Constraint Programming over Continuous Domains

Michel RUEHER

University of Nice Sophia-Antipolis / I3S – CNRS, France

September, 2011

Dagstuhl-Seminar 11371

"Uncertainty modeling and analysis with intervals"

Outline

Numeric CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$:

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of variables
- $\mathcal{D} = \{D_{x_1}, \ldots, D_{x_n}\}$ is a set of domains
  (D_{x_i} contains all acceptable values for variable x_i)
- $\mathcal{C} = \{c_1, \ldots, c_m\}$ is a set of constraints

Overall scheme

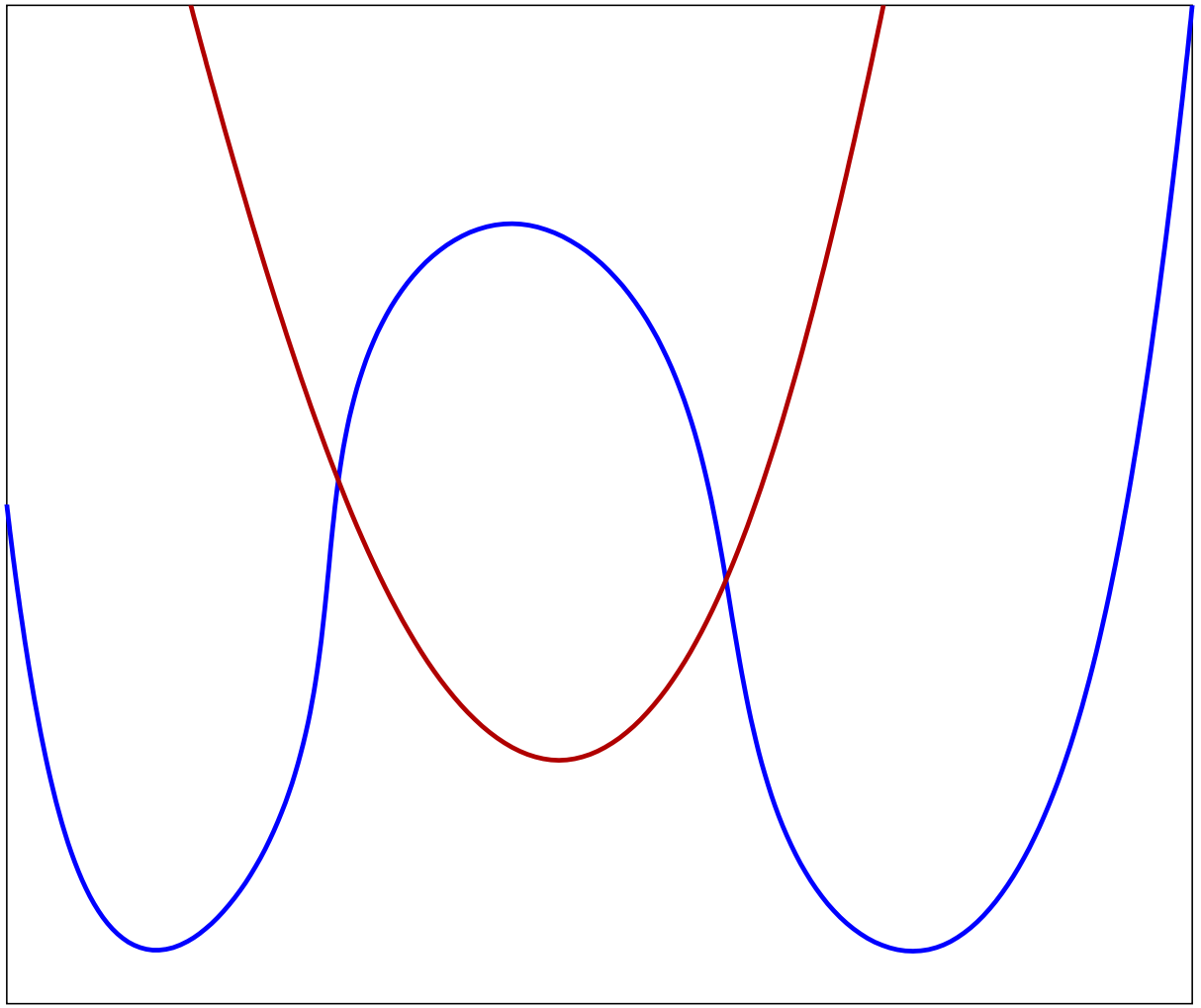
The constraint programming framework is based on a **branch & prune** schema which is best viewed as an iteration of two steps:

1. **Pruning the search space**
2. **Making a choice to generate two (or more) sub-problems**

- The pruning step $\longrightarrow$ **reduces an interval** when it can prove that the upper bound or the lower bound does not satisfy some constraint
- The branching step $\longrightarrow$ **splits the interval** associated to some variable in two intervals (often with the same width)

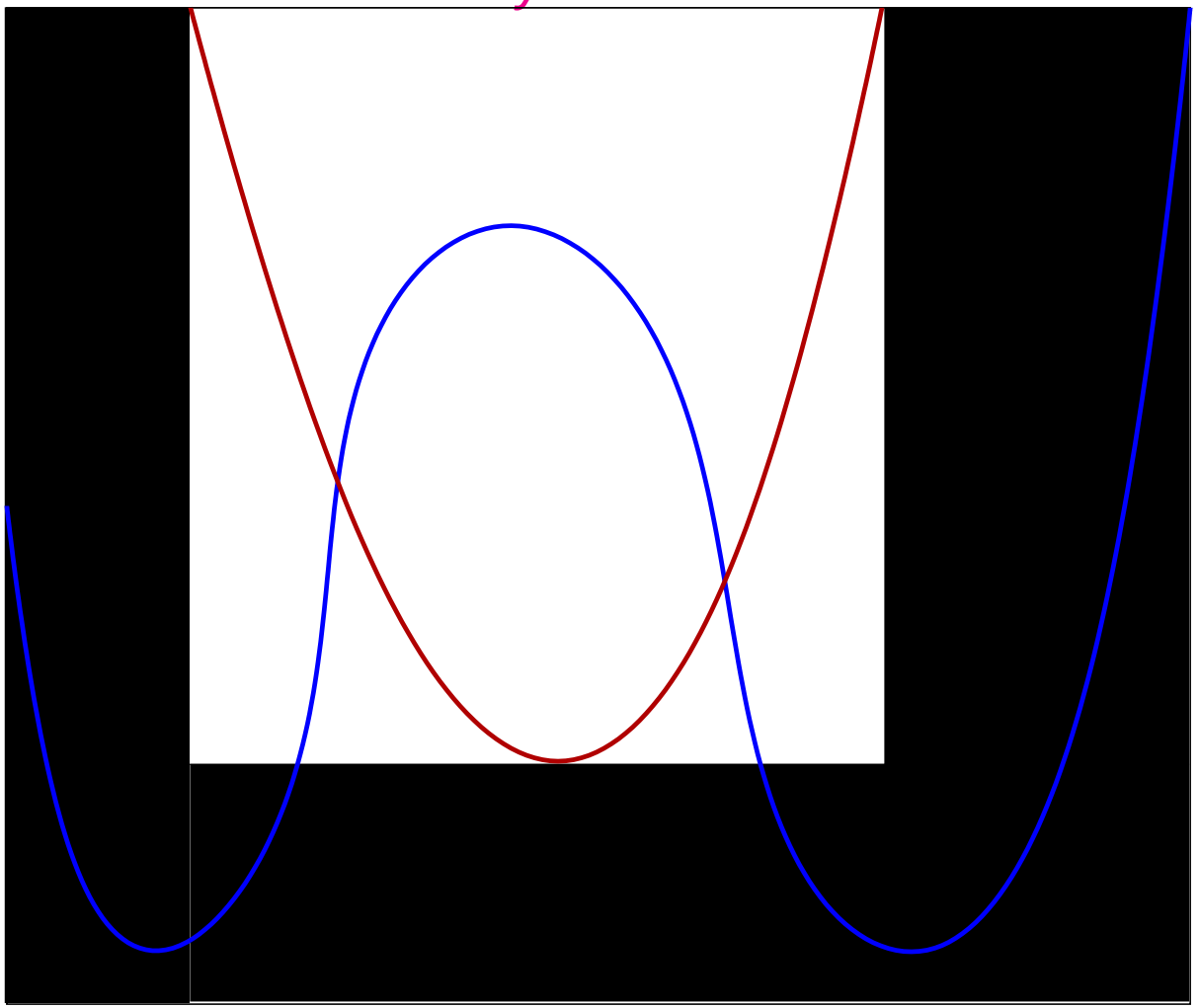
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



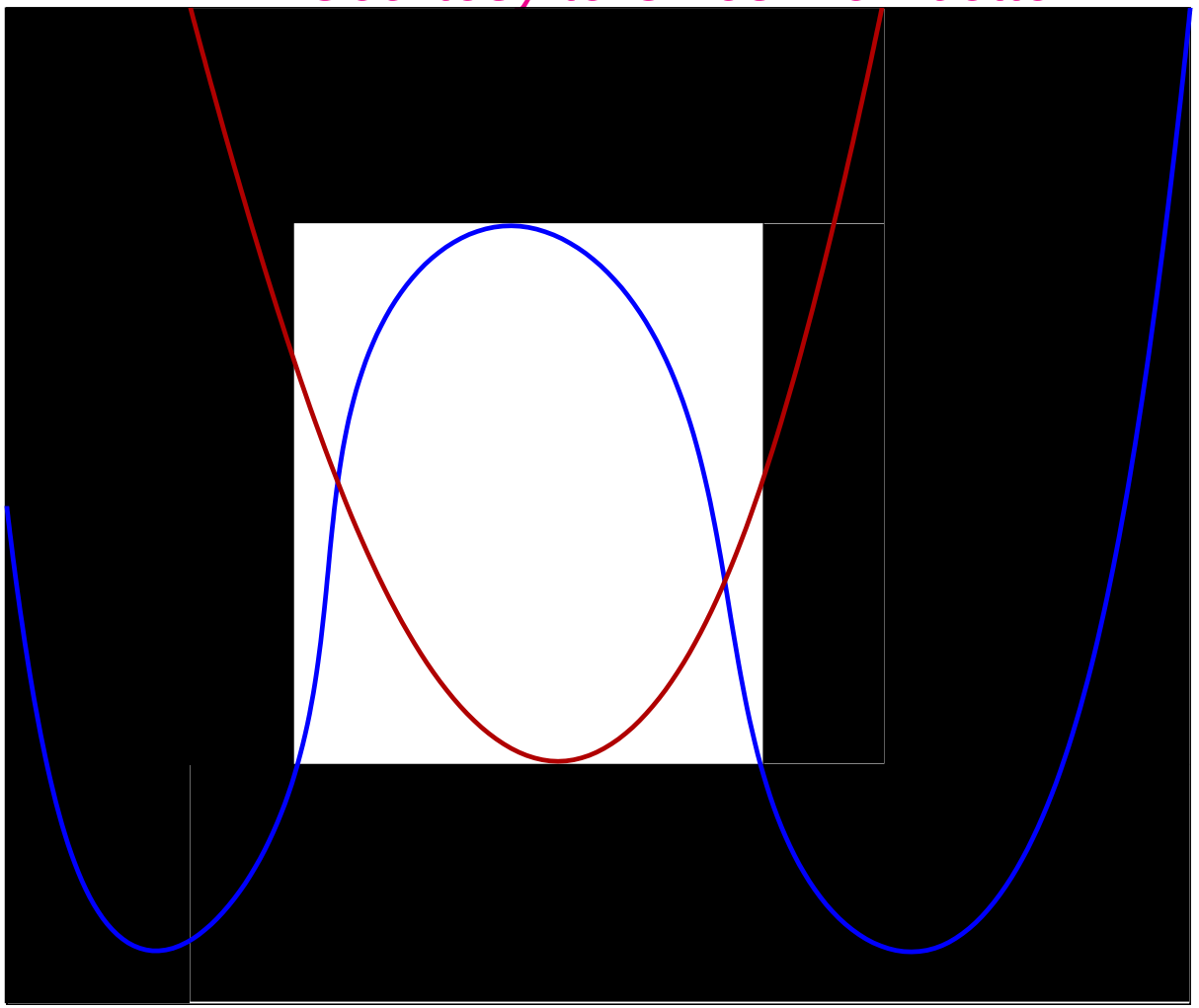
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



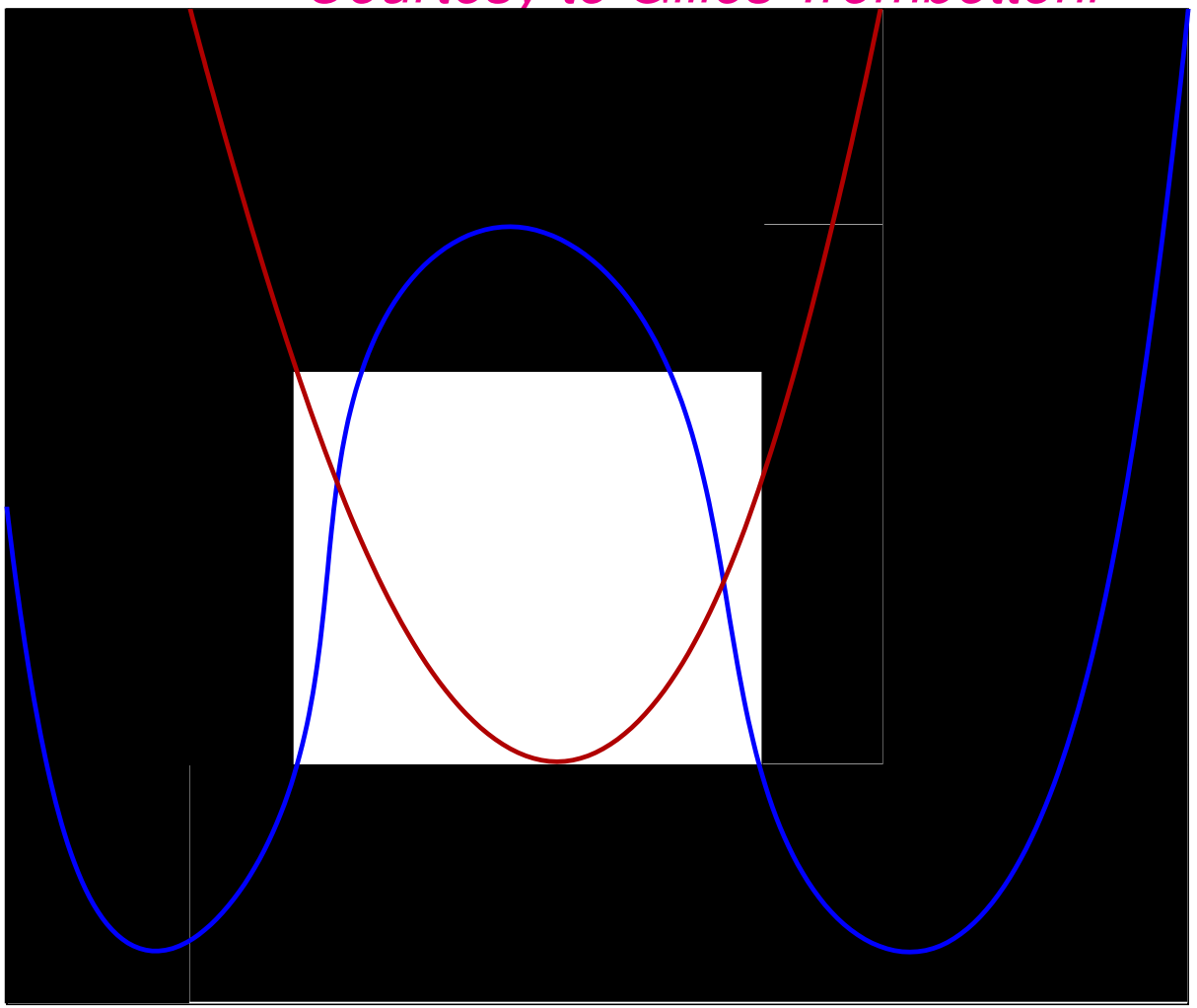
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



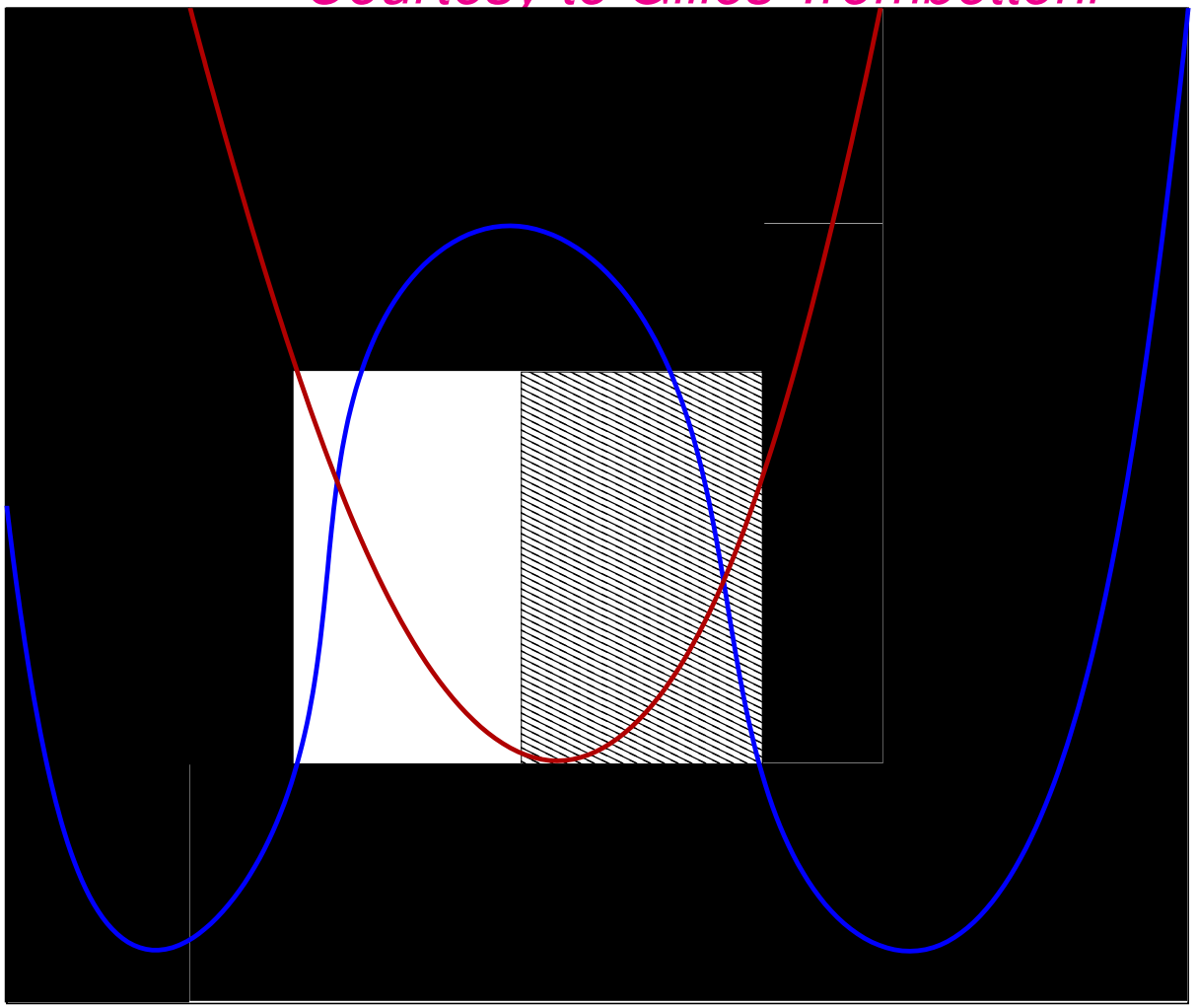
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



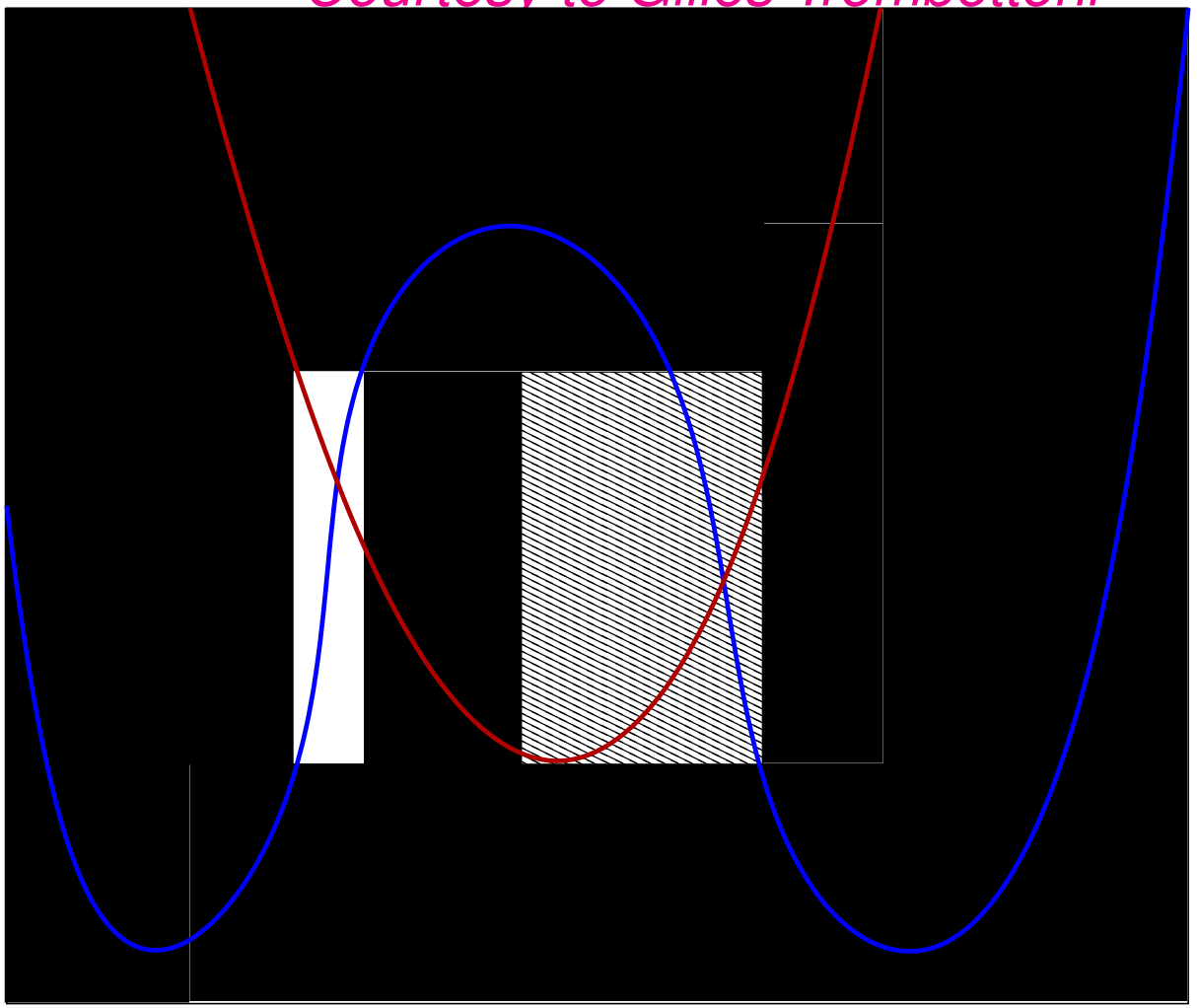
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



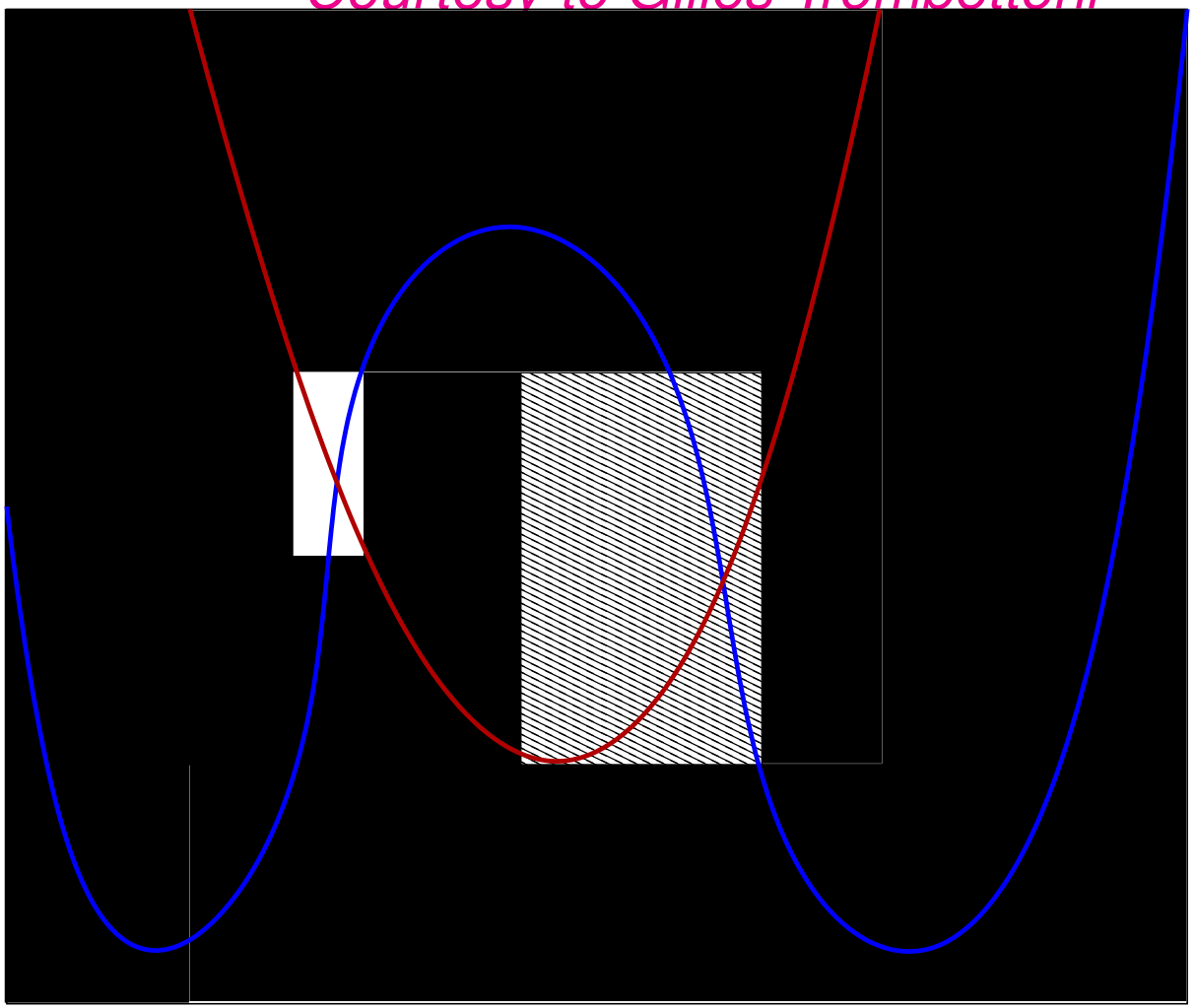
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



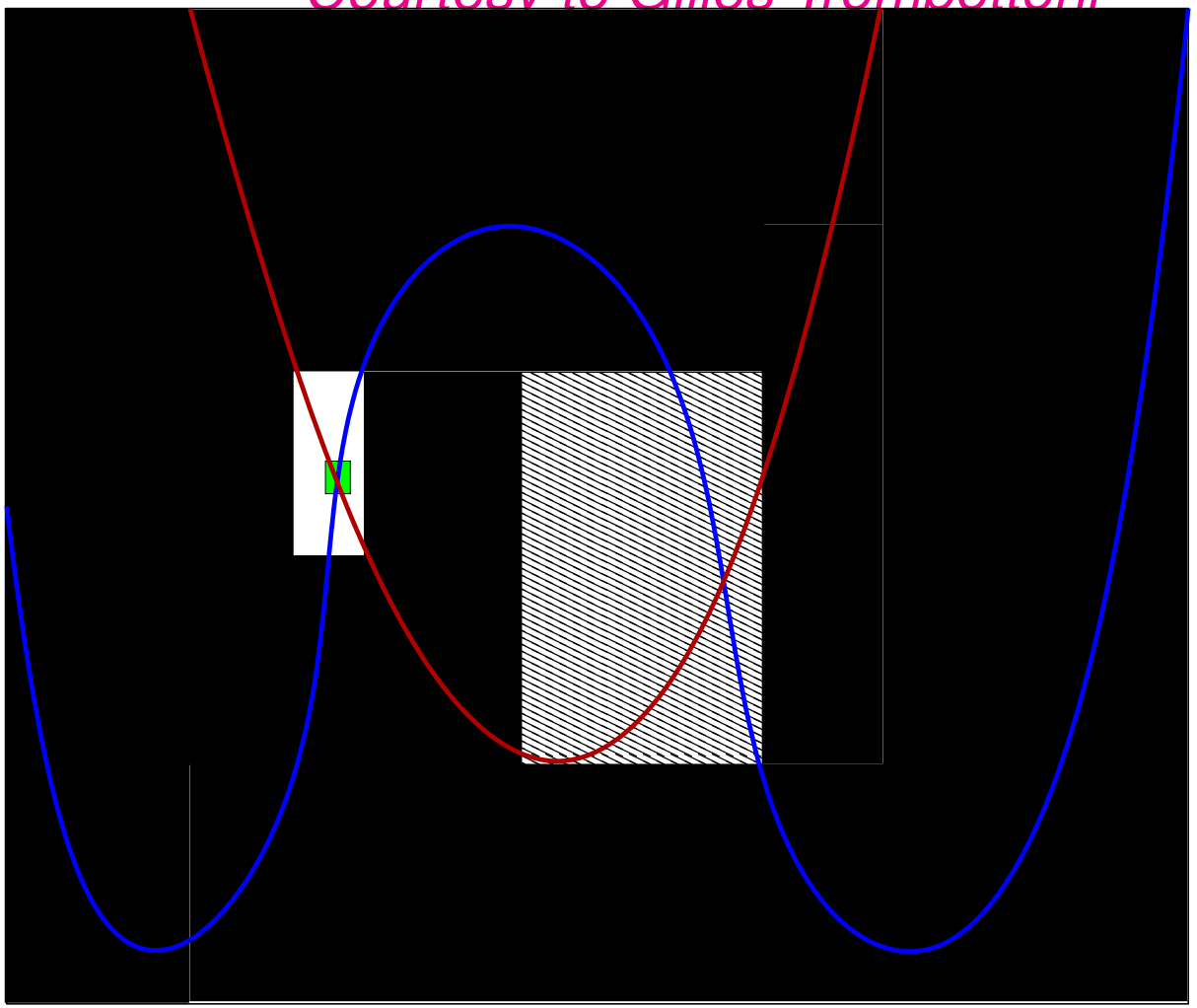
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



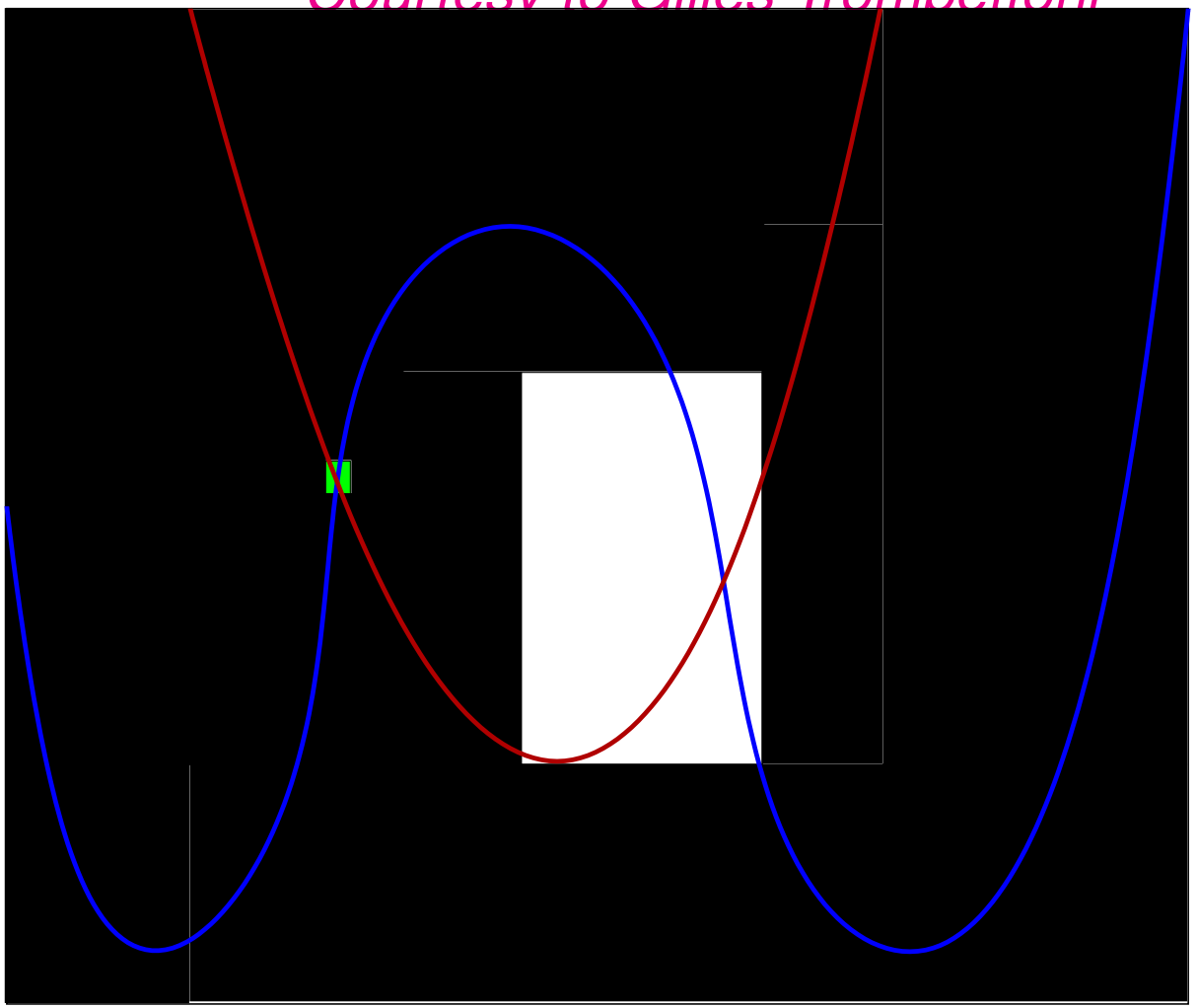
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



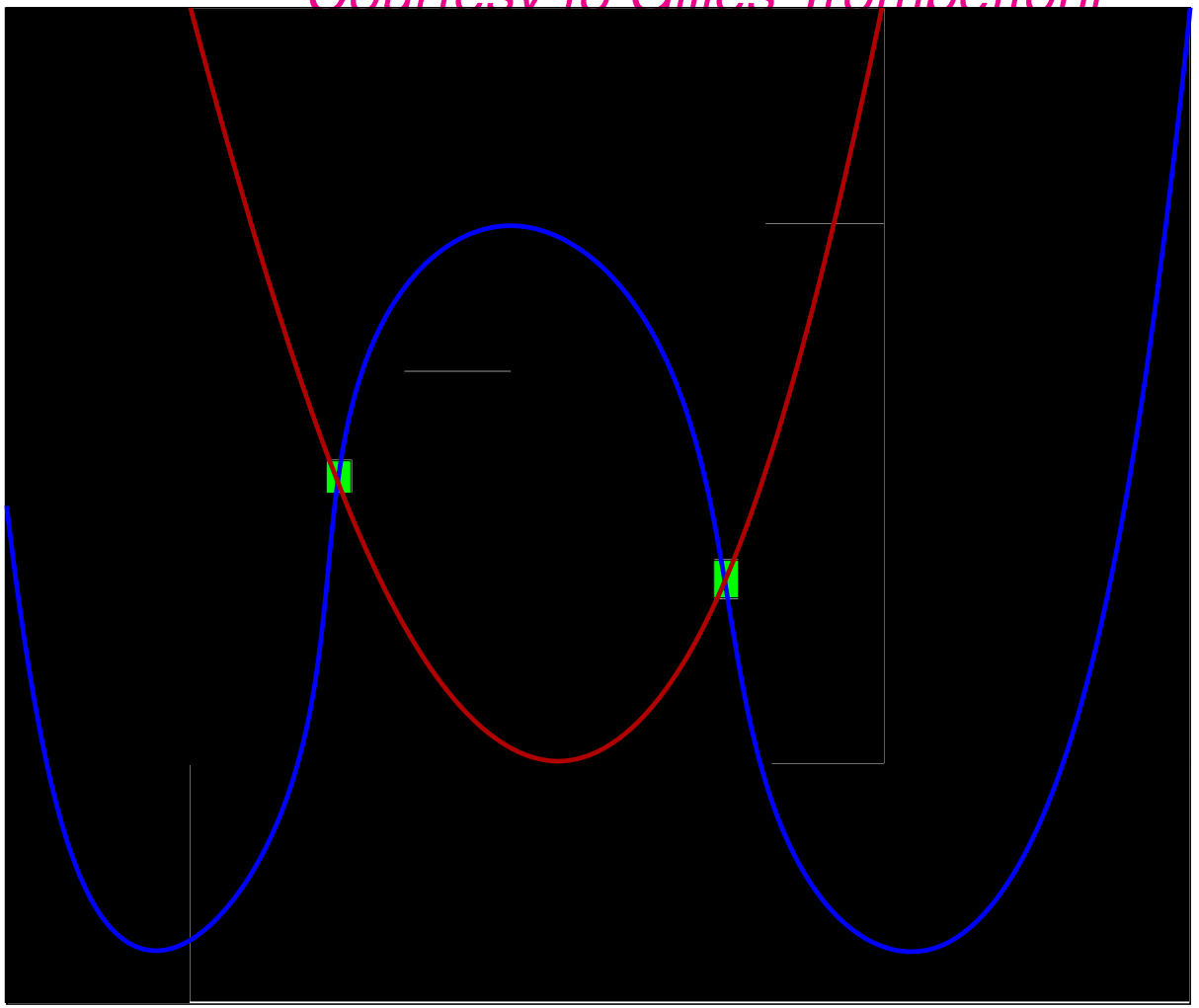
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



Notations

$c_j(x_1, \ldots, x_n)$	A relation over the real numbers
x	Real variable or vector of variables
X or D_x	The domain of variable x (i.e. intervals)
$X = [\underline{x}, \overline{x}]$	The set of real numbers x s.t. $\underline{x} \leq X \leq \overline{x}$
$\mathcal{C}$	The set of constraints
$\mathcal{R}^\infty$	$\mathcal{R} \cup \{-\infty, +\infty\}$, set of real numbers extended with infinity symbols
$\mathcal{F}$	The set of floating point numbers
a^+ (resp. a^-)	The smallest (resp. largest) number of $\mathcal{F}$ strictly greater (resp. lower) than a
$\tilde{k}$	smallest interval containing real number k
$\Phi_{cstc}(P)$	closure (filtering) by consistency of CSP P *cstc* stands for $2B, Box, 3B, Bound$

Interval extension

- In general, it is not possible to compute **the exact enclosure** of the range for an arbitrary function over the real numbers

$\rightarrow$ The interval extension of a function is an interval function that computes an **outer approximation** of the range of the function over a domain

Natural interval extension

F the **natural** interval extension of a real function f is obtained by replacing:

- Each constant k by its natural interval extension $\tilde{k}$
- Each variable by a variable over the intervals
- Each mathematical operator in f by its **optimal** interval extension

Optimal extensions for basic operators:

- $[a,b] \ominus [c,d] = [a-d, b-c]$
- $[a,b] \oplus [c,d] = [a+c, b+d]$
- $[a,b] \otimes [c,d] =$
  $[\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)]$
- $[a,b] \oslash [c,d] = [\min(\frac{a}{c},\frac{a}{d},\frac{b}{c},\frac{b}{d}), \max(\frac{a}{c},\frac{a}{d},\frac{b}{c},\frac{b}{d})]$
  if $0 \notin [c,d]$
  otherwise $\rightarrow [-\infty, +\infty]$

Interval extension: properties

- If $\mathbf{0 \notin F(X)}$, then no value exists in the box X such that $f(x) = 0$

  $\rightarrow$

  Equation $f(x)$ does not have any root in the box X

- Interval arithmetic can be implemented taking into account **round-off errors**

Problems when computing the image of an interval function

- **Outward Rounding** (required for safe computations with floating point numbers)
- **Non continuity** of interval functions: the image of an interval is in general not an interval
  $\rightarrow$ The **wrapping effect**, which overestimates by a unique vector the image of an interval vector
  Example:
  $f(x) = \frac{1}{x}$ with $X = [-1, 1]$
  $F([-1, 1]) = \frac{1}{[-1,1]} = [-\infty, -1] \cup [1, +\infty]$
  $\rightarrow = [-\infty, +\infty]$
- **Dependency problem** when a variable has multiple occurrences
  $X = [0, 10] \rightarrow X - X = [-10, 10]$

Pruning

- **Intuition:**
  Consider $X = [\underline{x}, \overline{x}]$ and $C(x, x_1, \ldots, x_n) \in \mathcal{C}$: if $C(x, x_1, \ldots, x_n)$ does not hold for any values $a \in [\underline{x}, x']$, then X may be shrunken to $X = [x', \overline{x}]$
- ***AC*-consistency (finite domains):**
  $c(x, y)$ is *AC*-consistent if each value in the domain of x has a support in the domain of y

  $\rightarrow$ A **constraint C_j is *AC*-like-consistent** if for any variable x_i in $\mathcal{X}_j$, **the bounds $\underline{D_i}$ and $\overline{D_i}$ have a support** in the domains of all other variables of $\mathcal{X}_j$

  $\rightarrow$ Constraint system C satisfies a partial consistency property if **a relaxation of C is consistent**

Local consistencies (1)

- **2B–consistency / Hull–consistency** only requires to check the Arc–Consistency property **for each bound** of the intervals

  Variable x with $X = [\underline{x}, \overline{x}]$ is 2B–consistent for constraint $f(x, x_1, \ldots, x_n) = 0$ if $\underline{x}$ and $\overline{x}$ are the leftmost and the rightmost zero of $f(x, x_1, \ldots, x_n)$

- **Box–consistency** :

  $\rightarrow$ coarser relaxation of AC than 2B–consistency

  $\rightarrow$ **better filtering**

  Variable x with $X = [\underline{x}, \overline{x}]$ is Box–Consistent for constraint $f(x, x_1, \ldots, x_n) = 0$ if $\underline{x}$ and $\overline{x}$ are the leftmost and the rightmost zero of $\mathbf{F(X, X_1, \ldots, X_n)}$, the optimal interval extension of $f(x, x_1, \ldots, x_n)$

Local consistencies (2)

- Let be $F : \mathcal{I}^n \rightarrow \mathcal{I}$ the natural interval extension of $f : \mathcal{R}^n \rightarrow \mathcal{R}$ and
  $f_{sol} = \square\{f(v_1, \ldots, v_n) \mid v_1 \in I_1, \ldots, v_n \in I_n\}$
  $f_{sol} \equiv$ 2B–consistency / Hull–consistency

  If each variable has **only one occurrence** in f
  then $f_{sol} \equiv F(I_1, \ldots, I_n)$
  else $f_{sol} \subseteq F(I_1, \ldots, I_n)$

Local consistency filtering (1)

Algorithms that achieve a local consistency filtering are based upon projection functions

- **Solution functions** express the variable x_i in terms of the other variables of the constraint. The solution functions of $x + y = z$ are:
  $f_x = z - y, f_y = z - x, f_z = x + y$

- For complex constraints, **no analytic solution function may exist**
  Example: $x + log(x) = 0$

Local consistency filtering (2)

Analytic functions always exist when the variable to express in terms of the others appears only once in the constraint

$\longrightarrow$ Considers that each occurrence is a different new variable

For $x + log(x) = 0$ we obtain $\{x_1 + \log(x_2) = 0, x_1 = x_2\}$

$\longrightarrow$ $f_{x_1} = -\log(x_2)$, $f_{x_2} = \exp^{-x_1}$

- Decomposition does not change the semantics of the initial constraints system
- ... but **it amplifies the dependency problem**

2B-consistency filtering(1)

Algorithms that achieve 2B-consistency filtering are based upon projection functions

$\longrightarrow$ considers that **each occurrence is a different new variable**

$\longrightarrow$ initial constraints are decomposed into “primitive” constraints

Decomposition of a constraint system (1)

- The decomposition of constraint $\mathcal{C}$ in a set of primitives constraints $decomp(\mathcal{C})$ does not change the semantics: $\mathcal{C}$ and $decomp(\mathcal{C})$ the same solutions
- The **scope** of the verification done by 2B–filtering algorithms is **reduced** by the decomposition: if variable x has multiple occurrences in constraint $c \in \mathcal{C}$, then the different occurrences of x in $decomp(c)$ may take **different values**

$\rightarrow$ 2B–filtering of $decomp(\mathcal{C})$ will be **weaker** than 2B–filtering of $\mathcal{C}$

Decomposition of a constraint system (2)

Example:

Let be $c : x_1 + x_2 - x_1 = 0$ with $D_{x_1} = [-1, 1],\ D_{x_2} = [0, 1]$

$\rightarrow$ $decomp(c) = \{x_1 + x_2 - x_3 = 0, x_1 = x_3\}$
with $D_{x_3} = [-1, 1]$

Each projection function of $decomp(c)$ can be computed with operations of the interval calculation

Constraint c is not 2B-consistent since $x_2 = 1$ does not have any support
...whereas $decomp(c)$ **is 2B-consistent**: $x_1 = -1$ and $x_3 = 0$ satisfy $x_1 + x_2 - x_3 = 0$ when $x_2 = 1$

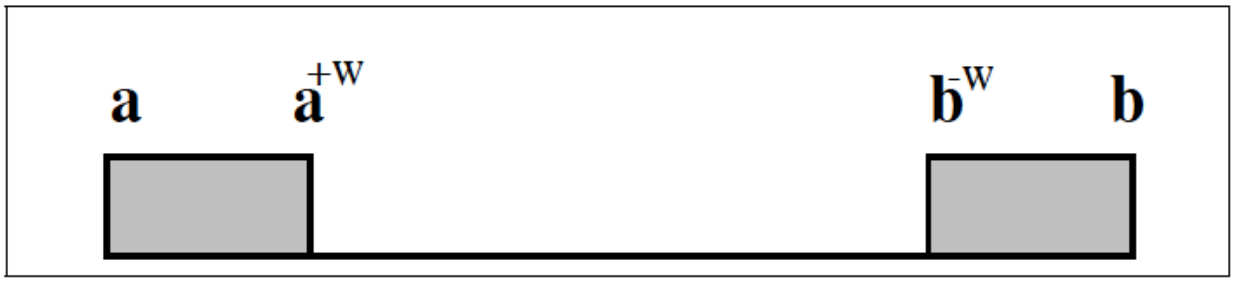
Early stopping of the propagation algorithm

In case of **asymptotic convergence**, it is not realistic to try to reduce the intervals until no more floating point number can be removed !

$\longrightarrow$ **To Stop the propagation** before reaching the fixed point

“Width” of the bound

a^{+w} stands for $(w+1)^{th}$ float after a
a^{-w} stands for $(w+1)^{th}$ float before a



2B(w)–Consistency

Let be $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ a CSP, $x \in \mathcal{X}$, $D_x = [a, b]$, w a positive integer D_x is **2B(w)–Consistent** for variable x if:

1. $\exists v \in [a, a^{+w})$ and v is the leftmost zero of $f(x, x_1, \ldots, x_n)$
2. $\exists v' \in (b^{-w}, b]$ and v' is the rightmost zero of $f(x, x_1, \ldots, x_n)$

Problems with 2B(w)-Consistency

- 2B(w)-Consistency filtering **depends on the evaluation order** of projection functions (no fixed point)

- There is no direct relationship between the value of w and the accuracy of filtering

Example of 2B(w)-Consistency filtering (1)

Let be:

$c_1 : x = y, \quad c_2 : x = z, \quad c_3 : x^2 = 4$

$D_x = [-10, 2], D_y = [-2, 2], D_z = [-1.5, 2]$ and $w = 1$

- **Order 1 :** $\langle c_1, c_2, c_3 \rangle$

  $Q = \{\langle c_1, x\rangle, \langle c_1, y\rangle, \langle c_2, x\rangle, \langle c_2, z\rangle, \langle c_3, x\rangle$

  1. Selection of $\langle c_1, x\rangle$, $D_x \leftarrow [-\mathbf{2}, \mathbf{2}]$
  2. Selection of $\langle c_1, y\rangle$, D_y is not changed
  3. Selection of $\langle c_2, x\rangle$, D_x is not changed because the reduction is lower than w
  4. Selection of $\langle c_2, z\rangle, \langle c_3, x\rangle$ cannot achieve any reductions

Example of 2B(w)-Consistency filtering (2)

$c_1 : x = y, \quad c_2 : x = z, \quad c_3 : x^2 = 4$
$D_x = [-10, 2], D_y = [-2, 2], D_z = [-1.5, 2]$ and $w = 1$

- **Order 2:** $\langle c_2, c_1, c_3 \rangle$
  $Q = \{\langle c_2, x \rangle, \langle c_2, z \rangle, \langle c_1, x \rangle, \langle c_1, y \rangle, \langle c_3, x \rangle$
  1. Selection of $\langle c_2, x \rangle$, $D_x \leftarrow [-1.5, 2]$
     No addition in Q
  2. Selection of $\langle c_2, z \rangle, \langle c_1, x \rangle, \langle c_1, y \rangle$ cannot achieve any reductions
  3. Selection of $\langle c_3, x \rangle$, $D_x \leftarrow [\mathbf{2}, \mathbf{2}]$
     $\langle c_1, y \rangle$ and $\langle c_2, z \rangle$ are pushed in Q
  4. Selection of $\langle c_1, y \rangle$, $D_y \leftarrow [\mathbf{2}, \mathbf{2}]$
  5. Selection of $\langle c_2, z \rangle$, $D_z \leftarrow [\mathbf{2}, \mathbf{2}]$

Box–consistency filtering

Transformation of the constraint $C_j(x_{j_1}, \ldots x_{j_k})$ into k **mono-variable constraints** by substituting all variables but one by their intervals

- The two extremal zeros of $C_{j,I}$ can be found by a **dichotomy algorithm** combined with a mono-variable version of the **interval Newton method**
- Box–consistency does not amplify the locality problem but it may generate **a huge number of constraints**

3B–Consistency

3B–Consistency, a relaxation of path consistency

$\longrightarrow$

checks whether 2B–Consistency can be enforced when the domain of a variable is **reduced to the value of one of its bounds** in the whole system

3B–Consistency (2)

Definition: 3B–Consistency

Let $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP and x a variable of $\mathcal{X}$ with $D_x = [a, b]$.

Let also:

- Let $P_{D_x^1 \leftarrow [a, a^+)}$ be the CSP derived from P by substituting D_x in $\mathcal{D}$ with $D_x^1 = [a, a^+)$
- Let $P_{D_x^2 \leftarrow (b^-, b]}$ be the CSP derived from P by substituting D_x in $\mathcal{D}$ with $D_x^2 = (b^-, b]$

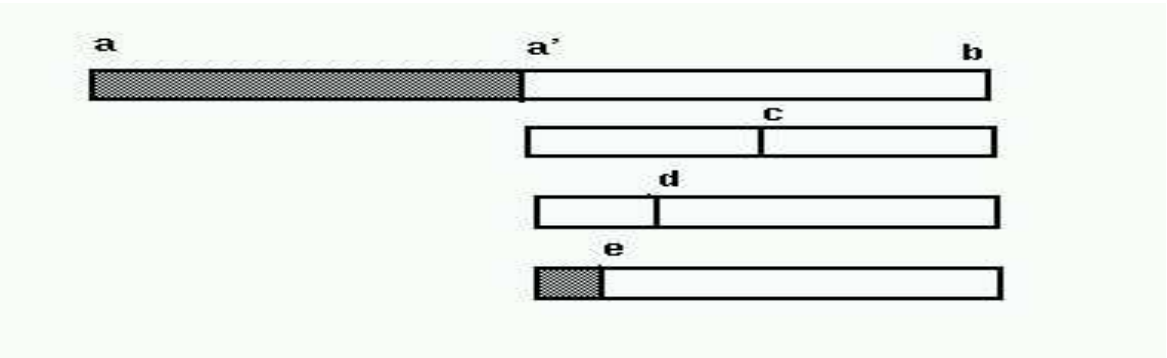
X is 3B–Consistent iff

$\Phi_{2B}(P_{D_x^1 \leftarrow [a, a^+)}) \neq P_\emptyset$ and $\Phi_{2B}(P_{D_x^2 \leftarrow (b^-, b]}) \neq P_\emptyset$

3B–Consistency (3)

Let $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP and $D_x = [a, b]$, if $\Phi_{2B}(P_{D_x \leftarrow [a, \frac{a+b}{2}]}) = \emptyset$

- then the part $[a, \frac{a+b}{2})$ of D_x will be removed and the filtering process continues on the interval $[\frac{a+b}{2}, b]$
- otherwise, the filtering process continues on the interval $[a, \frac{3a+b}{4}]$.



Relations between numeric CSP

- $\mathcal{D}' \subseteq \mathcal{D}$ means $D'_{x_i} \subseteq D_{x_i}$ for all $i \in 1..n$
- CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is **smaller** than $P' = (\mathcal{X}, \mathcal{D}', \mathcal{C})$ if $\mathcal{D} \subseteq \mathcal{D}'$, we note $P \prec P'$
- $P_\emptyset$ denotes the class of empty CSP (CSP with at least one empty domain)
  By convention $P_\emptyset$ is the smallest CSP.

Relation between Box–consistency and 2B-consistency (1)

General case: $\Phi_{2B}(P) \preceq \Phi_{Box}(P)$

Particular case: $\Phi_{2B}(P) \equiv \Phi_{Box}(P)$

if **no variable has multiple occurrences** in any constraint

Relation between Box–consistency and 2B–consistency (2)

A 2B-Consistent CSP is Box–consistency but a Box–consistent CSP may not be 2B-Consistent

Example:

$c : x_1 + x_2 - x_1 = 0, D_{x_1} = [-1, 1], D_{x_2} = [0, 1]$
is not be 2B-Consistent for x_2 but is Box–consistency for x_2 because
$([-1, 1] \oplus [0, 0^+] \ominus [-1, 1]) \cap [0, 0]$
and $([-1, 1] \oplus [1^-, 1] \ominus [-1, 1]) \cap [0, 0]$ are not empty

Box versus 2B(decomp)

2B-consistency on the decomposed system is weaker than Box–consistency on the initial system

$$\Phi_{Box}(P) \preceq \Phi_{2B}(\mathbf{P_{decomp}})$$

Proof:

For local consistencies CSP P_{decomp} is a relaxation of P

$\rightarrow$ 2B–consistency (P) $\preceq$ 2B–consistency (P_{decomp}).

Since there aren't any multiple occurrences of variables in P_{decomp}, $\Phi_{Box}(P_{decomp}) \equiv \Phi_{2B}(P_{decomp})$

and thus $\Phi_{Box}(P) \preceq \Phi_{2B}(P_{decomp})$

Implementation issues

- **Standard narrowing algorithm**
- **HC4-Revise** computes the optimal box (under continuity assumptions) when no constraint contains **multiple occurrences** of a variable
- **Box-Revise** computes the optimal box (under continuity assumptions) when each constraint contains at most **one** variable appearing several times

Standard narrowing algorithm (schema) (1)

1 **IN–1** (**in** $\mathcal{C}$, **inout** $\mathcal{D}$)
2
rangle $Q \leftarrow \{\langle x_i, C_j\rangle |\ C_j \in \mathcal{C}$ and $x_i \in Var(C_j)\}$
3 **while** $Q \neq \emptyset$
4 extract $\langle x_i, C_j\rangle$ from Q
5 $\mathcal{D}' \leftarrow \texttt{narrowing}(\mathcal{D}, x_i, C_j)$
6 **if** $\mathcal{D}' \neq \mathcal{D}$ **then**
7 $\mathcal{D} \leftarrow \mathcal{D}'$
8 $Q \leftarrow Q \cup \{\langle x_l, C_k\rangle | (x_l, x_i) \in Var(C_k)\}$
10 **endif**
11 **endwhile**

Standard narrowing algorithm (schema) (1)

$\rightarrow$ Computation of extremum functions in function `narrowing` of algorithm `IN-1`

1 **function narrow** $(\mathcal{D}, x_i, C_j)$: set of domains
2 $m \leftarrow Min_{x_i}(C, D_{x_i})$
3 $M \leftarrow Max_{x_i}(C, D_{x_i})$
4 **return** $\mathcal{D}[D_{x_i} \leftarrow [m, M]]$

2B-consistency filtering

Algorithm schema

- Generate **projection functions** for each variable of each constraint
- Use **interval extension** of the projection functions to compute $Min_{x_i}(C, D_{x_i})$ and $Max_{x_i}(C, D_{x_i})$

Computing Box–consistency filtering

La function $narrow(c, \mathcal{D})$ (generic algorithm `IN`) reduces the variable domains of c until c is Box–consistency :

- For each variable x of constraint , a **uni-variate interval function** is generated by replacing all variables but x by their domains
- Searching the leftmost zero and the rightmost zero of these uni-variate functions on intervals that are of the form:

$$C(D_{x_1}, .., D_{x_{i-1}}, x, D_{x_{i+1}}, ..., D_{x_k}) = \tilde{0}.$$

$\longrightarrow$ Computation of extremum functions in function `narrowing` of algorithm `IN-1`

Function LNAR (1)

Function LNAR computes the leftmost zero F_x for variable x with an initial domain I_x

- LNAR first reduces the interval I_x with function $NEWTON(F_x, I_x)$ (interval extension of Newton's method)
- If $NEWTON(F_x, I_x)$ cannot shrink enough I_x to make it Box–consistency , the domain is divided to check whether the left bound of I_x actually contains a zero
- Function $SPLIT$ splits interval I in two intervals I_1 and I_2; I_1 being the left part of the initial interval

Function LNAR (2)

```
function LNAR (IN: F_x, I_x, return Interval)
    r ← right(I_x)
    if  0 ∉ F_x(I_x) then return ∅
        else  I_x ← NEWTON(F_x, I_x)
              if  0 ∈ F_x([left(I_x), left(I_x)^+]) then return [left(I_x), r]
                  else  SPLIT(I_x, I_1, I_2)
                        L_1 ← LNAR(F_x, I_1)
                        if  L_1 ≠ ∅ then return [left(L_1), r]
                            else return [left(LNAR(F_x, I_2)), r]
                        endif
              endif
    endif
```

Figure: Function LNAR

Algorithm HC4

Goal

Limit the loss of information due to the decomposition of the constraints required by 2B–consistency filtering

Principle of algorithm HC4

- **HC4** works on a CSP where each constraint is represented by its **syntax tree**
  (no explicit decomposition: the nodes of the tree are primitive constraints)
- **HC4:** standard propagation scheme
- A projection is implemented by the function **HC4Revise** which shrinks the current box with a constraint c

BC4: similar to HC4, adapted for **Box-consistency filtering**

Algorithm HC4-Revise

Implementation of HC4-Revise

- Double exploration of the syntax tree of c
- Synthesis : **evaluation** (over intervals) at each node of the tree
- Heritage : **elementary projection** at each node of the tree

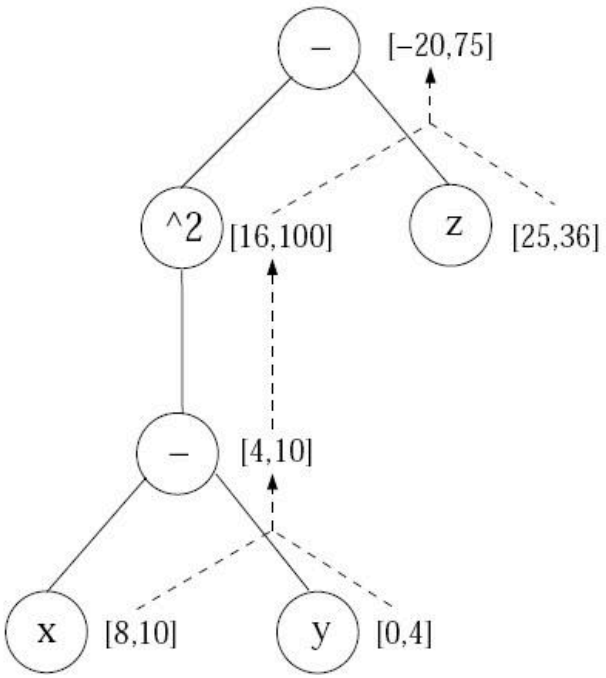
Algorithm HC4Revise

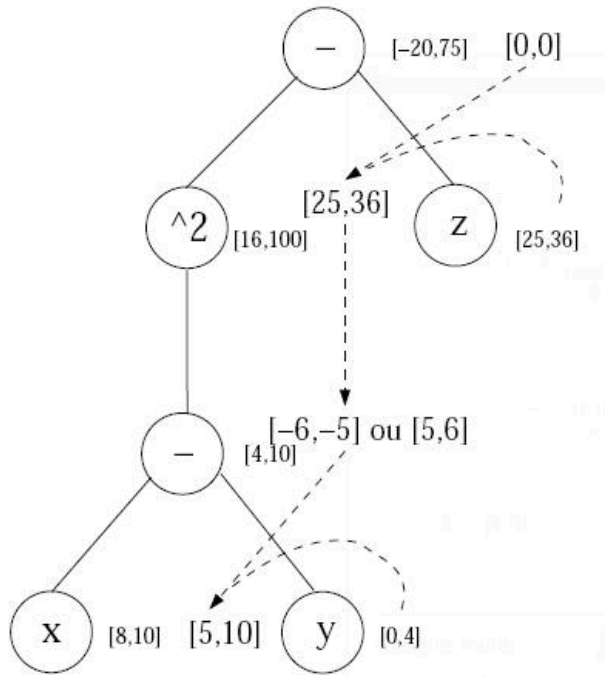
Principle of algorithm HC4Revise *on one constraint c*

- Double exploration of the syntax tree of c.
- Synthesis : **evaluation** (over intervals) at each node of the tree
- Heritage : **elementary projection** at each node of the tree

Example

Evaluation of $(x-y)^2 = z$ **Projection** over x
with $X = [8,10], Y = [0,4], Z = [25,36]$





What computes HC4Revise ?

For any constraint c without multiple occurrences of variables:

- **HC4Revise** computes **Hull-consistency**, the optimal projection of c
- If **gaps** are collected ($\longrightarrow$ domain is an union of intervals), HC4Revise computes **arc-consistency**

Global constraints

- Global constraints played a key role in the **success of CP on finite domains**
- QUAD
  - A **tight linear relaxation** of the quadratic constraints adapted from a classical **RLT techniques** (Sherali-Tuncbilek 92,Sherali-Adams 99)
  - Use of **LP algorithm** to narrow the domain of each variable
    $\longrightarrow$ the coefficient of these linear constraints are updated

Courtesy to Yahia Lebbah, Claude Michel

QUAD: Motivations

- **Quadratic equations and inequations are widely** used to model distance relations in numerous applications (kinematics, robotics,chemistry)
- Classical (local) filtering algorithms are unable to achieve a significant pruning because these constraints are **handled independently**
  - No way to catch the dependencies between constraints
  - Splitting is behind the success for small dimensions

QUAD: Safe use of Linear Relaxation

- **A global constraint** to handle a tight approximation of the constraint system with an LP solver
- ***Combines***
  - **safe and rigorous** linear relaxations
  - **local consistencies** and **interval methods**

The QUAD framework

- **Reformulation**
  - capture the linear part of the problem
    $\longrightarrow$ replace each non linear term by a new variable
    eg x^2 by y_i
- **Linearisation/relaxation**
  - introduce **redundant linear constraints**
    $\longrightarrow$ tight approximations of the non-linear terms (RLT)
- **Computing min(x)** $= \underline{x_i}$ and **max(x)** $= \overline{x_i}$ in LP

Linearisation of x^2

- $f(x) = x^2$ with $\underline{x} \leq x \leq \overline{x}$ is approximated by :

$$\begin{array}{ll} L_1(y, \alpha) & \equiv [(x - \alpha)^2 \geq 0]_I \text{ where } \alpha \in [\underline{x}, \overline{x}] \\ L_2(y) & \equiv (\underline{x} + \overline{x})x - y - \underline{x} * \overline{x} \geq 0 \end{array}$$

- $[(x - \alpha_i)^2 = 0]_I$ generates the tangents to $y = x^2$ at $x = \alpha_i$
- $L_1(y, \overline{x})$ and $L_1(y, \underline{x})$: underestimations of y
  $L_2(y)$: overestimation of y

QUAD **only computes** $L_1(y, \overline{x})$ and $L_1(y, \underline{x})$

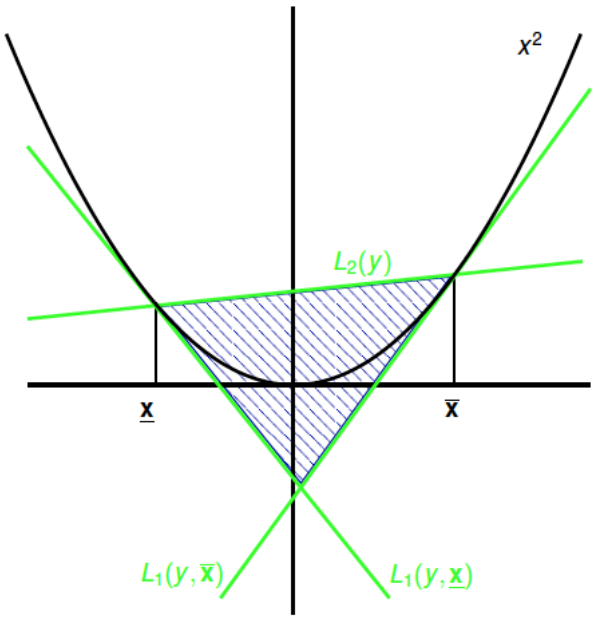
Linearisation of x^2

Example 1: relaxation of x^2 with $x \in [-4, 5]$

- $L_1(y, \alpha) \equiv y \geq 2\alpha x - \alpha^2$

$L_1(y, -4) : y \geq -8x - 16$

$L_1(y, 5) : y \geq 10x - 25$



- $L_2(y) \equiv y \leq (\underline{x} + \overline{x})x - \underline{x} * \overline{x}$

$L_2(y) : y \leq x + 20$

Linearisation of xy

Relaxation of xy

$$L_3(z) \equiv [(x - \underline{x_i})(y - \underline{x_j}) \geq 0]_I$$
$$L_4(z) \equiv [(x - \underline{x_i})(\overline{x_j} - y) \geq 0]_I$$
$$L_5(z) \equiv [(\overline{x_i} - x)(y - \underline{x_j}) \geq 0]_I$$
$$L_6(z) \equiv [(\overline{x_i} - x)(\overline{x_j} - y) \geq 0]_I$$

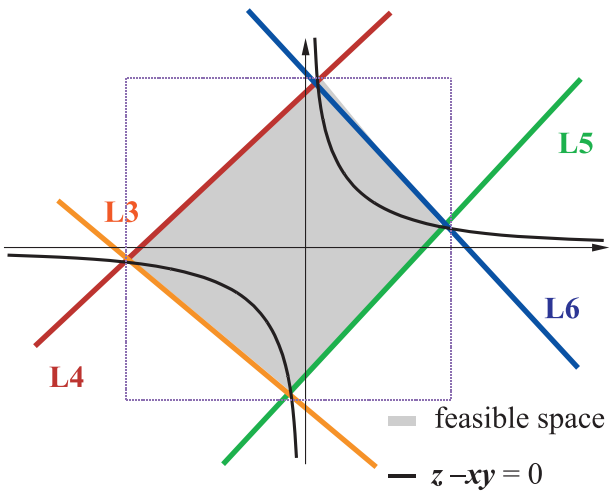
Example 2:

$z = xy$ with $x \in [-5, +5], y \in [-5, +5]$

$$L3(z) : z + 5x + 5y + 25 \geq 0$$
$$L4(z) : -z + 5x - 5y + 25 \geq 0$$
$$L5(z) : -z - 5x + 5y + 25 \geq 0$$
$$L6(z) : z - 5x - 5y + 25 \geq 0$$

Let's take $z = 5$

$$L3(z) : y \geq -x - 6$$
$$L4(z) : y \leq 4 - x$$
$$L5(z) : y \geq x - 4$$
$$L6(z) : y \leq 6 - x$$



QUAD filtering algorithm (1)

Function QUAD_filtering(IN: $\mathcal{X}$, $\mathcal{D}$, $\mathcal{C}$, ϵ) **return** $\mathcal{D}'$

1. **Reformulation**
   $\rightarrow$ linear inequalities $[\mathcal{C}]_R$ for the nonlinear terms in $\mathcal{C}$
2. **Linearisation/relaxation of the whole system** $[\mathcal{C}]_L$
   $\rightarrow$ a linear system $LR = [\mathcal{C}]_L \cup [\mathcal{C}]_R$
3. $\mathcal{D}' := \mathcal{D}$
4. **Pruning**

QUAD filtering algorithm (2)

- **Pruning**

  While reduction of some bound $> \epsilon$ and $\emptyset \notin \mathcal{D}'$ **Do**

  1. **Update the coefficients** of $[\mathcal{C}]_R$ according to $\mathcal{D}'$
  2. **Reduce the lower and upper bounds** $\underline{x}'_i$ and $\overline{x}'_i$ of each **initial** variable $x_i \in \mathcal{X}$ by computing min and max of x_i subject to LR with a LP solver
  3. **Propagate reductions** with local consistencies, newton

Issues in the use of linear relaxation

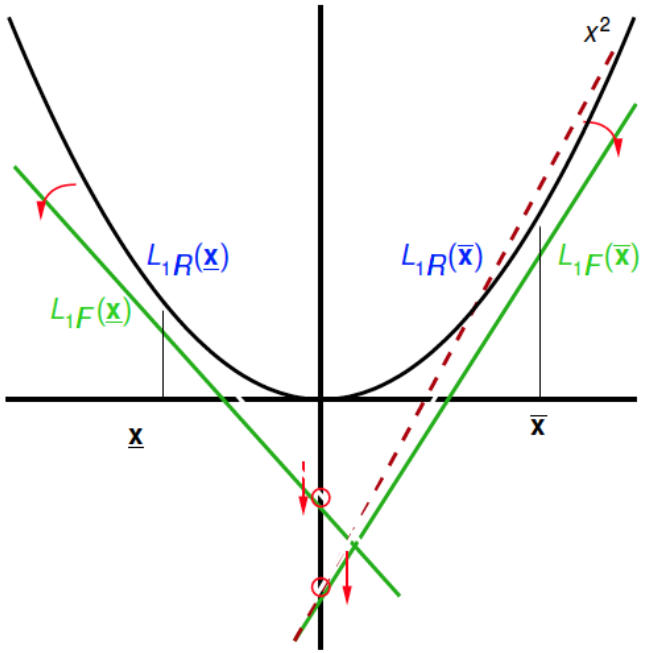
- Coefficients of linear relaxations are scalars
  $\Rightarrow$ computed with **floating point numbers**
- Efficient implementations of the simplex algorithm
  $\Rightarrow$ **floating point numbers**
- **All the computations with floating point numbers require right corrections**

Safe approximations of L_1

$$L_1(y, \alpha) \equiv y \geq 2\alpha x - \alpha^2$$

Effects of rounding:

- rounding of 2α
  $\Rightarrow$ rotation on y axis
- rounding of α^2
  $\Rightarrow$ translation on y axis



Safe approximations of L_1

[$L_{1F}(y, \alpha)$ approximations]

Let $\alpha \in F$ and

$$L_{1F}(y, \alpha) \equiv \begin{cases} y - \lfloor 2\alpha \rfloor x + \lceil \alpha^2 \rceil \geq 0 & \text{iff } \alpha \geq 0 \\ y - \lceil 2\alpha \rceil x + \lceil \alpha^2 \rceil \geq 0 & \text{iff } \alpha < 0 \end{cases}$$

$\forall x \in \mathbf{x}$, and $y \in [0, max\{\underline{x}^2, \overline{x}^2\}]$,

if $L_1(y, \alpha)$ holds, then $L_{1F}(y, \alpha)$ holds too

Generalisation to n-ary linearisations

Let $\sum_{i=1}^{n} a_i x_i + b \geq 0$
then $\forall x_i \in \mathbf{x}_i$:

$$\sum_{i=1}^{n} \overline{a}_i x_i + \sup(\overline{b} + \sum_{i=1}^{n} \sup(\sup(\mathbf{a}_i x_i) - \overline{a}_i \underline{x}_i)) \geq \sum_{i=1}^{n} a_i x_i + b \geq 0$$

Correction of the Simplex algorithm

Consider the following LP :

$$\text{minimise } c^T x$$

$$\text{subject to } \underline{b} \leq Ax \leq \overline{b}$$

- Solution = vector $x_R \in R^n$
- CPLEX computes a vector $x_F \in F^n \neq x_R$.
- x_F is safe for the objective if $c^T x_R \geq c^T x_F$
- Neumaier and Shcherbina
  - → ***cheap method to obtain a rigorous bound of the objective***
  - → ***rigorous computation of the certificate of infeasibility***

Quadrification: Power terms

A **power term** of the form x^n can be approximated by $n+1$ **inequalities** with a procedure proposed by Sherali and Tuncbilek , called “bound-factor product RLT constraints”
It is defined by the following formula:

$$[x^n]_R = \{[(x-\underline{x})^i(\overline{x}-x)^{n-i} \geq 0]_L, i = 0...n\} \quad (1)$$

Quadrification: product term (1)

For the **product term**

$$x_1 x_2 \ldots x_n \tag{2}$$

The **Quadrification** step brings back the multi-linear term into a **set of quadratic terms** as follows:

$$\begin{array}{rcccc}
\underbrace{x_1 x_2 \ldots x_n} & = & \underbrace{x_1 \ldots x_{d1}} & & \underbrace{x_{d1+1} \ldots x_n} \\
x_{1\ldots n} & = & x_{1\ldots d1} & \times & x_{d1+1\ldots n} \\
\hline
 & & \underbrace{x_1 \ldots x_{d2}} & & \underbrace{x_{d2+1} \ldots x_{d1}} \\
x_{1\ldots d1} & = & x_{1\ldots d2} & \times & x_{d2+1\ldots d1} \\
\hline
 & & \underbrace{x_{d1+1} \ldots x_{d3}} & & \underbrace{x_{d3+1} \ldots x_n} \\
x_{d1+1\ldots n} & = & x_{d1+1\ldots d3} & \times & x_{d3+1\ldots n} \\
\hline
\ldots & & & & \\
\hline
\end{array}$$

where $x_{i\ldots j} = [x_i x_{i+1} \ldots x_j]_L$.

Quadrification: product term (2)

For instance, consider the term $x_1 x_2 x_3 x_4 x_5$. The proposed quadrification process would operate in the following way:

$$
\begin{array}{|rcl|}
\hline
\underbrace{x_1 x_2 x_3 x_4 x_5}_{y_1} & = & \underbrace{x_1 x_2 x_3}_{y_2} \times \underbrace{x_4 x_5}_{y_3} \\
\hline
y_2 = \underbrace{x_1 x_2}_{y_4} & & \times \underbrace{x_3}_{x_3} \\
\hline
y_3 = \underbrace{x_4}_{x_4} & & \times \underbrace{x_5}_{x_5} \\
\hline
y_4 = \underbrace{x_1}_{x_1} & & \times \underbrace{x_2}_{x_2} \\
\hline
\end{array}
$$

Search

- Main **heuristics**
- Mind the **Gaps**

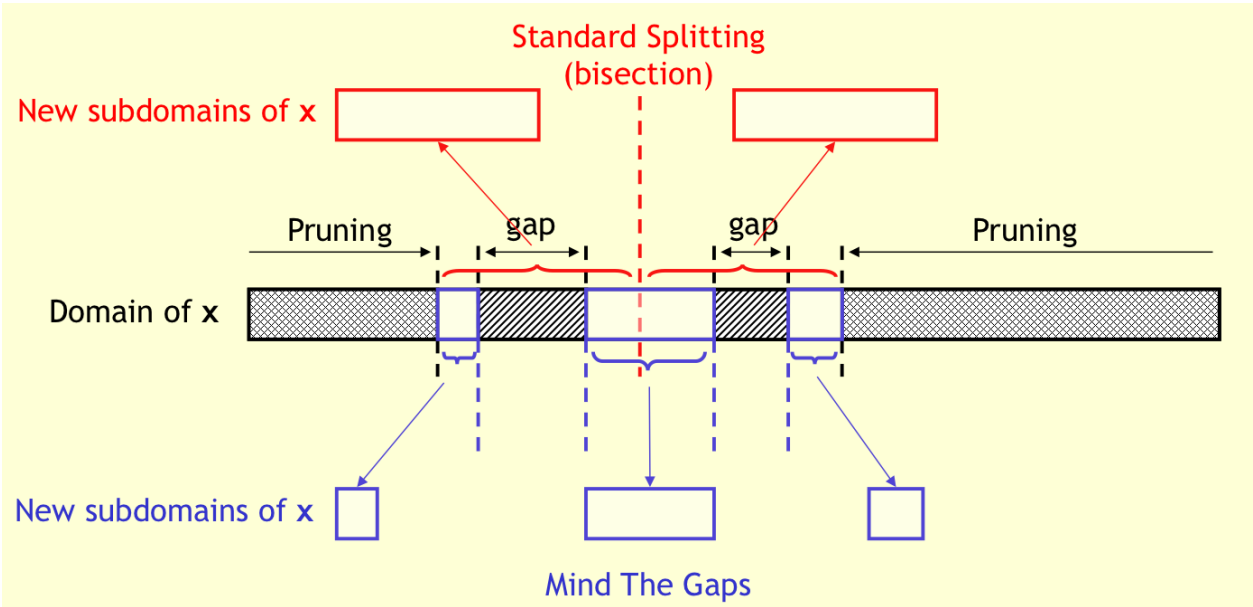
Main heuristics

In the search tree, **the choice of the next variable to bisect is very important**
Three heuristics are commonly used:

- **Round robin**
- **Select first the largest interval**
- **Smear** function (Kearfott 1990)
  - For each (f, x), in the current box $[B]$:
    compute $smear(f, x) = |\frac{\partial f}{\partial x}([B])| \times Diam([x])$;
  - For some variable x:
    $smear(x) = \sum_j (smear(f_j, x))$ (or $Max_j(smear(f_j, x))$) ;
  - Bisect the variable with the strongest impact.

Standard splitting vs Mind The Gaps

- **Collect** gaps while filtering (HC4 Revise)
- **Eliminate** non relevant gaps
- **Select** relevant gaps
- Generate **sub problems**



Conclusion

- **Local consitencies** $\longrightarrow$ power-full **refutation capabilities**
- **Main difficulty:** finding a good trade-off between pruning and search

- **Applications**
  - Global optimisation: **Boosting OBR**
    `http://www.essi.fr/~rueher/Publis/rc11.pdf`
  - Program verification: **Refining Approximations**
    `http://www.essi.fr/~rueher/Publis/nsv11.pdf`