Exploiting Monotonicity and Common Subexpressions for improving interval constraint propagation algorithms

Ignacio Araya

COPRIN, INRIA, Université de Nice-Sophia, France

MEA, December 2009, Paris, France

Outline

1. Introduction
   - Interval Arithmetics
   - Interval Methods for Solving NCSP
2. Exploiting monotonicity
   - Evaluation by monotonicity
   - Ocurrence Grouping
   - The `Mohc` algorithm
3. Exploting Common Subexpressions
   - Common Subexpression Elimination
   - Improving `HC4` by CSE
   - The I-CSE Algorithm

Outline

1. Introduction
   - Interval Arithmetics
   - Interval Methods for Solving NCSP
2. Exploiting monotonicity
   - Evaluation by monotonicity
   - Ocurrence Grouping
   - The `Mohc` algorithm
3. Exploting Common Subexpressions
   - Common Subexpression Elimination
   - Improving `HC4` by CSE
   - The I-CSE Algorithm

Basic notions of interval arithmetics

- An **interval** $[x] = [a, b]$ is the set of real numbers between a and b.
- $\underline{x}$, $\overline{x}$ are the left and right bounds of $[x]$.
- $diam([x]) = \overline{x} - \underline{x}$ is the **diameter** of $[x]$.
- A **box** $[B] = \{[x_1], ..., [x_n]\}$ is a vector of intervals.

Interval extensions

- $[f]$ is an **interval extension** of $f(x_1, x_2, ..., x_n)$ if the image $[f]([B])$ contains the image of $[B]$ under f, i.e.

$$[f]([B]) \supset \mathcal{I}f([B])$$

- The **optimal image** $[f]_{opt}([B])$ is the sharpest interval containing $\mathcal{I}f([B])$.

The natural interval extension

- The **natural (interval) extension** $[f]_n$ of f evaluates with interval arithmetics all the operators and elementary functions in f.
- Example: Consider the function $f(x) = x + x^2$. The natural extension of f with $[x] = [-1, 1]$ is:
  $[f]_n([-1, 1]) = [-1, 1] + [-1, 1]^2 = [-1, 2]$.
- If each variable occurs once in f, then $[f]_n = [f]_{opt}$.
- If a variable occurs several times in f, then $[f]_n \supset [f]_{opt}$, due to the **dependency problem**.
  - For example: $f(x) = x - x$ with $[x] \in [0, 1] \Rightarrow$
    $[f]_n([x]) = [-1, 1]$ ($[f]_{opt}([B]) = [0, 0]$).

Outline

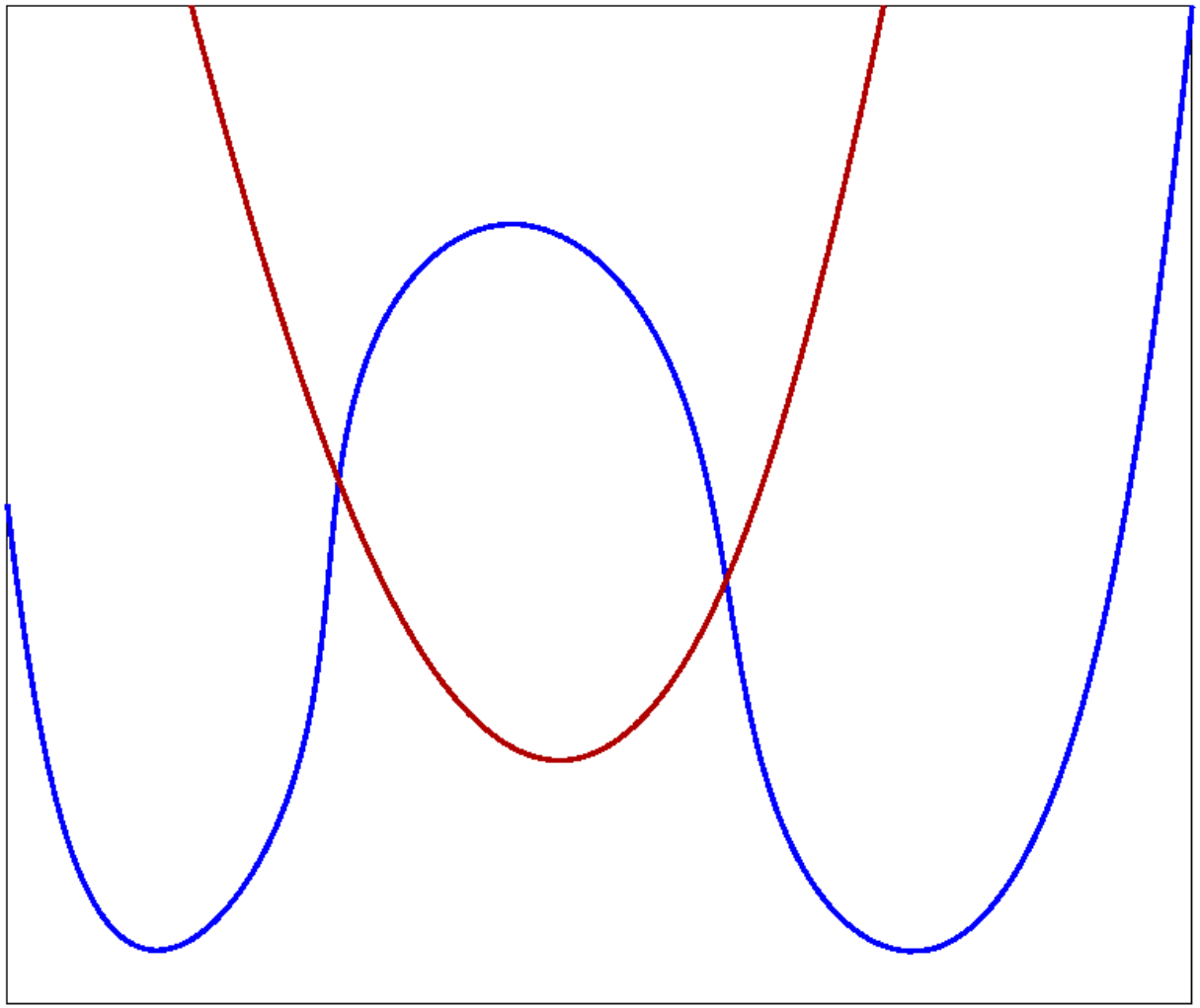
1. Introduction
   - Interval Arithmetics
   - Interval Methods for Solving NCSP
2. Exploiting monotonicity
   - Evaluation by monotonicity
   - Ocurrence Grouping
   - The `Mohc` algorithm
3. Exploting Common Subexpressions
   - Common Subexpression Elimination
   - Improving `HC4` by CSE
   - The I-CSE Algorithm

Basic concepts

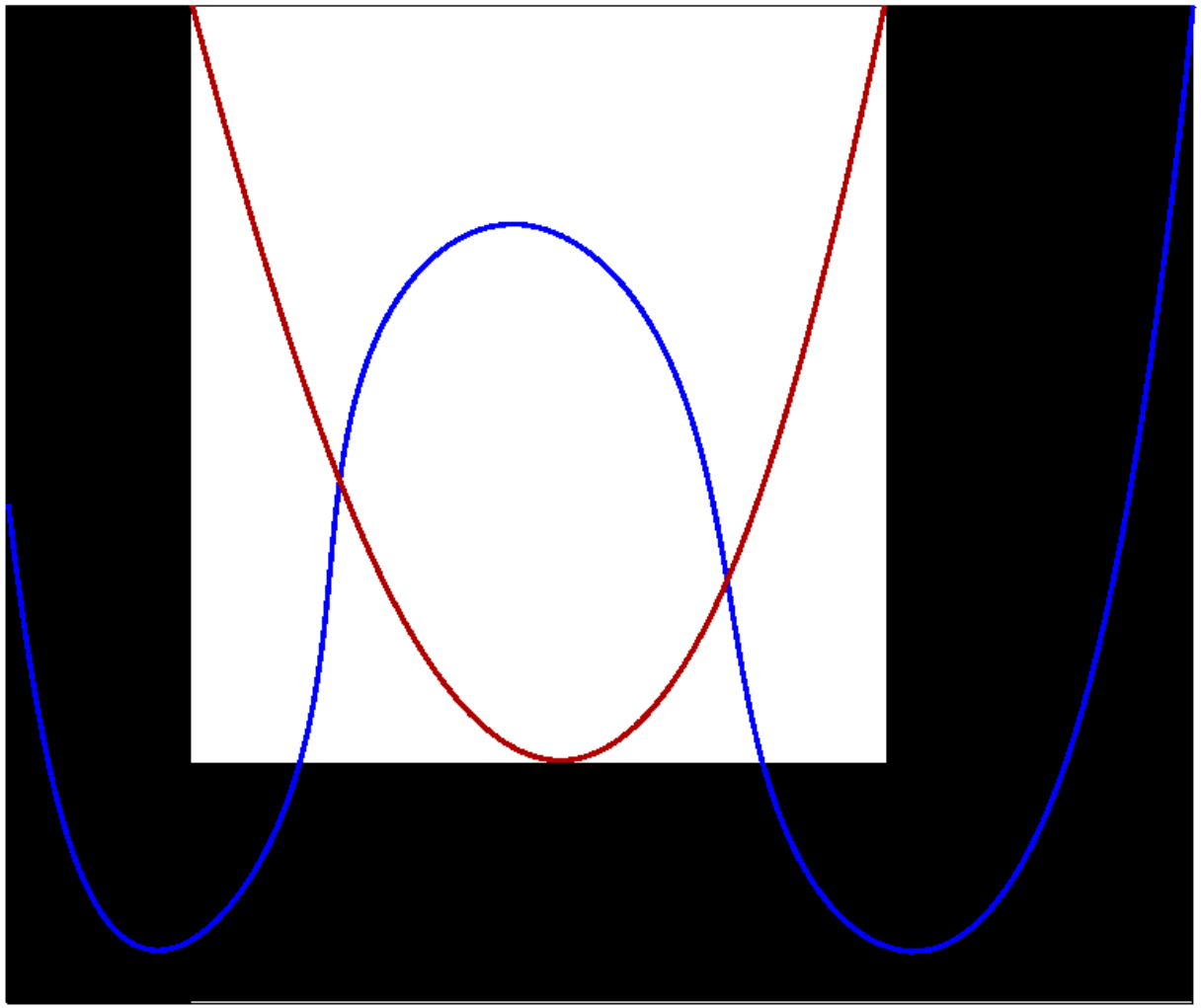
Definitions:

- **NCSP:** $P = (X, C, [B])$
  X: set of variables.
  C: set of constraints.
  $[B]$: box containing the domains of the variables in X.
- **Prune/contract:** To reduce a box on the bounds.
  - Interval Newton methods handle **the whole system** like a global contraint.
  - Propagation algorithms handle each constraint independently.
- **Branching:** Bisecting or splitting a box in 2 sub-boxes.

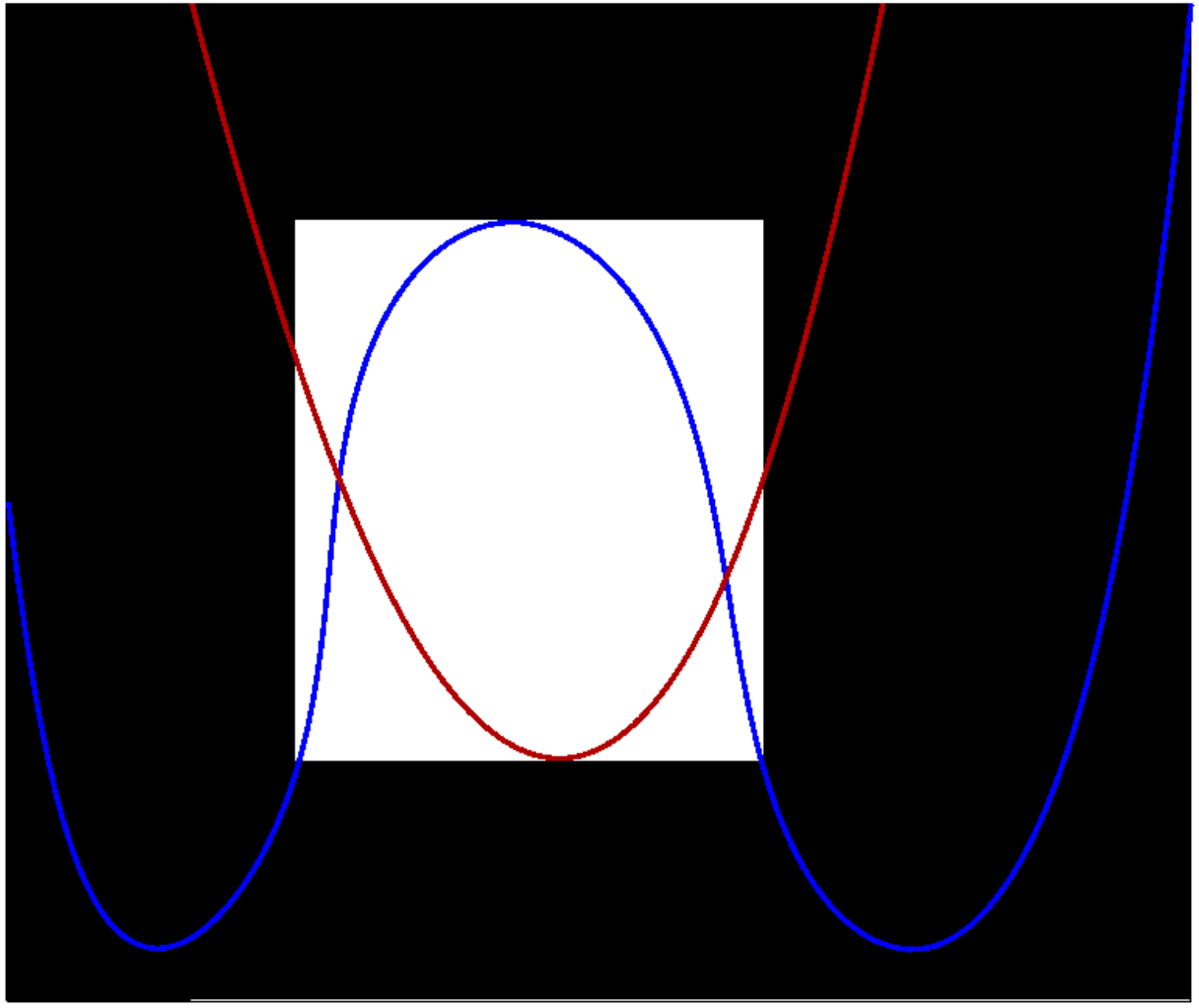
Branch and prune/contract



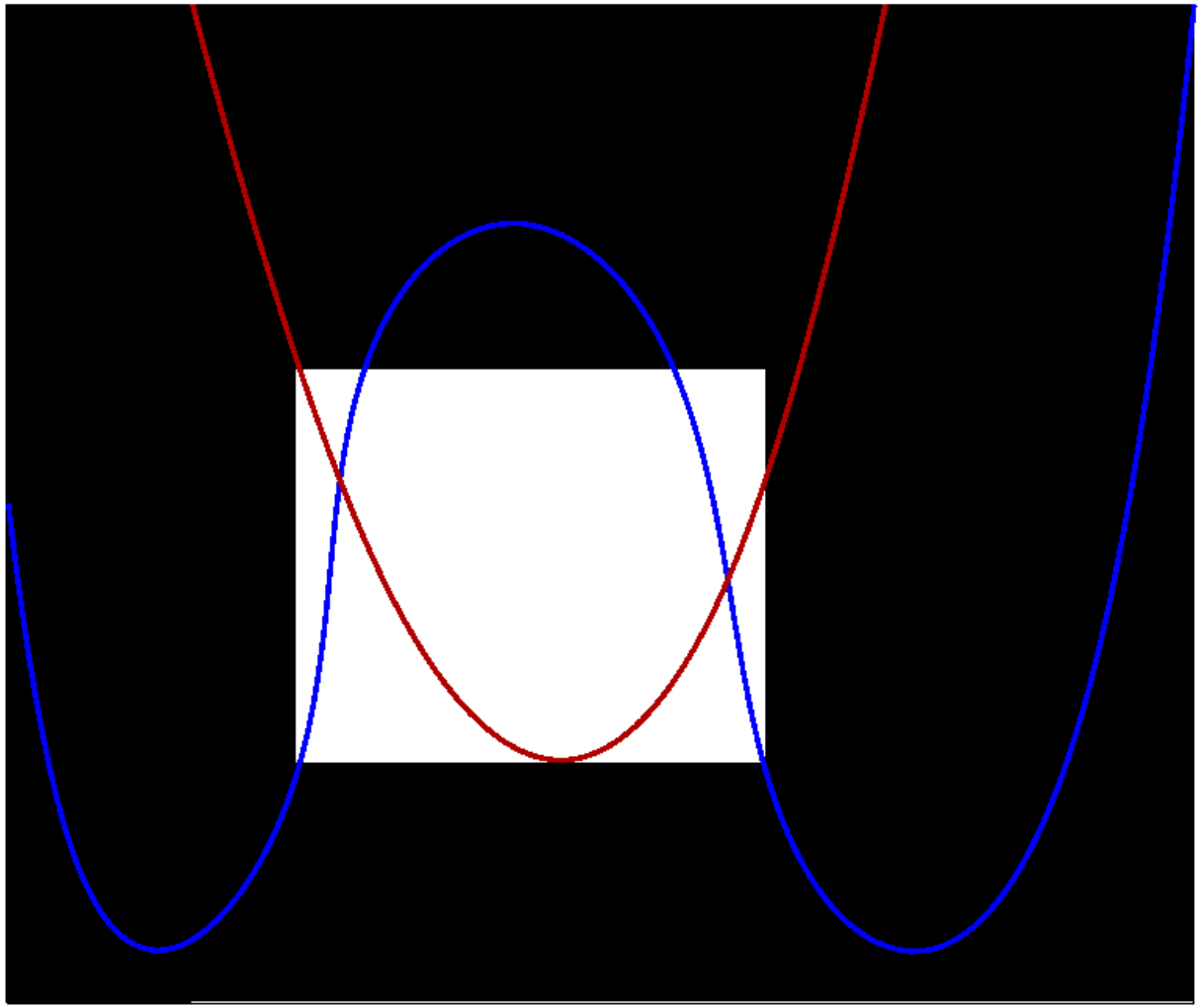
Branch and prune/contract



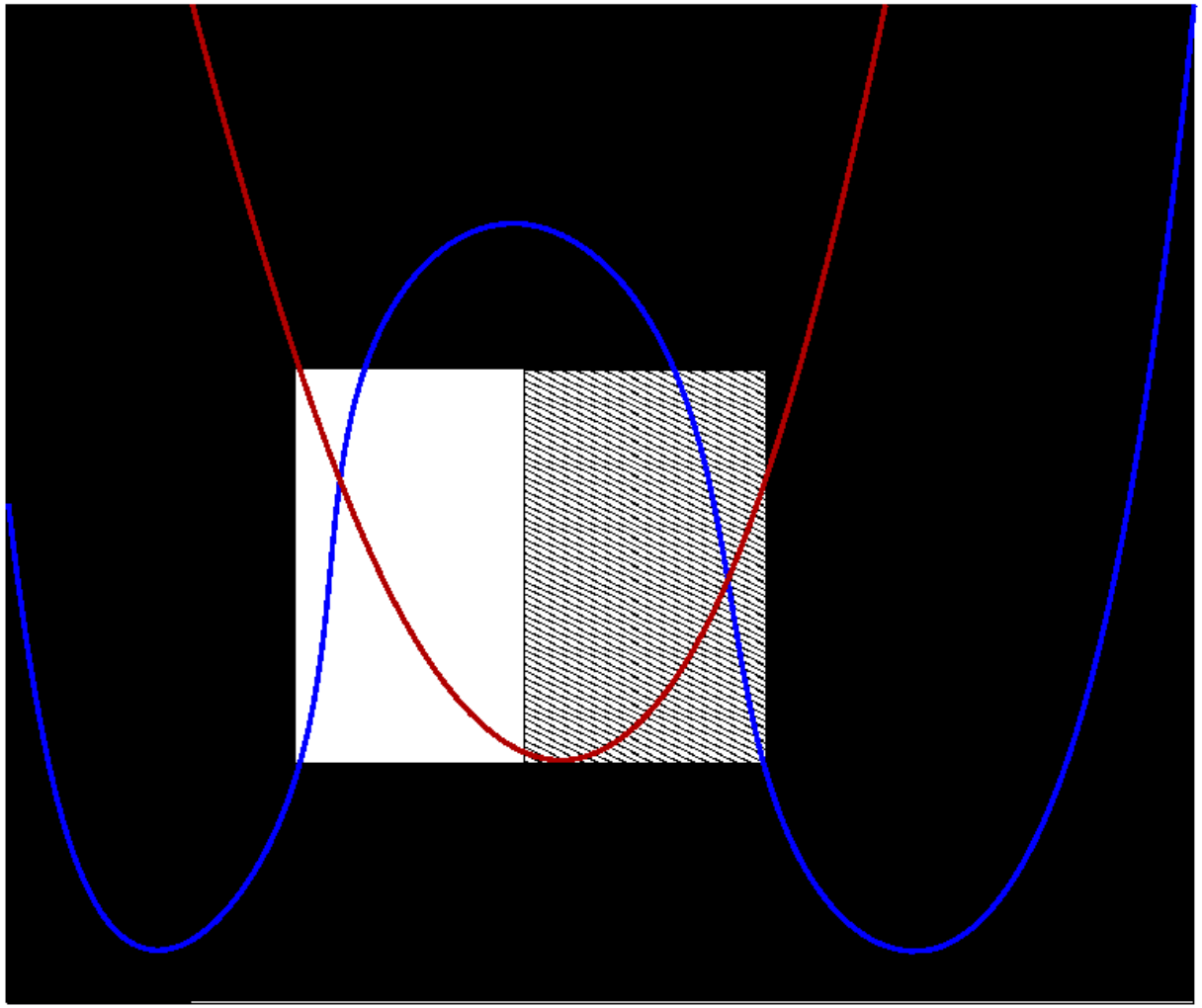
Branch and prune/contract



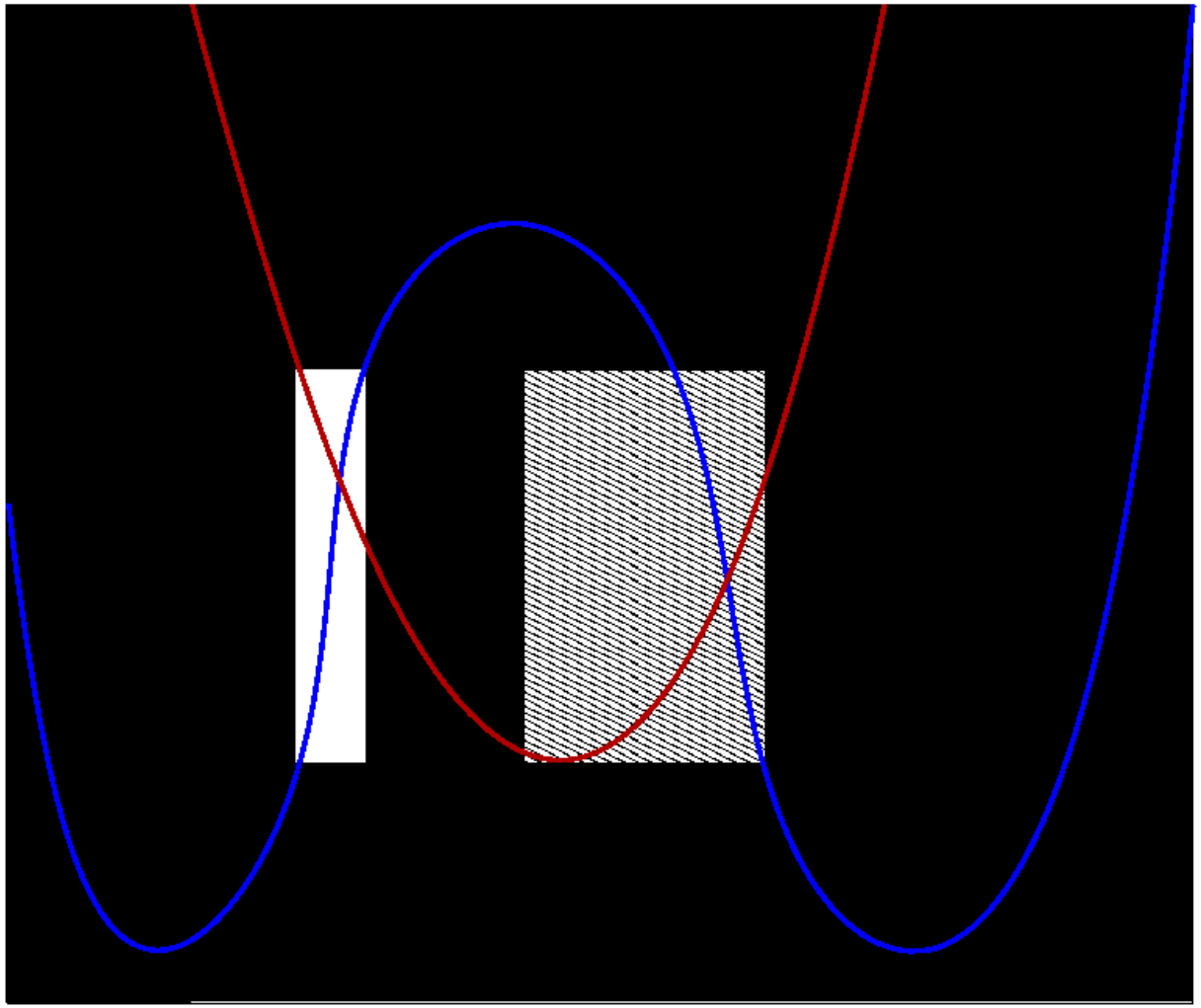
Branch and prune/contract



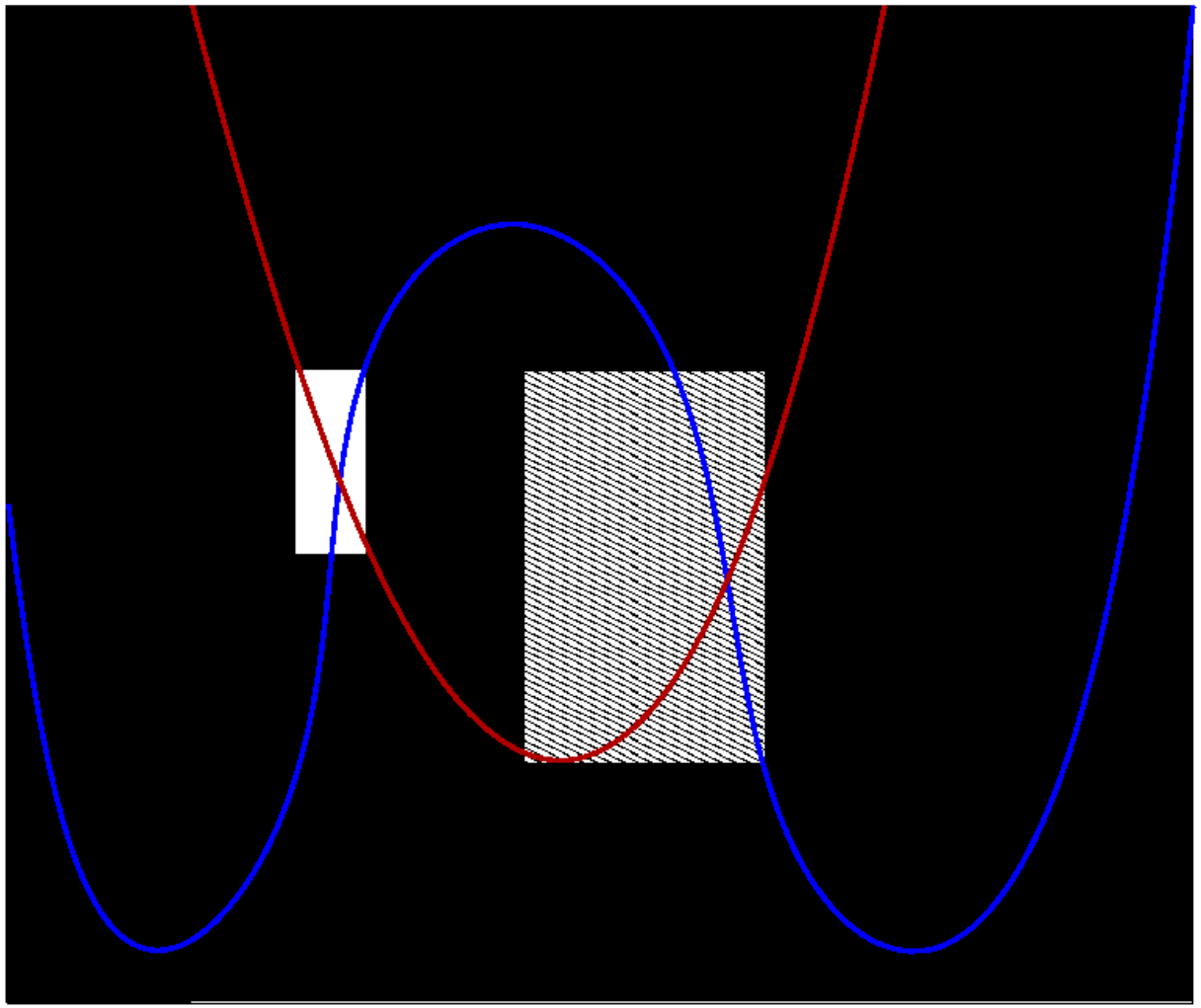
Branch and prune/contract



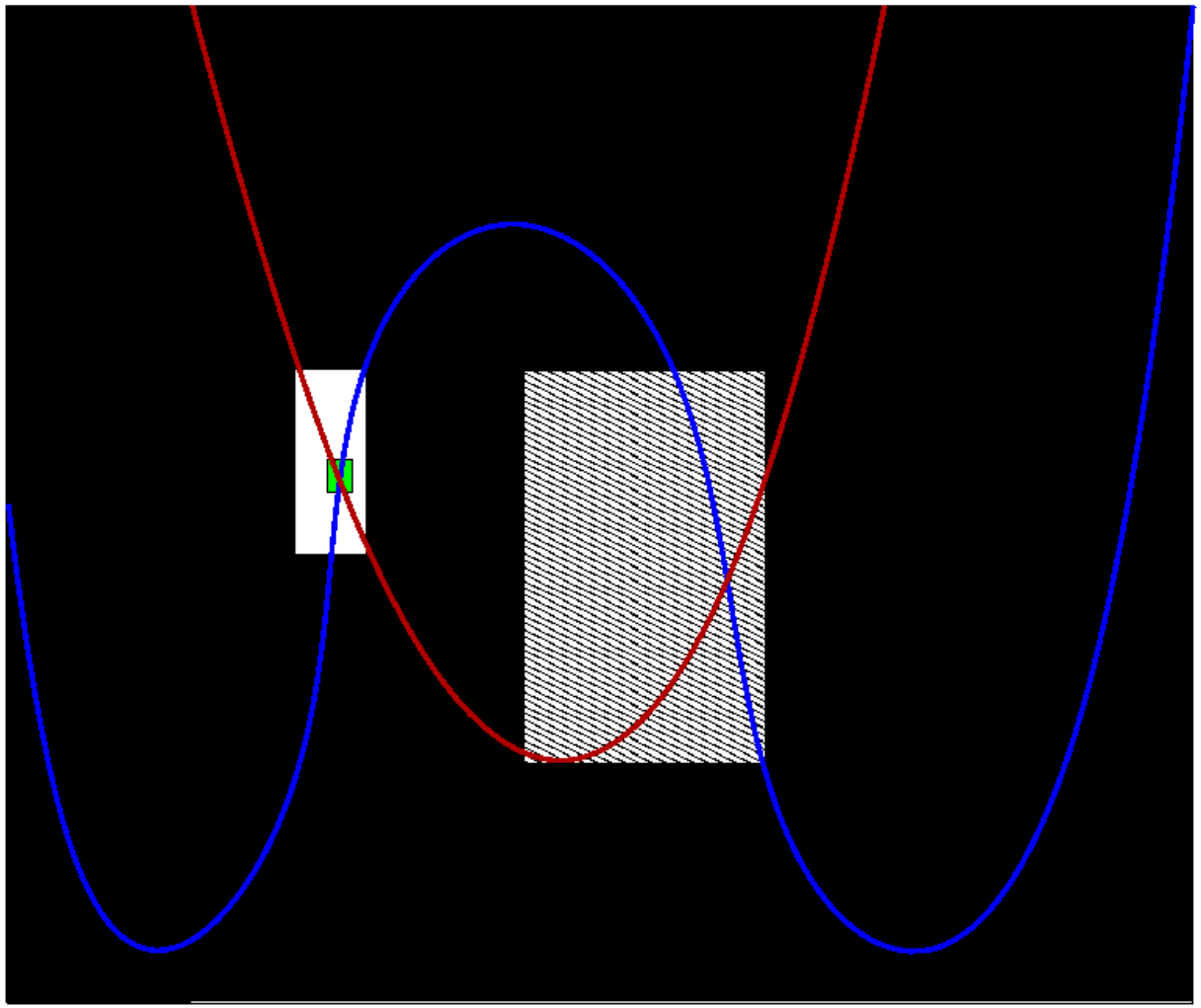
Branch and prune/contract



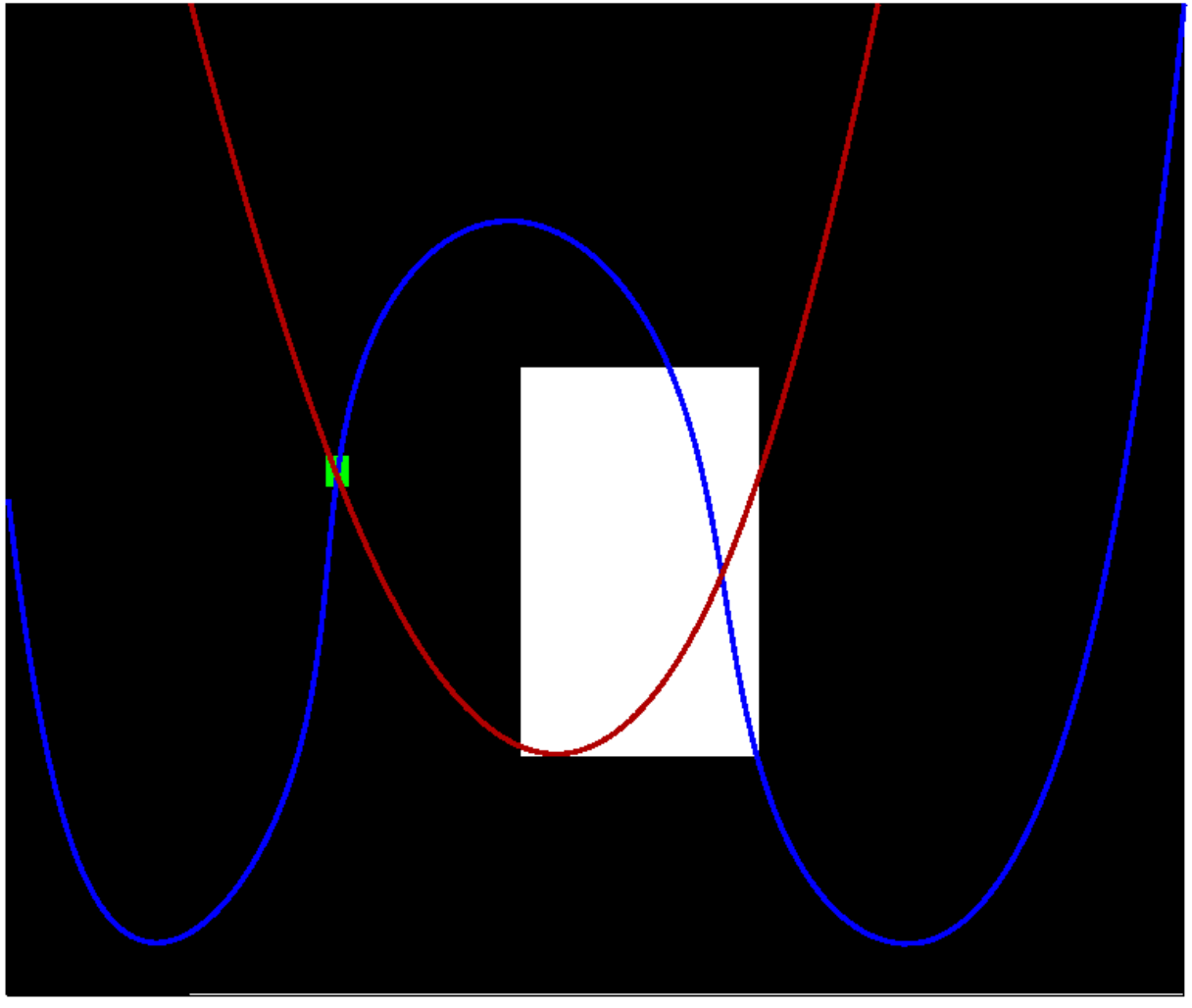
Branch and prune/contract



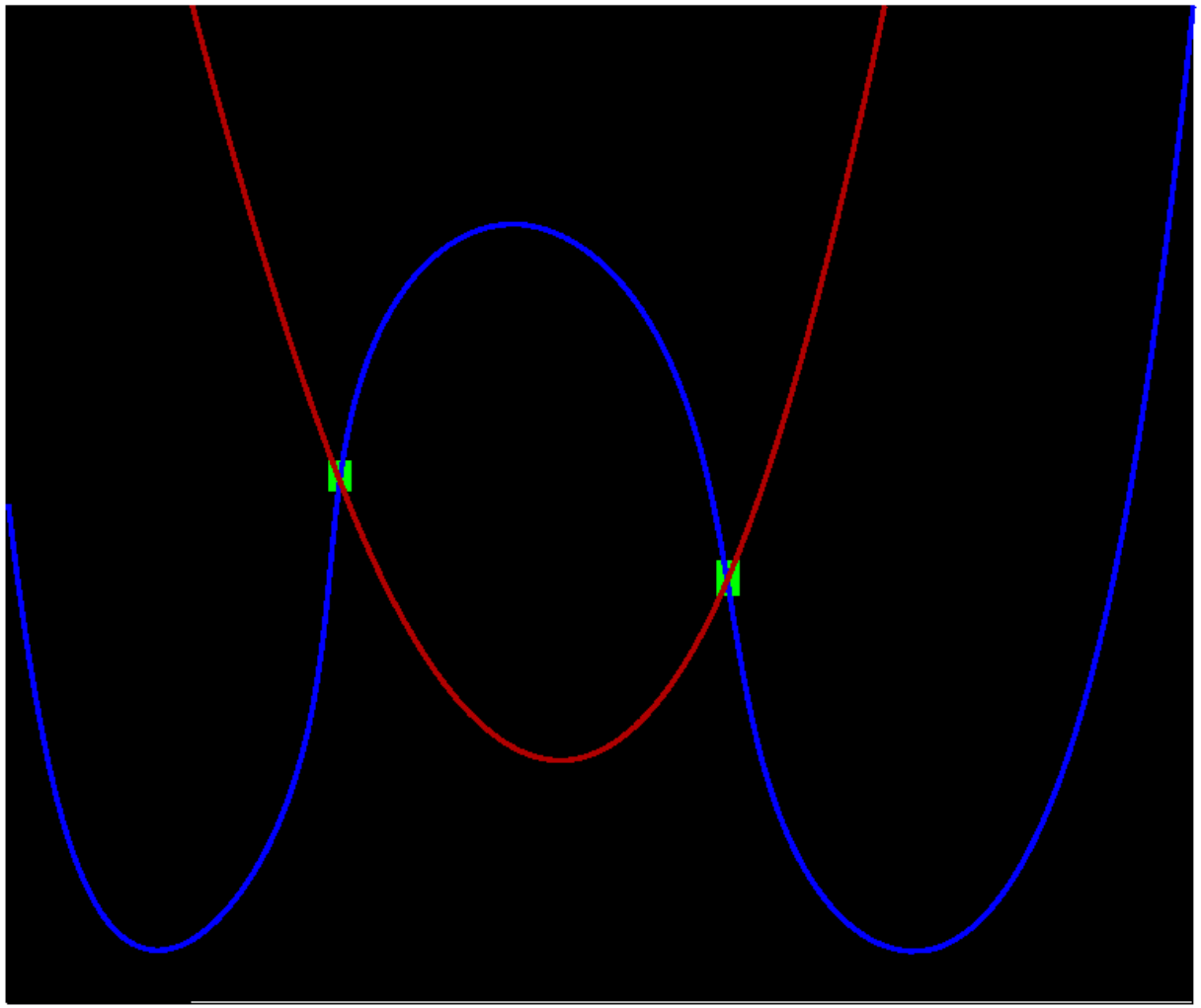
Branch and prune/contract



Branch and prune/contract



Branch and prune/contract



Constraint Propagation Algorithms

- Objective: Narrowing of the current box (on the bounds) with no loss of solution.
- Two procedures:
  - Revise procedure: Contract the box using one constraint.
  - Propagation procedure: The changes on the domains are propagated to the other constraints.

Hull-consistency

- Similar to arc consistency and bound consistency in finite domains.
- Enforcing hull-consistency in a constraint is equivalent to compute the smallest/optimal box containing all the solution of the constraint.
- The difficult comes from **multiple occurrences of variables**: the **dependency problem**.

Enforcing hull-consistency

Consider the constraint $c : (x - y)^2 = z$ with initial domains $[x] = [8, 10]$, $[y] = [0, 4]$ and $[z] = [25, 36]$.

- Projection over y ($y = x - \sqrt{z}$):

$$[y] \leftarrow [y] \cap \left([x] - \sqrt{[z]}\right) = [0, 4] \cap \left([8, 10] - \sqrt{[25, 36]}\right) = [2, 4]$$

- The optimal box is: {[8,10],[2,4],[25,36]}
- If we replace x by $2x - x$ then we lead with the dependency problem.
  Projection over y ($y = \mathbf{2x - x} - \sqrt{z}$):

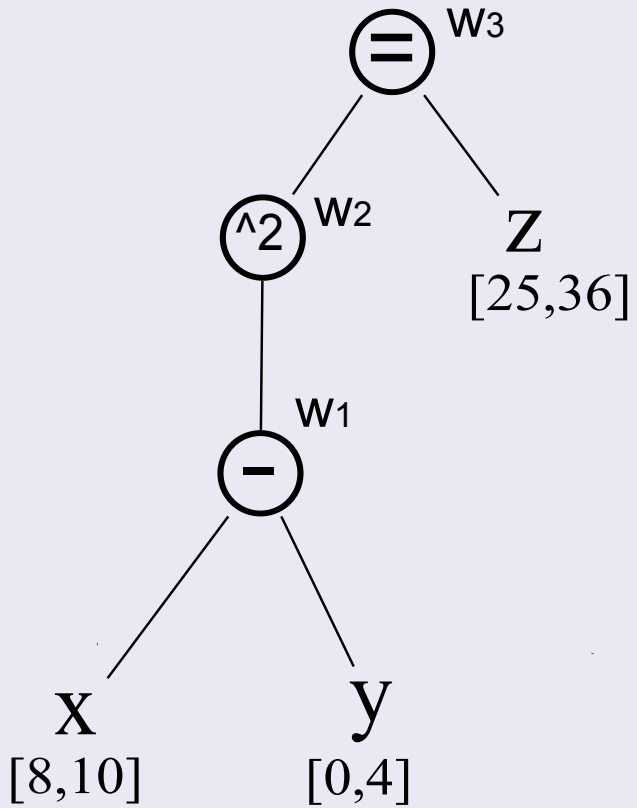
$$[y] \leftarrow [0, 4] \cap \left(\mathbf{[6, 12]} - \sqrt{[25, 36]}\right) = \mathbf{[0, 4]}$$

Revise procedures

- `HC4-Revise` computes the optimal box when a constraint contains **no** multiple occurrences of variables.
- The more costly `Box-Revise` procedure computes the optimal projection over a variable x, if x is the **only** variable appearing several times in the constraint.
- A new procedure called `MoHC-Revise` handles better the dependency problem when **several** variables occurs several times. (`MoHC-Revise` exploits the monotonicity of functions.)

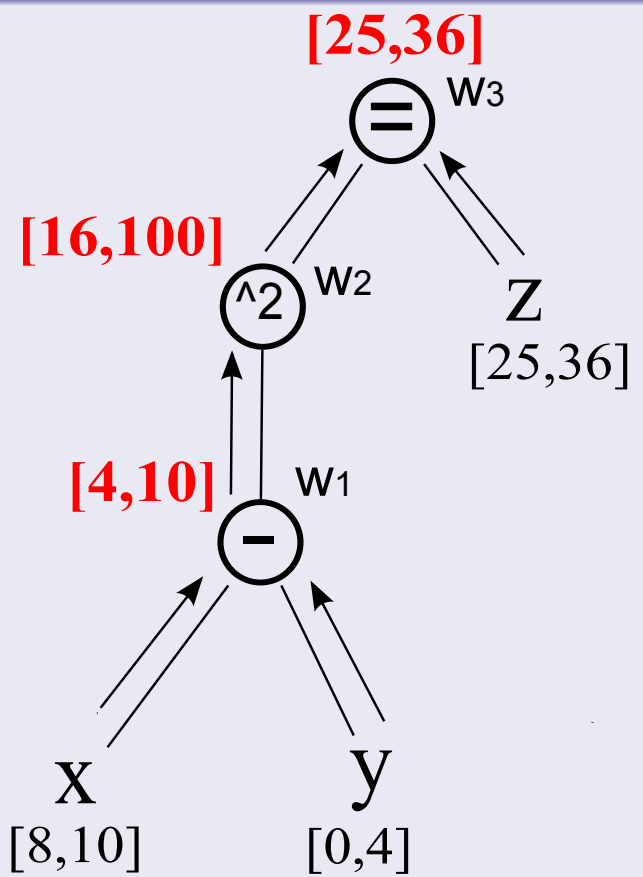
HC4-Revise

$(x - y)^2 = z$



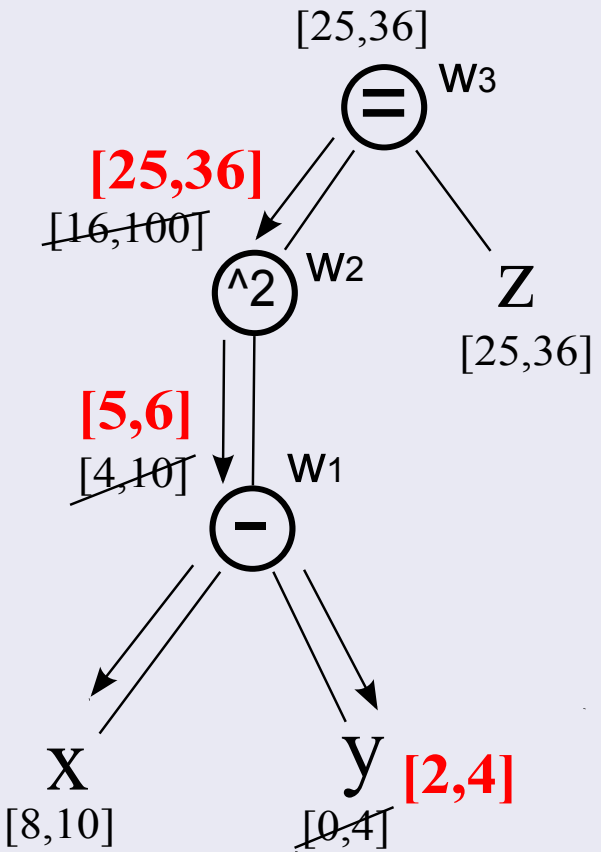
HC4-Revise

$(x - y)^2 = z$



HC4-Revise

$(x - y)^2 = z$



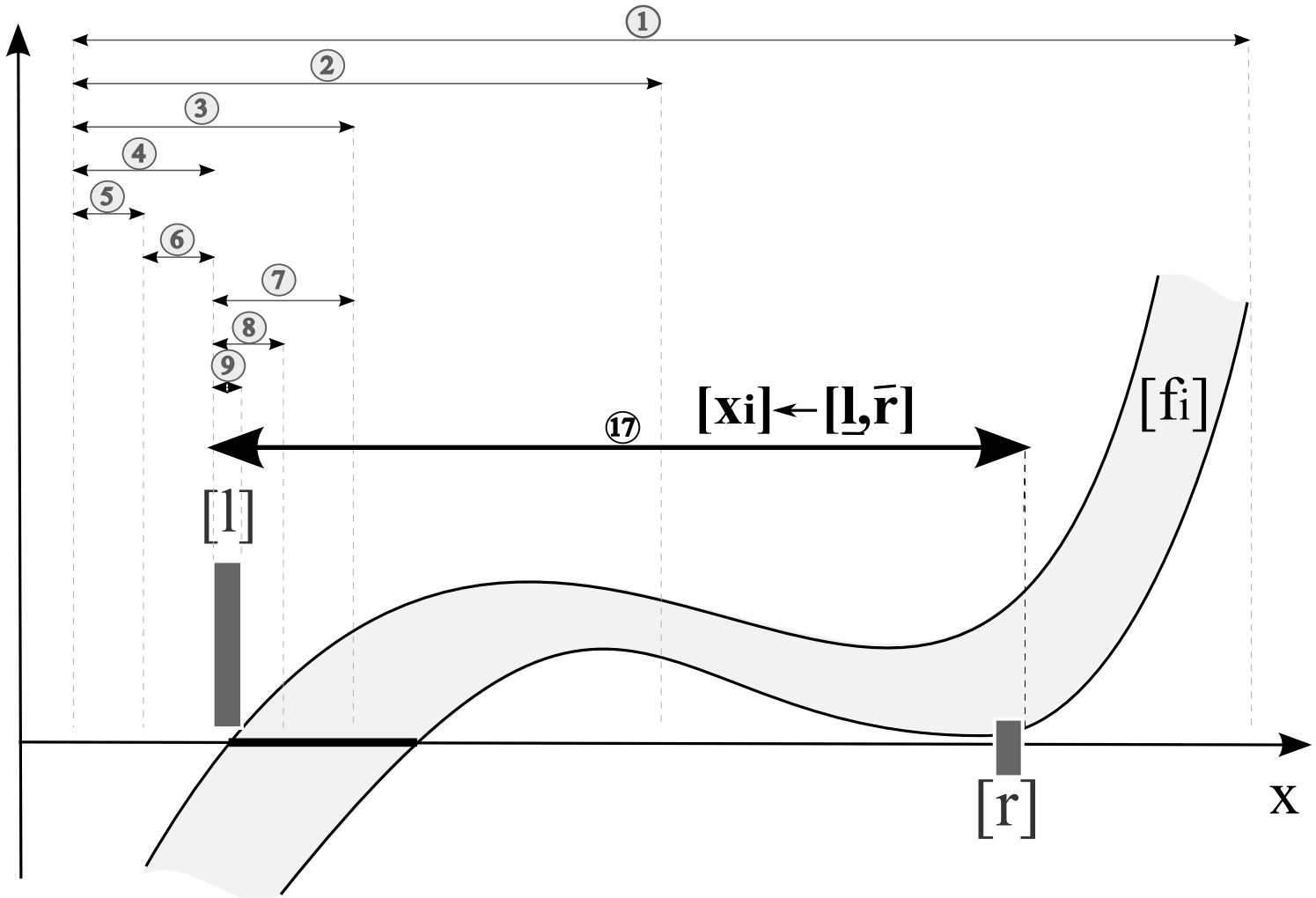
BoxNarrow

- The procedure narrows the domain of one variable (x_i) using a constraint $c : [f] = 0$.
- For performing the narrowing the algorithm works with the interval function:

$$[f_i](x) = [f]([x_1], ..., [x_{i-1}], x, [x_{i+1}], ..., [y_k])$$

- Two procedures (`LeftNarrow` and `RightNarrow`) are performed for obtaining the leftmost (l) and the rightmost (r) zeros of $[f_i]$.
- Finally: $[x_i] \leftarrow [l, r]$

BoxNarrow



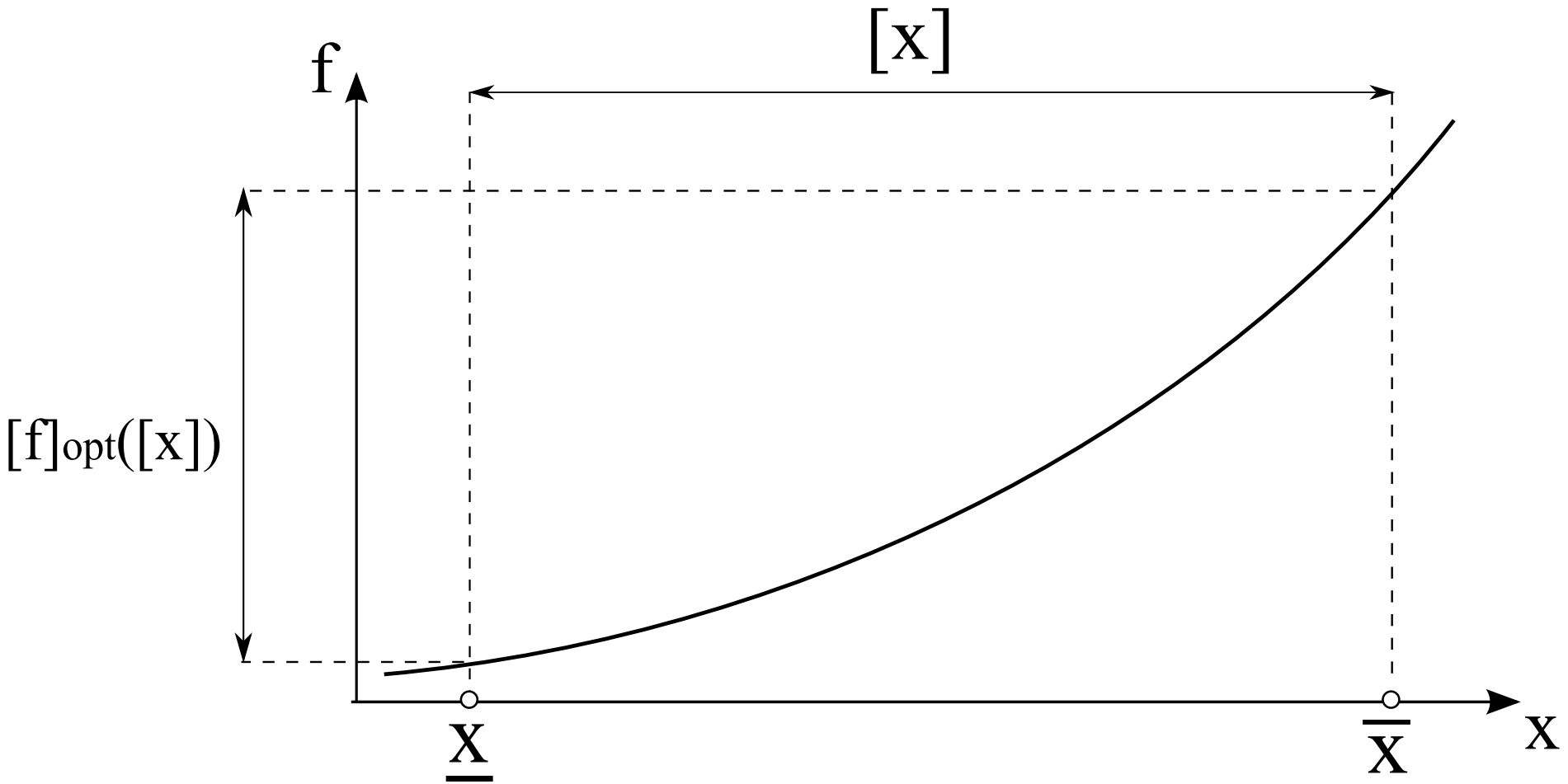
Outline

1. Introduction
   - Interval Arithmetics
   - Interval Methods for Solving NCSP
2. Exploiting monotonicity
   - Evaluation by monotonicity
   - Ocurrence Grouping
   - The Mohc algorithm
3. Exploting Common Subexpressions
   - Common Subexpression Elimination
   - Improving HC4 by CSE
   - The I-CSE Algorithm

Outline

1. Introduction
   - Interval Arithmetics
   - Interval Methods for Solving NCSP
2. Exploiting monotonicity
   - Evaluation by monotonicity
   - Ocurrence Grouping
   - The Mohc algorithm
3. Exploting Common Subexpressions
   - Common Subexpression Elimination
   - Improving HC4 by CSE
   - The I-CSE Algorithm

Evaluation by monotonicity



The evaluation by monotonicity

- The **evaluation by monotonicity** $[f]_m([B])$ of a function f eliminates the dependency problem in each **monotonic variable** x (i.e. $\frac{\partial f(X)}{\partial x} \geq 0$ or $\frac{\partial f(X)}{\partial x} \leq 0, \forall X \in [B]$).
- Consider the function $f(x_1, x_2, x_3)$. If f is increasing w.r.t. x_1 and decreasing w.r.t. x_2. The **evaluation by monotonicity** of f is given by:

$$[f]_m([B]) = [\underline{f_{min}}, \overline{f_{max}}]$$

  where $[f_{min}] = [f]_n(\underline{x_1}, \overline{x_2}, [x_3])$ and $[f_{max}] = [f]_n(\overline{x_1}, \underline{x_2}, [x_3])$
- $[f]_m([B]) \subseteq [f]_n([B])$

Outline

1. Introduction
   - Interval Arithmetics
   - Interval Methods for Solving NCSP
2. Exploiting monotonicity
   - Evaluation by monotonicity
   - Ocurrence Grouping
   - The Mohc algorithm
3. Exploting Common Subexpressions
   - Common Subexpression Elimination
   - Improving HC4 by CSE
   - The I-CSE Algorithm

Occurrence Grouping Method

- The method improves the evaluation by monotonicity when f is not monotonic w.r.t. a variable x.
- Consider the function $f(x) = -x^3 + 2x^2 + 6x$ with $[x] = [-1.2, 1]$. f **is not monotonic** w.r.t. x.
- $[f]([-1.2, 1]) = [f]_m([-1.2, 1]) = [-8.2, 10.61]$
- The method consists in grouping the occurrences of x in three sets (x_a, x_b, x_c) such that f^{og_1} is increasing w.r.t. x_a and decreasing w.r.t. x_b:

$$f^{og_1}(x_a, x_b) = -x_b^3 + 2x_a^2 + 6x_a$$

$$f^{og_2}(x_a, x_c) = -x_a^3 + 2x_c^2 + 6x_a$$

- $[f^{og_1}]_m([x]) = [-5.32, 9.73]$ and $[f^{og_2}]_m([x]) = [-5.47, 7.88]$

Finding a *good* Occurrence Grouping

- We proposed an algorithm that finds a good grouping in time O(k log k) (k is the number of occurrences).
- The algorithm allows each occurrence of x in f to be replaced by a convex combination of auxiliary variables, xa, xb and xc. For example consider $f_1 = -x^3 + 2x^2 + 6x$, then :

$$f_1^{og} = -x_a^3 + 2(0.35x_a + 0.65x_c)^2 + 6x_a$$

Results

- Experiments show gains in execution time between 1 to 30 using the Mohc algorithm.
- On these same experiments, OG takes between 0.04 and 1.6 times the time of a natural evaluation.

Outline

1. Introduction
   - Interval Arithmetics
   - Interval Methods for Solving NCSP
2. Exploiting monotonicity
   - Evaluation by monotonicity
   - Ocurrence Grouping
   - The Mohc algorithm
3. Exploting Common Subexpressions
   - Common Subexpression Elimination
   - Improving HC4 by CSE
   - The I-CSE Algorithm

What is Mohc?

- New constraint propagation algorithm (as HC4 and Box)
- Exploit **monotonicity** of functions to better filter/contract
- Better filter w.r.t. one constraint having several variables with **multiple occurrences**

Revise procedure of Mohc

Mohc-Revise (in-out $[B]$; in f, Y, W, ρ_{mohc}, τ_{mohc}, ϵ)

HC4-Revise$(f(Y,W)=0, Y, W, [B])$
if $W \neq \emptyset$ **and** $\rho_{mohc}[f] < \tau_{mohc}$ **then**
  $[G] \leftarrow$ GradientCalculation$(f, W, [B])$ /* G[i]$=\frac{\partial f}{\partial x_i}([B])$ */
  $(f^{og}, W) \leftarrow$ OccurrenceGrouping$(f, W, [B], [G])$
  $(f_{max}, f_{min}, X, W) \leftarrow$ ExtractMonotonicVars$(f^{og}, W, [B], [G])$
  MinMaxRevise$([B], f_{max}, f_{min}, Y, W)$
  MonotonicBoxNarrow$([B], f_{max}, f_{min}, X, [G], \epsilon)$
end if

- X: monotonic variables; Y: variables with single occurrence in f; W: vars with multiple occurrences, but not detected monotonic.
- MinMaxRevise narrows the intervals of variables in Y and W.
- MonotonicBoxNarrow narrows the intervals of variables in X.

The MinMaxRevise procedure

$$[f]_M([B]) = \left[\underline{[f_{min}]([Y],[W])}, \overline{[f_{max}]([Y],[W])}\right]$$

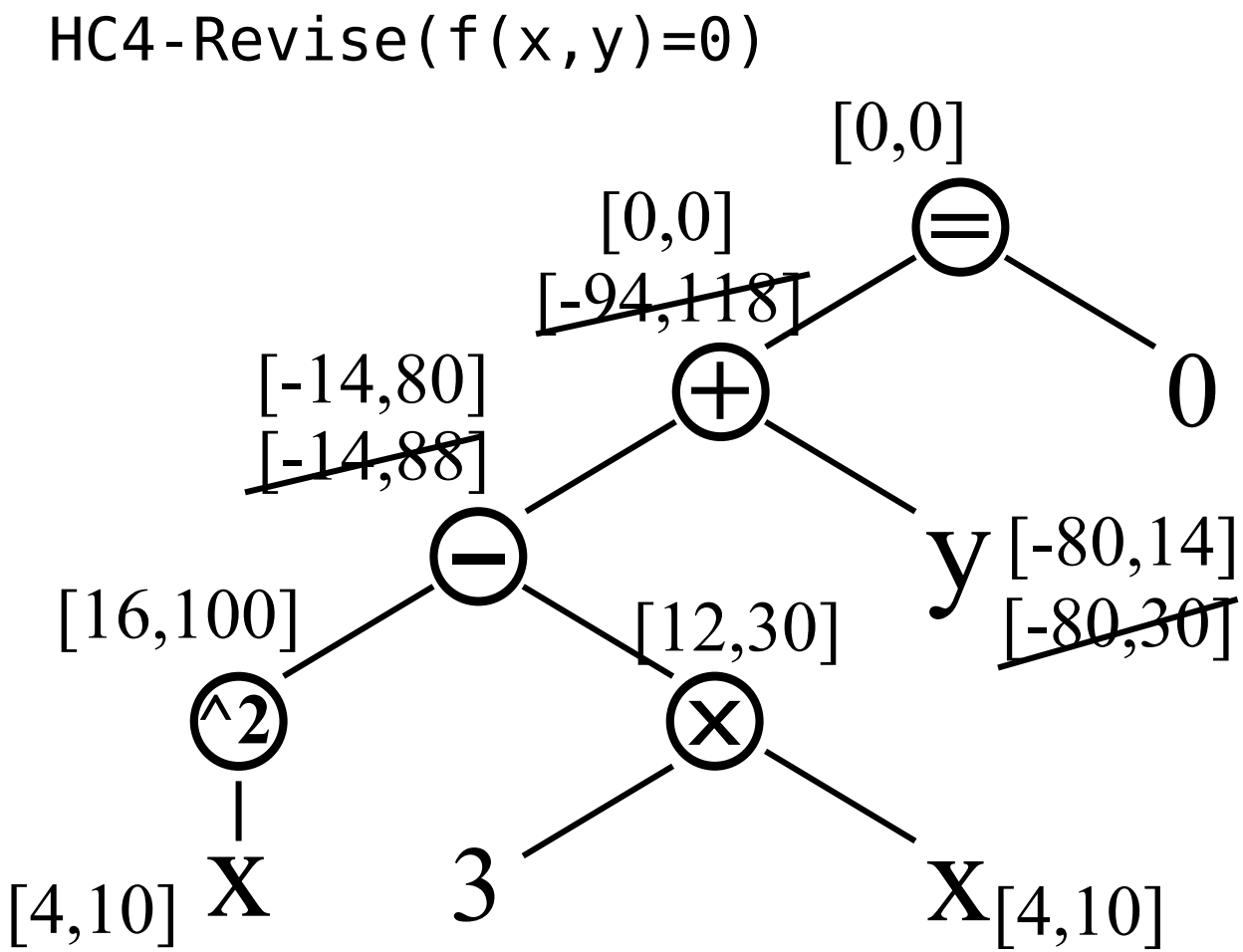
$$0 \in [f]_M([B]) \quad \equiv \quad [f_{min}]([Y],[W]) \leq [0,0] \leq [f_{max}]([Y],[W])$$

MinMaxRevise (in-out $[B]$; in f_{max}, f_{min}, Y, W)

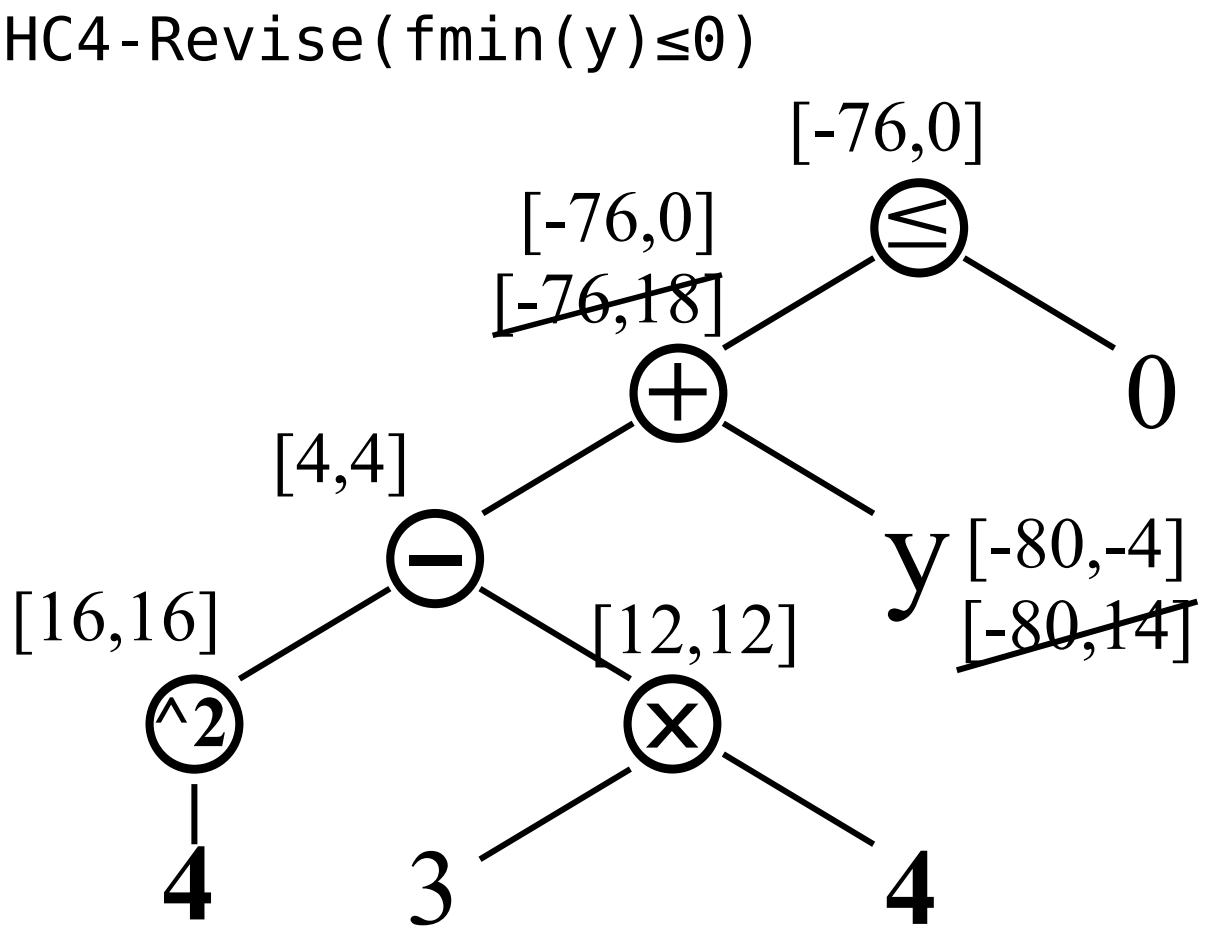
HC4-Revise($f_{min}(Y,W) \leq 0,\ Y, W, [B]$) /* MinRevise */

HC4-Revise($f_{max}(Y,W) \geq 0,\ Y, W, [B]$) /* MaxRevise */

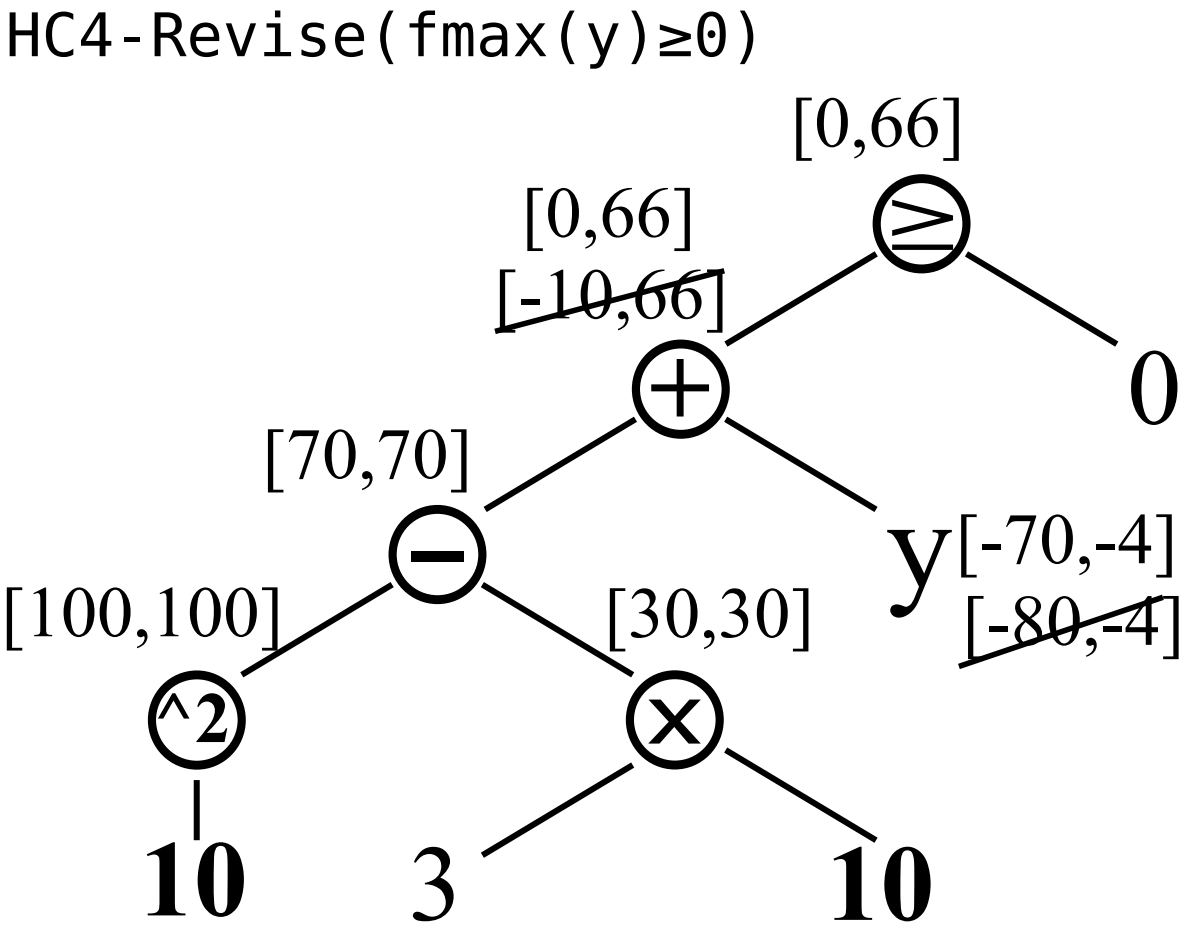
Example: HC4-Revise on $x^2 - 3 \times x + y = 0$



Example: MinRevise on $x^2 - 3 \times x + y = 0$



Example: MaxRevise on $x^2 - 3 \times x + y = 0$

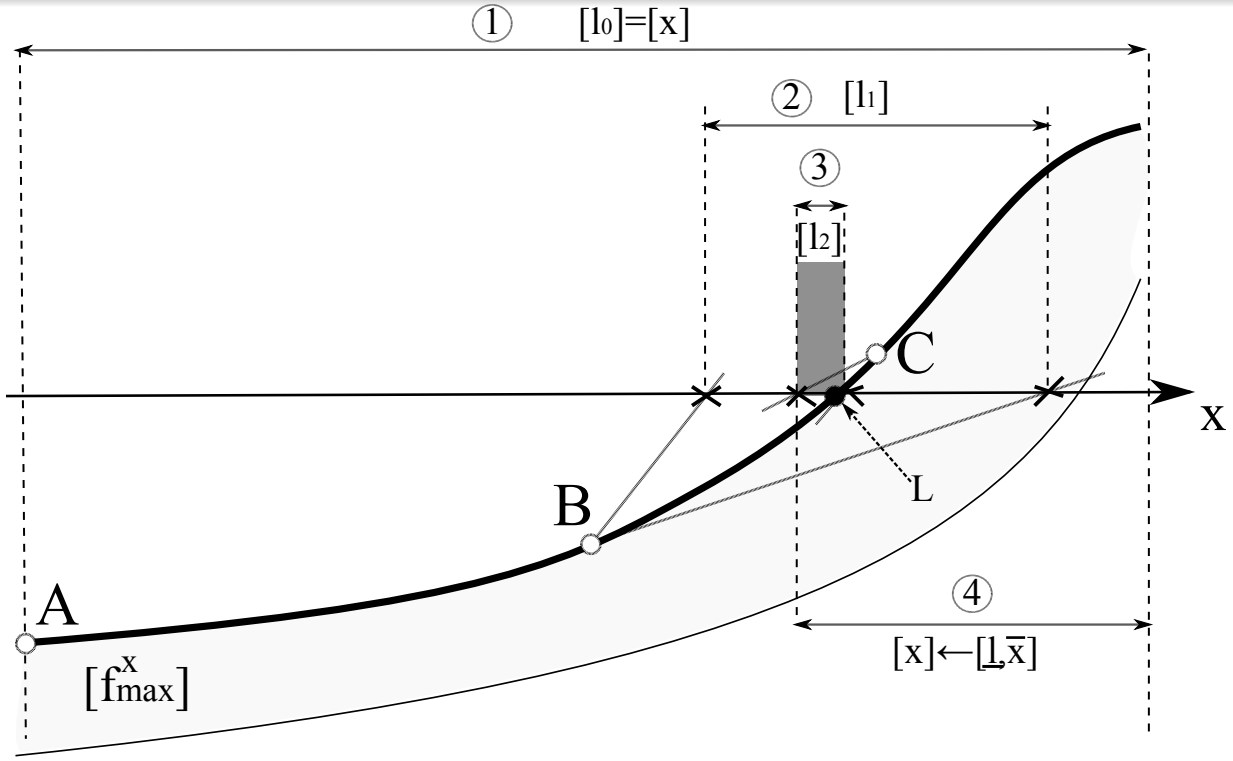


The MonotonicBoxNarrow procedure

For every variable $x \in X$ s.t. f is monotonic (e.g., increasing) w.r.t. x, MonotonicBoxNarrow:

- uses a univariate interval function f^x_{max} (like standard BoxNarrow) which is monotonic w.r.t. x,
- calls a procedure LeftNarrowFmax to contract $[x]$ by the **left** side,
- calls a procedure RightNarrowFmin to contract $[x]$ by the **right** side.

Illustration of LeftNarrowFmax



Principle

- An existence test checks that $\overline{[f^x_{max}](\underline{x})} < 0$, i.e., the point A is below zero (otherwise: no leftside contraction).
- A dichotomic process is then run to sharply enclose L ($[l] \leftarrow [x]$).

Properties that helps to improve Mohc

- If MaxRevise contracts the box (with f_{max}), then MonotonicBoxNarrow (using f_{min}) is useless.
- After applying once MaxRevise and MonotonicBoxNarrow, the Mohc-Revise procedure reaches the fixed point.
- If one bound of $[x_i]$ is improved (using MonotonicBoxNarrow), then only one bound of any $[x_j]$ ($x_j \neq x_i$) can be improved. (Inspired from Octum.)

Hull consistency

If:

- W is empty, i.e., all variables are monotonic (X) or appears once (Y)

Then:

- `Mohc-Revise` computes an **optimal box** (with a precision ratio ϵ).

User-defined parameters: ϵ and τ_{mohc}

Mohc-Revise (in-out $[B]$; in f, Y, W, ρ_{mohc}, τ_{mohc}, ϵ)

HC4-Revise$(f(Y,W)=0, Y, W, [B])$
if $W \neq \emptyset$ **and** $\rho_{mohc}[f] < \tau_{mohc}$ **then**
  $[G] \leftarrow$ GradientCalculation$(f, W, [B])$ /* G[i]$=\frac{\partial f}{\partial x_i}([B])$ */
  $(f^{og}, W) \leftarrow$ OccurrenceGrouping$(f, W, [B], [G])$
  $(f_{max}, f_{min}, X, W) \leftarrow$ ExtractMonotonicVars$(f^{og}, W, [B], [G])$
  MinMaxRevise$([B], f_{max}, f_{min}, Y, W)$
  MonotonicBoxNarrow$([B], f_{max}, f_{min}, X, [G], \epsilon)$
end if

User-defined parameters: ϵ and τ_{mohc}

- Experiments show that finely tuning ϵ is useless $\Rightarrow \epsilon = 10\%$ (see paper).
- $\tau_{mohc} \in [0, 1]$ allows the monotonicity-based procedures to be called more or less often: Condition: $\rho_{mohc}[f] < \tau_{mohc}$
- For every constraint f: $\rho_{mohc}[f] = \frac{Diam([f]_M([B]))}{Diam([f]([B]))}$
- $\rho_{mohc}[f]$ is computed only once after every bisection.
- Experiments show that τ_{mohc} should be tuned between 60% and 99% (see paper).

Experiments

- 17 instances from the COPRIN benchmarks.
- Implementation of Mohc in the Ibex C++ library (by Chabert).
- Strategy: Between two choice points:
  1. Monotonicity test,
  2. 3BCID(Mohc),
  3. Interval Newton.
- Comparison with 3BCID(HC4).

Good results

NCSP	HC4	Box	Mohc(0.7)	Mohc(0.99)	Gain
Butcher	282528	25867	5026	**1842**	153
8 3	1.8e+8	1.7e+6	2.2e+6	**324669**	554
Direct kin.	17515	>28800	458	**363**	48
11 2	1.4e+6		9541	**5609**	250
Fourbar	13121	11011	366	**353**	37
4 3	8.5e+6	732429	58571	**45695**	186
Virasoro	7158	>28800	1241	**902**	7.9
8 224	2.6e+6		79211	**38739**	67
Geneig	598	>7200	116	**87.6**	6.8
6 10	205859		15341	**6975**	30
Yamam.1	11.8	15.3	**2.02**	2.69	5.8
8 7	3017	**183**	303	297	10
Pramanik	95.9	278	19.6	**19.6**	4.9
3 2	124661	23017	12691	**8435**	15
Hayes	41.7	282	17.3	**13.9**	3.0
8 1	17763	7247	4437	**1717**	10
Trigo1	150.7	773	**55.8**	71.9	2.7
10 9	2565	1005	461	455	5.6

Not so bad results

NCSP	HC4	Box	Mohc(0.7)	Mohc(0.99)	Gain
Caprasse	2.77	32.2	2.74	**2.34**	1.18
4 18	1309	719	903	**391**	3.3
Kin1	**1.95**	68.7	1.97	3.36	0.99
6 16	87	**65**	87	81	1.1
Trigexp2	**90.1**	>3600	91.4	169	0.99
11 0	15187		15099	7717	2.0
I5	**55.7**	>3600	58.5	82.9	0.95
10 30	10621		9811	**8715**	1.2
Eco9	**13.9**	102.0	14.6	26.0	0.95
9 16	6193	4991	6037	**4343**	1.4
Brent	**19.0**	311.0	20.2	41.4	0.94
10 1008	3923	**2137**	3815	3189	1.2
Redeco8	**6.23**	69.8	6.82	10.88	0.91
8 8	2441	1913	2347	**1537**	1.6
Katsura	**77.4**	2265	103	245	0.75
12 7	4251	3557	3573	**3151**	1.3

Perspectives

- Mohc can be seen as an generalization of the classic propagation algortihms (HC4 and Box) using monotonicity properties of functions.
- Implement a more efficient and effective version of the algorithm MinMaxrevise.
- Render Mohc adpatative.

Outline

1. Introduction
   - Interval Arithmetics
   - Interval Methods for Solving NCSP
2. Exploiting monotonicity
   - Evaluation by monotonicity
   - Ocurrence Grouping
   - The Mohc algorithm
3. Exploting Common Subexpressions
   - Common Subexpression Elimination
   - Improving HC4 by CSE
   - The I-CSE Algorithm

Outline

1. Introduction
   - Interval Arithmetics
   - Interval Methods for Solving NCSP
2. Exploiting monotonicity
   - Evaluation by monotonicity
   - Ocurrence Grouping
   - The Mohc algorithm
3. Exploting Common Subexpressions
   - Common Subexpression Elimination
   - Improving HC4 by CSE
   - The I-CSE Algorithm

Common Subexpressions Elimination (CSE)

- **CSE** is an important feature in optimization of code.
- CSE consists in replacing common subexpressions (CS) by auxiliary variables.
  For example:

```
a = b * c + g
d = b * c * d
```

  It may be worth (in performance) transforming the code to:

```
tmp = b * c
a = tmp + g
d = tmp * d
```

Common Subexpressions Elimination (CSE)

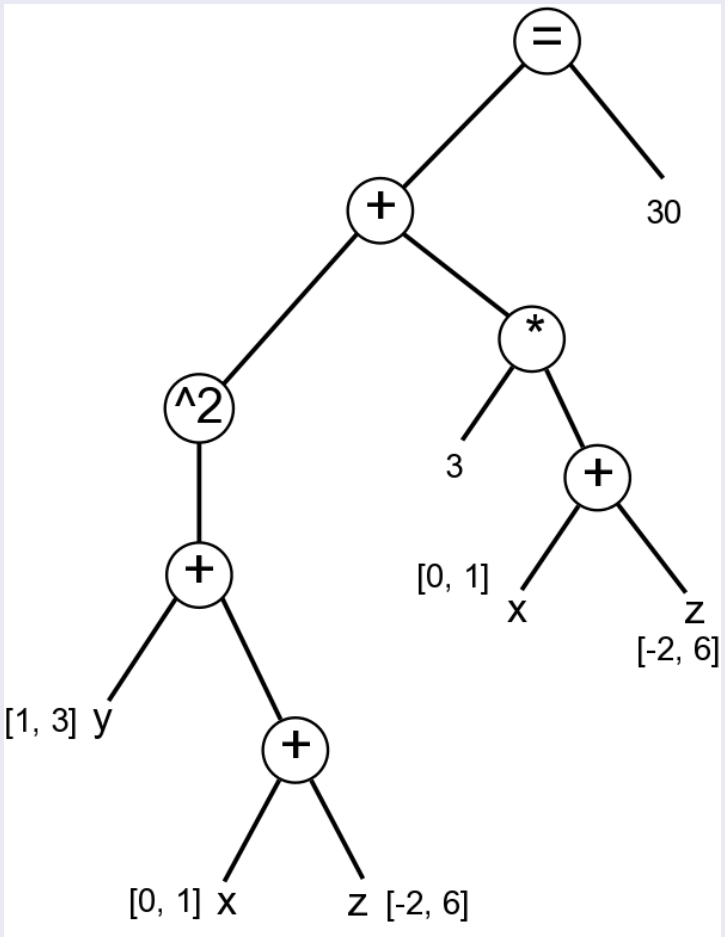
- In interval analysis, several experts have been interested in eliminating the common subexpressions of system of constraints.
- Schichl and Neumaier proposed a unique DAG to represent a system of equations. Common subexpressions (CS) are represented by nodes with several parents.
- The community of interval analysis thought that the obtained gains were **due to a reduction of the number of operations**.
- However, CSE also may be useful to improve the performance of interval solvers (bringing *a better contraction/filtering*).

Outline

1. Introduction
   - Interval Arithmetics
   - Interval Methods for Solving NCSP
2. Exploiting monotonicity
   - Evaluation by monotonicity
   - Ocurrence Grouping
   - The Mohc algorithm
3. Exploting Common Subexpressions
   - Common Subexpression Elimination
   - Improving HC4 by CSE
   - The I-CSE Algorithm

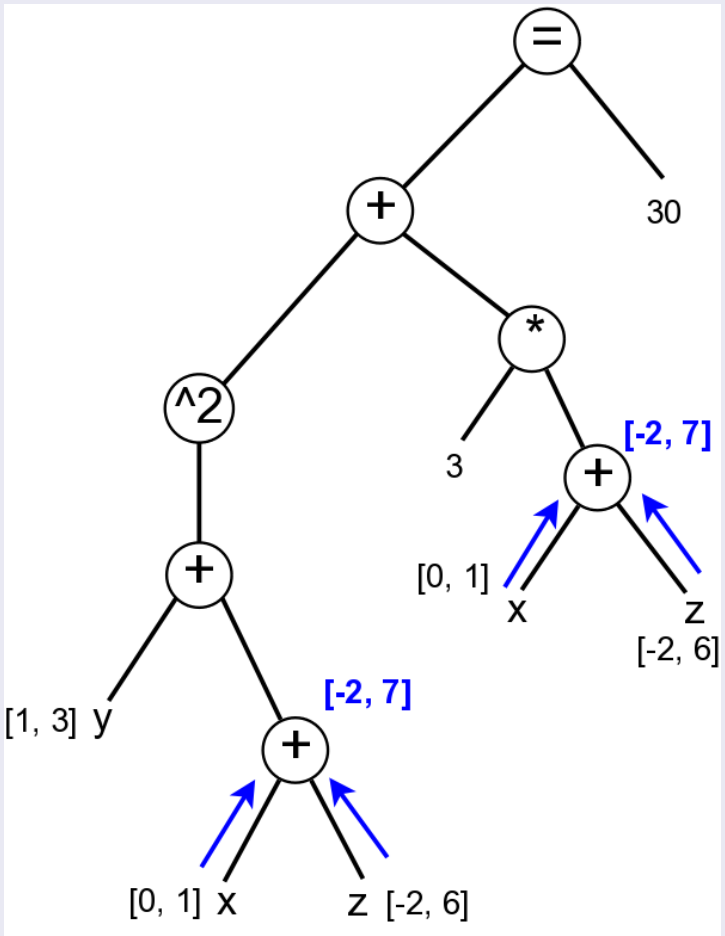
HC4-Revise Algorithm

Example of HC4-revise: $(x + y + z)^2 + 3(x + z) = 30$



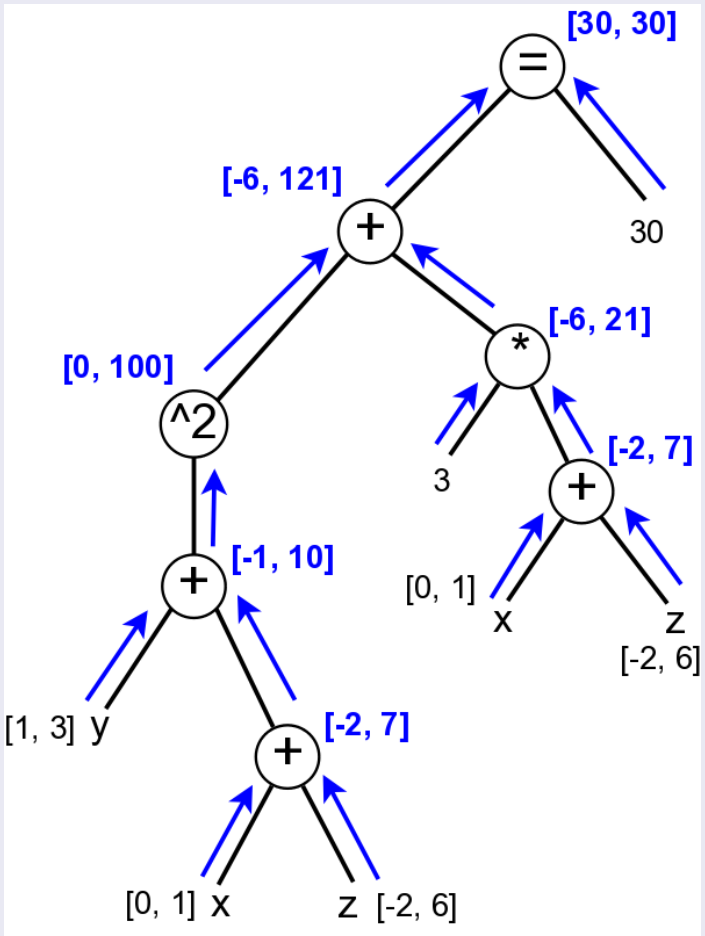
HC4-Revise Algorithm

Example of HC4-revise: $(x + y + z)^2 + 3(x + z) = 30$



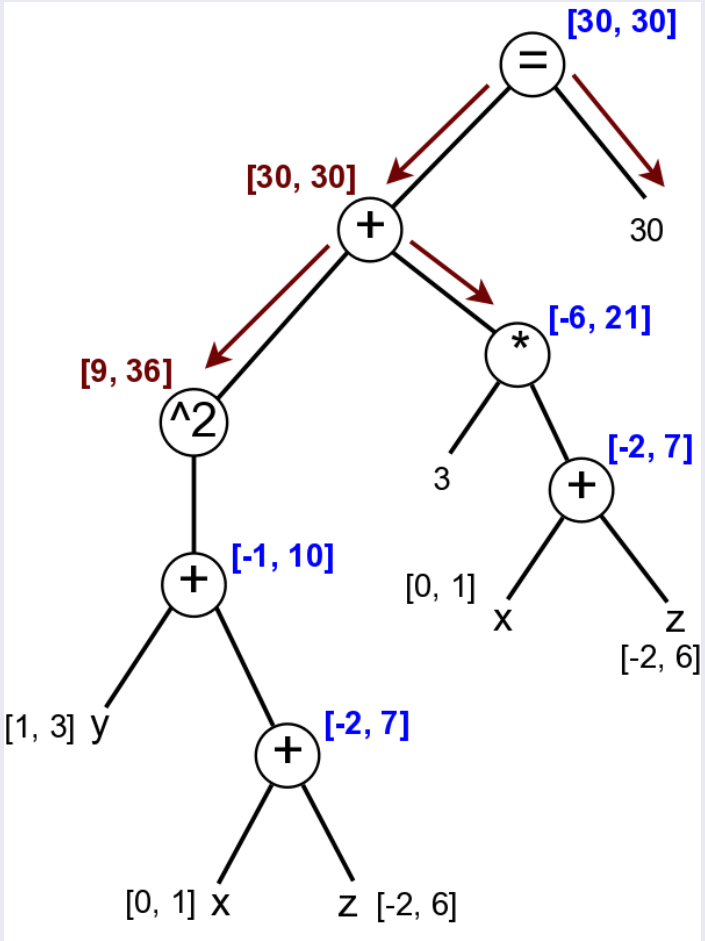
HC4-Revise Algorithm

Example of HC4-revise: $(x + y + z)^2 + 3(x + z) = 30$



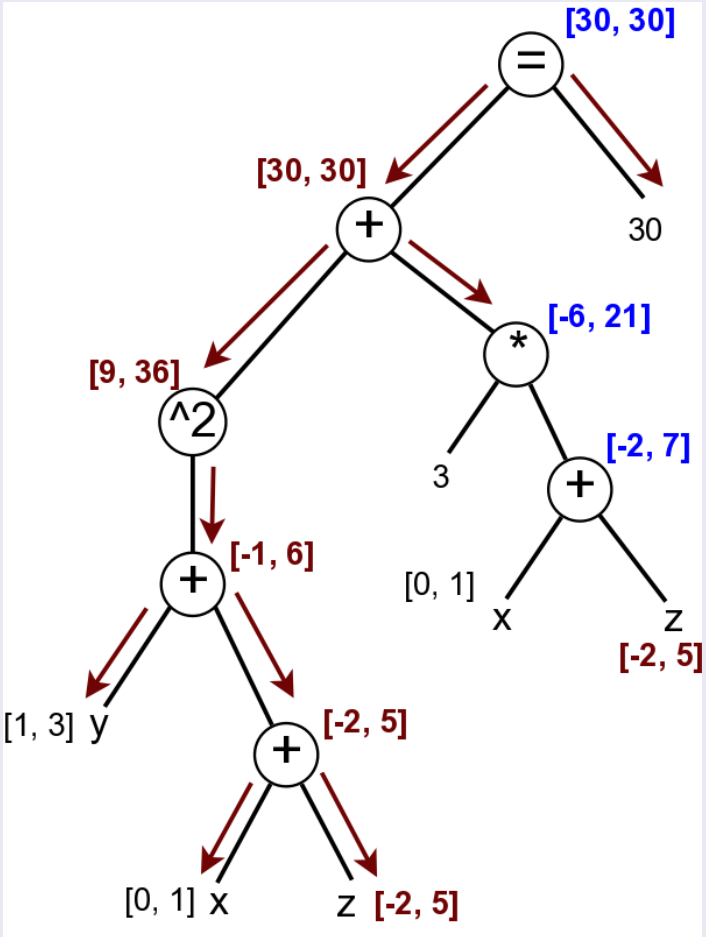
HC4-Revise Algorithm

Example of HC4-revise: $(x + y + z)^2 + 3(x + z) = 30$



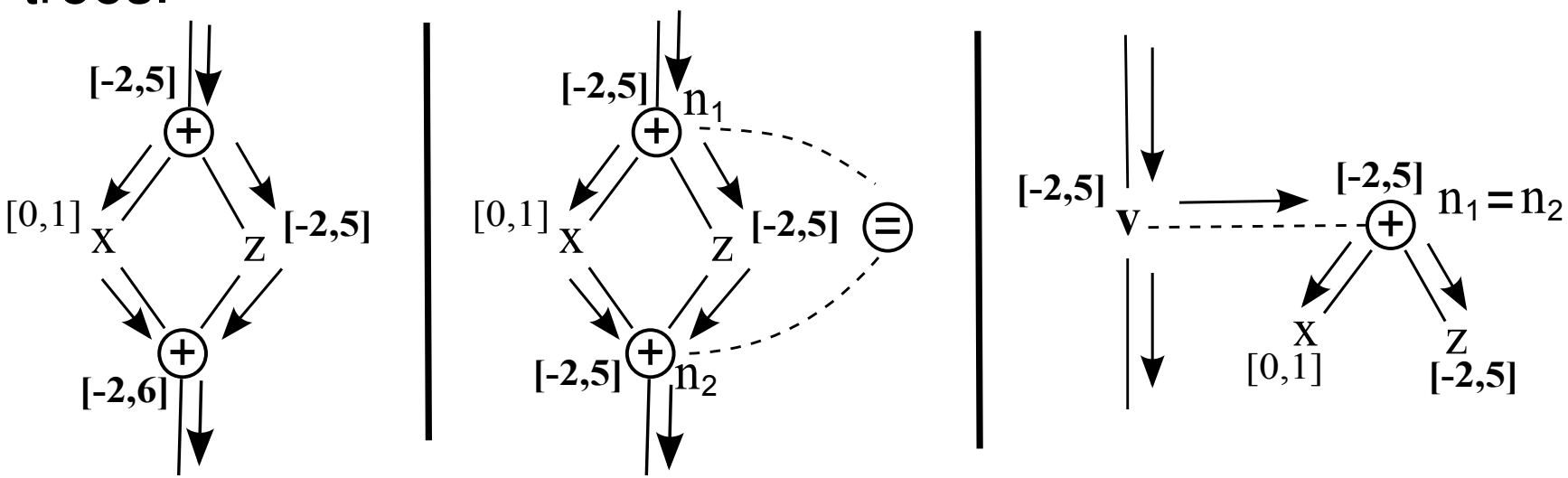
HC4-Revise Algorithm

Example of HC4-revise: $(x + y + z)^2 + 3(x + z) = 30$



CSE may improve the filtering

Example: The sum $x + z$ is common to two expression trees.



Replace n1 and n2 by a common variable v, and add a new constraint v = x + z.

Outline

1. Introduction
   - Interval Arithmetics
   - Interval Methods for Solving NCSP
2. Exploiting monotonicity
   - Evaluation by monotonicity
   - Ocurrence Grouping
   - The Mohc algorithm
3. Exploting Common Subexpressions
   - Common Subexpression Elimination
   - Improving HC4 by CSE
   - The I-CSE Algorithm

The I-CSE Algorithm

- The novelty of I-CSE lies in the way **additive and multiplicative** CSs are taken into account.
- I-CSE manages **conflictive subexpressions**. Example: $x_1 + x_2 + x_3$, $x_1 + x_2$ and $x_2 + x_3$
- Algorithm divided into 4 steps.

The I-CSE Algorithm

- Every equation in the system is represented by an n-ary tree. The algorithm is divided in 4 steps:
  1. The trees are compacted into a DAG, merging together equivalent subtrees.
  2. N-ary sums and products expressions are pairwise intersected obtaining all the maximal subexpressions shared by two expressions.
  3. The CSs obtained in step 2 are integrated into the DAG. Conflictive CSs generate redundant equations.
  4. The new system is generated.

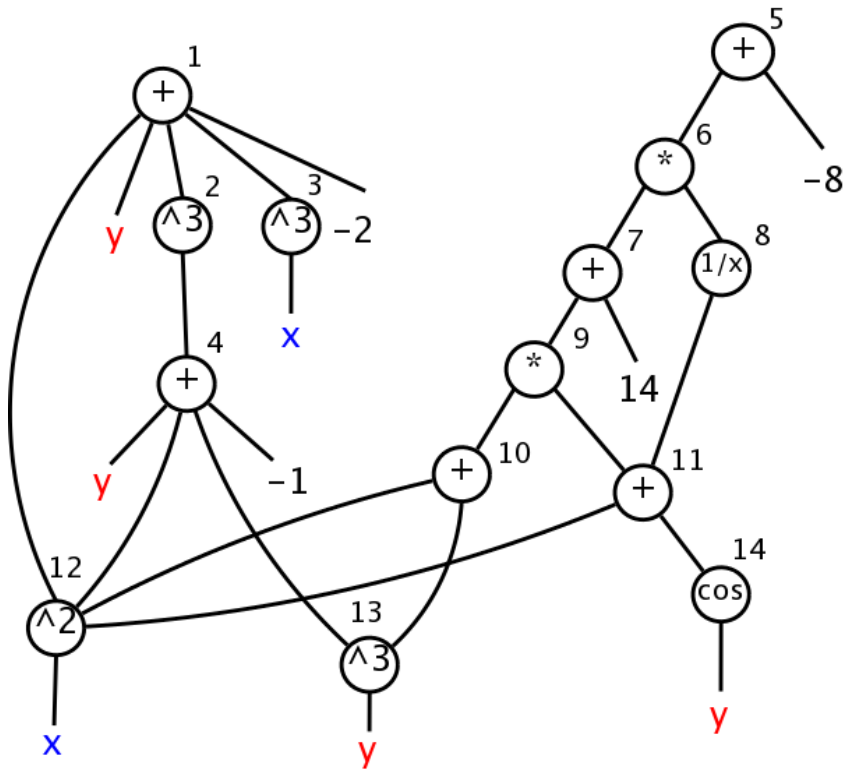
Step 1: DAG Generation

Initial system:

$$\begin{aligned} x^2 + y + (y + x^2 + y^3 - 1)^3 + x^3 &= 2 \\ \frac{(y^3 + x^2) \times (x^2 + cos(y)) + 14}{x^2 + cos(y)} &= 8 \end{aligned}$$

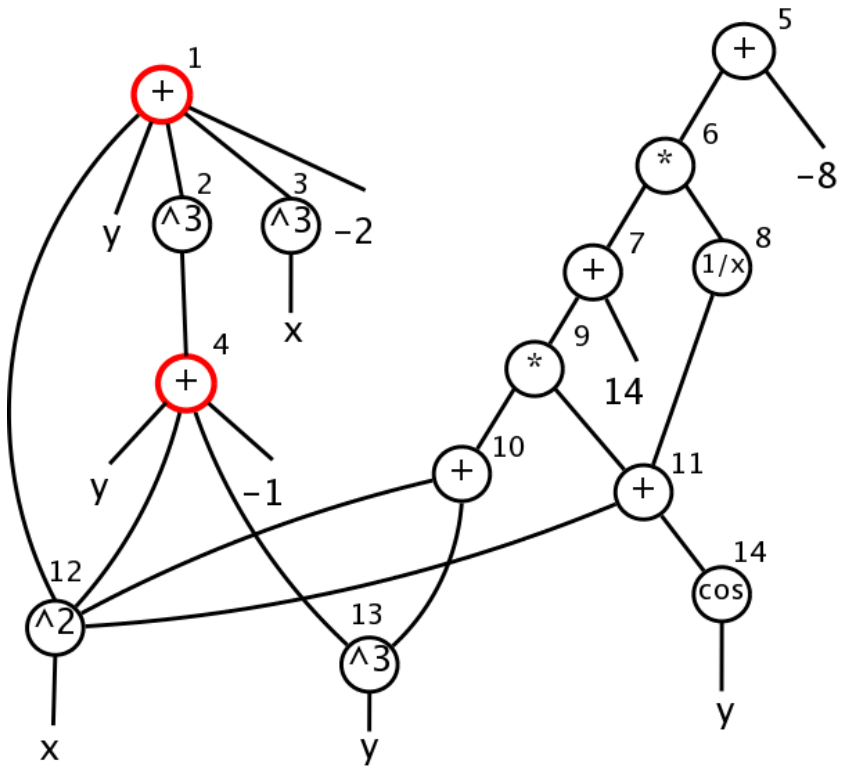
Step 1: DAG Generation

$$\begin{aligned} \mathbf{x^2} + y + (y + \mathbf{x^2} + y^3 - 1)^3 + x^3 &= 2 \\ \frac{(y^3 + \mathbf{x^2}) \times (\mathbf{x^2} + \mathbf{cos(y)}) + \mathbf{14}}{\mathbf{x^2} + \mathbf{cos(y)}} &= 8 \end{aligned}$$



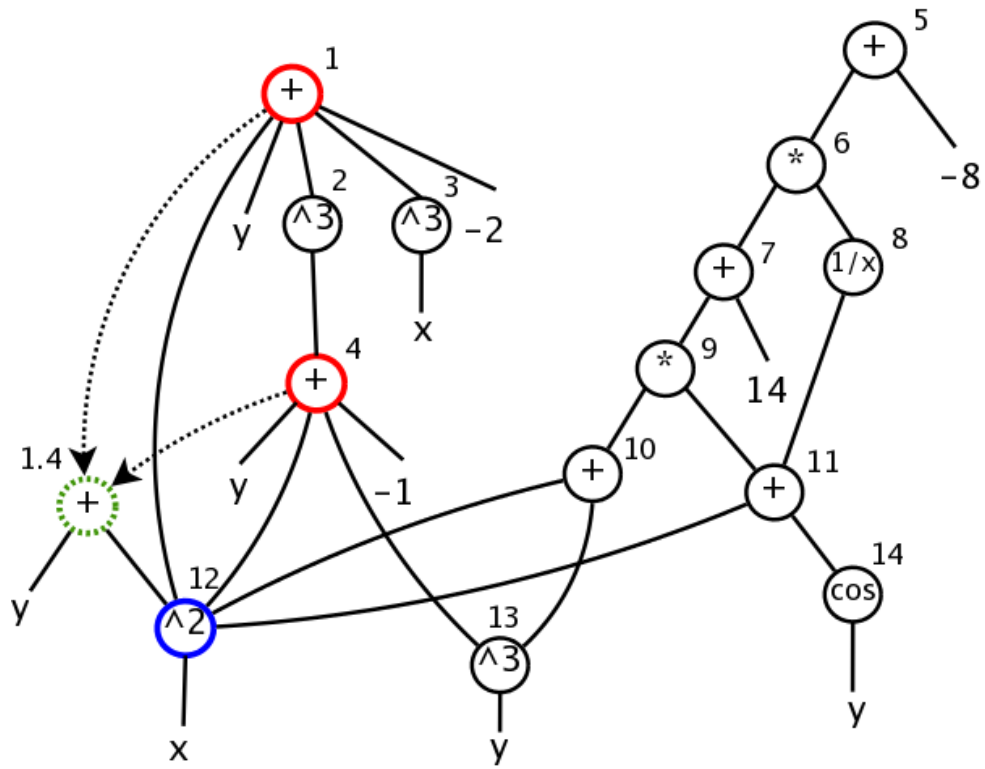
Step 2: Pairwise Intersection

$$+(\mathbf{x^2}, \mathbf{y}, node_2, x^3, -2)_1 \cap +(\mathbf{y}, \mathbf{x^2}, y^3, -1)_4 = +(\mathbf{y}, \mathbf{x^2})$$



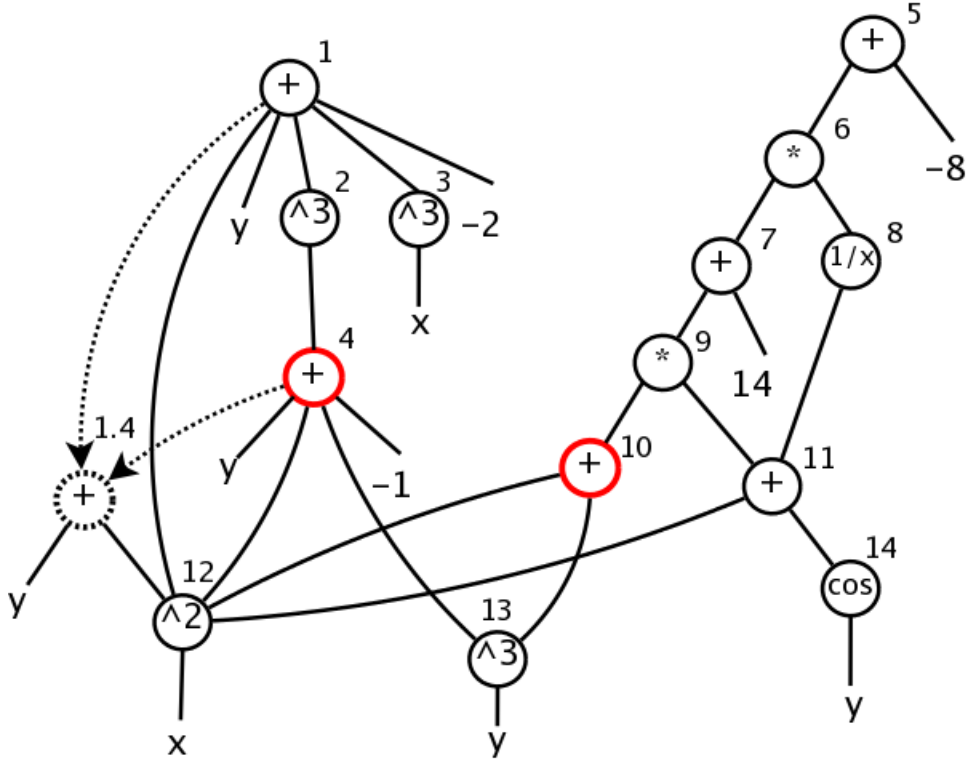
Step 2: Pairwise Intersection

$$+(\mathbf{x^2}, \mathbf{y}, node_2, x^3, -2)_1 \cap +(\mathbf{y}, \mathbf{x^2}, y^3, -1)_4 = +(\mathbf{y}, \mathbf{x^2})_{1.4}$$



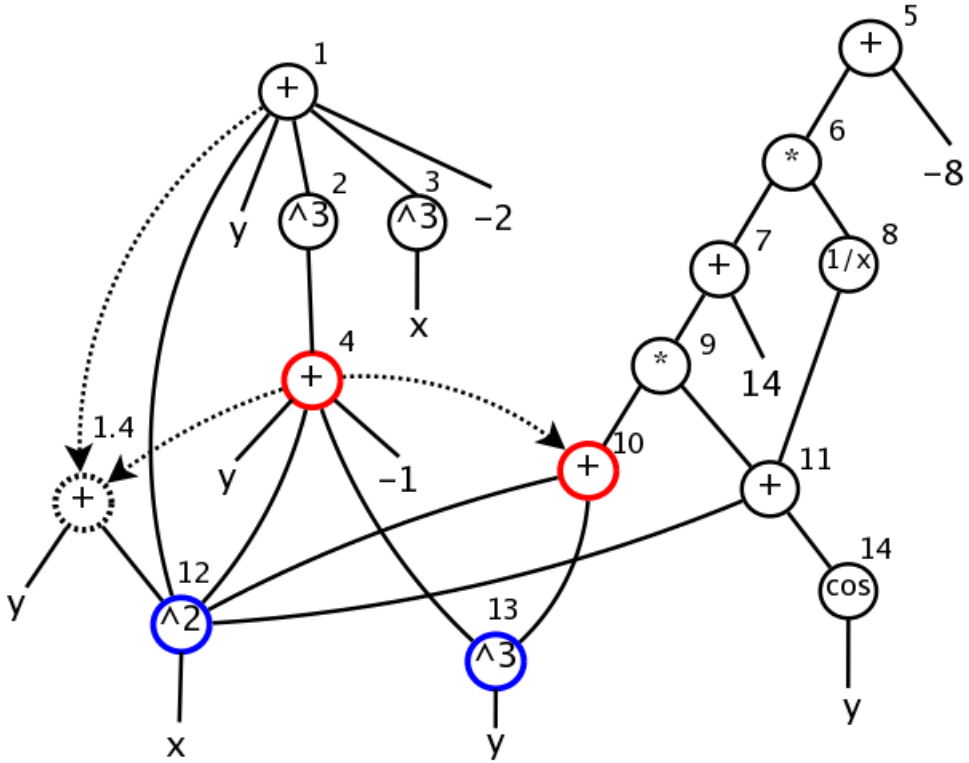
Step 2: Pairwise Intersection

$$+(y, \mathbf{x^2}, \mathbf{y^3}, -1)_4 \cap +(\mathbf{x^2}, \mathbf{y^3})_{10} = +(\mathbf{x^2}, \mathbf{y^3})$$

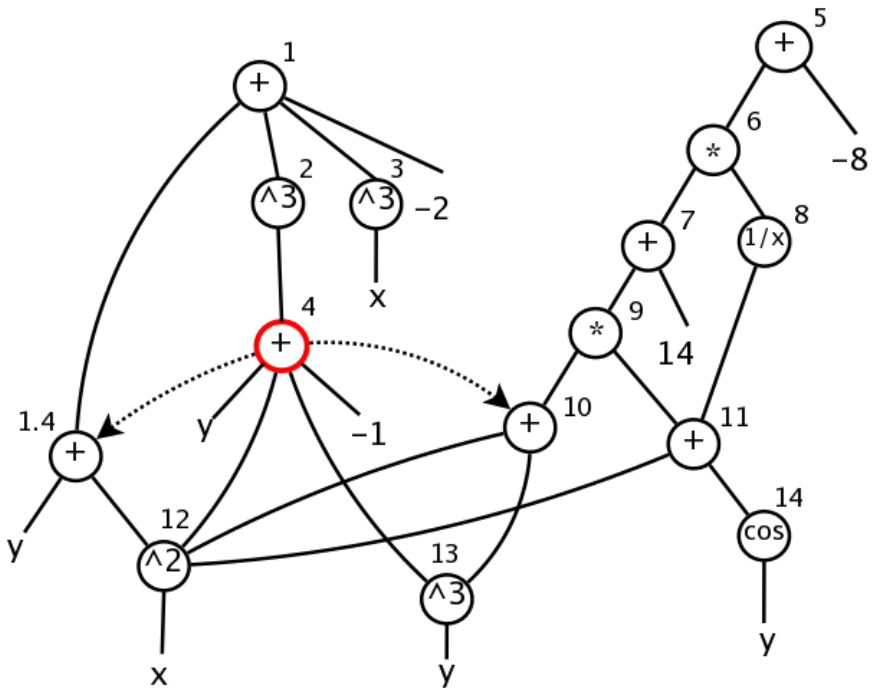


Step 2: Pairwise Intersection

$$+(y, \mathbf{x^2}, \mathbf{y^3}, -1)_4 \cap +(\mathbf{x^2}, \mathbf{y^3})_{10} = +(\mathbf{x^2}, \mathbf{y^3})_{10}$$



Step 3: Integrating CSs into the DAG



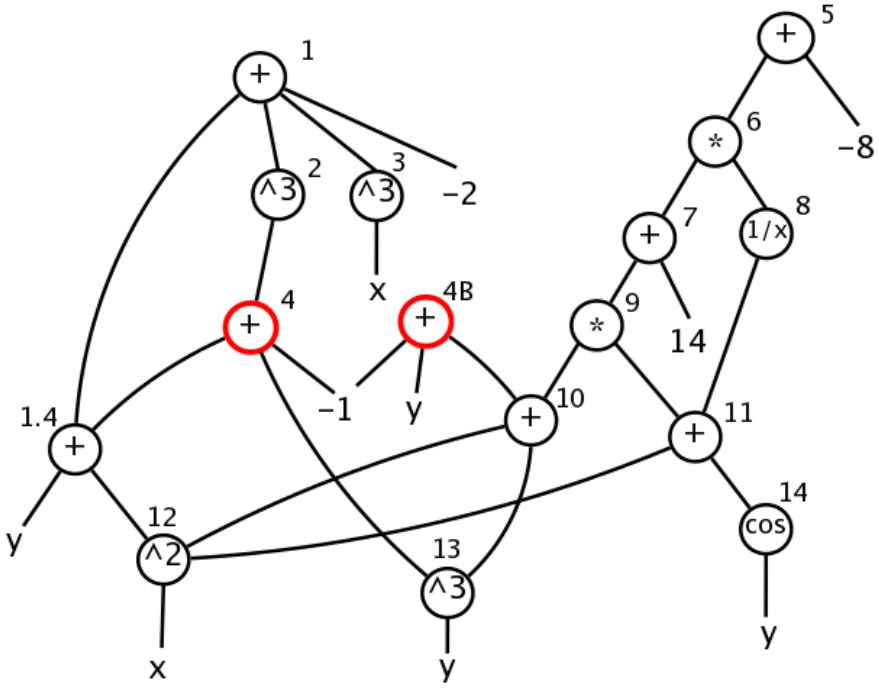
Observation

CSs related to node 4 are in conflict ($n_4 = y + x^2 + y^3$):

$$n_{1.4} := y + \mathbf{x^2}$$

$$n_{10} := \mathbf{x^2} + y^3$$

Step 3: Integrating CSs into the DAG



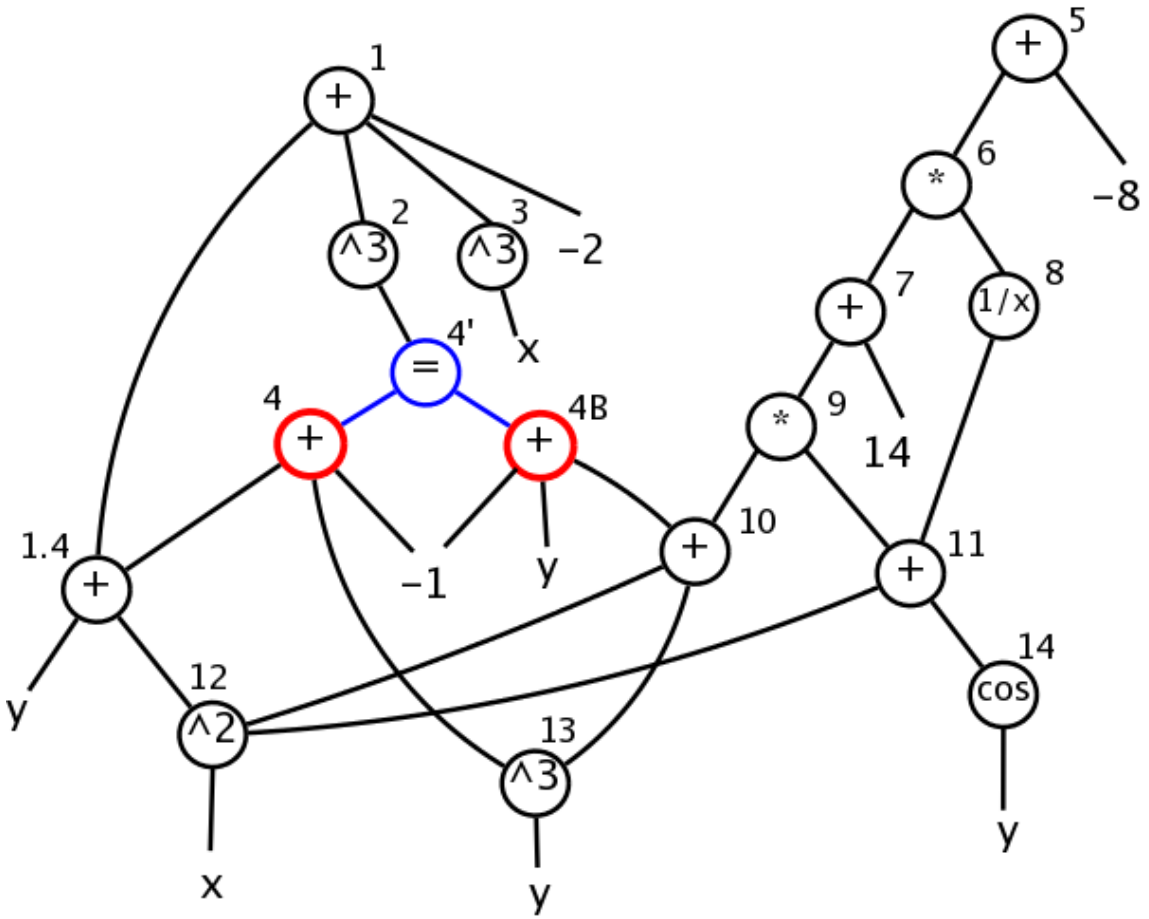
Observation

A redundant node is created ($n_4 = y + x^2 + y^3$):

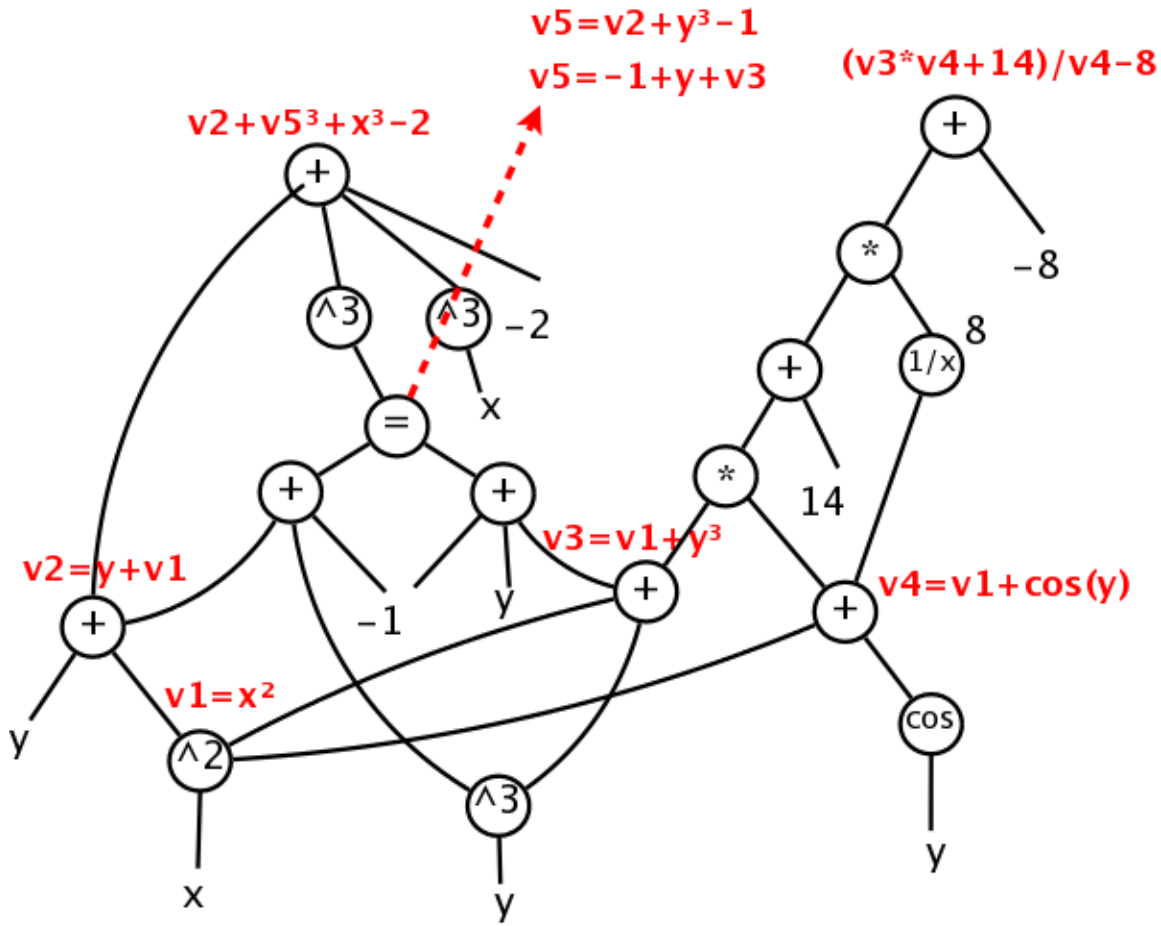
$$n'_4 := n_{1.4} + y^3 - 1$$

$$n_{4B} := y + n_{10} - 1$$

Step 3: Integrating CSs into the DAG



Step 4: Generation of the new system



Step 4: Generation of the new system

The new generated system is:

$$\begin{aligned} v_2 + v_5^3 + x^3 - 2 &= 0 \\ \frac{v_3 \times v_4 + 14}{v_4} - 8 &= 0 \end{aligned}$$

$$\begin{aligned} v_1 &= x^2 \\ v_2 &= y + v_1 \\ v_3 &= v_1 + y^3 \end{aligned} \qquad \begin{aligned} v_4 &= v_1 + cos(y) \\ v_5 &= v_2 + y^3 - 1 \\ v_5 &= -1 + y + v_3 \end{aligned}$$

Experiments

Benchmark			I-CSE-B	ICSE-NC	I-CSE	
	#s	N	#cs	#cs	#cs	#rc
6body	5	6	2	3	3	0
Bellido	8	9	0	1	1	0
Brown-7	3	7	3	7	21	24
Brown-7*	3	7	3	1	1	0
Brown-30	**2**	**30**	**26**	**53**	**435**	**783**
BroyBand-20	1	20	22	37	97	73
BroyBand-100	1	100	102	119	479	473
Caprasse	18	4	6	7	11	2
Design	1	9	3	3	3	0
Dis-Integral-6	1	6	4	6	18	9
Dis-Integral-20	3	20	18	34	207	171
Eco9	16	8	0	3	7	1
EqCombustion	4	5	7	8	11	1
ExtendWood-4	3	4	2	2	2	0
Geneig	10	6	11	14	14	0
Hayes	1	8	9	8	8	0
I5	30	10	3	4	10	5
Katsura-19	5	20	81	81	81	0
Katsura-20	7	21	90	90	90	0

Benchmark			I-CSE-B	ICSE-NC	I-CSE	
	#s	N	#cs	#cs	#cs	#rc
Kin1	16	6	13	13	19	3
Pramanik	2	8	0	15	15	0
Prolog	0	21	0	7	7	0
Rose	16	3	5	5	5	0
Trigexp1-30	1	30	29	29	29	0
Trigexp1-50	1	50	49	49	49	0
Trigexp2-11	0	11	15	15	15	0
Trigexp2-19	0	19	27	27	27	0
Trigonom-5	2	5	7	9	20	14
Trigonom-5*	2	5	7	6	6	0
Trigonom-10	24	10	15	15	26	15
Trigonom-10*	24	10	15	12	12	0
Yamamura-8	7	8	5	10	36	48
Yamamura-8*	7	8	5	1	1	0
Yamamura-10	9	12	7	14	55	79
Yamamura-10*	9	12	7	1	1	0
Yamamura-12	9	12	9	18	78	119
Yamamura-12*	9	12	9	1	1	0
Yamamura-16	9	16	13	26	136	224

Results with HC4 and Interval Newton

Benchmark	TIME in second				TIME(Osys) / TIME			#Boxes		
	Osys	ICSE-B	ICSE-NC	I-CSE	ICSE-B	ICSE-NC	I-CSE	Osys	ICSE-NC	I-CSE
EqCombustion	>3600	26.1	0.35	**0.14**	>137	>10000	**>25000**	>1e+08	3967	**1095**
Rose	>3600	500	**101**	**101**	>7.2	**>35**	**>35**	>3e+07	**865099**	**865099**
Hayes	141	51.9	**15.7**	**15.7**	2.7	**9**	**9**	550489	**44563**	**44563**
6-body	0.22	**0.07**	**0.07**	**0.07**	**3.1**	**3.1**	**3.1**	4985	**495**	**495**
Design	176	65.2	**63.2**	**63.2**	2.7	**2.8**	**2.8**	425153	**122851**	**122851**
I5	>3600	>3600	**1534**	1565	?	**>2.3**	>2.3	>3e+07	**7e+06**	7e+06
Geneig	3323	2910	2722	2722	1.14	**1.22**	**1.22**	7e+08	4e+08	4e+08
Kin1	8.52	8.32	8.32	**8.01**	1.02	1.02	**1.06**	**905**	909	**905**
Pramanik	89.3	92.1	**84.9**	**84.9**	0.97	**1.05**	**1.05**	487255	**378879**	**378879**
Bellido	15.7	15.9	**15.6**	**15.6**	0.99	**1.01**	**1.01**	29759	**29319**	**29319**
Eco9	**23.9**	**23.9**	24	24.1	**1.00**	**1.00**	0.99	126047	117075	**110885**
Caprasse	**1.56**	1.81	1.68	2.16	0.86	0.93	0.72	8521	7793	**7491**
Brown-7*	500	350	**0.01**	**0.01**	1.42	**49500**	**49500**	6e+06	**95**	**95**
Dis-Integral-6	201	0.46	1.3	**0.03**	437	155	**6700**	653035	4157	**47**
ExtendWood-4	29.9	**0.03**	**0.03**	**0.03**	**997**	**997**	**997**	422705	**353**	**353**
Brown-7	500	350	30.7	**1.49**	1.42	16.1	**332**	6e+06	258601	**3681**
Trigexp2-11	1118	208	**56.2**	**56.2**	5.38	**19.9**	**19.9**	1e+06	**316049**	**316049**
Yamamura-8*	13	13.3	**0.75**	**0.75**	0.98	**17.3**	**17.3**	29615	**2161**	**2161**
Broy-Banded-20	778	759	261	**58.1**	1.03	2.98	**13.4**	172959	46761	**12623**
Trigonometric-5*	15.8	12.3	**1.49**	**1.49**	1.28	**10.6**	**10.6**	10531	**1503**	**1503**
Trigonometric-5	15.8	12.3	8.94	**6.97**	1.28	1.77	**2.27**	10531	7369	**5307**
Yamamura-8	13	13.3	44.6	**10.8**	0.98	0.3	**1.20**	29615	115211	**13211**
Katsura-19	**1430**	1583	1583	1583	0.90	0.90	0.90	**145839**	153193	153193
Trigexp1-30	**2465**	3244	3244	3244	0.76	0.76	0.76	1e+07	1e+07	1e+07

Conclusion

- CSs can bring significant gains in filtering and not only a decrease in the number of operations.
- Gains of several orders of magnitude.

Exploiting Monotonicity and Common Subexpressions for improving interval constraint propagation algorithms

Ignacio Araya

COPRIN, INRIA, Université de Nice-Sophia, France

MEA, December 2009, Paris, France