Constraint Programming

Lecture 2. Local consistency.

Ruslan Sadykov

INRIA-Bordeaux

13 January 2022

Lignes directrices

Definitions

- A constraint is binary if it contains at most two variables.
- A CSP is binary if all its constraints are binary.

Network of binary constraints

A constraint network is a set of constraints on variables with discrete and finite domain.

A binary constraint network can be represented by a special graph :

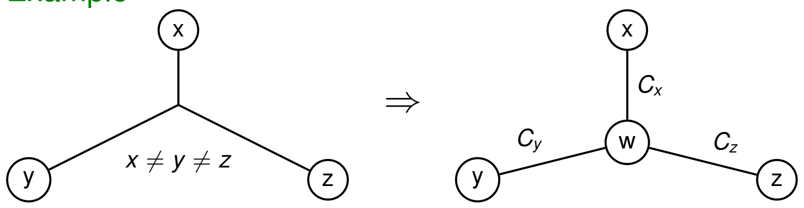
- Vertices represent variables.
- Edges represent constraints.

If two vertices are adjacent, there are constraints containing the corresponding variables.

Making any CSP binary

For each non-binary CSP, there exists an equivalent binary CSP.

Example

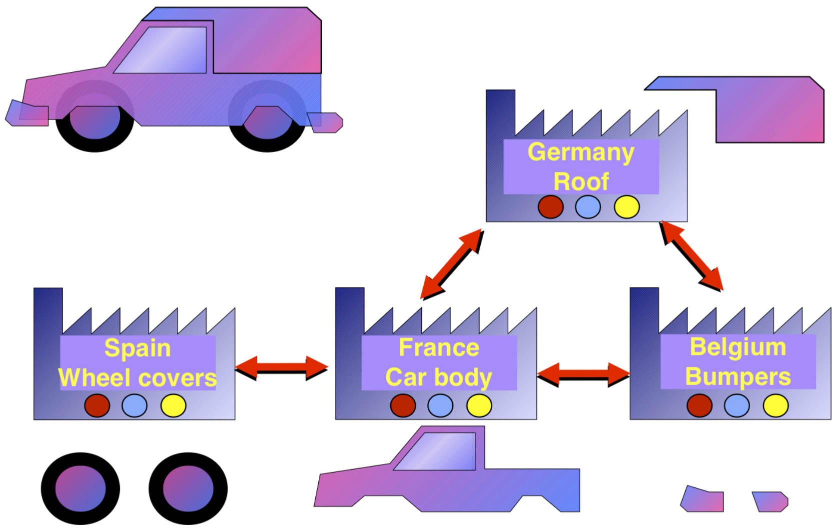


$D_x = D_y = D_z = \{1, 2, 3\}$

$D_w = \{abc\}_{a \neq b \neq c}$
$C_x = \{(a, abc)\}_{a \in D_x, abc \in D_w}$
$C_y = \{(b, abc)\}_{b \in D_y, abc \in D_w}$
$C_z = \{(c, abc)\}_{c \in D_z, abc \in D_w}$

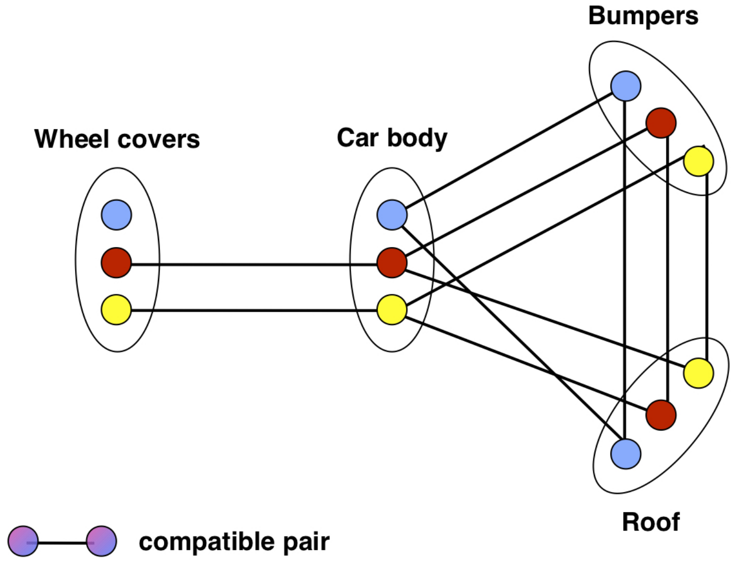
Lignes directrices

A trivial example

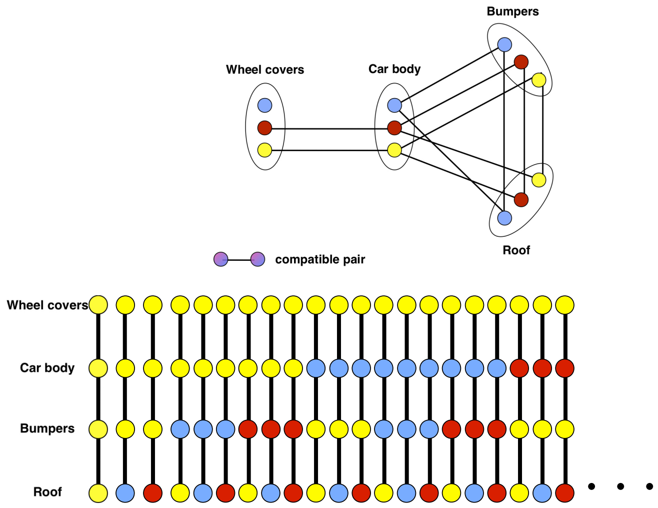


Source : Philippe Baptiste

A trivial example : the extended constraint network



Trivial solution algorithm for our example



Trivial solution of discrete CSPs

- Suppose that all domains are finite
- Then the number $|A|$ of different assignments of values in variable domains is finite too :

$$|A| = |D_{x_1}| \times \cdots \times |D_{x_n}|.$$

- Then, we can consider these assignments one by one and verify whether at least one of them satisfies the constraints.
- Computational complexity (number of operations to do) is at least $|A| \times |C|$.
- It is too large !
  - 8 queens : $8^8 \times 96 \approx 10^{10}$
  - Sudoku : $81^9 \times 27 \approx 10^{17}$

Inconsistency

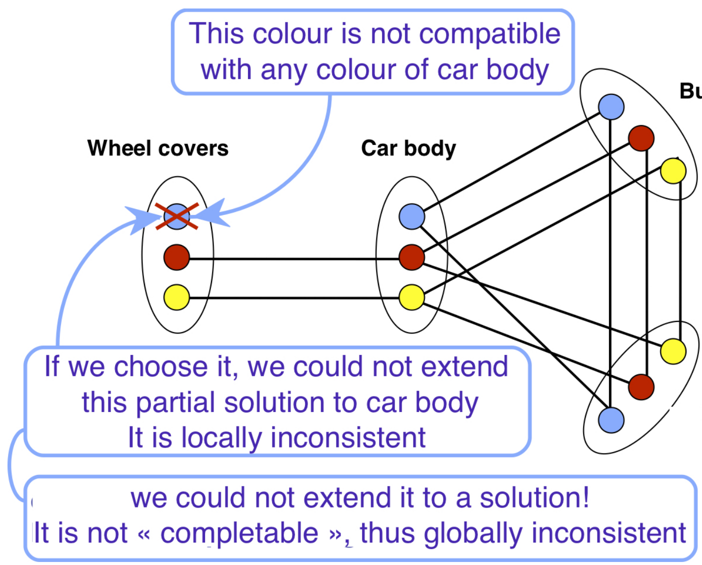
To solve faster, we try to remove values (from domains of variables) which do not lead to any solution.

Example

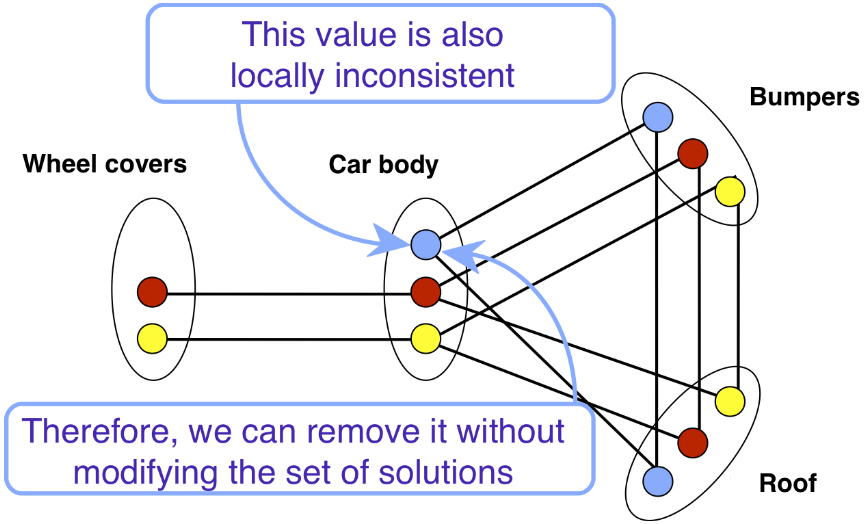
- $D_x = \{1, 2, 3\}$, $D_y = \{2, 3, 4\}$, $x \geq y$.
- If $x = 1$, there are no values in D_y which satisfy the constraint.
- Therefore, we can delete 1 from D_x.

We say that these values are « inconsistent » with one or several constraints.

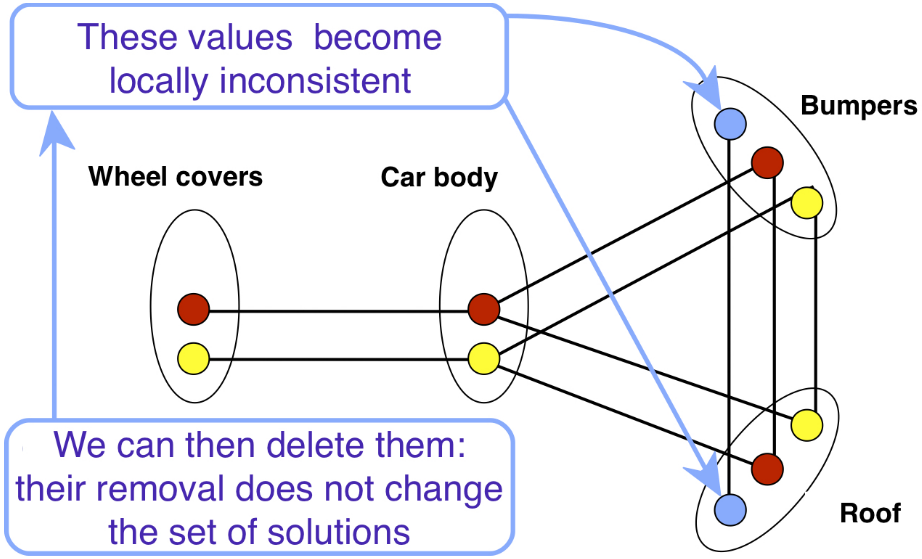
Inconsistency : illustration I



Inconsistency : illustration II



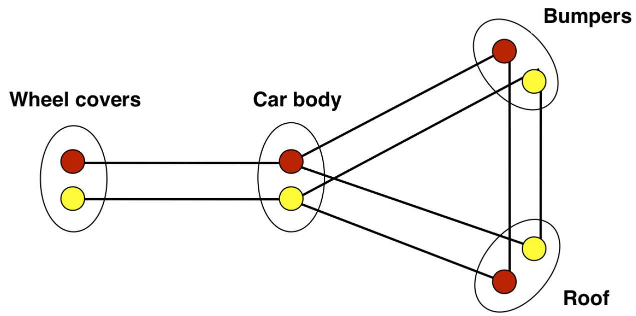
Inconsistency : illustration III



Local consistency

- We do not check consistency of values with all constraints at the same time (this would mean solving the whole CSP)
- We check consistency with a subset of constraints (usually, one), therefore the consistance is local.
- A CSP is locally consistent if all values in all variable domains are locally consistent
- When a CSP achieves local consistency, the set of solutions does not change. Therefore, the CSP remains equivalent, but simpler (or smaller).

Local consistency : illustration



- We did not change the set of solutions : the constraint network is equivalent.
- We reduced the search space !

Lignes directrices

Définitions I

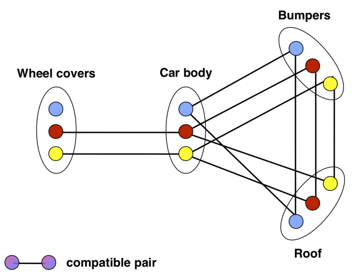
Here we concentrate on the binary case.

- There exist different levels of local consistency
- If we check consistency of a value with only binary constraint at a time, it is the arc-consistency.
- Value a of variable x est arc-consistent if and only if it has at least one compatible value (support) in each neighbour domain.
- **Formally :**

$$\langle x, a\rangle \text{ is arc-consistent } \Leftrightarrow \forall C(x,y)\ \exists b \in D_y :\ C(a,b).$$

$$\langle x, a\rangle \text{ is not arc-consistent } \Leftrightarrow \exists C(x,y) :\ \forall b \in D_y\ \neg C(a,b).$$

Arc-consistency : illustration



(Wheel covers, ●) is arc-consistent as there exists a compatible value for Car body.
(Car body, ●) is arc-consistent as

- there exists a compatible value (●) for wheel covers,
- there exists a compatible value (●) for bumpers,
- there exists a compatible value (●) for the roof

(Car body, ●) is not arc-consistent as it does not have a compatible value for wheel covers.
Removal of (Car body, ●) makes (Roof, ●) and (Bumpers, ●) arc-inconsistent

Definitions II

- A contraint is arc-consistent if and only if all values in domains of its variables are arc-consistent.
- A CSP is arc-consistent if and only if all its constraints are arc-consistent.

Why arc-consistency ?

There exist polynomial (and efficient !) algorithms to achieve the arc-consistency for a binary CSP.

Algorithm AC-1

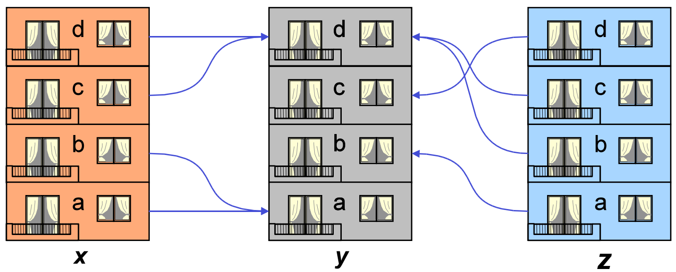
repeat
  finished $\leftarrow$ TRUE;
  foreach *contraint* $C(x, y)$ **do**
    if *there exist values in* D_x *which do not have a support in* D_y **then**
      remove them;
      finished $\leftarrow$ FALSE;
until *finished* $=$ *TRUE*;

Computational complexity in the worst case :

$$O(nd \times ed^2) = O(ned^3),$$

where n — number of variables, e — number of constraints, d — size of the largest domain.

Friends analogy



Town↔CSP
Building↔Variable
Neighbour buildings↔Variables « connected » by a constraint
Resident↔Value
Friends↔Pair of values satisfying the constraint

A resident is arc-consistent while it has at least on friend in each neighbour building.

Friends analogy — AC-1

A resident is arc-consistent while it has at least on friend in each neighbour building.

AC-1 :

1. Each arc-inconsistent resident leaves the town and sends a letter « I leave the town - Anonymous » to every resident of the town.
2. When a resident receives a letter, he verifies whether he is still arc-consistent.

Algorithm AC-3

toTest $\leftarrow \{C(x, y)\}_{C(x,y) \in C}$;
foreach $C(x, y) \in$ *toTest* **do**
  toTest $\leftarrow$ *toTest* $\setminus \{C(x, y)\}$;
  remove all values from D_x which do not have a support in D_y;
  if *at least on value has been removed* **then**
    toTest $\leftarrow$ *toTest* $\cup \{C(z, x) : \exists C(z, x) \in C, z \neq x\}$;

Computational complexity in the worst case :

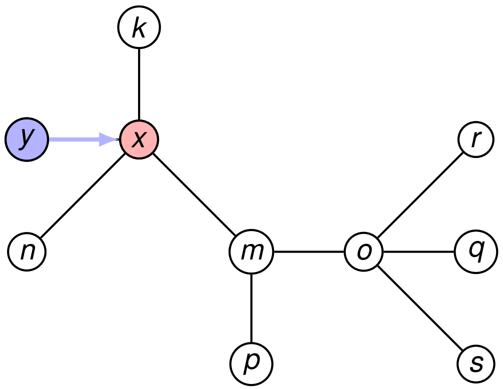
$$O(ed \times d^2) = O(ed^3).$$

Spacial complexity in the worst case :

$$O(dn + e).$$

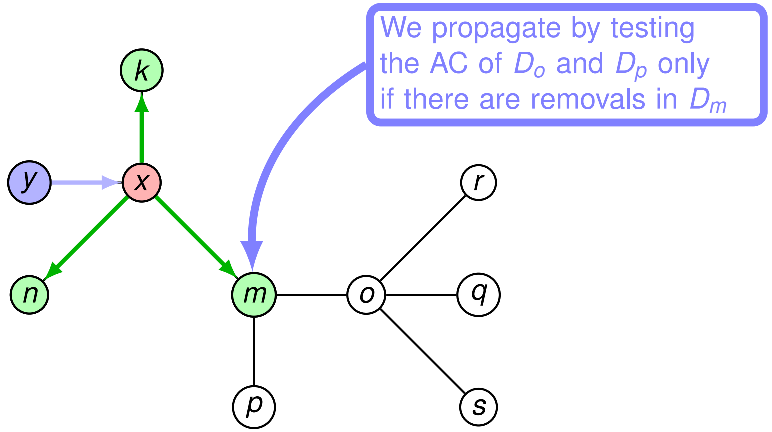
Algorithm AC-3 : illustration

If a value is removed, we verify arc-consistency only for values of neighbour variables.



Algorithm AC-3 : illustration

If a value is removed, we verify arc-consistency only for values of neighbour variables.



Friends analogy — AC-3

AC-1 :

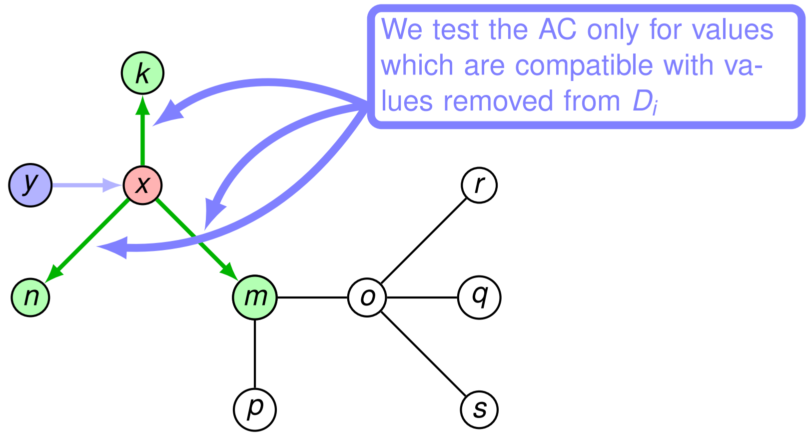
1. Each arc-inconsistent resident leaves the town and sends a letter « I leave the town - Anonymous » to **every resident of the town**.
2. When a resident receives a letter, he verifies whether he is still arc-consistent.

AC-3 :

1. Each arc-inconsistent resident leaves the town and sends a letter « I leave building x - Anonymous » to **residents of neighbour buildings**.
2. When a resident receives a letter, he verifies whether he still has friends **in building** x.

Algorithm AC-4 : illustration

If a value is removed, we verify arc-consistency only for values of neighbour variables which are compatible with the removed value.



Friends analogy — AC-4

AC-3 :

1. Each arc-inconsistent resident leaves the town and sends a letter « I leave building x - Anonymous » **to all residents** of neighbour buildings.
2. When a resident receives a letter, he verifies whether he still has friends **in building** x.

AC-4 :

1. Each arc-inconsistent resident leaves the town and sends a letter « I leave building x - Signed $\langle x, a\rangle$ » **to its friends** in neighbour buildings.
2. When a resident receives a letter, he verifies whether he still has friends **in building** x.

Algorithm AC-4

Overview

- We use structure $\mathcal{S}$: $\mathcal{S}_{i,a}$ is the set of pairs (variable, value) which support (i, a)
- We count the number of supports `counter`$[(i, j), a]$
- As soon as one `counter`$[(i, j), a]$ becomes zero, value a is removed from D_i and the pair (i, a) is added to list Q of revisions to do

Complexity analysis

- Spatial complexity is $O(ed^2)$ (memory needed to keep structure $\mathcal{S}$)
- Initialization of structure $\mathcal{S}$ takes time $O(ed^2)$
- In the course of the algorithm, each value `counter`$[(i, j), a]$ can be decreased at most d times — time complexity is also $O(ed^2)$

Friends analogy — AC-6

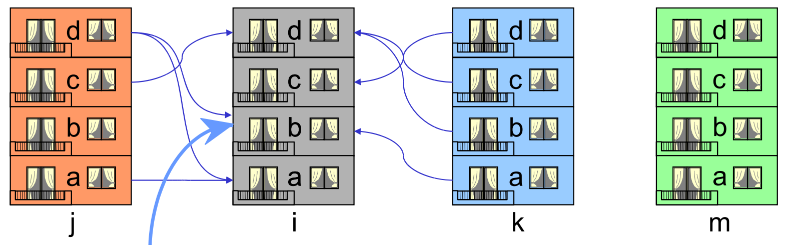
AC-4 :

1. Each arc-inconsistent resident leaves the town and sends a letter « I leave building x - Signed $\langle x, a\rangle$ » **to its friends** in neighbour buildings.
2. When a resident receives a letter, he verifies whether he still has friends **in building** x.

AC-6 :

1. Each arc-inconsistent resident leaves the town and sends a letter « I leave building x - Signed $\langle x, a\rangle$ » **to its friends for which he is the friend living in the smallest apartement number in** c
2. When a resident receives a letter, he **searches another friend in** x **who lives in an apartment with larger number than** $\langle x, a\rangle$

Algorithm AC-6 : illustration



- If resident $\langle i, b\rangle$ leaves, only $\langle k, a\rangle$ will be notified.
- The presence of others will not be challenged by the departure of $\langle i, b\rangle$
- We will search for another friend of $\langle k, a\rangle$ living on a higher floor than $\langle i, b\rangle$.

Arc-consistency algorithms in practice

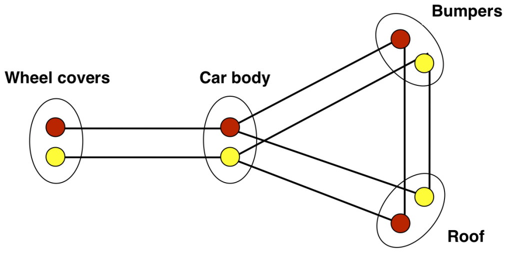
	AC-3	**AC-4**	**AC-6**
Time complexity	$O(ed^3)$	$O(ed^2)$	$O(ed^2)$
« Practical » complexity	$\Omega(ed^2)$	$O(ed^2)$	$O(ed^2)$
Spacial complexity	$O(dn + e)$	$O(ed^2)$	$O(ed)$
Implementation difficulty	easy	medium	hard

Algorithms **AC-3** and **AC-4** are mostly used in practice.

Arc-consistent $\neq$ globally consistent

If a CSP is arc-consistent, this does not mean that CSP is globally consistent (has a solution).

The same example



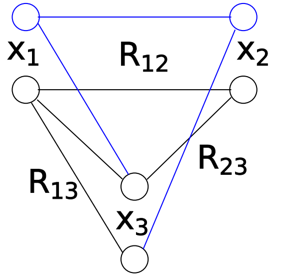
This CSP is arc-consistent, but does not have a solution.

Lignes directrices

Path consistency

A pair of variables are path consistent with the third variable if each arc-consistent pair of values can be extended to another variable in such a way that all binary constraints are satisfied.

Example



Variables x_1 and x_2 are not path consistent with x_3.

We can make them path consistent by removing blue values.

K-consistency

There exist « stronger » local consistencies for finite CSPs..

- A CSP is K-consistent if each set of values of $K - 1$ variables which satisfy all the constraints between them can be « extended » to K-th variable (there exists a value for this K-th variable such that all constrains between these K variables are satisfied).
- The algorithms to make a CSP K-consistent are exponential in K.
- Generally, the « expenses » for this type of local consistency are larger than the advantages of their use.

K-consistency II

Equivalencies

- 1-consistency = node consistency
- 2-consistency = arc-consistency
- 3-consistency = path consistency (for binary CSPs)

Strong consistency

A CSP is strongly K-consistent if it is L-consistent for each $L \leq K$.

Arc-B-consistency

And if the domains are not discrete ? Intervals ?

Arc-B-consistency is the arc-consistency limited to bounds of intervals.

- Weaker that the arc-consistency.
- Easy to implement, therefore broadly used.

Definition

A constraint $C(x_1, \ldots, x_k)$ is arc-B-consistent if and only if

$$\forall x_i \quad \forall a_i \in \{\min(D_{x_i}), \max(D_{x_i})\}$$

$$\exists a_1 \in D_{x_1}, \ldots,\ a_{i-1} \in D_{x_{i-1}},\ a_{i+1} \in D_{x_{i+1}}, \ldots,\ a_k \in D_{x_k}$$

such that $C(a_1, \ldots, a_k)$ is true.

Arc-B-consistency : example

Variables : x, y, z. Domains are intervals.
Constraint : $z = x + y$

$\min(D_z)$ and $\max(D_z)$ are arc-consistent if

$$\begin{cases} \min(D_z) \geq \min(D_x) + \min(D_y) \\ \max(D_z) \leq \max(D_x) + \max(D_y) \end{cases}$$

otherwise

$$D_z \leftarrow \Big[\min(D_x) + \min(D_y), \max(D_x) + \max(D_y)\Big]$$

$\min(D_x)$, $\max(D_x)$?
$\min(D_y)$, $\max(D_y)$?
Constraint $z = \max\{x_1, x_2, \ldots, x_n\}$?