

## ***Parte II: L'Interpretazione Astratta***

Vedi: Jones-Nielson

[www.cs.unibo.it/~laneve/html/metodi1/paper2.ps](http://www.cs.unibo.it/~laneve/html/metodi1/paper2.ps)

I problemi:

***propagazione delle costanti*** : individuare le variabili all'interno di un programma che sono usate senza essere riassegnate (da rimpiazzare con costanti).

***variabili vive*** : una variabile  $x$  è *morta* ad un certo punto del programma se il suo valore non è più necessario da quel punto in poi, altrimenti è *viva* (le variabili morte possono essere riutilizzate per ottimizzare gli sprechi di memoria).

***analisi di interferenza*** : individuare frammenti di un programma che usano insiemi disgiunti di variabili (questi frammenti possono essere eseguiti in parallelo).

***terminazione di programmi*** (per alcuni casi particolari).

**osservazione.** Questi tipi di analisi sono fatte dai compilatori per ottimizzare il codice (cf. *data-flow analysis*).

L'interpretazione astratta può essere vista come *data-flow analysis formale*: le tecniche di verifica sono matematicamente corrette (e non empiriche, talvolta sbagliate, come accadeva nei primi compilatori).

**esempio:** *propagazione delle costanti*.

<code>x := h ;</code>	<code>(x: const, y: const)</code>
<code>if x = y</code>	<code>(x: const, y: const)</code>
<code>then x := f(y)</code>	<code>(x: var, y: const)</code>
<code>else x := g(y)</code>	<code>(x: var, y: const)</code>
<code>print(x)</code>	<code>(x: var, y: const)</code>



può essere ottimizzato come

```
x := h ;
if x = k
  then x := f(k)
  else x := g(k)
print(x)
```

osservazione: si può anche fare con type-checking!

esempio: *variabili vive*.

```
x := h ;
if x = k
  then x := f(k)
  else x := g(k)
print(x)
```

```
(x: live, y: dead)
(x: live, y: dead)
(x: live, y: dead)
(x: live, y: dead)
(x: live, y: dead)
(x: dead, y: dead)
```



## CASO DI STUDIO: *il problema di Collatz*

Verificare la terminazione (per ogni  $n$ ) del programma (ancora non risolto):

```
A:   while n ≠ 1 do
B:       if n % 2 = 0
C:           then   n := n/2 ;      D:
E:           else   n := 3×n + 1 ; F:
G:
```

Astrazione su una singola esecuzione : sia 5 il valore iniziale di  $n$ . Allora  $n$  prende i seguenti valori successivi ai vari punti del programma:

```
B:   5, 16, 8, 4, 2
C:   -, 16, 8, 4, 2
....
```

**osservazione:** visto che non siamo in grado di dimostrare la terminazione del problema di Collatz per ogni  $n$ , forse è possibile dimostrarla per sottoinsiemi "interessanti" dei naturali.

## tentativo (pari e dispari):

- "T"                    indica "pari o dispari" ( $\mathbb{N}$ ),  
"pari"                indica l'insieme dei *numeri pari*,  
"dispari"            indicare l'insieme dei *numeri dispari*,  
" "                    indica l'insieme vuoto.

allora abbiamo la seguente esecuzione (astratta):

n a A	n a B	n a C	n a D	n a E	n a F	n a G
dispari	dispari	pari	T	dispari	pari	dispari
T	T	pari	T	dispari	pari	dispari
T	T	pari	T	dispari	pari	dispari

conseguenza ovvia: **n è dispari a G**, semai il controllo raggiunge G !

**osservazione:** le *configurazioni astratte* sono sottoinsiemi delle possibili configurazioni reali.

**la speranza:** è possibile dimostrare proprietà interessanti dell'algoritmo per domini astratti.

## la definizione formale:

usiamo un dominio astratto:  $\{ \perp, \text{pari}, \text{dispari}, \top \}$  con la relazione d'ordine  $(\{ \perp, \text{pari}, \text{dispari}, \top \}, \leq)$  è un ordinamento parziale

definiamo le operazioni sul dominio astratto. Ad esempio:

$$n/2 = \begin{cases} \top & \text{se } n = \perp \\ \text{pari, dispari, } \top & \text{se } n = \text{pari, dispari, } \top \end{cases}$$

$$3 \times n + 1 = \begin{cases} \text{dispari} & \text{se } n = \text{pari} \\ \text{pari} & \text{se } n = \text{dispari} \\ \top & \text{se } n = \top \\ \perp & \text{se } n = \perp \end{cases}$$

la formalizzazione di questa tecnica:

la semantica astratta di Cousot

## la semantica astratta di Cousot

1. associa ad ogni punto del programma *una astrazione dell'insieme delle configurazioni di memoria* che possono presentarsi quando il controllo raggiunge quel punto;
2. definisce delle *operazioni di astrazione* che trasformano gli insiemi del punto 1, simulando il comportamento del programma. Molte (ma non tutte !) data-flow analysis sono state formalizzate in questo modo.
3. *dimostra la stabilità delle funzioni di astrazione* : da un certo punto in poi esse diventano stabili sugli insiemi delle configurazioni.

### osservazione:

- (a) la proprietà che si vuole dimostrare per il programma è espressa attraverso la stabilità di una funzione
- (b) la stabilità è "**computazionalmente interessante**" se è raggiungibile in tempo *ragionevole*, ad esempio, polinomiale.

## *un semplice linguaggio imperativo*

sintassi: il linguaggio consiste di *comandi*  $C$  ed *espressioni*  $E$  :

$C$	$::=$	<code>skip</code>	
		<code>x := E</code>	(assegnamento)
		<code>C ; C'</code>	(comp. sequenziale)
		<code>if E then C else C'</code>	(condizionale)
		<code>while E do C</code>	(iterazione)
$E$	$::=$	<code>x</code>	(variabili)
		<code>Z</code>	(numeri interi)
		<code>E # E</code>	
		<code>E</code>	
$\#$		<code>{+, -, , x , /, %, =, &lt; , &gt;}</code>	

**osservazione:** assumiamo che una memoria  $\Sigma$  mappa un identificatore ad un valore  $n$ .

la semantica concreta "a grandi passi". Si definiscono due funzioni (una per le espressioni e una per i comandi):  $\Sigma \Vdash E \quad n$ ,  $\Sigma \Vdash C \quad \Sigma'$  in maniera induttiva.

regole per le espressioni.

$$\begin{array}{c} \Sigma \Vdash n \quad n \\ \Sigma \Vdash E \quad n \quad \Sigma \Vdash E' \rightarrow n' \quad n = n' \# n'' \\ \hline \Sigma \Vdash E \# E' \quad n \end{array}$$

commenti:

1. non è possibile valutare espressioni le cui variabili non sono definite in memoria
2. la valutazione delle espressioni non modifica la memoria (non ci sono side-effects)

regole per i comandi. Formato  $\Sigma \Vdash C \quad \Sigma'$

$\Sigma \Vdash \text{skip} \quad \Sigma$

$$\frac{x \in \text{dom}(\Sigma) \quad \Sigma \Vdash E \quad s \quad \Sigma \Vdash c \quad \Sigma' \quad \Sigma' \Vdash c' \quad \Sigma''}{\Sigma \Vdash \mathbf{x} := E \quad \Sigma [x: s] \quad \Sigma \Vdash c; c' \quad \Sigma''}$$

$$\frac{\Sigma \Vdash E \quad n \quad n \neq 0 \quad \Sigma \Vdash c \quad \Sigma' \quad \Sigma \Vdash E \quad 0 \quad \Sigma \Vdash c' \quad \Sigma'}{\Sigma \Vdash \text{if } E \text{ then } C \text{ else } C' \quad \Sigma' \quad \Sigma \Vdash \text{if } E \text{ then } C \text{ else } C' \quad \Sigma'}$$

$$\frac{\Sigma \Vdash E \quad 0 \quad \Sigma \Vdash E \quad n \quad n \neq 0 \quad \Sigma \Vdash c \quad \Sigma' \quad \Sigma' \Vdash \text{while } E \text{ do } C \quad \Sigma''}{\Sigma \Vdash \text{while } E \text{ do } C \quad \Sigma \quad \Sigma \Vdash \text{while } E \text{ do } C \quad \Sigma''}$$

## la semantica astratta dei comandi

la semantica "a grandi passi". Si definiscono due funzioni (una per le espressioni e una per i comandi):  $\Sigma \Vdash E \quad S$ ,  $\Sigma \Vdash C \quad \Sigma$  in maniera induttiva.

regole per le espressioni.

$$\begin{array}{c}
 \Sigma \Vdash n \quad \{n\} \quad \Sigma \Vdash n \# n' \quad \{n''\} \quad (n \# n' = n'') \\
 \\
 \frac{\Sigma(x) = S}{\Sigma \Vdash x \quad S} \quad \frac{\Sigma \Vdash E \quad S \quad \Sigma \Vdash E' \quad S'}{\Sigma \Vdash E \# E' \quad \{n \mid n = n' \# n'', n' \in S, n'' \in S\}}
 \end{array}$$

commenti:

1. non è possibile valutare espressioni le cui variabili non sono definite in memoria
2. la valutazione delle espressioni non modifica la memoria (non ci sono side-effects)

regole per i comandi. Formato  $\Sigma \Vdash C \quad \Sigma'$

$\Sigma \Vdash \text{skip} \quad \Sigma$

$\Sigma \Vdash \mathbf{x} := \mathbf{E} \quad \{ \Sigma' \mid \Sigma \quad \Sigma \text{ and } \Sigma \Vdash \mathbf{x} := \mathbf{E} \quad \Sigma' \}$

$$\frac{\Sigma \Vdash C \quad \Sigma' \quad \Sigma' \Vdash C' \quad \Sigma''}{\Sigma \Vdash C ; C' \quad \Sigma''}$$

$\Sigma \Vdash \text{if } \mathbf{E} \text{ then } C \text{ else } C' \quad \{ \Sigma' \mid \Sigma \quad \Sigma \text{ and } \Sigma \Vdash \text{if } \mathbf{E} \text{ then } C \text{ else } C' \quad \Sigma' \}$

$\Sigma \Vdash \text{while } \mathbf{E} \text{ do } C \quad \{ \Sigma' \mid \Sigma \quad \Sigma \text{ and } \Sigma \Vdash \text{while } \mathbf{E} \text{ do } C \quad \Sigma' \}$

**esempio:** `if x>y then x:= x+1 else x:= x-1`

con memoria  $[x \mapsto \{0,1\} ; y \mapsto \{0,2\}]$

**conseguenza:** l'algoritmo astratto trasforma domini astratti.

**la tecnica generale (memoria con una sola variabile):**

*input dell'algoritmo:* sottoinsieme dei naturali (dominio astratto)

*output dell'algoritmo:* sottoinsieme dei naturali (dominio astratto)

## la *semantica astratta* di Cousot (un esempio)

Le possibili configurazioni del programma di Collatz sono sottinsiemi di:

$$[n : \{0, 1, 2, 3, 4\}]$$

che abbrevieremo con  $\{0, 1, 2, 3, 4, \dots\}$ .

Sia  $S_A$  la configurazione astratta iniziale. Le configurazioni nei vari punti sono:

$$S_B = (S_A \quad S_D \quad S_F) \setminus \{1\}$$

$$S_C = S_B \quad \{0, 2, 4, 6, 8, \dots\}$$

$$S_D = \{n/2 \quad | \quad n \in S_C\}$$

$$S_E = S_B \quad \{1, 3, 5, 7, 9, \dots\}$$

$$S_F = \{3 \times n + 1 \quad | \quad n \in S_E\}$$

$$S_G = (S_A \quad S_D \quad S_F) \setminus \{1\}$$

**osservazione:** queste equazioni possono essere derivate automaticamente (da un programma) **definendo le modifiche fatte dai singoli comandi.**

## proprietà del dominio astratto (caso dei naturali)

$(\mathcal{P}(\mathbb{N}), \subseteq, \cup, \cap)$  è un reticolo completo.

$\mathcal{P}(\mathbb{N})$  è l'insieme dei sottoinsiemi di  $\mathbb{N}$

**reticolo** significa che  $(\mathcal{P}(\mathbb{N}), \subseteq)$  è un ordinamento parziale tale che, per ogni  $A, B \in \mathcal{P}(\mathbb{N})$ , esistono  $A \cup B$  (il *least upper bound*) e  $A \cap B$  (il *greatest lower bound*)

**completo** significa che ogni collezione di insiemi (anche infinita) ha least upper bound e greatest lower bound.

**osservazione:** ogni reticolo completo ha elemento minimo (nel caso di sopra,  $\emptyset$ ) e massimo (nel caso di sopra,  $\mathbb{N}$ )

$(\mathbb{Z}, \leq)$  non è un reticolo completo.

**osservazione:** il sistema di equazioni di pg. 8

$$\begin{aligned}S_B &= (S_A \quad S_D \quad S_F) \setminus \{ 1 \} \\S_C &= S_B \quad \{ 0, 2, 4, 6, 8, \dots \} \\S_D &= \{ n/2 \quad | \quad n \quad S_C \} \\S_E &= S_B \quad \{ 1, 3, 5, 7, 9, \dots \} \\S_F &= \{ 3 \times n + 1 \quad | \quad n \quad S_E \} \\S_G &= (S_A \quad S_D \quad S_F) \quad \{ 1 \}\end{aligned}$$

può essere riscritto in questo modo:

$$\begin{aligned}S_B &= f_B (S_A, S_B, S_C, S_D, S_E, S_F, S_G) \\S_C &= f_C (S_A, S_B, S_C, S_D, S_E, S_F, S_G) \\S_D &= f_D (S_A, S_B, S_C, S_D, S_E, S_F, S_G) \\S_E &= f_E (S_A, S_B, S_C, S_D, S_E, S_F, S_G) \\S_F &= f_F (S_A, S_B, S_C, S_D, S_E, S_F, S_G) \\S_G &= f_G (S_A, S_B, S_C, S_D, S_E, S_F, S_G)\end{aligned}$$

dove  $f_B, f_C, f_D, f_E, f_F, f_G$  sono opportune funzioni.

**osservazione:** risolvere il sistema di sopra significa trovare un insieme di valori per cui  $(S_A, S_B, S_C, S_D, S_E, S_F, S_G) = F(S_A, S_B, S_C, S_D, S_E, S_F, S_G)$

**definizione:**

**punto fisso** : ogni A per cui  $A = f(A)$

**funzione monotona (crescente)** :  $A \leq B \quad f(A) \leq f(B)$

Il teorema importante:

**Teorema di Tarski:** *una funzione monotona crescente su un reticolo completo ha un reticolo completo di punti fissi.*

Prova: dimostrare che

- (1) **un punto fisso almeno esiste.** Si consideri l'insieme  $P = \{ x \mid x \geq f(x) \}$  (insieme dei post-fixed points) e sia  $H = \bigcap_{x \geq f(x)} x$ , allora  $H \leq x$ , per ogni  $x \in P$ . Quindi per monotonia e per definizione di P,

$$f(H) \leq f(x) \leq x, \text{ per ogni } x \in P$$

Poichè H è il più grande dei minoranti di P, deve essere  $H \geq f(H)$ . Ma da questa equazione deriva che  $f(H) \geq f(f(H))$ , quindi anche  $f(H) \in P$ . Perciò  $H \leq f(H)$ .

- (2) questo è il minimo punto fisso (facile perchè ogni punto fisso appartiene a  $P$ );
- (3) se una collezione  $A$  di punti fissi esiste allora il loro least upper bound (unione) e il loro greatest lower bound (intersezione) sono punti fissi: per l'unione, prendere  $s = \bigcup A$  e sia  $B = \{ y \mid s \leq y \}$  e mostrare che  $B$  è un reticolo completo. Quindi, poichè  $f : B \rightarrow B$  è monotona, esiste  $s'$  minimo punto fisso di  $f$  su  $B$ . Ogni maggiorante di  $A$  e punto fisso deve essere maggiore di  $s'$ .  $\square$

Le equazioni di prima hanno soluzione perchè si usano funzioni monotone su un reticolo completo.

**problema:** *come fare a calcolare il risultato ?* Il sistema di equazioni non si può risolvere direttamente perchè l'insieme  $S_A$  non è noto.

**funzione continua** :  $f$  è continua se

$$f(\bigcup_i A_i) = \bigcup_i f(A_i)$$

**Teorema di continuità di Kleene:** *una funzione monotona continua su un reticolo completo ha minimo punto fisso  $\bigcup_n f^n(\perp)$ .*

(nel caso di  $(\mathbb{N}, \subseteq, \cup, \cap), \perp = \emptyset$ )

**osservazione:** il teorema di continuità fornisce un algoritmo per calcolare il minimo punto fisso:

$$\perp \rightsquigarrow f^1(\perp) \rightsquigarrow f^2(\perp) \rightsquigarrow f^3(\perp) \rightsquigarrow f^4(\perp) \rightsquigarrow \dots$$

**problema:** l'algoritmo può non fornire il punto fisso in tempo finito.

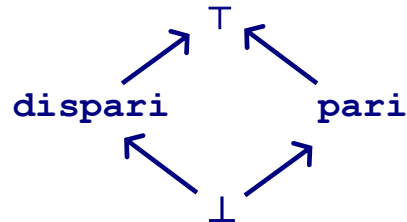
*esempio.*  $f : A \mapsto A \cup \{ \max(A)+1 \}$

$\rightsquigarrow \{ 0 \} \rightsquigarrow \{ 0, 1 \} \rightsquigarrow \{ 0, 1, 2 \} \rightsquigarrow \{ 0, 1, 2, 3 \} \rightsquigarrow \dots$

**osservazione:** se il dominio non ha catene crescenti infinite la stabilità si raggiunge in tempo finito!

ragionare su domini astratti finiti!

**esempio** (programma di Collatz, cont.) Il dominio astratto è:



che è un reticolo completo finito.

sia la funzione di astrazione così definita:

$$(S) = \begin{cases} \perp & \text{se } S = \\ \text{pari} & \text{se } S \subseteq \{ 0, 2, 4, \dots \} \\ \text{dispari} & \text{se } S \subseteq \{ 1, 3, 5, \dots \} \\ \top & \text{altrimenti} \end{cases}$$

il sistema di equazioni diventa:

$$\text{abs}_A = (S_A)$$

$$\text{abs}_B = (\text{abs}_A \quad \text{abs}_D \quad \text{abs}_F) \quad \top$$

$$\text{abs}_C = \text{abs}_B \quad \text{pari}$$

$$\text{abs}_D = f_{n/2}(\text{abs}_C)$$

$$\text{abs}_E = \text{abs}_B \quad \text{dispari}$$

$$\text{abs}_F = f_{3n+1}(\text{abs}_E)$$

$$\text{abs}_G = (\text{abs}_A \quad \text{abs}_D \quad \text{abs}_F) \quad \text{dispari}$$

dove

$$f_{n/2}(S) = \begin{cases} \perp & \text{se } S = \perp \\ T & \text{altrimenti} \end{cases} \quad f_{3n+1}(S) = \begin{cases} \perp & \text{se } S = \perp \\ \text{pari} & \text{se } S = \text{dispari} \\ \text{dispari} & \text{se } S = \text{pari} \\ T & \text{altrimenti} \end{cases}$$

Il calcolo iterativo del punto fisso: (input `dispari`)

<code>abs<sub>A</sub></code>	<code>abs<sub>B</sub></code>	<code>abs<sub>C</sub></code>	<code>abs<sub>D</sub></code>	<code>abs<sub>E</sub></code>	<code>abs<sub>F</sub></code>	<code>abs<sub>G</sub></code>	iterazione
<code>⊥</code>	<code>⊥</code>	<code>⊥</code>	<code>⊥</code>	<code>⊥</code>	<code>⊥</code>	<code>⊥</code>	0
<code>dispari</code>	<code>⊥</code>	<code>⊥</code>	<code>⊥</code>	<code>⊥</code>	<code>⊥</code>	<code>⊥</code>	1
<code>dispari</code>	<code>dispari</code>	<code>⊥</code>	<code>⊥</code>	<code>⊥</code>	<code>⊥</code>	<code>dispari</code>	2
<code>dispari</code>	<code>dispari</code>	<code>⊥</code>	<code>⊥</code>	<code>dispari</code>	<code>⊥</code>	<code>dispari</code>	3
<code>dispari</code>	<code>dispari</code>	<code>⊥</code>	<code>⊥</code>	<code>dispari</code>	<code>pari</code>	<code>dispari</code>	4
<code>dispari</code>	<code>T</code>	<code>⊥</code>	<code>⊥</code>	<code>dispari</code>	<code>pari</code>	<code>dispari</code>	5
<code>dispari</code>	<code>T</code>	<code>pari</code>	<code>⊥</code>	<code>dispari</code>	<code>pari</code>	<code>dispari</code>	6
<code>dispari</code>	<code>T</code>	<code>pari</code>	<code>T</code>	<code>dispari</code>	<code>pari</code>	<code>dispari</code>	7, 8, ...

(input pari)

abs <sub>A</sub>	abs <sub>B</sub>	abs <sub>C</sub>	abs <sub>D</sub>	abs <sub>E</sub>	abs <sub>F</sub>	abs <sub>G</sub>	iterazione
⊥	⊥	⊥	⊥	⊥	⊥	⊥	0
pari	⊥	⊥	⊥	⊥	⊥	⊥	1
pari	pari	⊥	⊥	⊥	⊥	⊥	2
pari	pari	pari	⊥	⊥	⊥	⊥	3
pari	pari	pari	T	⊥	⊥	⊥	4
pari	T	pari	T	⊥	⊥	dispari	5
pari	T	pari	T	dispari	⊥	dispari	6
pari	T	pari	T	dispari	pari	dispari	7, 8, ...

**osservazione:** lo stato alla posizione F è sempre **pari**. Quindi, nel caso di iterazione, il test " $n \% 2 = 0$ " è sempre verificato. Dunque il programma si può ottimizzare come segue:

```
A:  while n ≠ 1 do
B:      if n % 2 = 0
C:          then    n := n/2 ;      D:
E:          else    n := 3×n + 1 ; n := n/2 ; F:
G:
```

che evita due test ogni volta che  $n$  è dispari. L'interpretazione astratta consente di validare una simile ottimizzazione.

**osservazione (dead code).** I valori "stabili" dei domini astratti nei vari punti del programma non sono mai " $\perp$ ". Infatti " $\perp$ " non rappresenta alcuna configurazione reale. Quando il valore stabile di un punto del programma è " $\perp$ " significa che quel punto non è mai raggiungibile e quindi il relativo codice può essere rimosso.

Dunque



è un dominio astratto importante per la raggiungibilità del codice.

**esercizio.** Verificare che il codice seguente

```
if E then C else if E then C' else C''
```

può essere ottimizzato in

```
if E then C else C'
```

**Problema: *quando una verifica con l'interpretazione astratta è corretta?***

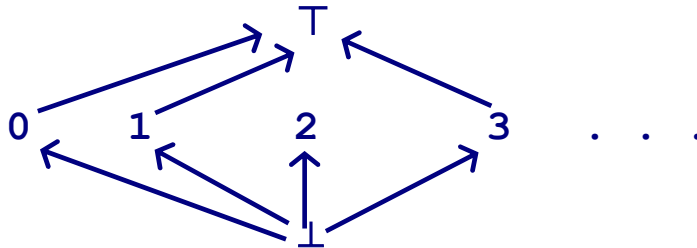
**risposta** : bisogna stabilire una relazione tra

- 1. il dominio concreto e il dominio astratto*
- 2. la trasformazione concreta dell' input durante la computazione e quella astratta.*

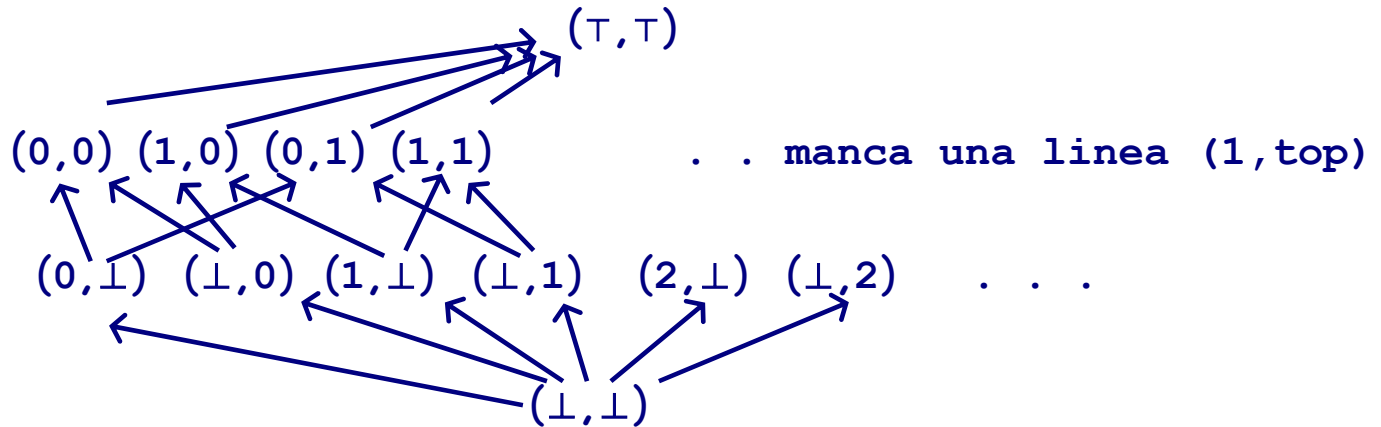
## Correttezza della verifica con Interpretazione Astratta: Astrazioni e Concretizzazioni

Il dominio concreto che abbiamo usato per il semplice linguaggio imperativo (pagina 8) consiste di t-uple di naturali (t-uple perchè le variabili di un programma possono essere più d'una).

il dominio concreto di una variabile:



il dominio concreto di due variabili:



(ogni punto del reticolo è una coppia di valori, uno per la prima variabile e uno per la seconda).

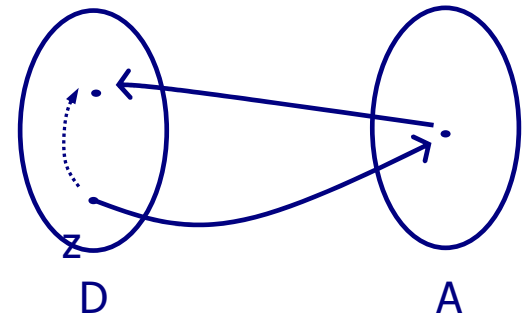
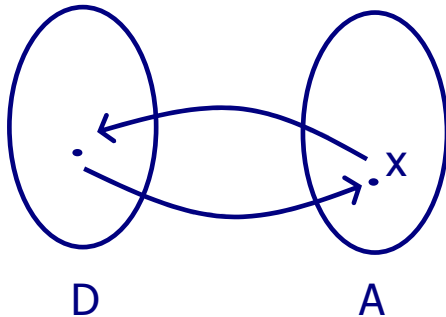
**definizione (astrazione e concretizzazione).** Un dominio astratto  $(A, \leq_A)$  è **consistente** rispetto a un dominio concreto  $(D, \leq_D)$  se esistono due funzioni  $\alpha : D \rightarrow A$  (astrazione) e  $\gamma : A \rightarrow D$  (concretizzazione) tale che

1.  $\alpha$  e  $\gamma$  preservano l'ordine; cioè

$$d \leq_D d' \implies \alpha(d) \leq_A \alpha(d') \quad \text{e} \quad a \leq_A a' \implies \gamma(a) \leq_D \gamma(a')$$

2. per ogni  $x \in A$ ,  $x = \alpha(\gamma(x))$

3. per ogni  $z \in D$ ,  $z \leq_D \gamma(\alpha(z))$



**esempio:** astrazione e concretizzazione nel dominio astratto per il programma di Collatz:

$$\begin{array}{l}
 (x) = \begin{cases} \perp \\ \text{pari} \\ \text{dispari} \\ \top \end{cases} & \begin{array}{l} \text{se } x = \\ \text{se } x \subseteq \{ 0, 2, 4, \dots \} \\ \text{se } x \subseteq \{ 1, 3, 5, \dots \} \\ \text{altrimenti} \end{array} \\
 \\
 (z) = \begin{cases} \{ 0, 2, 4, \dots \} \\ \{ 1, 3, 5, \dots \} \\ \mathbb{N} \end{cases} & \begin{array}{l} \text{se } z = \perp \\ \text{se } z = \text{pari} \\ \text{se } z = \text{dispari} \\ \text{se } z = \top \end{array}
 \end{array}$$

**osservazione:** al punto 3 c' è " c " invece che "=" perchè la funzione di astrazione "dimentica delle informazioni". Quando si torna indietro con la concretizzazione si perde la "precisione" (pur preservando l'ordine).

di solito: è surgettiva, è iniettiva. (c.f. connessioni di Galois)

## la correttezza delle trasformazioni di stato

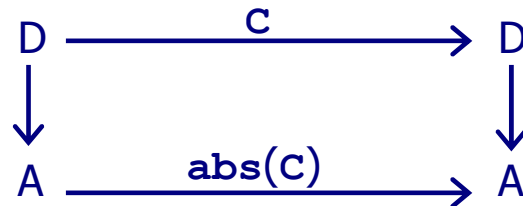
Come si evince dalla semantica del linguaggio imperativo, l'esecuzione di un comando ha come unico effetto quello di modificare la memoria.

Per interpretare questo effetto nel dominio astratto occorre definire la semantica astratta dei comandi (vedi pg. 9-10)

Siano  $(D, \_D)$  e  $(A, \_A)$  il *dominio concreto* ed il *dominio astratto*. Sia inoltre  $\text{abs}(C)$  la semantica astratta del programma  $C$ .

**che proprietà deve soddisfare  $\text{abs}(C)$  affinché sia corretta?**

**1. risposta "naïve"**: deve commutare il seguente diagramma ( $C$  è un comando):



Questa proprietà è **troppo forte** per molte situazioni.

esempio dell'inadeguatezza con il dominio concreto dei numeri naturali ed il dominio astratto del programma di Collatz ( $\{\perp, \text{dispari}, \text{pari}, \top\}$ ).

Sia

$$c = x := x/2$$

allora

$$\text{abs}(c)(\Sigma) = \begin{cases} \perp & \text{se } (\Sigma)(x) = \perp \\ \top & \text{se } (\Sigma)(x) \neq \perp \end{cases}$$

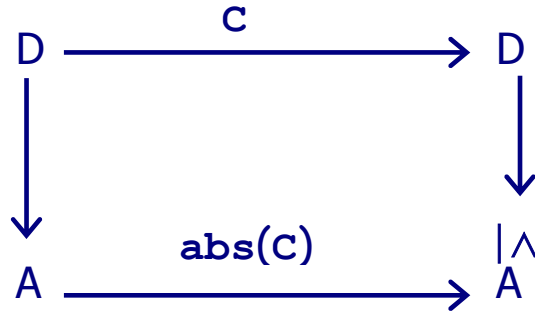
Ora sia  $\Sigma = [x : 2]$ . Allora, per la semantica operativa definita in precedenza:

$$\Sigma \Vdash c \quad * \quad [x : 1]$$

quindi  $c(\Sigma) = [x : 1]$ . D'altra parte  $\text{abs}(c)(\Sigma) = \top$ . Quindi

$$(c(\Sigma)) = \text{dispari} \quad \top = \text{abs}(c)(\Sigma).$$

2. **risposta più accurata:** il diagramma che deve commutare è il seguente:



difatti, nell'esempio di sopra otteniamo:

$$(c(\Sigma)) = \text{dispari} \quad \top = \text{abs}(c)(\Sigma).$$

**Motivazione intuitiva per il " ":** la computazione sul dominio astratto è meno accurata rispetto alla computazione del dominio concreto ("essere `dispari`" è meno preciso di "essere `⊤`"). Dunque,

*partire dal dominio concreto, astrarre e calcolare in maniera astratta è meno preciso che calcolare concretamente e poi astrarre.*

### 3. **risposta definitiva:** i domini astratti sono **insiemi di configurazioni concrete**.

le computazioni del dominio concreto non possono essere simulate singolarmente poichè il dominio astratto non ha informazioni sufficienti.

*esempio:*  $C = \text{if } x > y \text{ then } x := x+y \text{ else } x := x-y$

di solito il dominio astratto non ha info sufficienti per determinare il risultato di un test: la semantica astratta confonde i due rami dell' if:

$$\text{abs}(\text{if } E \text{ then } C \text{ else } C')(\Sigma) = \\ [\text{abs}(C)(\Sigma_E \cap \Sigma)] \cup [\text{abs}(C')(\Sigma_{\neg E} \cap \Sigma)]$$

dove  $\Sigma$  è una configurazione astratta e  $\Sigma_E$  ed  $\Sigma_{\neg E}$  sono definiti appropriatamente.

**conseguenza operativa:** i domini astratti devono contemplare **insiemi di configurazioni concrete** piuttosto che singole configurazioni.

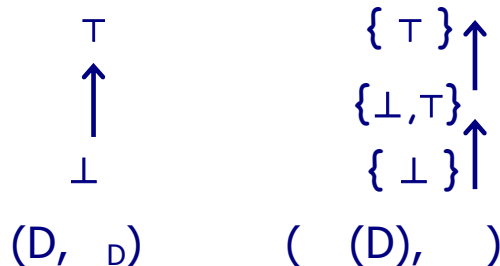
Il più concreto dominio astratto è l'insieme dei sottoinsiemi delle configurazioni

**definizione (reticolo power-set).** *Sia  $(D, \sqsubseteq_D)$  un reticolo completo. Allora  $(\mathcal{P}(D), \sqsubseteq_{\mathcal{P}(D)})$  è il reticolo completo definito come segue:*

*$\mathcal{P}(D)$  è l'insieme dei sottoinsiemi di  $D$ ;*

*$S \sqsubseteq_{\mathcal{P}(D)} S'$  se e solo se, per ogni  $d \in S$ , esiste  $d' \in S'$  tale che  $d \sqsubseteq_D d'$  e, viceversa, per ogni  $d' \in S'$ , esiste  $d \in S$  tale che  $d \sqsubseteq_D d'$ .*

**osservazione:**  $(\mathcal{P}(D), \sqsubseteq_{\mathcal{P}(D)})$  è un reticolo completo. Esempio:



(esistono altre definizioni del reticolo power-set: cf. *power-set di Scott*)

**conclusione:** *definire astrazione e concretizzazione su  $(D, \dots)$ ,  
cioè:*

$\alpha : (D) \rightarrow A$  (*astrazione*)

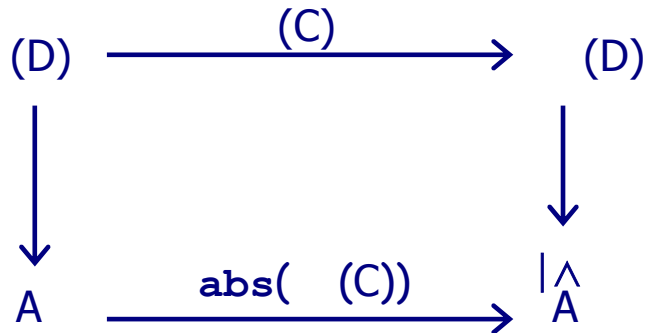
$\gamma : A \rightarrow (D)$  (*concretizzazione*)

con le solite proprietà.

La trasformazione di domini astratti è

$$(C) : S \mapsto \{ d' \mid d \in S \text{ e } d \Vdash C \ast d' \}$$

per cui, **deve commutare** il seguente diagramma:

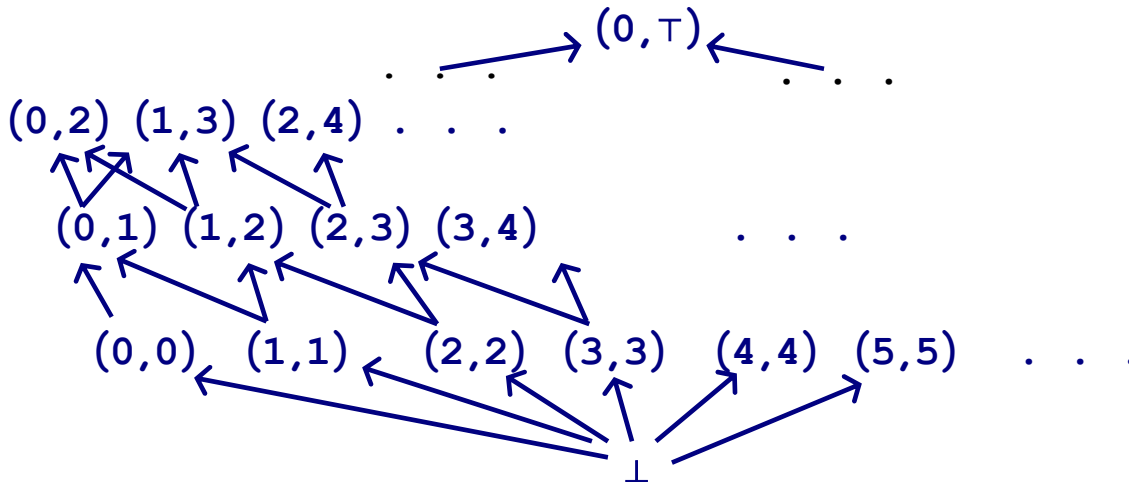


## Widening e Narrowing

L'approccio all'interpretazione astratta discusso finora presuppone domini astratti **con catene crescenti finite** (per rendere effettivo l'algoritmo della ricerca del minimo punto fisso).

*purtoppo questa restrizione è troppo forte.*

**esempio:** un programma che opera sugli array non dovrebbe mai accedere al di fuori del range degli indici. Il dominio astratto per verificare tale proprietà dovrebbe essere  $(\mathbb{N} \cup \{\top\})^2 \cup \{\perp\}$ :



*Osservazione* : non è possibile astrarre ulteriormente su questo dominio senza perdere informazioni utili sugli estremi degli array.

Si consideri il seguente frammento di codice, detto  $C_{m,n}$ :

```
(a)   i := m ;                               (b)  
      while (c) i < n do  
(d)         i := i+1 ;                       (e)  
(f)
```

Il sistema di equazioni corrispondente è:

$$A_{(b)} = [m, m]$$

$$A_{(c)} = A_{(b)} \cup A_{(e)}$$

$$A_{(d)} = A_{(c)} \cap [0, n]$$

$$A_{(e)} = A_{(d)} + [1, 1]$$

$$A_{(f)} = A_{(c)} \cap [n+1, \tau]$$

dove l'operazione "+" su intervalli è definita da:

$$[i, j] + [h, k] = [i+h, j+k]$$

$$X + Y = \perp \quad (\text{se } X=\perp \text{ oppure } Y=\perp)$$

la soluzione è la seguente:

$A_{(a)}$	$A_{(b)}$	$A_{(c)}$	$A_{(d)}$	$A_{(e)}$	$A_{(f)}$	iterazione
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	0
$\perp$	$[m,m]$	$\perp$	$\perp$	$\perp$	$\perp$	1
$\perp$	$[m,m]$	$[m,m]$	$\perp$	$\perp$	$\perp$	2
$\perp$	$[m,m]$	$[m,m]$	$[m,m]$	$\perp$	$\perp$	3
$\perp$	$[m,m]$	$[m,m]$	$[m,m]$	$[m+1,m+1]$	$\perp$	4
$\perp$	$[m,m]$	$[m,m+1]$	$[m,m+1]$	$[m+1,m+1]$	$\perp$	5
$\perp$	$[m,m]$	$[m,m+1]$	$[m,m+1]$	$[m+1,m+2]$	$\perp$	6
...	...	...	...	...	...	...
$\perp$	$[m,m]$	$[m, n+1]$	$[m,n]$	$[m+1,n+1]$	$[n+1, n+1]$	k

raggiunge la forma stabile dopo k iterazioni (k lineare rispetto a n-m)

quindi ogni elemento del dominio è rilevante.

Adesso si consideri il seguente programma:

```
(a)  i := m ; (b)
      while (c) i < n do
(d)      i := i+1 ; (e)
(f)
```

Il sistema di equazioni corrispondente è:

$$A_{(b)} = [m, m]$$

$$A_{(c)} = A_{(b)} \cup A_{(e)}$$

$$A_{(d)} = A_{(c)} \cap [n, \top]$$

$$A_{(e)} = A_{(d)} + [1, 1]$$

$$A_{(f)} = A_{(c)} \cap [0, n-1]$$

la soluzione è la seguente:  $(m \quad n)$

$A_{(a)}$	$A_{(b)}$	$A_{(c)}$	$A_{(d)}$	$A_{(e)}$	$A_{(f)}$	iterazione
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	0
$\perp$	$[m,m]$	$\perp$	$\perp$	$\perp$	$\perp$	1
$\perp$	$[m,m]$	$[m,m]$	$\perp$	$\perp$	$\perp$	2
$\perp$	$[m,m]$	$[m,m]$	$[m,m]$	$\perp$	$\perp$	3
$\perp$	$[m,m]$	$[m,m]$	$[m,m]$	$[m+1,m+1]$	$\perp$	4
$\perp$	$[m,m]$	$[m,m+1]$	$[m,m+1]$	$[m+1,m+1]$	$\perp$	5
$\perp$	$[m,m]$	$[m,m+1]$	$[m,m+1]$	$[m+1,m+2]$	$\perp$	6
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
$\perp$	$[m,m]$	$[m, \top]$	$[m, \top]$	$[m+1, \top]$	$\perp$	$+\infty$

non raggiunge la forma stabile in tempo finito



usare un operatore che acceleri la convergenza

definiamo l'operatore di **widening**  $\nabla$  di intervalli nel modo seguente:

$$\perp \nabla X = X$$

$$X \nabla \perp = X$$

$$[i, j] \nabla [h, k] = [ \text{if } h < i \text{ then } 0 \text{ else } i, \\ \text{if } k > j \text{ then } \top \text{ else } j ]$$

Se modifichiamo il sistema di equazioni sostituendo

$$A_{(c)} = A_{(b)} \cup A_{(e)}$$

con

$$A_{(c)} = A_{(c)} \nabla (A_{(b)} \cup A_{(e)})$$

si ottiene:

$A_{(a)}$	$A_{(b)}$	$A_{(c)}$	$A_{(d)}$	$A_{(e)}$	$A_{(f)}$	iterazione
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	0
$\perp$	$[m,m]$	$\perp$	$\perp$	$\perp$	$\perp$	1
$\perp$	$[m,m]$	$[m,m]$	$\perp$	$\perp$	$\perp$	2
$\perp$	$[m,m]$	$[m,m]$	$[m,m]$	$\perp$	$\perp$	3
$\perp$	$[m,m]$	$[m,m]$	$[m,m]$	$[m+1,m+1]$	$\perp$	4
$\perp$	$[m,m]$	$[m, \top]$	$[m,n]$	$[m+1,m+1]$	$\perp$	5
$\perp$	$[m,m]$	$[m, \top]$	$[m, \top]$	$[m+1,n+1]$	$\perp$	6
$\perp$	$[m,m]$	$[m, \top]$	$[m, \top]$	$[m+1, \top]$	$\perp$	7, ...

cioè la sequenza converge alla 7<sup>o</sup> iterazione.

## la teoria dell'operatore di widening

**Definizione.** Dato un reticolo completo  $(L, \leq)$ ,  $W : L \times L \rightarrow L$  è un operatore di *widening* se

1. per ogni  $x, y \in L$ ,  $x \leq W(x, y)$  e  $y \leq W(x, y)$
2. per ogni catena crescente  $x_0 \leq x_1 \leq x_2 \leq \dots$ , la catena definita da  $z_0 = x_0$ ,  $z_1 = W(z_0, x_1)$ ,  $z_2 = W(z_1, x_2)$ , ... non è strettamente crescente.

**Proposizione.** Se  $\nabla$  è un operatore di widening ed  $f$  una funzione monotona e continua, allora la sequenza

$$z_0 = \perp$$
$$z_{n+1} = z_n \nabla f(z_n)$$

si stabilizza ed il suo limite  $X$  è un pre-punto-fisso di  $f$  ( $f(X) \leq X$ ) che è maggiore del minimo punto fisso di  $f$ .

Prova. Si consideri la catena crescente

$$\perp \leq f(\perp) \leq f^2(\perp) \leq f^3(\perp) \leq \dots$$

e sia  $F = \bigcup_i \mathbb{N} f^i(\perp)$ .

Allora la sequenza stabilizzante

$$\perp \leq \perp \nabla f(\perp) \leq (\perp \nabla f(\perp)) \nabla f^2(\perp) \leq ((\perp \nabla f(\perp)) \nabla f^2(\perp)) \nabla f^3(\perp) \leq \dots$$

$\parallel$       $\parallel$

$\parallel$

$\parallel$

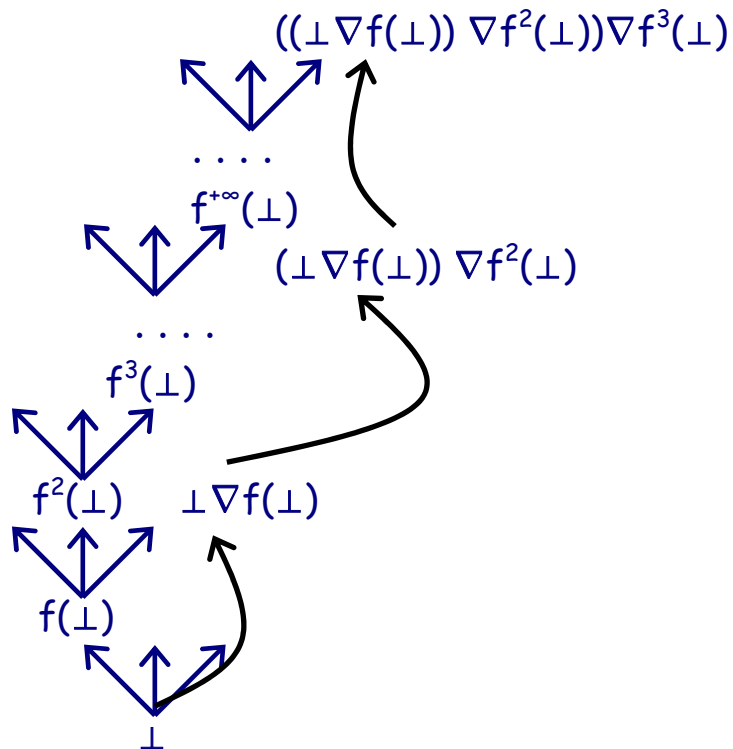
$z_0$       $z_1$

$z_2$

$z_3$

è tale che  $f^i(\perp) \leq z_i$ , per ogni  $i$ . Quindi  $F \leq X = \bigcup_i \mathbb{N} z_i$ . Inoltre, si osservi che, per definizione di widening e di  $X$ ,  $f(X) \leq X$ . ✓

graficamente:



## osservazione:

1. l'operatore di widening restituisce un pre-punto-fisso. Questi pre-punti-fissi sono molto importanti per le funzioni monotone (su reticoli completi) perchè hanno un minimo, che è il *minimo punto fisso*.
2. l'operatore di widening va inserito in tutti i punti in cui il programma cicla (tipicamente prima della guardia di un while o di una chiamata ricorsiva)
3. l'operatore di widening è un operatore ad-hoc. Individuare il corretto operatore può non essere banale.

## come progettare un buon operatore di widening

**Tecnica 1.** Sia  $(\mathcal{D}, \sqsubseteq)$  il dominio concreto, e sia  $(A, \sqsubseteq_A)$  quello astratto, con  $\alpha$  e  $\gamma$  le due funzioni di *astrazione* e *concretizzazione*. Allora l'operatore

$$x \nabla y = \alpha(\alpha(x) \cup \alpha(y)) \quad (1)$$

è un operatore di widening se il dominio astratto è finito (si basa sul fatto che, per definizione di astrazione e concretizzazione,  $\alpha(\gamma(z)) \sqsubseteq z$ ). Dimostrarlo!

*esempio (analisi degli intervalli con la regola dei segni).* Si consideri il seguente reticolo



con le funzioni

$$(x) = \begin{cases} \perp & \text{se } x = \perp \\ 0 & \text{se } x = [0, 0] \\ & \text{altrimenti} \end{cases}$$

$$(y) = \begin{cases} \perp & \text{se } y = \perp \\ [0, 0] & \text{se } y = 0 \\ [0, \top] & \text{se } y = \end{cases}$$

l'operatore di *widening*  $\nabla$  di intervalli secondo la formula (1) è definito

da:

$$\perp \nabla X = X$$

$$X \nabla \perp = X$$

$$X \nabla Y = \text{if } (X=0) \ \&\& \ (Y=0) \ \text{then } 0 \\ \text{else}$$

che, applicato al sistema di equazioni di prima (a sx), si ottiene quello di dx

$$A_{(b)} = [m, m]$$

$$A_{(b)}$$

$$A_{(c)} = A_{(b)} \cup A_{(e)}$$

$$A_{(c)} = A_{(c)} \nabla (A_{(b)} \cup A_{(e)})$$

$$A_{(d)} = A_{(c)} \cap [0, n]$$

$$A_{(d)} = A_{(c)} \cap$$

$$A_{(e)} = A_{(d)} + [1, 1]$$

$$A_{(e)} = A_{(d)} +$$

$$A_{(f)} = A_{(c)} \cap [n+1, \top]$$

$$A_{(f)} = A_{(c)} \cap$$

che si risolve in questo modo:

$A_{(a)}$	$A_{(b)}$	$A_{(c)}$	$A_{(d)}$	$A_{(e)}$	$A_{(f)}$	iterazione
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	0
$\perp$		$\perp$	$\perp$	$\perp$	$\perp$	1
$\perp$			$\perp$	$\perp$	$\perp$	2
$\perp$				$\perp$		3
$\perp$						4

**osservazione:** questa tecnica sembra artificiosa perchè l'interpretazione astratta coi segni si ottiene come composizione del dominio "meno" astratto degli intervalli.

(si veda Cousot&Cousot: "*Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation*".)

**Tecnica 2.** L'idea del widening è di eliminare componenti instabili durante una iterazione. Un modo brutale per forzare il widening è:

```
x ∇ y      if y ≤ x then x else ⊤
```

Dunque il widening esiste sempre, ma non è una argomentazione convincente.

Un buon approccio è di usare una **tecnica di estrapolazione**, che limita il numero massimo di iterazioni ad un naturale  $n$ . Ad esempio:

```
(x, i) ∇ (y, i+1)      if (y ≤ x && i ≤ n) then (x, i+1)  
                       else if i ≤ n then (x ∪ y, i+1)  
                       else (⊤, ⊤)
```

**osservazione.** La tecnica del widening può dare risultati peggiori che se si considerano domini astratti finiti. Ad esempio: il  $\top$  è il punto fisso di ogni funzione monotona su un reticolo completo.

Per ovviare a tale problema sono stati studiati gli operatori di narrowing.

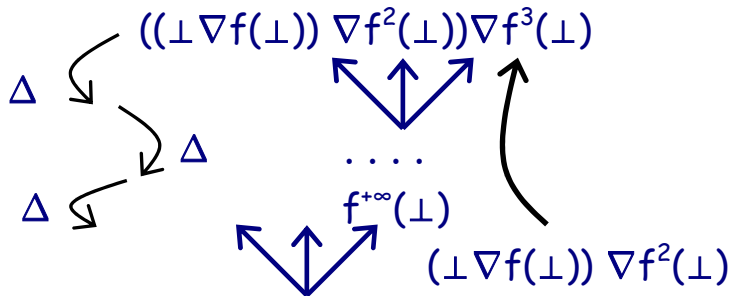
# Narrowing

la tecnica del widening consente di approssimare il punto fisso *per eccesso*

questa approssimazione può essere poco significativa (si pensi al  $\top$ ).

Per cercare approssimazioni più interessanti: gli *operatori di narrowing*

**graficamente:** un operatore di narrowing  $\Delta$  consente di "scendere" verso il punto fisso.



**definizione.** Dato un reticolo completo  $(L, \leq)$ ,  $N : L \times L \rightarrow L$  è un operatore di narrowing se

1. per ogni  $x, y \in L$ ,  $x \geq N(x, y) \geq y$  ;

2. per ogni catena decrescente  $x_0 \geq x_1 \geq x_2 \geq \dots$ , la catena definita da  $z_0 = x_0$ ,  $z_1 = N(z_0, x_1)$ ,  $z_2 = N(z_1, x_2)$ , ... non è strettamente decrescente.

**esempio (analisi degli intervalli):** sia  $\Delta$  il seguente operatore

$$\perp \Delta X = \text{non definito quando } X \neq \perp$$

$$X \Delta \perp = \perp$$

$$[i, j] \Delta [h, k] = [ \text{if } h \geq i \text{ then } i, \text{ if } j \geq k \text{ then } j ]$$

quindi il  $\Delta$  semplicemente scarta l'estremo superiore e non migliora gli estremi finiti.

Nell'analisi che abbiamo fatto, avevamo inizialmente il sistema di equazioni

$$A_{(b)} = [m, m]$$

$$A_{(c)} = A_{(b)} \cup A_{(e)}$$

$$A_{(d)} = A_{(c)} \cap [0, n]$$

$$A_{(e)} = A_{(d)} + [1, 1]$$

$$A_{(f)} = A_{(c)} \cap [n+1, \top]$$

E, sostituendo  $A_{(c)} = A_{(b)} \cup A_{(e)}$  con  $A_{(c)} = A_{(c)} \nabla (A_{(b)} \cup A_{(e)})$  si otteneva la soluzione:

$A_{(a)}$	$A_{(b)}$	$A_{(c)}$	$A_{(d)}$	$A_{(e)}$	$A_{(f)}$	iterazione
$\perp$	$[m, m]$	$[m, \top]$	$[m, n]$	$[m+1, n+1]$	$[n+1, \top]$	6, 7, ...

Applicando a questa soluzione il narrowing, sostituendo l'equazione di  $A_{(c)}$  del sistema di sopra con

$$A_{(c)} = A_{(c)} \Delta (A_{(b)} \cup A_{(e)})$$

otteniamo la soluzione:

$A_{(a)}$	$A_{(b)}$	$A_{(c)}$	$A_{(d)}$	$A_{(e)}$	$A_{(f)}$	iterazione
$\perp$	$[m,m]$	$[m, \top]$	$[m,n]$	$[m+1,n+1]$	$[n+1,\top]$	1
$\perp$	$[m,m]$	$[m, n+1]$	$[m,n]$	$[m+1,n+1]$	$[n+1,\top]$	2, 3, ...

e si osservi che questa era la soluzione che si otteneva con l'iterazione infinita (senza widening e narrowing).