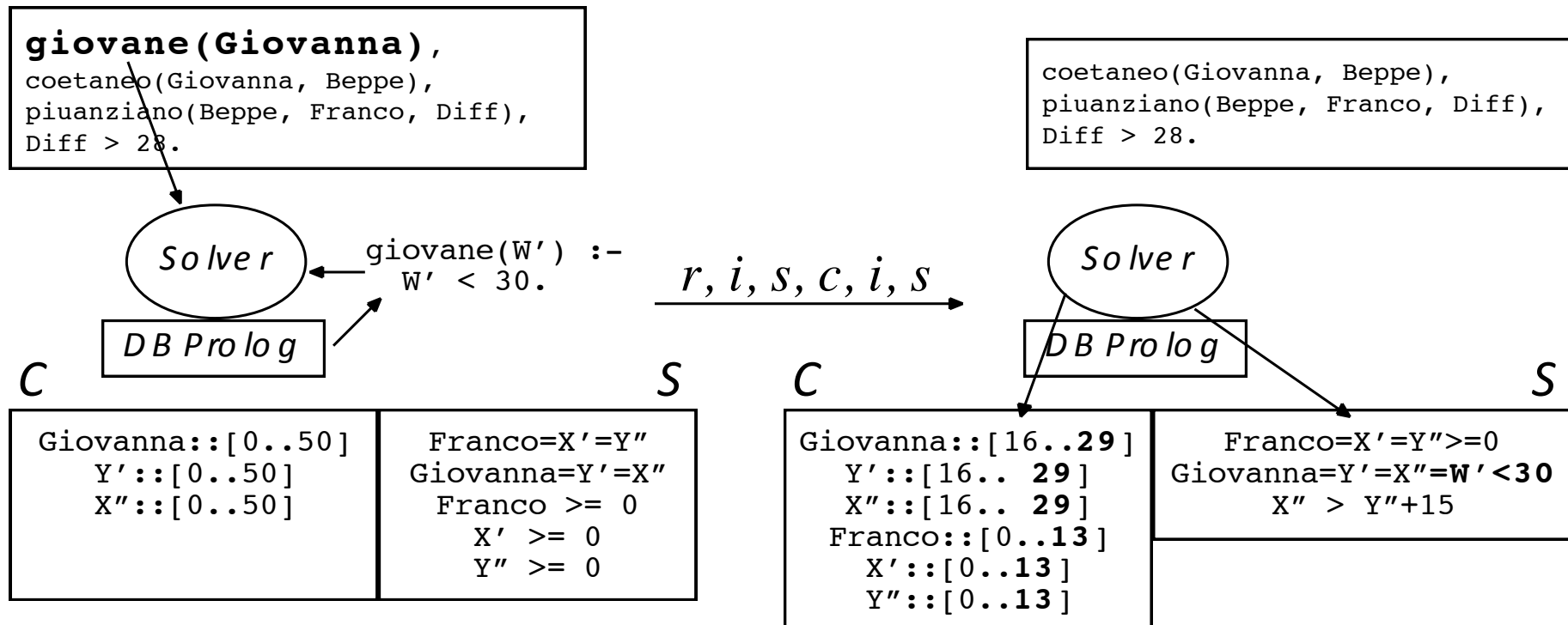
```
Giovanna::[0..50]
Y'::[0..50]
X''::[0..50]
```

```
Franco=X'=Y''
Giovanna=Y'=X''
Franco >= 0
X' >= 0
Y'' >= 0
```

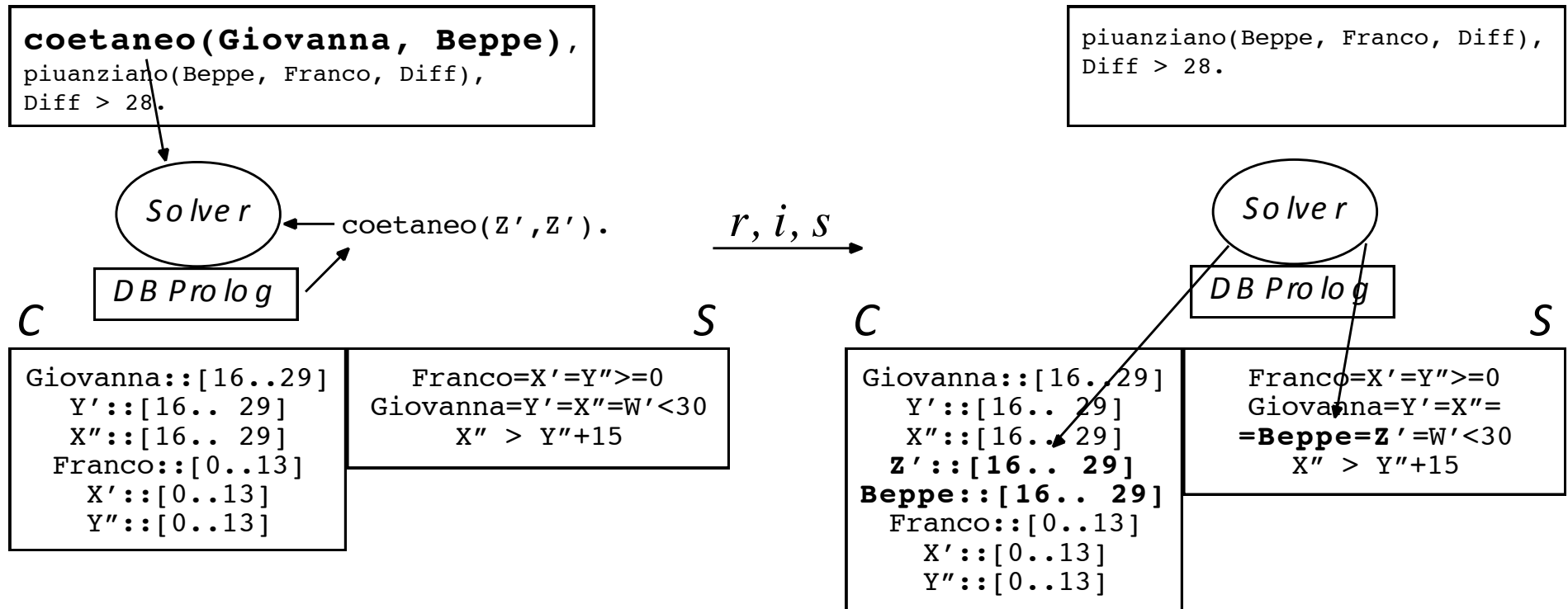
Esempio: i passi computazionali (8)



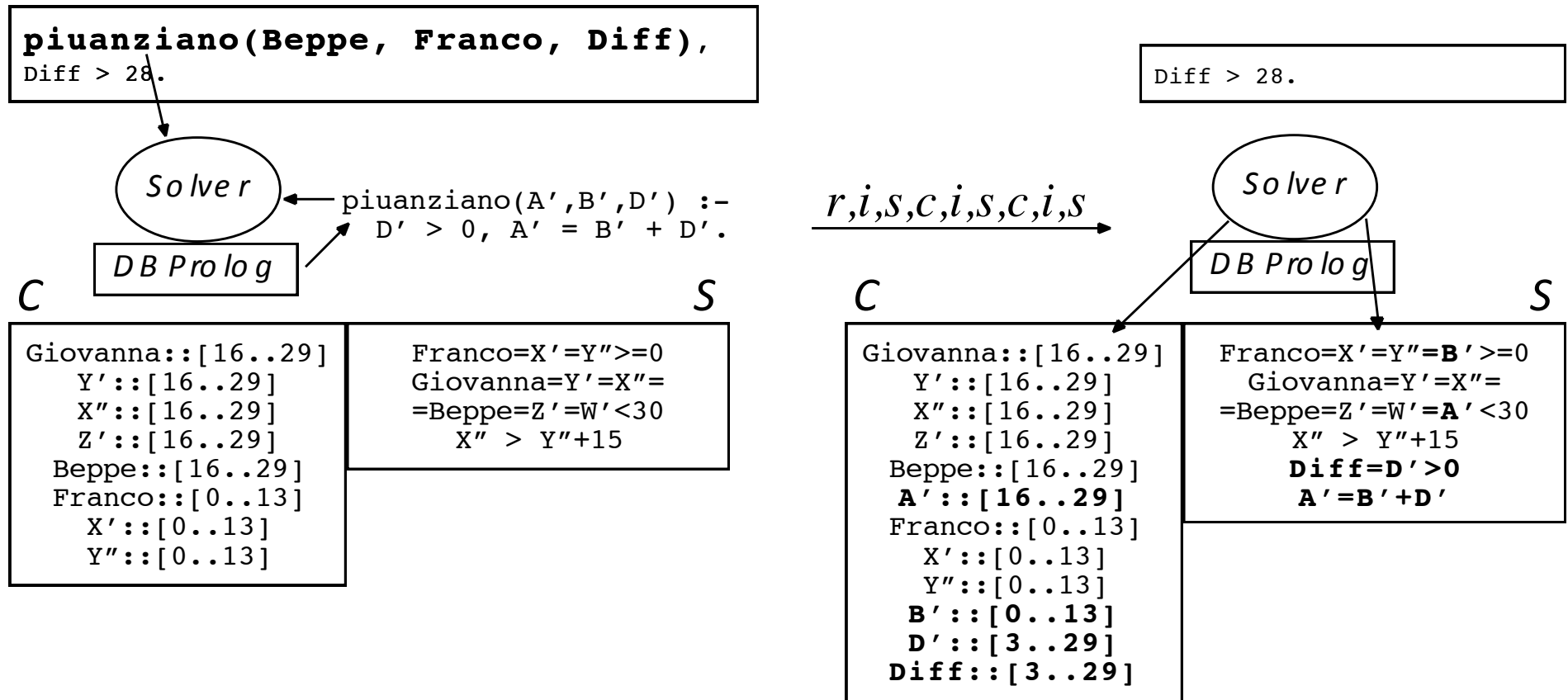
Esempio: i passi computazionali (9)



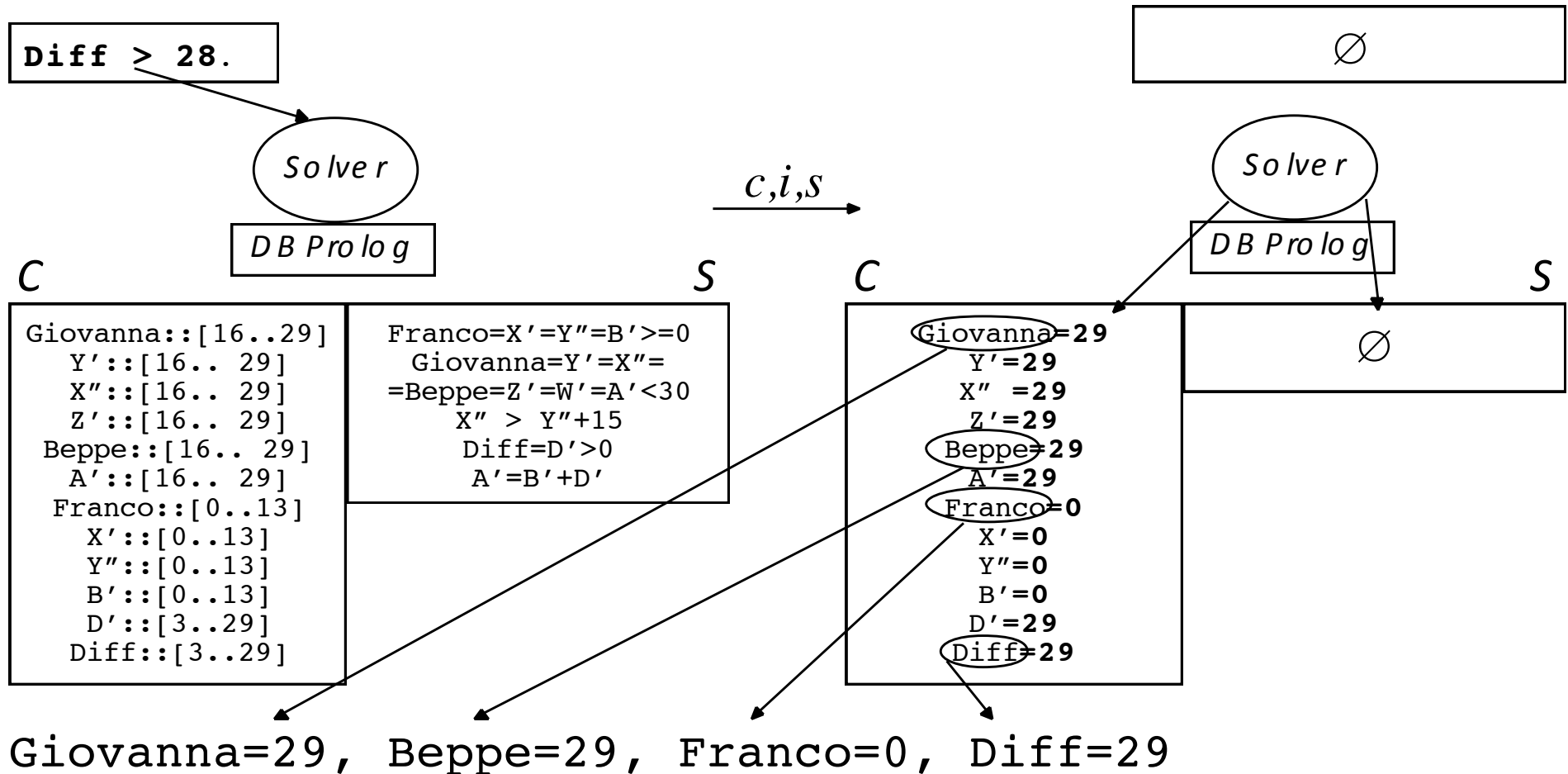
Esempio: i passi computazionali (10)



Esempio: i passi computazionali (11)



Esempio: i passi computazionali (12)



Applicazioni

- **Analisi di circuiti: utilizziamo CLP(R)**

- I vincoli descrivono le relazioni tra le grandezze corrispondenti a sottosistemi (legge di Ohm) e relazioni tra sottosistemi (leggi di Kirchhoff)

```
circuito(res(R),V,I):- V=R*I.
```

```
circuito(serie(S1,S2),V,I):-
```

```
    I=I1, I=I2, V=V1+V2,
```

```
    circuito(S1,V1,I1), circuito(S2,V2,I2).
```

```
circuito(parallelo(S1,S2),V,I):-
```

```
    I=I1+I2, V=V1, V=V2,
```

```
    circuito(S1,V1,I1), circuito(S2,V2,I2).
```

```
?- circuito(serie(serie(res(R),res(R)),res(R)),V,5).
```

```
    R = 0,0666667 *V
```

- Risultato: vincolo in forma *intensionale*

Applicazioni (2)

- **Problemi di ottimizzazione combinatoria**
 - definito su un numero finito di variabili $(X_1, \dots, X_n)$ legate da vincoli e da una funzione obiettivo

Struttura di un programma CLP

```
solve( $X_1, \dots, X_n, \text{Cost}$ ) :-  
    constraint( $X_1, \dots, X_n$ ),  
    generate( $X_1, \dots, X_n$ ),  
    cost( $X_1, \dots, X_n, \text{Cost}$ ).
```

```
?- solve( $X_1, \dots, X_n, \text{Cost}$ ), Cost < BestCost.
```

```
?- minimize(solve( $X_1, \dots, X_n, \text{Cost}$ ), BestCost).
```

- Pruning: risultato in forma *estensionale*

Applicazioni: Cutting Stock

- Si devono produrre degli scaffali a partire da **T** tavole di legno di dimensione data
- esiste un numero fisso **N** di modelli (forma e dimensione) di scaffali e un numero fisso **M** di configurazioni di taglio
- **S_{ij}** è il numero di scaffali di tipo **j** ($0 \leq j \leq N$) prodotto dal taglio di tipo **i** ($0 \leq i \leq M$) che produce lo scarto **W_i**
- **R_j** è il numero di scaffali di tipo **j** richiesti.
- **X_i** è il numero di tavole tagliate secondo la configurazione di tipo **i**

```
solve(X1, ..., XM, Cost) :-  
    X1 + ... + XM = T,  
    X1*S1,1 + ... + XM*SM,1 >= R1,  
    ...,  
    X1*S1,N + ... + XM*SM,N >= RN,  
    Cost = X1*W1 + ... + XM*WM,  
    generate(X1, ..., XM).
```

Applicazioni: Scheduling

- **Problema di scheduling**
 - assegnamento a **risorse** di **task** di **durata** data in un certo intervallo di tempo
- **Obiettivi tipici**
 - minimizzazione della durata dello schedule
 - massimizzazione dello sfruttamento delle risorse
- **Vincoli tipici**
 - due date e release date
 - precedenza tra task
 - incompatibilità delle risorse
 - capacità delle risorse

Applicazioni: Scheduling (2)

- **Esempio**

- M macchine, J job
- job k : sequenza di n_k task $T_{k,1}, \dots, T_{k,n_k}$ di durata $D_{k,1}, \dots, D_{k,n_k}$

```
solve( $T_{1,1}, \dots, T_{j,n_j}, \text{Cost}$ ) :-  
    precedence( $[T_{1,1}, \dots, T_{1,n_1}], [D_{1,1}, \dots, D_{1,n_1}]$ ),  
    ..., % uno per job  
    precedence( $[T_{j,1}, \dots, T_{j,n_j}], [D_{j,1}, \dots, D_{j,n_j}]$ ),  
    defineCost( $T_{1,1}, \dots, T_{j,n_j}, \text{Cost}$ ),  
    enumerate( $T_{1,1}, \dots, T_{j,n_j}$ ),  
    generateStartTime( $T_{1,1}, \dots, T_{j,n_j}$ ).
```

Applicazioni: DNA Sequencing

- **Sequenza DNA**

- sequenza finita sulle lettere $\{A, C, G, T\}$
- enzimi che partizionano la sequenza
- semplificazione: consideriamo le lunghezze

- **Esempio**

- due enzimi, A e B, che spezzano il DNA rispettivamente in $A_1, \dots, A_N$ e $B_1, \dots, B_M$
- usiamo i due enzimi contemporaneamente: $D_1, \dots, D_K$
- un frammento D_i è allineato o con un frammento A_j o con uno B_k

$$\text{len}(D_1 \dots D_i) = \text{len}(A_1 \dots A_j) \vee \text{len}(D_1 \dots D_i) = \text{len}(B_1 \dots B_k)$$

Applicazioni: DNA Sequencing (2)

```
solve(A, B, D, [AF|AR], [BF|BR], [DF|DR]) :-  
    chooseInit(A, B, D, AF, BF, DF, A2, B2, D2),  
    rs(A2, B2, D2, AF, BF, DF, AR, BR, DR).  
rs(A, B, D, LenA, LenB, LenD, [], [], []) :-  
    empty(A), empty(B), empty(D).  
rs(A, B, D, LenA, LenB, LenD, [Aj|AR], BR, [Di|DR]) :-  
    LenA = LenD, LenA < LenB,  
    Di <= LenB - LenA, Aj >= Di,  
    choose(Di, D, D2), choose(Aj, A, A2),  
    rs(A2, B, D2, LenA + Aj, LenB, LenD + Di, AR, BR, DR).  
rs(A, B, D, LenA, LenB, LenD, AR, [Bk|BR], [Di|DR]) :-  
    LenB = LenD, LenB < LenA,  
    Di <= LenA - LenB, Bk >= Di,  
    choose(Di, D, D2), choose(Bk, B, B2),  
    rs(A, B2, D2, LenA, LenB + Bk, LenD + Di, AR, BR, DR).
```

Limiti di CLP

- **Combinazione di domini**
 - un unico solver che comprende tutti i domini
 - combinazione di constraint solver differenti (mediatori, metalivelli)
- **Vincoli definiti dall'utente**
 - domini “built-in” nei sistemi CLP
 - capacità di definire nuovi vincoli e nuove regole di inferenza
- **Domini tempo-varianti**
 - limite ereditato da LP: assiomi come verità *immutabili*
 - necessità di esplicitare il parametro tempo

Temi di ricerca

- **Programmazione a vincoli orientata agli oggetti**
 - relazioni (ereditarietà, istanza, ecc.) come vincoli inter-oggetto
 - capacità di esprimere computazioni vincolate su domini non interpretati
 - vincoli definiti dall'utente
 - definizione e realizzazione di linguaggi a vincoli orientati agli oggetti
- **Vincoli sulle modifiche di stato**
 - domini tempo-varianti
 - tempo come parametro implicito/esplicito su cui esprimere vincoli

Conclusioni

- **CLP estende LP superandone molti limiti quali:**
 - dominio non interpretato
 - strategia di ricerca inefficiente (vincoli come test)
 - vincoli come ingresso/uscita
- **CLP: schema generale**
 - specializzabile definendo
 - *il dominio*
 - *il meccanismo di inferenza*
 - *il test di consistenza*
- **CLP e programmazione non-algoritmica**
- **Applicazioni tipiche IA con CLP:**
 - cutting stock
 - scheduling
 - sequencing

alta espressività, rapida prototipizzazione, buona efficienza
- **Limiti ed estensioni**