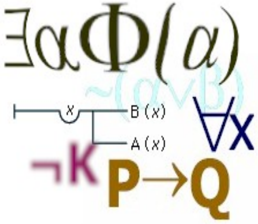
*Πανεπιστήμιο Μακεδονίας*
*Τμ. Εφαρμοσμένης Πληροφορικής*

# ECLiPSe Integer/Real Constraints



# Διαφορές Συστημάτων Λογικού και CLP προγραμματισμού

- Διαφορά στην **ενοποίηση** και στην **απόδοση τιμών** στις μεταβλητές.
  - Οι μεταβλητές πλέον έχουν **πεδίο ορισμού**.
  - Αν δοθεί **τιμή εξω από το πεδίο**, ο στόχος **αποτυγχάνει**.
  - Η τιμή μπορεί να είναι και ένα **υποσύνολο** του αρχικού πεδίου.
- Έστω για παράδειγμα ο παρακάτω κανόνας:
  - positive_less(X,Y):- X>0, X < Y.
- Εάν υποβάλλουμε την ερώτηση:
  - ?- less(3,4).
- στην κλασσική Prolog θα πάρουμε την απάντηση:
  - yes.

# Διαφορές Συστημάτων Λογικού και CLP προγραμματισμού (cont)

- Αν όμως υποβάλλουμε την ερώτηση:
  - ?- less(X,10).

  δεν θα πάρουμε καμία απάντηση (εκτός ίσως από κάποιο μήνυμα λάθους).
- Ο λόγος είναι ότι στην κλασσική Prolog οι αριθμητικές σχέσεις (ισότητες, ανισότητες) μπορούν να ελεγχθούν μόνο εφόσον οι μεταβλητές **έχουν όλες πάρει τιμή.**

# Λογικός Προγραμματισμός με Περιορισμούς

- Οι νεώτερες εκδόσεις της Prolog έχουν επεκταθεί έτσι ώστε να χειρίζονται μεταβλητές με πεδίο ορισμού και περιορισμούς μεταξύ των μεταβλητών.
- Έτσι η ερώτηση:
  - ?- positive_less(X,10).
  - 1=<X=<9
- Μία λύση σε ένα πρόβλημα λογικού προγραμματισμού με περιορισμούς (μπορεί να) είναι ένα πιο συγκεκριμένο σύνολο περιορισμών πάνω στις μεταβλητές της ερώτησης.

# ECLiPSe Prolog CLP

- Η ECLiPSe έχει μια πληθώρα βιβλιοθηκών οι οποίες υποστηρίζουν περιορισμούς.
  - Interval Constraints (**ic**)
  - Finite Domain Constraints (**ic_global**, **ic_cummulative**, **ic_edge_finder**)
  - Integer Sets (**fd_sets, ic_sets**)
  - Symbolic Domain (**ic_symbolic**)
  - Generalised Propagation (**Propia**)
  - Constraint Handling Rules (**chr**)
  - External Solvers (**eplex**)
  - Βιβλιοθήκη αλγορίθμων επιδιόρθωσης (**repair**)

# Βιβλιοθήκες και χρήση Περιορισμών

- Χρήση των βιβλιοθηκών της ECLiPSe, προϋποθέτει την "**φόρτωσή**" τους.
- Υπάρχουν δύο επιλογές.
  - Στην **αρχή του προγράμματος (κώδικα)** να υπάρχει μια από τις παρακάτω εντολές:
    - :-lib(ic).
    - :-use_module(library(ic)).
  - **ή** στην γραμμή ερωτήσεων της ECLiPSe να δώσουμε μία από τις παρακάτω «ερωτήσεις»:
    - ?-lib(ic).  ή ?-use_module(library(ic)).

# Γενική Δομή ενός Προγράμματος CLP

- Η γενική δομή ενός προγράμματος CLP είναι η ακόλουθη:
  - "***Φόρτωση***" των κατάλληλων βιβλιοθηκών
  - Δήλωση των **μεταβλητών** και των αντίστοιχων **πεδίων** τους.
  - Δήλωση των **περιορισμών** στις προηγούμενες μεταβλητές.
  - **Αναζήτηση** λύσης.

# Παράδειγμα

- Πρόβλημα βιομηχανικού μύλου.
  - Έστω ότι πρέπει να ορισθεί η σειρά με την οποία θα εισαχθούν τα προϊόντα Α, Β, Γ, Δ μέσα σε ένα βιομηχανικό μύλο. Λόγω κάποιων παρασκευαστικών παραμέτρων του τελικού προϊόντος, το προϊόν Α πρέπει να εισαχθεί στο μύλο μετά από το Δ, το Γ πριν από το Β, και το Β πριν από το Α.

# Μοντελοποίηση Προβλήματος

- **Μεταβλητές:** $V_A$, $V_B$, $V_Γ$ και $V_Δ$
- **Πεδία Μεταβλητών:** [1..4] (σειρά με την οποία θα μπουν τα προϊόντα.
- **Περιορισμοί**:
  - $V_A \neq V_B$ και $V_A \neq V_Γ$ και $V_A \neq V_Δ$
  - $V_B \neq V_Γ$ και $V_B \neq V_Δ$ και $V_Γ \neq V_Δ$
  - $V_A > V_Δ$ *το προϊόν Α μετά από το Δ*
  - $V_Γ < V_B$ *το προϊόν Γ πριν από το Β*
  - $V_B < V_A$ *το προϊόν Β πριν από το Α*

*Δύο προϊόντα δεν μπορούν να πάρουν την ίδια σειρά.*

# Υλοποίηση σε ECLiPSe Prolog

```
:-lib(ic).                                  % Φόρτωση Βιβλιοθήκης


solve_mill(A,B,C,D):-
        [A,B,C,D] #:: 1..4,                 % Δήλωση πεδίων
        A #\= B, A #\= C, A #\= D,
        B #\= C, B #\= D,                   % Δήλωση περιορισμών
        C #\= D,
        A #> D, C #< B, B #< A,
        labeling([A,B,C,D]).                % Αναζήτηση κατά βάθος.
```

# Η βιβλιοθήκη ic

- Η βιβλιοθήκη ic (interval constraints) υποστηρίζει περιορισμούς επί διαστημάτων ακεραίων και πραγματικών αριθμών.
- Αλγόριθμος επίλυσης AC (bounds consistency).
- Επιτρέπει να αναμιχθούν ακέραιες και πραγματικές μεταβλητές στους περιορισμούς.
- Υποστηρίζει τόσο γραμμικές όσο και μη γραμμικές αριθμητικές εκφράσεις στους υπολογισμούς.

# Δηλώσεις Πεδίων Τιμών (1/4)

- Δήλωση πεδίου ορισμού μιας ή περισσοτέρων μεταβλητών:
  - Vars :: Domain .
- όπου:
  - **Vars** μια **μεταβλητή** ή μια **λίστα** μεταβλητών
  - Domain ένα διάστημα της μορφής Low..High ή μια λίστα τέτοιων διαστημάτων καθώς και μεμονωμένων τιμών.

    X:: 1..10

# Δηλώσεις Πεδίων Τιμών (2/4)

- Παραδείγματα:
  - X :: -10..10 .
  - X :: [-2..3, 8..15, 19].
  - [X, Y] :: -20..1.0Inf.
  - X:: -1.0Inf .. 1.0Inf
  - X :: -10.0..15.0.
- Εάν όλα τα διαστήματα στο domain είναι ακέραια, η μεταβλητή θεωρείται ακέραια (integer), αλλιώς θεωρείται πραγματική (real).

*Τέτοια διαστήματα ισχύουν μόνο για ακεραίους*

# Δηλώσεις Πεδίων Τιμών (3/4)

- Δυο παραλλαγές δήλωσης του πεδίου ορισμού είναι οι ακόλουθες:
  - **Vars #:: Domain**, καθορίζει ότι οι μεταβλητές της Vars είναι υποχρεωτικά ακέραιες
  - **Vars $:: Domain**, καθορίζει ότι οι μεταβλητέςείναι υποχρεωτικά πραγματικές
- integer(X).
  - Δηλώση ότι μια μεταβλητή(ες) είναι ακέραια, χωρίς να ορίσουμε πεδίο ορισμού.
- real(X).
  - Παρόμοια, δηλώνει ότι η μεταβλητή είναι πραγματική.

# Δηλώσεις Πεδίων Τιμών (4/4)

- Τέλος, μπορούμε να μην δηλώσουμε καθόλου το πεδίο ορισμού μιας μεταβλητής, αρκεί να την χρησιμοποιήσουμε μέσα σε έναν περιορισμό.
  - Εάν βρίσκεται μέσα σε περιορισμούς που αφορούν **πραγματικούς**, τότε η μεταβλητή είναι **πραγματική**.

    Y+X $>0.

  - Εάν βρίσκεται μέσα σε περιορισμούς που αφορούν **ακεραίους**, τότε η μεταβλητή είναι **ακέραια**.

    Y+X #>0.

# Περιορισμοί Ακέραιων Εκφράσεων

- Οι παρακάτω περιορισμοί αφορούν ακέραιες εκφράσεις και υποεκφράσεις.
  - ExprX #= ExprY
  - ExprX #>= ExprY
  - ExprX #=< ExprY
  - ExprX #> ExprY
  - ExprX #< ExprY
  - ExprX #\= ExprY

# Περιορισμοί Πραγματικών Εκφράσεων

- Οι παρακάτω περιορισμοί αφορούν πραγματικές εκφράσεις και υποεκφράσεις.
  - ExprX $= ExprY
  - ExprX $>= ExprY
  - ExprX $=< ExprY
  - ExprX $> ExprY
  - ExprX $< ExprY
  - ExprX $\= ExprY

# Εκφράσεις

- Στις εκφράσεις μπορούν να εμφανίζονται τα ακόλουθα:
  - μεταβλητές (περιορισμών), αριθμητικές σταθερές,
  - εκφράσεις αριθμητικών πράξεων (+,-,*,/, κλπ)
  - **min(E1,E2), max(E1,E2)**
  - **sqr(Expr), sqrt(Expr), exp(Expr) (**εκθετικό**), ln(Expr)**
  - **sin(Expr), cos(Expr), atan(Expr)**
  - **rsqr(Expr)** (αντιστοιχεί στο +-sqrt(Expr) ).
  - **rpow(E1,E2) X in E1 = X^E2.**
  - **sub(Expr)** (υποδιάστημα της έκφρασης)
  - **sum(ExprList), min(ExprList), max(ExprList)**

# Παραδείγματα

*?- Z #:: [11 .. 15], [X, Y] #:: [1 .. 10], W #= X + min(Z, Y).*

*Z = Z{11 .. 15}        X = X{1 .. 10}*

*Y = Y{1 .. 10}        W = W{2 .. 20}*

*There are 3 delayed goals. Yes (0.00s cpu)*

*?- [X, Y, Z] #:: [1 .. 10], W #= max([3 + X, 4 + Y, 10 + Z]).*

*X = X{1 .. 10}        Y = Y{1 .. 10}*

*Z = Z{1 .. 10}        W = W{11 .. 20}*

*There are 4 delayed goals. Yes (0.00s cpu)*

# Εκφράσεις (διάζευξη)

- **+- E** : θα ισχύει είτε το + είτε το - και θα επιλεγεί εκείνο που ικανοποιεί τον περιορισμό, πχ

  **[X,Y] #::[1..10], X #= Y + (+- 3), indomain(Y).**

# Reified Περιορισμοί

- Βασική Ιδέα
  - Ένας περιορισμός μπορεί να συνεπάγεται λογικά (entailed) να μη συνεπάγεται λογικά (disentailment) ή να μην γνωρίζουμε ακόμη αν θα συνεπάγεται ή όχι στην τελική λύση.
  - Συνεπάγεται λογικά = 1, συνεπάγεται η άρνησή του = 0 , δεν γνωρίζω [0,1]
- Παράδειγμα
  - X ανήκει [1..5]
    - X > 0 συνεπάγεται λογικά (1)
    - X > 20 συνεπάγεται η άρνηση (0)
    - X > 3 στην τελική λύση μπορεί να ισχύει μπορεί και όχι ([0,1])

# Reified Περιορισμοί

- Περιορισμοί των οποίων η τιμή αλήθειας μπορεί να χρησιμοποιηθεί:
  - ως τιμή μιας μεταβλητής,
  - μέρος λογικής έκφρασης.
- Διαθέσιμοι λογικοί σύνδεσμοι είναι οι **and**, **or**, **neg** και **=>** :
  - **and**: X$> 3 and X$<8
  - **or**: X$<3 or X$>8
  - **neg**: neg X$>3
  - **=>:** X $>3 => Y$<8

# Παράδειγμα Reified Περιορισμών

```
?- X :: 1 .. 10, T #= (X #> 2).
X = X{1 .. 10}
T = T{[0, 1]}
There is 1 delayed goal.
Yes (0.00s cpu)

?- X :: 1 .. 10, T #= (X #> 20).
X = X{1 .. 10}
T = 0
Yes (0.00s cpu)
```

# Χρήση reified περιορισμών

- Αν στη reified μεταβλητή αποδοθεί τιμή τότε ο περιορισμός ενεργοποιείται και “κόβει” από το πεδίο τιμών.

*?- X :: 1 .. 5, B #= (X #> 3).*

*X = X{1 .. 5}*

*B = B{[0, 1]}*

- αλλά

*?- X :: 1 .. 5, B #= (X #> 3), B #= 1.*

*X = X{[4, 5]}*

*B = 1*

# Reified Περιορισμοί Πεδίων

- **::(Var,Domain,Bool)**
  - Αν το τρέχον πεδίο της Var, είναι **υποσύνολο** του πεδίου Domain, τότε η μεταβλητή Bool γίνεται 1,
  - Αν τα δύο πεδία δεν έχουν κοινά στοιχεία τότε η μεταβλητή Bool, γίνεται 0,
  - αλλιώς ανήκει στο διάστημα 0..1

# Παράδειγματα

*?- X #:: [1 .. 10], ::(X, 1 .. 15, B).*

*X = X{1 .. 10}    B = 1    Yes (0.00s cpu)*

*?- X #:: [1 .. 10], ::(X, 11 .. 15, B).*

*X = X{1 .. 10}    B = 0    Yes (0.00s cpu)*

*?- X #:: [1 .. 10], ::(X, 1 .. 5, B).*

*X = X{1 .. 10}    B = B{[0, 1]}*

*There is 1 delayed goal. Yes (0.00s cpu)*

*?- X #:: [1 .. 10], ::(X, 1 .. 5, B), B = 1.*

*X = X{1 .. 5}    B = 1    Yes (0.00s cpu)*

# Διάφοροι περιορισμοί

- alldifferent(Vars)
  - όπου Vars είναι μια λίστα μεταβλητών περιορισμών. Δηλώνει ότι οι μεταβλητές είναι ανα ζεύγη διαφορετικές.
  - πχ. alldifferent([A,B,C,D]).
- element(Index, List, Value)
  - Περιορίζει την τιμή Value να είναι το Index'th στοιχείο της λίστας ακεραίων List.
  - element(2,[10,20,30,40],X) η τιμή X θα γίνει ίση με 20.
  - element(Y,[10,20,30,40],20) η τιμή Y θα γίνει ίση με 2.

# Εύρεση Λύσης σε ακέραια πεδία

- Υπάρχουν δύο κατηγορήματα που ξεκινούν τη διαδικασία ανάθεσης τιμών στις μεταβλητές:
  - **indomain(X)**: Η μεταβλητή X είναι μια μεταβλητή περιορισμών. Η κλήση indomain(X) αναθέτει στην X μια από τις τιμές που αυτή μπορεί να πάρει, ενώ ταυτόχρονα ενημερώνει όλες τις μεταβλητές που συμμετέχουν σε περιορισμούς με την X.
  - **labeling(List)**: Η λίστα List περιλαμβάνει ένα σύνολο μεταβλητών περιορισμών. Η κλήση στην labeling επιχειρεί να βρει μια ανάθεση τιμών στις μεταβλητές της List (λύση).
- Συνήθως σε απλά προβλήματα καλούμε την labeling με όλες τις μεταβλητές του προβλήματος.

# Το κατηγόρημα search/6

search(+L, ++Arg, ++Select, +Choice, ++Method, +Option)

- Μια γενική ρουτίνα αναζήτησης η οποία παραμετροποιείται για να δώσει μια πληθώρα αλγορίθμων.
  - L, λίστα μεταβλητών περιορισμών
  - Arg, σε ποιο όρισμα του όρου είναι οι μεταβλητές, συνήθως μηδέν.
  - Select, **επιλογή μεταβλητής** για ανάθεση τιμής (*heuristics*).
    - Τιμές: input_order, first_fail, smallest, largest, occurrence, most_constrained, max_regret, anti_first_fail
    - <user_defined> ** κατηγόρημα με arity 2, όπου το πρώτο όρισμα είναι μεταβλητή περιορισμών και το δεύτερο μια αριθμητική τιμή.

# Το κατηγόρημα search/6

- Choice: **Επιλογή τιμής** από το πεδίο της μεταβλητής.
  - Διαθέσιμες τιμές indomain, indomain_min, indomain_max, indomain_middle, indomain_median, indomain_split, indomain_random, indomain_interval
- Method: **Μέθοδος αναζήτησης** που μπορεί να είναι:
  - complete, bbs(Steps:integer), lds(Disc:integer), credit(Credit:integer, Extra:integer or bbs(Steps:integer) or lds(Disc:integer)), dbs(Level:integer, Extra:integer or bbs(Steps:integer) or lds(Disc:integer)), sbds, gap_sbds, gap_sbdd
- Option, Λίστα από επιλογές.

# Παράδειγμα

- Στο πρόβλημα του βιομηχανικού μύλου, πλήρης αναζήτηση με ευριστική συνάρτηση την αρχή της συντομότερης αποτυχίας (first-fail) και χρήση alldifferent περιορισμού.

```
solve_mill(A,B,C,D):-
    [A,B,C,D] #:: 1..4,
    alldifferent([A, B, C, D]),
    A #> D, C #< B, B #< A,
    search([A,B,C,D],0,first_fail,indomain,complete,[]).
```

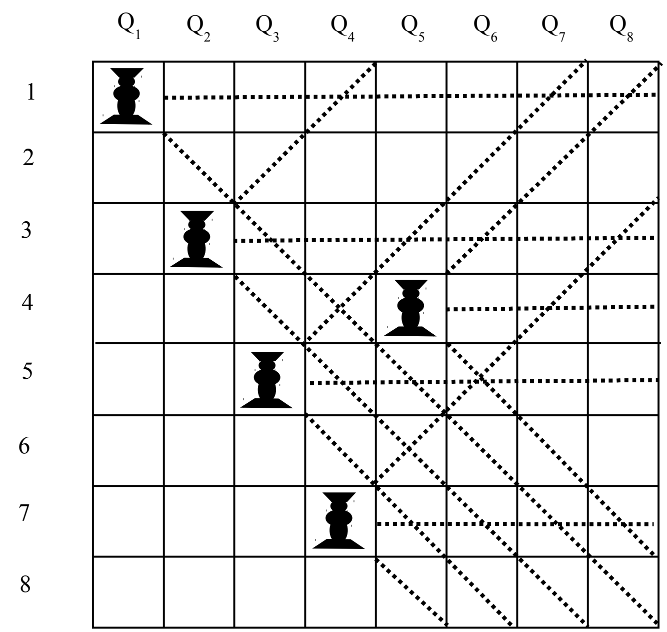
# Ένα κλασικό παράδειγμα

- Classic N-queens Problem
- Να τοποθετηθούν σε μια σκακιέρα NxN, N βασίλισσες ώστε να μην επιτίθενται η μία στην άλλη.
  - Κλασική εκδοχή να τοποθετηθούν 8 βασίλισσες σε μια σκακιέρα 8x8.
  - Η δυσκολία στην επίλυσή του αυξάνει εκθετικά.
  - Χρησιμοποιείται για την μέτρηση της απόδοσης αλγορίθμων ικανοποίησης περιορισμών.
  - Το πλέον κλασσικό πρόβλημα στο CLP.

# NQueens

- Θεωρούμε ότι κάθε βασίλισσα είναι σε διαφορετική στήλη.
- Συνθήκη μη απειλής μεταξύ των βασιλισσών:
- Όλες οι βασίλισσες πρέπει να είναι σε διαφορετική γραμμή:

  $\forall$ i, j: $Q_j \neq Q_i$.




- Ισχύουν οι περιορισμοί:

$Q_j \neq Q_{j+n} + n$ για $n>1$ και $n+j \leq 8$

$Q_j \neq Q_{j+n} - n$ για $n>1$ και $n+j \leq 8$

# Απλή Prolog

- Η λύση είναι μια λίστα. Η **θέση** στη λίστα δίνει τη στήλη. Άρα ασχολούμαστε με την γραμμή μόνο.
- Εφόσον ο κώδικας είναι γενικός πρέπει να δώσουμε την λίστα με τους διαθέσιμους αριθμούς γραμμών.

```
solve(N,X):- coords(N,ListOfYCoordinates),
             solution(X,ListOfYCoordinates).

coords(0,[]):-!.
coords(N,[N|RestCoordinates]):-
   N1 is N - 1,
   coords(N1,RestCoordinates).
```

# Επίλυση

- Αναδρομή.
  - Σε κάθε βήμα βάλε μια βασίλισσα σε μια από τις διαθέσιμες συντεταγμένες.
  - Τοποθέτηση βασιλισσών στις υπόλοιπες συντεταγμένες.
  - Έλεγχος αν παραβιάστηκαν περιορισμοί.
- Δεν είναι generate-and-test. Η λύση δημιουργείται μερικώς και ελέγχεται σε κάθε βήμα.

# Κώδικας

```
solution([],[]).
solution([ X | Rest], Coordinates):-
        delete(X,Coordinates,NewCoordinates),
        solution(Rest, NewCoordinates),
        noattack(X,Rest,1).

noattack(_,[],_).
noattack(X, [Y|Rest],Pos):-
        X =\= Y + Pos,
        X =\= Y - Pos,
        NewPos is Pos + 1,
        noattack(X,Rest, NewPos).
```

# Προσέγγιση

- Όπως παραπάνω:
  - Δήλωση πεδίων τιμών
  - Δήλωση Περιορισμών
  - Αναζήτηση

# CLP Κώδικας (1)

```
queens(N,List):-
    length(List,N),
    List #:: 1..N,
    alldifferent(List),
    apply_constraints(List),
    labeling(List).

apply_constraints([]).
apply_constraints([X|Rest]):-
    safe(X,Rest,1),
    apply_constraints(Rest).
```

```
safe(_,[],_).
safe(X,[Y|Rest],Pos):-
    noattack_clp(X,Y,Pos),
    NewPos is Pos + 1,
    safe(X,Rest,NewPos).

noattack_clp(X,Y,Pos):-
    Y #\= X+Pos,
    Y #\= X-Pos.
```

# CLP Κώδικας (2) first_fail

```
queens(N,List,Strategy,ValueOrdering):-
        length(List,N),
        List #:: 1..N,
        alldifferent(List),
        apply_constraints(List),
       search(List,0,Strategy,ValueOrdering,complete, []).

apply_constraints([]).
apply_constraints([X|Rest]):-
        safe(X,Rest,1),
        apply_constraints(Rest).
```

# Συμπεράσματα

- Μικρές αλλαγές στον κώδικα τεράστια βελτίωση στην απόδοση.
- Heuristics, μεγάλη βελτίωση στην εύρεση λύσης.
- Προσοχή στη μοντελοποίηση των προβλημάτων! (επόμενα μαθήματα).