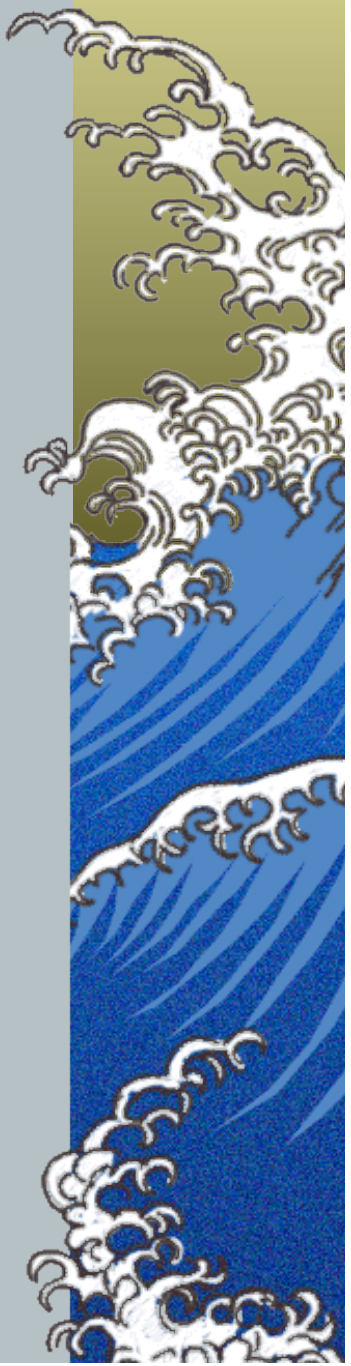
Theorem proving via Resolution

- insieme di fbf (logica del prim'ordine)

  --> clausole

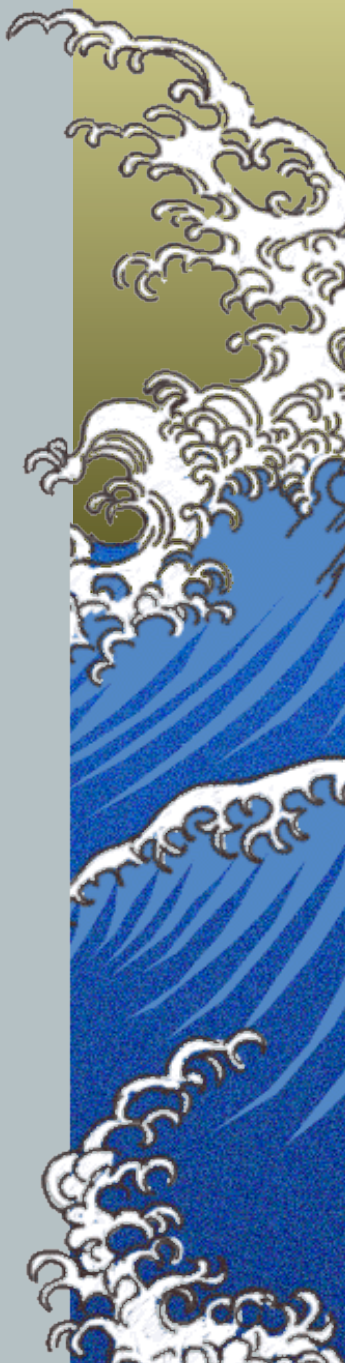
  - skolemizzazione, eliminazione quantificatori universali
  - equivalente per l'insoddisfacibilità
- dimostrazioni per assurdo
  - la formula da provare si nega
  - si tenta di derivare la contraddizione (clausola vuota)
- la risoluzione di Robinson è una regola corretta e completa
  - insoddisfacibilità <--> derivazione clausola vuota



La Risoluzione di Robinson

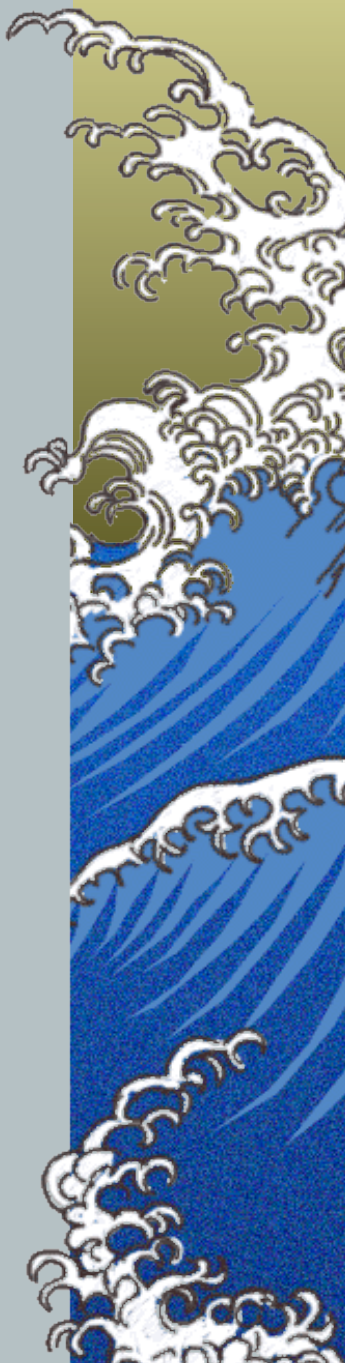
- $A_1 \vee B_1 \vee \ldots \vee B_n \qquad \neg A_2 \vee C_1 \vee \ldots \vee C_m$
- senza variabili, se $A_1 = A_2$ il risolvente è

  $B_1 \vee \ldots \vee B_n \vee C_1 \vee \ldots \vee C_m$
- con variabili, bisogna prima applicare alle clausole la sostituzione unificante più generale (mgu) di A_1 e A_2
  - l'mgu è calcolato dall'algoritmo di unificazione (Robinson)



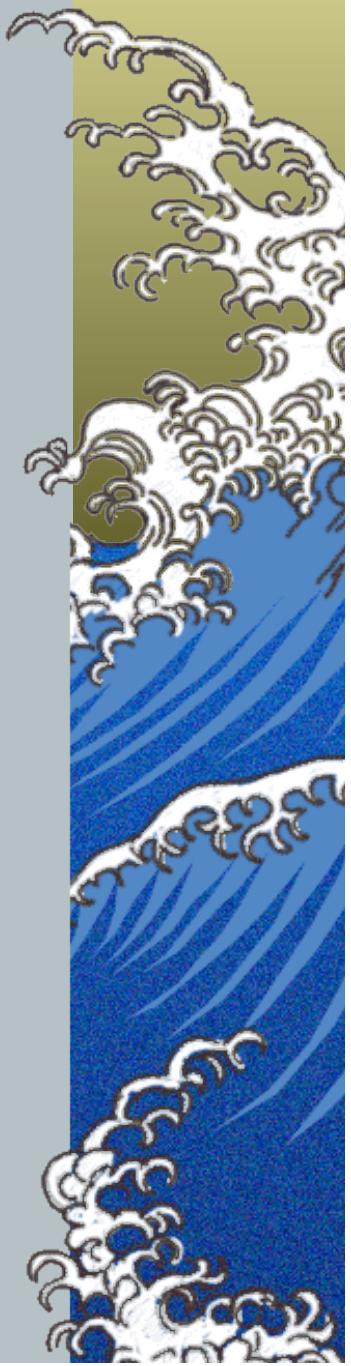
Nessuno è perfetto

- l'uso dell'unificazione riduce drasticamente il numero di clausole inferite
- da una coppia di clausole derivo risolventi solo istanziandole con l'mgu
  - ogni risolvente sta per un numero (possibilmente infinito) di sue istanze
- comunque esplosione combinatoria
- ricerca di strategie (possibilmente complete)
  - la risoluzione lineare è una specie di riscrittura nondeterministica (purtroppo incompleta)



Kowalski

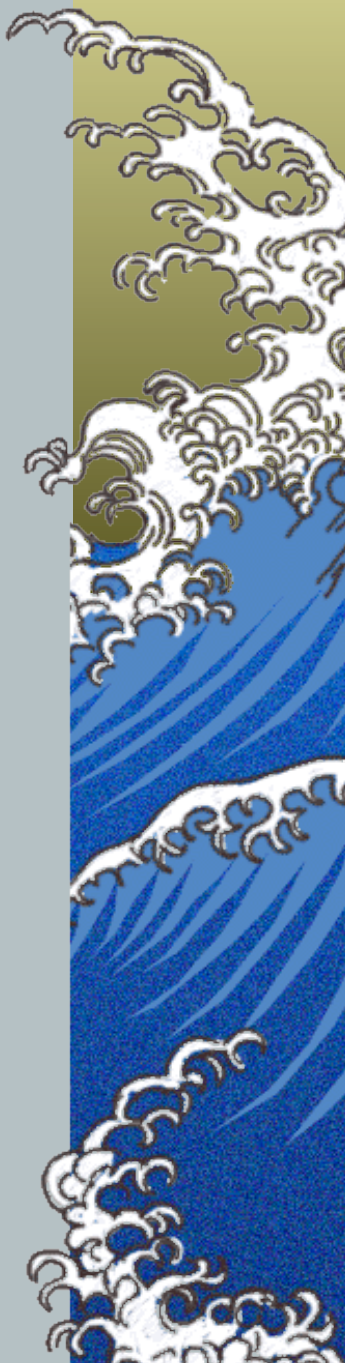
- Bob Kowalski identifica un sottoinsieme (le clausole Horn definite, ***clausole con al più un letterale positivo***), per cui
  - la risoluzione lineare è completa
    - una dimostrazione è una derivazione della clausola vuota (terminazione!) attraverso una riscrittura nondeterministica (risoluzione SLD, semantica operazionale)
  - i vari tipi di clausole hanno una naturale interpretazione procedurale
    - definizione e chiamata di sottoprogrammi, passaggio di parametri sofisticato (unificazione, non distinzione tra input e output, strutture dati parziali)
    - i sottoprogrammi sono nondeterministici
  - un programma logico ha un modello minimo di Herbrand (unico, la semantica!)
    - tale modello può essere costruito con un calcolo di punto fisso (bottom-up) nello stile della semantica denotazionale



Un programma logico

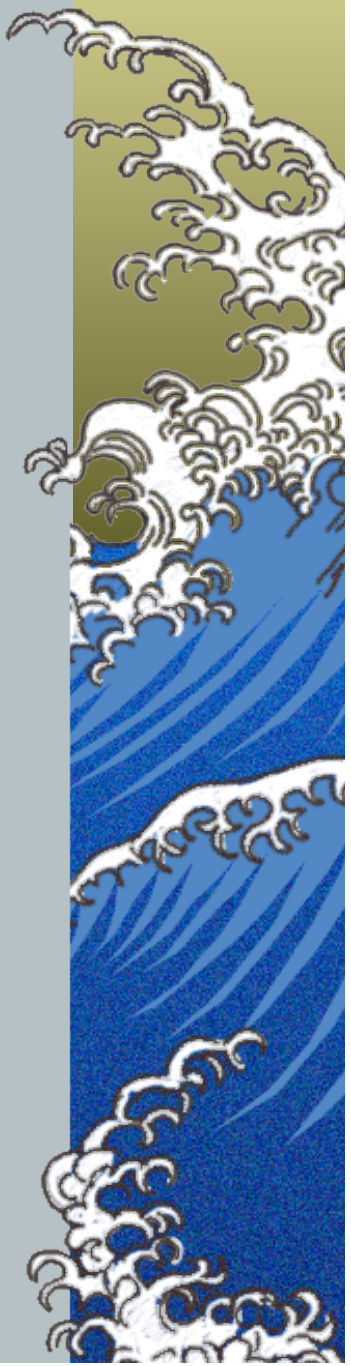
- sort(X,Y) :- sorted(Y), perm(X, Y).
- sorted([]).
- sorted([X]).
- sorted([X, Y|Z]) :- X <= Y, sorted(Y|Z).
- perm([], []).
- perm(X|Y, U|V) :- delete(U, X|Y, Z), perm(Z, V).
- delete(X, X|Y, Y).
- delete(X, Y|Z, Y|W) :- delete(X, Z, W).

- **un goal**
- ?- sort([1, 0, 2], X).



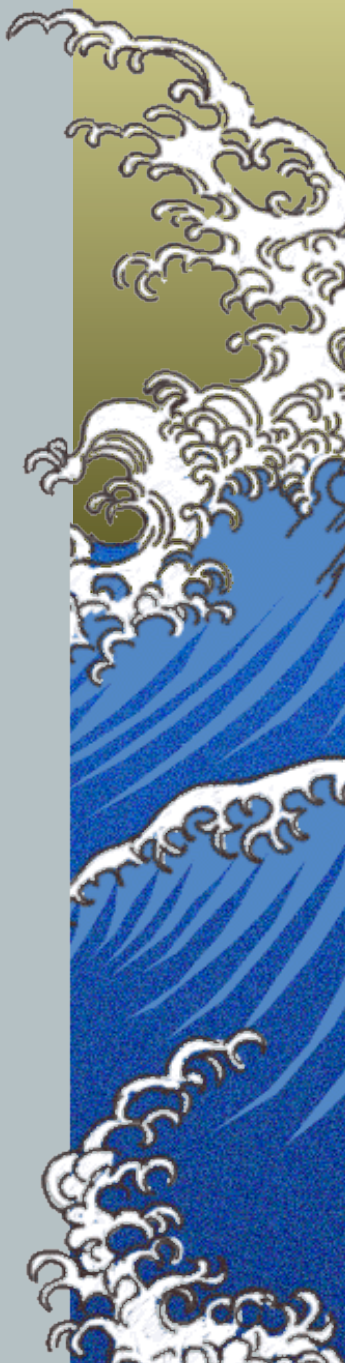
Colmerauer

- Kowalski visita Alain Colmerauer a Marsiglia
  - interessato alla formalizzazione del linguaggio naturale
- Kowalski non ha ancora capito proprio tutto, ma Colmerauer (con l'aiuto di Roussel) implementa il linguaggio che gli serve
- PROLOG = clausole Horn definite
  - + un po' di cose strane ma utilissime (assert, retract, !, primitive meta)
  - + almeno un errore prolifico (l'unificazione senza occur check)



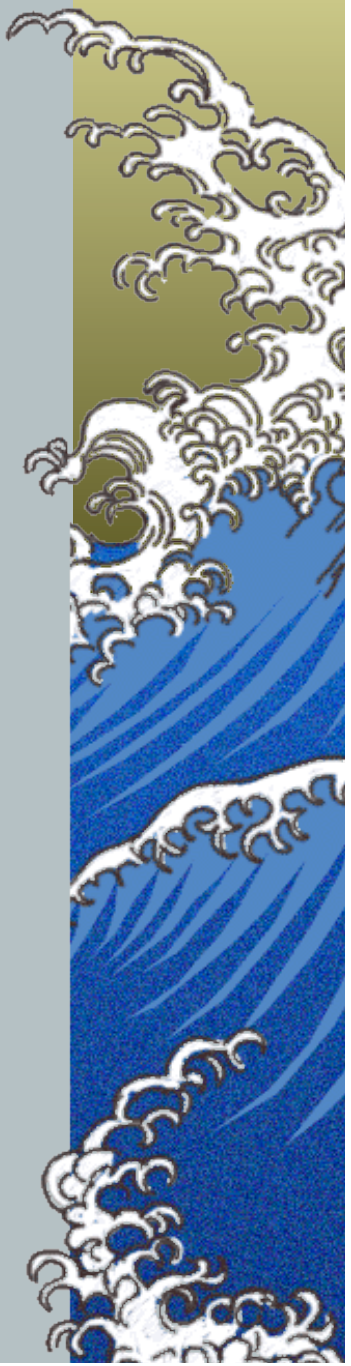
Un capolavoro tecnologico: il compilatore Edinburgh-Prolog

- i meccanismi operazionali di Prolog (unificazione, non-determinismo) offrono grandi margini di ottimizzazione
- David Warren (Edinburgo) progetta una macchina target per la compilazione, chiamata WAM (Warren Abstract Machine)
  - dotata di pila, heap, garbage collector
  - ricostruita dal punto di vista teorico con la valutazione parziale
- ed un compilatore molto astuto
  - i programmi Prolog diventano anche efficienti!



Tante belle implementazioni!

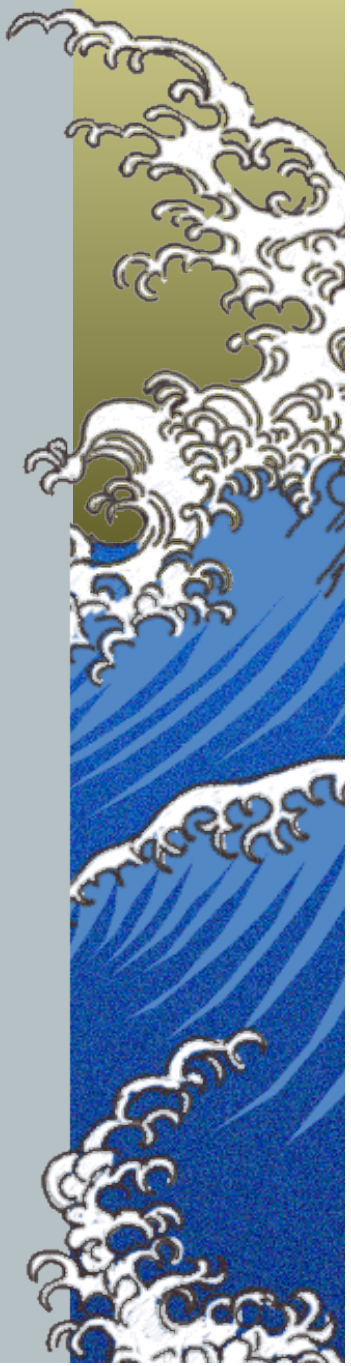
- altri compilatori, anche più sofisticati
  - per esempio, SICSTUS-Prolog
- molta ricerca su analisi statica e ottimizzazioni
  - banco di prova teorico e pratico dell'interpretazione astratta (Ciao-Prolog)
- molta ricerca sulle implementazioni parallele



Prolog con insiemi di equazioni

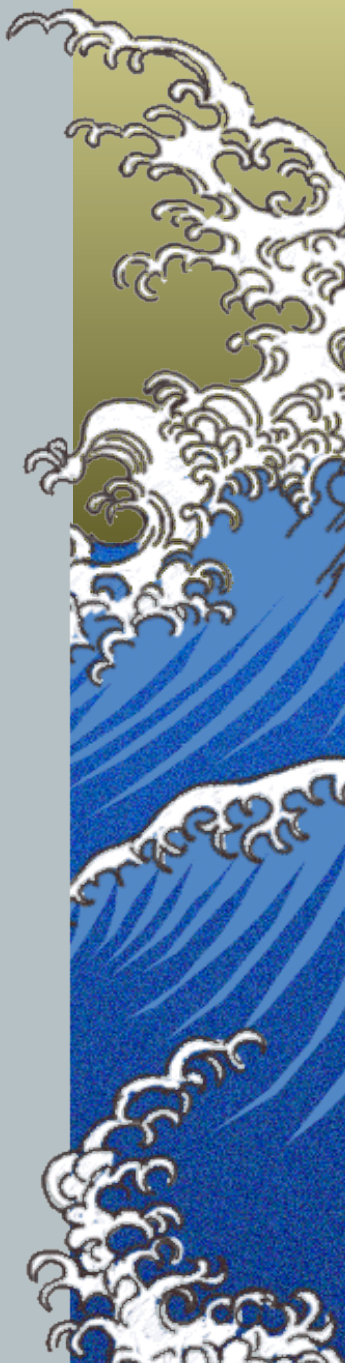
- perm([], []).
- perm(X|Y, U|V) :- delete(U, X|Y, Z), perm(Z, V).
- perm(X, Y) :- X = [], Y = [].
- perm(X, Y) :- X = X1|Y1, Y = U|V # delete(U, X, Z), perm (Z, V).

- l'unificazione è resa esplicita, specificando equazioni fra termini
  - il calcolo accumula, risolve e restituisce insiemi di equazioni
    - l'algoritmo di unificazione verifica la soddisfacibilità e genera la forma risolta



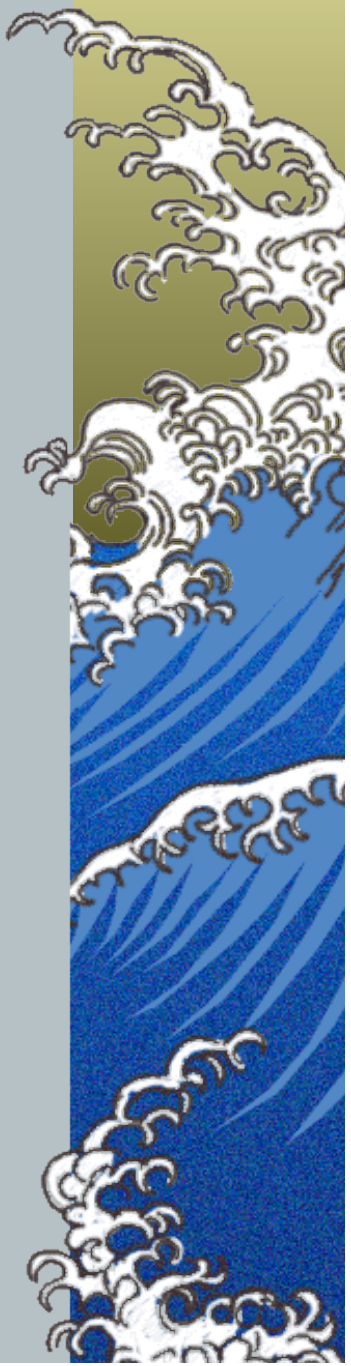
Da Prolog a CLP

- rimpiazziamo gli insiemi di equazioni con altri “sistemi di vincoli”
  - equazioni e disequazioni su termini
  - equazioni su termini razionali (senza occur check!)
  - vincoli su domini finiti numerici
  - equazioni e disequazioni su numeri reali
- rimpiazziamo l’algoritmo di unificazione con un algoritmo che
  - verifica la soddisfacibilità e genera una forma semplificata (normale, se possibile)
- ricicliamo algoritmi classici (per esempio, il simplesso)
- otteniamo potenti strumenti per formalizzare e risolvere problemi di tipo combinatorio su domini numerici



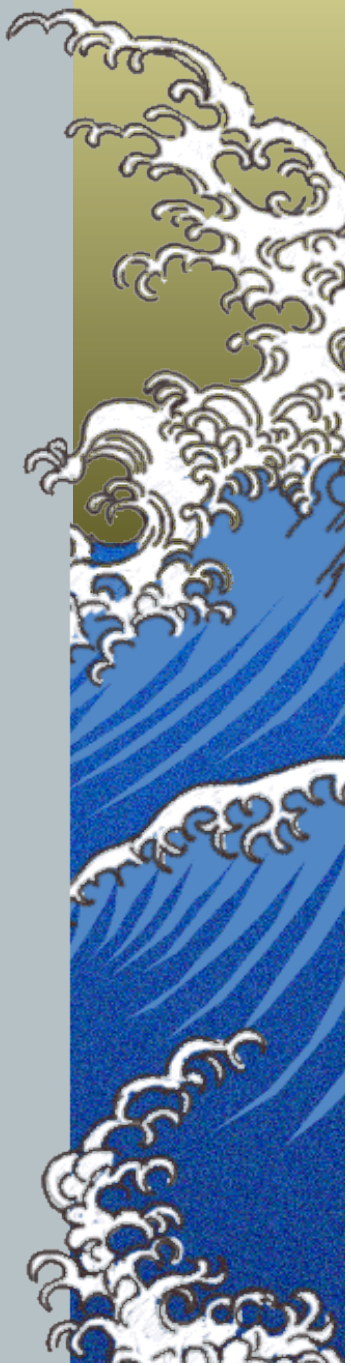
Semantica

- la semantica è molto semplice
  - con varie formulazioni alternative tra loro equivalenti
- in realtà, come sempre, si utilizzano semantiche diverse secondo le necessità
- ambiente ideale su cui studiare
  - le relazioni fra semantiche (interpretazione astratta)
  - le varie tecniche semantics-based
    - analisi
    - trasformazione
    - ottimizzazione



La groundness

- motivato dalla compilazione
  - vogliamo sapere se una variabile verrà legata ad un termine ground
- semantica concreta
  - deve modellare le risposte calcolate (non-ground)
  - è utile una formulazione goal independent
    - la S-semantica punto fisso
- tanti domini astratti per la groundness
  - per studiarne la precisione relativa
  - risultati generali per i raffinamenti di domini



Uno splendido linguaggio di specifica eseguibile

- il formalismo delle clausole Horn definite coincide praticamente con il metalinguaggio della semantica operazionale (regole di transizione)
  - relazioni
  - nondeterminismo
- ed è eseguibile
  - una specifica (di un linguaggio o di un sistema) data mediante regole di transizione è anche un prototipo funzionante
  - l'applicazione industriale più diffusa di Prolog!
- l'essenza del dichiarativo vs. procedurale!