

Local Consistency

Combinatorial Problem Solving (CPS)

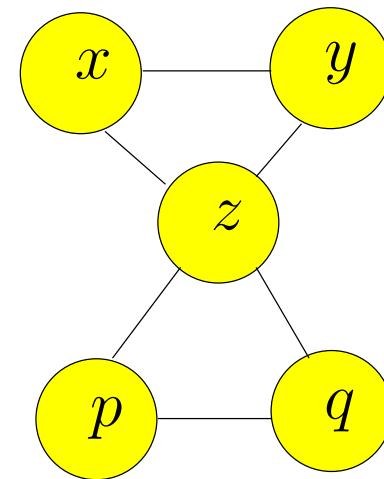
Enric Rodríguez-Carbonell (based on materials by Javier Larrosa)

February 14, 2020

Interaction Graph

- The **interaction graph** of a CSP is an undirected graph $G = (V, E)$ s.t.:
 - ◆ there is a vertex i associated to each variable x_i
 - ◆ there is an edge (i, j) if
there exists some constraint having both x_i and x_j in its scope

CSP with Boolean variables x, y, z, p, q
and constraints: $x + y = z, |p - q| = z$



- For example, the interaction graph of graph coloring is the same graph!

Interaction Graph

- The interaction graph of a CSP is interesting to study
- E.g., connected components of the interaction graph can be solved independently (then just join the solutions)
- Here it is used to describe some propagation notions

Binary CSP's

- A CSP is **binary** if all its constraints have arity 2
- When considering binary CSP's,
 c_{ij} indicates the constraint between variables x_i and x_j
- In what follows we will focus on binary CSP's.
We can do it wlog because of the following property:

Theorem. Any CSP can be transformed into an equisatisfiable binary one.

Binary CSP's

- Consider the CSP with Boolean variables x, y, z, p, q and constraints: $x + y = z, |p - q| = z$.

- The equivalent binary CSP has:

- ◆ *variables*: one for each original constraint

$$v_{x+y=z}, \quad v_{|p-q|=z}$$

- ◆ *domains*: the tuples that satisfy the original constraint

$$\begin{aligned} d_{x+y=z} &= \{(x, y, z) \in \{(0, 0, 0), (1, 0, 1), (0, 1, 1)\}\} \\ d_{|p-q|=z} &= \{(p, q, z) \in \{(0, 0, 0), (1, 0, 1), (0, 1, 1), (1, 1, 0)\}\} \end{aligned}$$

- ◆ *constraint*: satisfying tuples are consistent on common values

$$\begin{aligned} &\{((0, 0, \mathbf{0}), (0, 0, \mathbf{0})), ((0, 0, \mathbf{0}), (1, 1, \mathbf{0})), \\ &\quad ((1, 0, \mathbf{1}), (1, 0, \mathbf{1})), ((1, 0, \mathbf{1}), (0, 1, \mathbf{1})), \\ &\quad ((0, 1, \mathbf{1}), (1, 0, \mathbf{1})), ((0, 1, \mathbf{1}), (0, 1, \mathbf{1}))\} \end{aligned}$$

Binary CSP's

Proof: Let $P = (X, D, C)$ be the non-binary CSP.

An equisatisfiable binary one is $P' = (X', D', C')$ defined as follows:

- There is a variable associated to every constraint in P .
Let x'_i be the variable associated to constraint $c_i \in C$.
- Let $S = (x_{i_1}, \dots, x_{i_k})$ be the scope of c_i .
The domain of x'_i is the set of tuples $\tau \in d_{i_1} \times \dots \times d_{i_k}$ s.t. $c_i(\tau) = 1$.
- There is a binary constraint $c'_{ij} \in C'$ iff
 c_i and c_j have some common variable in their scopes.
- The constraint $c'_{ij}(\tau, \sigma)$ is true if
 τ and σ **match** in their common variables.
- This is known as the **dual graph translation**
- If σ is a solution to P , then a solution σ' to P' is obtained as follows:
the value for x'_i is the projection of σ on the scope of c_i .
- If σ' is a solution to P' , then a solution σ to P is obtained as follows:
the value for x_j is the value assigned in σ' by any of the constraints where x_j appears

Filtering, Propagation

- Let $P = (X, D, C)$ be a (binary) CSP,
 $x_i \in X$ a variable and $a \in d_i$ a domain value
- $P[x_i \rightarrow a]$ is the CSP obtained from P by replacing d_i by $\{a\}$
- $a \in d_i$ is **feasible** if $P[x_i \rightarrow a]$ has solutions, **unfeasible** otherwise
- Example:
 - ◆ Let x, y be two integer variables with domains $[1, 10]$
 - ◆ Consider constraint $|x - y| > 5 \equiv (x - y > 5) \vee (y - x > 5)$
 - ◆ Values 5 and 6 for both variables are unfeasible

Filtering, Propagation

- Let $P = (X, D, C)$ be a (binary) CSP,
 $x_i \in X$ a variable and $a \in d_i$ a domain value
- $P[x_i \rightarrow a]$ is the CSP obtained from P by replacing d_i by $\{a\}$
- $a \in d_i$ is **feasible** if $P[x_i \rightarrow a]$ has solutions, **unfeasible** otherwise
- Example:
 - ◆ Let x, y be two integer variables with domains $[1, 10]$
 - ◆ Consider constraint $|x - y| > 5 \equiv (x - y > 5) \vee (y - x > 5)$
 - ◆ Values **5** and **6** for both variables are unfeasible
- Detecting unfeasible values is useful because they can be removed without losing solutions
- In general, detecting if a value is feasible is an NP-Complete problem
- But in some cases unfeasible values can be easily detected
- **Filtering algorithms** identify and remove unfeasible values efficiently
Filtering is also called **propagation** (and filtering algorithms, **propagators**)

Local Consistency

- A clean way of designing filtering techniques is by means of the concept of **local consistency**
 - ◆ A **local consistency** property allows identifying unfeasible values: inconsistent values, i.e., not satisfying the property, are unfeasible
 - ◆ **Local** because only small pieces of the problem are considered (typically, one constraint)
 - ◆ **Enforcing a local consistency** property means propagating: removing the inconsistent values until the property is achieved
 - ◆ Enforcing local consistency should be cheap (polynomial time)

Local Consistency Properties

- The most important is Arc Consistency (AC)
(aka Domain Consistency)
- Weaker than AC:
 - ◆ Directional AC (DAC)
 - ◆ Bounds Consistency (BC)
 - ◆ ...
- Stronger than AC:
 - ◆ Singleton AC (SAC)
 - ◆ Neighborhood Inverse Consistency (NIC)
 - ◆ ...

Arc Consistency

- Let $P = (X, D, C)$ be a (binary) CSP
- Value $a \in d_i$ of variable $x_i \in X$ is **arc-consistent** wrt. x_j if there is $b \in d_j$ (the **support** of a in x_j) s.t. $c_{ij}(a, b) = \text{true}$
- The definition of arc-consistency is then lifted in the natural way:
 - ◆ Variable $x_i \in X$ is **arc-consistent** wrt. x_j if all values in its domain are arc-consistent wrt. x_j
 - ◆ Constraint $c_{ij} \in C$ is **arc-consistent** if x_i is arc-consistent wrt. x_j , and vice-versa
 - ◆ A CSP P is **arc-consistent** if all $c \in C$ are arc-consistent
- Notation: **AC** means **arc-consistent**
- If $a \in d_i$ is arc-inconsistent (not arc-consistent) wrt. some x_j , it is **unfeasible**. Hence, it can be safely removed!
- **Enforcing AC** means removing arc-inconsistent values until AC is achieved

Arc Consistency

- The AC name comes from the **arcs** (constraints) of the interaction graph
- **Example of enforcing AC.**

Consider the CSP with variables x, y, z ,
domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$,
and constraints $x < y, y < z$

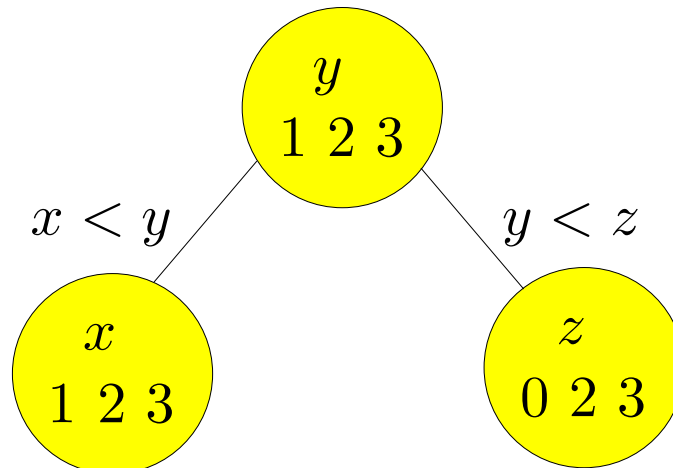
- Recall nodes are labelled with variables (here, also with their domains)
- Recall edges are labelled with constraints

Arc Consistency

- The AC name comes from the **arcs** (constraints) of the interaction graph
- **Example of enforcing AC.**

Consider the CSP with variables x, y, z ,
domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$,
and constraints $x < y$, $y < z$

- Recall nodes are labelled with variables (here, also with their domains)
- Recall edges are labelled with constraints

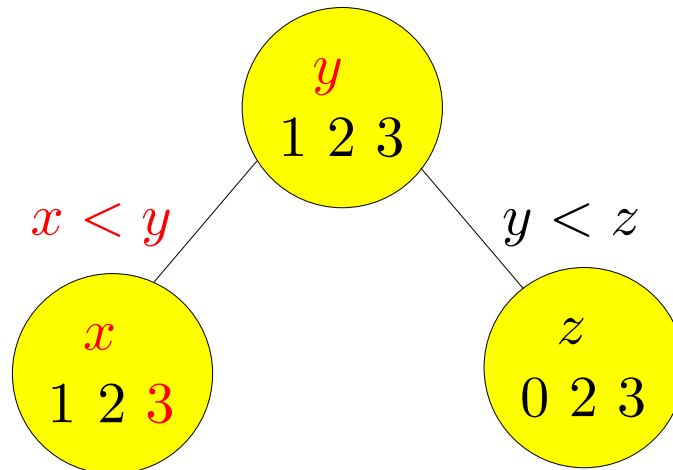


Arc Consistency

- The AC name comes from the **arcs** (constraints) of the interaction graph
- **Example of enforcing AC.**

Consider the CSP with variables x, y, z ,
domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$,
and constraints $x < y$, $y < z$

- Recall nodes are labelled with variables (here, also with their domains)
- Recall edges are labelled with constraints

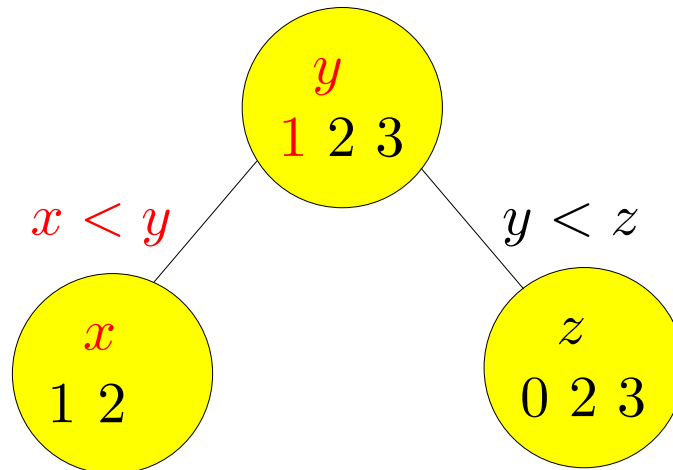


Arc Consistency

- The AC name comes from the **arcs** (constraints) of the interaction graph
- **Example of enforcing AC.**

Consider the CSP with variables x, y, z ,
domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$,
and constraints $x < y$, $y < z$

- Recall nodes are labelled with variables (here, also with their domains)
- Recall edges are labelled with constraints

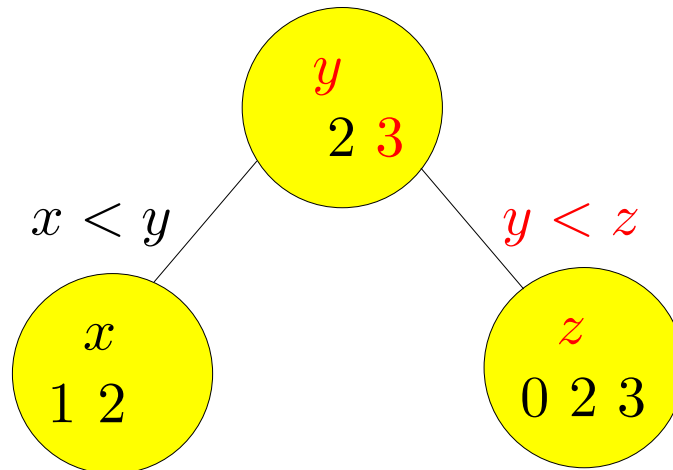


Arc Consistency

- The AC name comes from the **arcs** (constraints) of the interaction graph
- **Example of enforcing AC.**

Consider the CSP with variables x, y, z ,
domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$,
and constraints $x < y$, $y < z$

- Recall nodes are labelled with variables (here, also with their domains)
- Recall edges are labelled with constraints

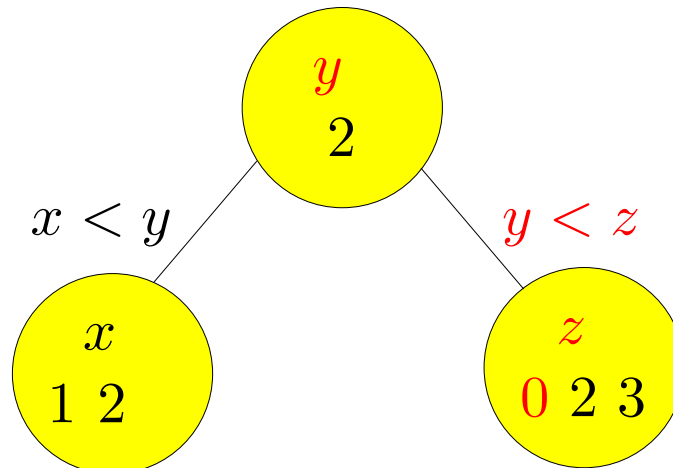


Arc Consistency

- The AC name comes from the **arcs** (constraints) of the interaction graph
- **Example of enforcing AC.**

Consider the CSP with variables x, y, z ,
domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$,
and constraints $x < y$, $y < z$

- Recall nodes are labelled with variables (here, also with their domains)
- Recall edges are labelled with constraints

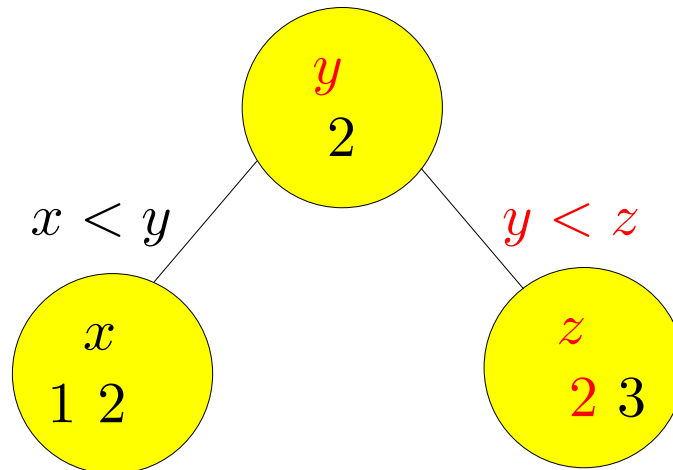


Arc Consistency

- The AC name comes from the **arcs** (constraints) of the interaction graph
- **Example of enforcing AC.**

Consider the CSP with variables x, y, z ,
domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$,
and constraints $x < y$, $y < z$

- Recall nodes are labelled with variables (here, also with their domains)
- Recall edges are labelled with constraints

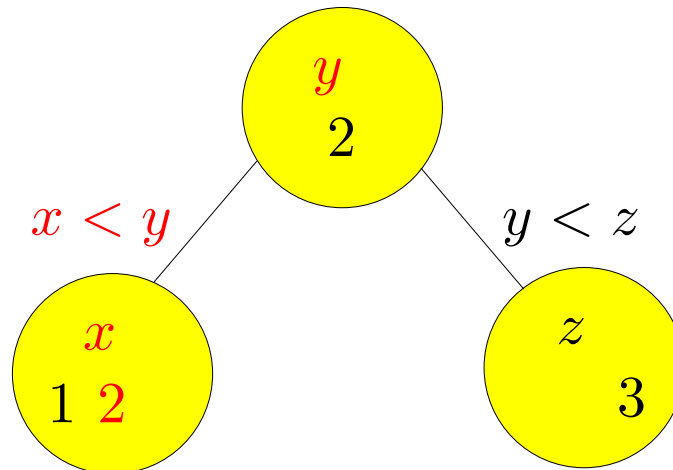


Arc Consistency

- The AC name comes from the **arcs** (constraints) of the interaction graph
- **Example of enforcing AC.**

Consider the CSP with variables x, y, z ,
domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$,
and constraints $x < y$, $y < z$

- Recall nodes are labelled with variables (here, also with their domains)
- Recall edges are labelled with constraints

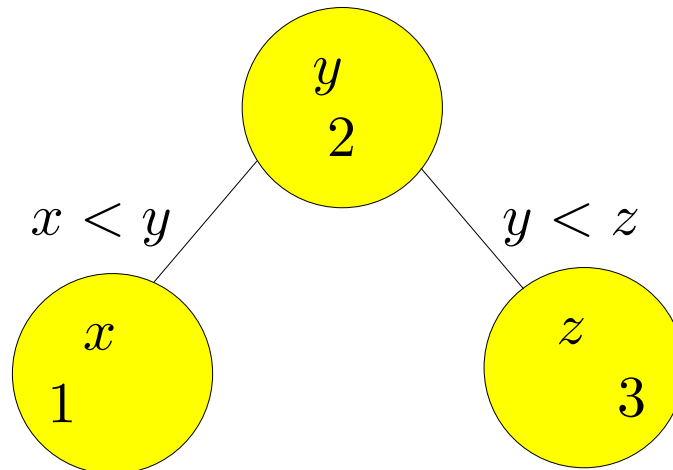


Arc Consistency

- The AC name comes from the **arcs** (constraints) of the interaction graph
- **Example of enforcing AC.**

Consider the CSP with variables x, y, z ,
domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$,
and constraints $x < y$, $y < z$

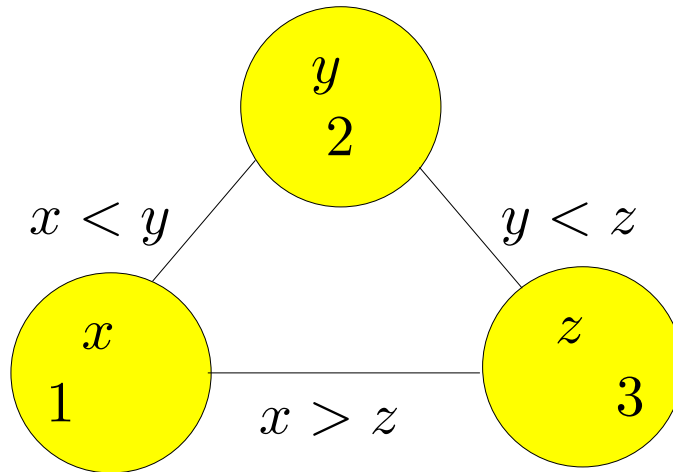
- Recall nodes are labelled with variables (here, also with their domains)
- Recall edges are labelled with constraints



Arc Consistency

■ Another example of enforcing AC.

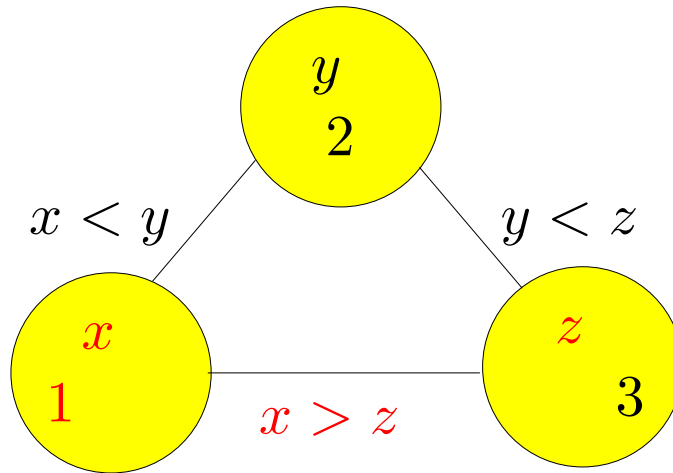
Consider the CSP with variables x, y, z ,
domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$,
and constraints $x < y$, $y < z$, $x > z$



Arc Consistency

■ Another example of enforcing AC.

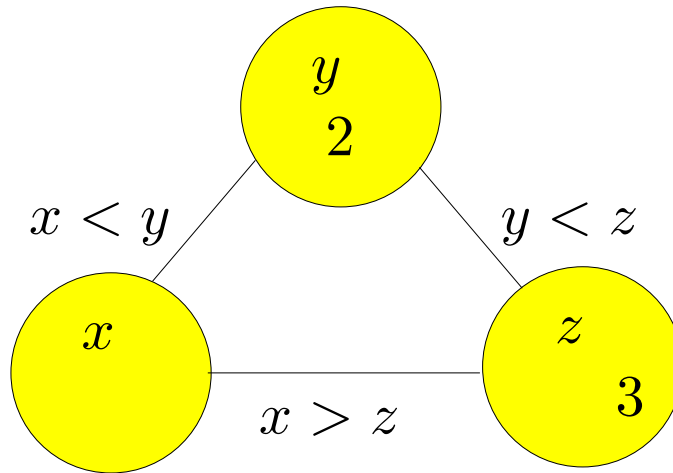
Consider the CSP with variables x, y, z , domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



Arc Consistency

■ Another example of enforcing AC.

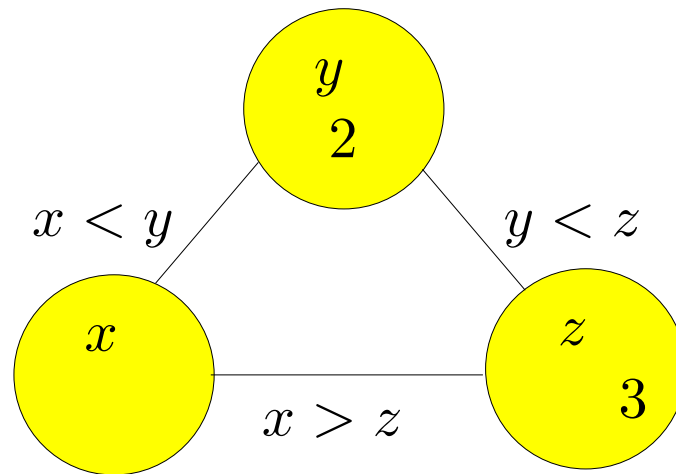
Consider the CSP with variables x, y, z , domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



Arc Consistency

- Another example of enforcing AC.

Consider the CSP with variables x, y, z , domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



- The domain of x has become empty!
- So the CSP **cannot** have any solution

Arc Consistency

- If while enforcing AC some **domain becomes empty**, then we know the original CSP has **no solution**
- Else, when there are no more arc-inconsistent values, we have a smaller equivalent CSP (no solution is lost)
- **Uniqueness:** the order of removal of arc-inconsistent values is irrelevant
- Now let us see algorithms for effectively enforcing AC

AC-1: $\text{Revise}(i, j)$

The simplest algorithm to enforce AC is called **AC-1**

Based on function $\text{Revise}(i, j)$,
which removes values from d_i without support in d_j .
Returns **true** if some value is removed

```
function  $\text{Revise}(i, j)$   
  change := false  
  for each  $a \in d_i$  do  
    if  $\forall b \in d_j \neg c_{ij}(a, b)$  then  
      change := true  
      remove  $a$  from  $d_i$   
  return change
```

The time complexity of $\text{Revise}(i, j)$ is $O(|d_i| \cdot |d_j|)$
(we assume that evaluating a binary constraint takes constant time)

AC-1

```
procedure AC-1( $X, D, C$ )  
  repeat  
     $\text{change} := \text{false}$   
    for each  $c_{ij} \in C$  do  
       $\text{change} := \text{change} \vee \text{Revise}(i, j) \vee \text{Revise}(j, i)$   
  until  $\neg \text{change}$ 
```

- The time complexity of AC-1 is $O(e \cdot n \cdot m^3)$,
with $n = |X|$, $m = \max_i \{|d_i|\}$ and $e = |C|$ (note $e = O(n^2)$)
- Whenever a value has been removed,
the domain should be checked if empty (not done here for simplicity)

AC-3

- A more efficient algorithm is **AC-3**, which only revises constraints with a chance of filtering domains
- AC-3 uses a set of pairs Q s.t. if $(i, j) \in Q$ then we can't ensure that all values in d_i have support in x_j

procedure AC-3(X, D, C)

$Q := \{(i, j), (j, i) \mid c_{ij} \in C\}$ // each pair is added twice

while $Q \neq \emptyset$ **do**

$(i, j) := \text{Fetch}(Q)$ // selects and removes from Q

if $\text{Revise}(i, j)$ **then**

$Q := Q \cup \{(k, i) \mid c_{ki} \in C, k \neq j\}$

- Space complexity: $O(e)$
- Time complexity: $O(e \cdot m^3)$

Complexity of AC-3

```
 $Q := \{(i, j), (j, i) \mid c_{ij} \in C\}$  // each pair is added twice  
while  $Q \neq \emptyset$  do  
     $(i, j) := \text{Fetch}(Q)$  // selects and removes from  $Q$   
    if  $\text{Revise}(i, j)$  then  
         $Q := Q \cup \{(k, i) \mid c_{ki} \in C, k \neq j\}$ 
```

- (i, j) is in Q because d_j has been pruned.
Therefore, it will be in Q at most m times.
Consequently, the loop iterates at most $O(e \cdot m)$ times
- Without the red part, the cost of AC-3 would be $O(e \cdot m^3)$
- For a given i , $\text{Revise}(i, j)$ will be true at most m times.
Aggregated cost of red part due to i is

$$O(|\{k \mid c_{ki} \in C\}| \cdot m) = O(\deg(i) \cdot m)$$

So aggregated cost of red part due to all vars is $O(e \cdot m)$

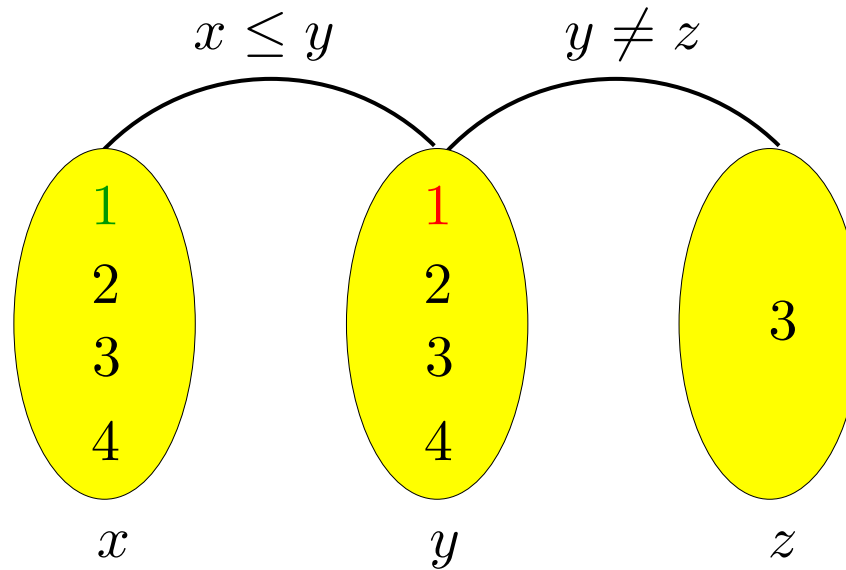
- So the total cost in time is $O(e \cdot m^3)$

Example of AC-3

- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$

Example of AC-3

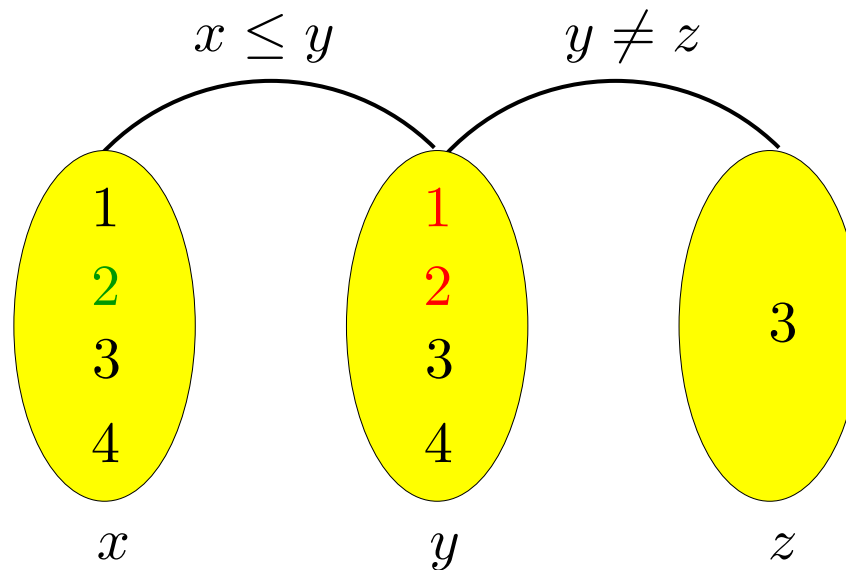
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(x, y)$
 - Finding support for $(x, 1)$: 1

Example of AC-3

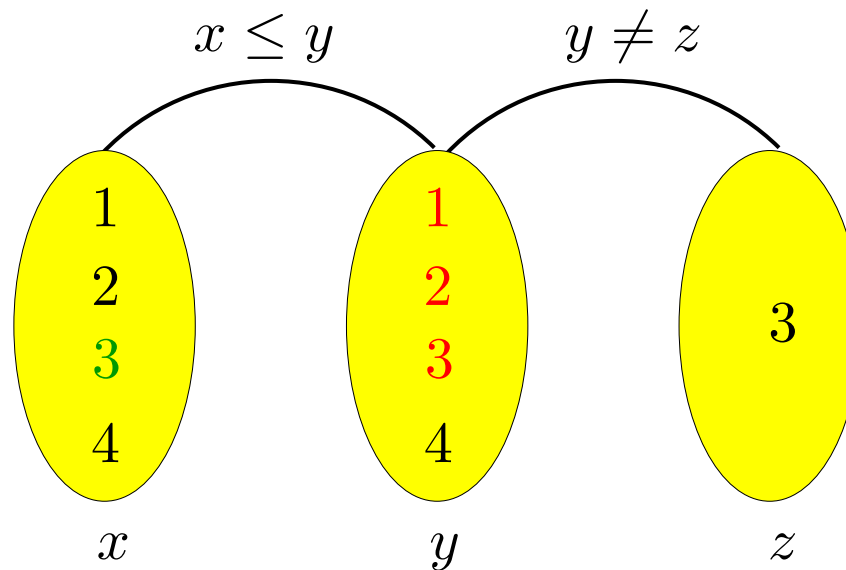
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(x, y)$
 - Finding support for $(x, 2)$: 2

Example of AC-3

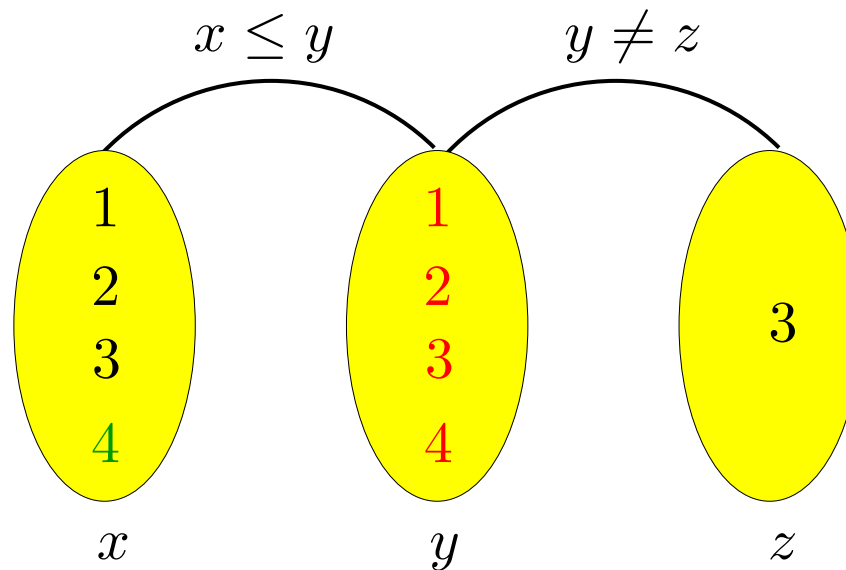
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(x, y)$
 - Finding support for $(x, 3)$: 3

Example of AC-3

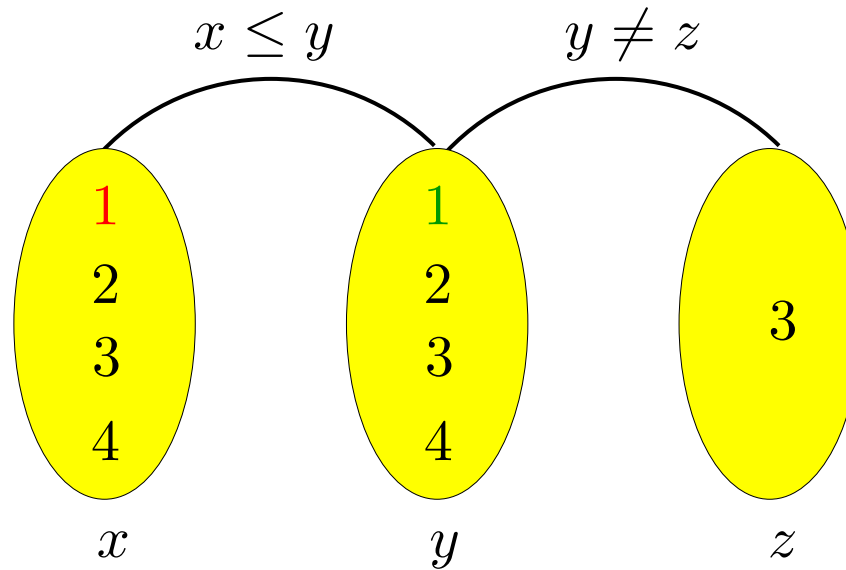
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(x, y)$
 - Finding support for $(x, 4)$: 4

Example of AC-3

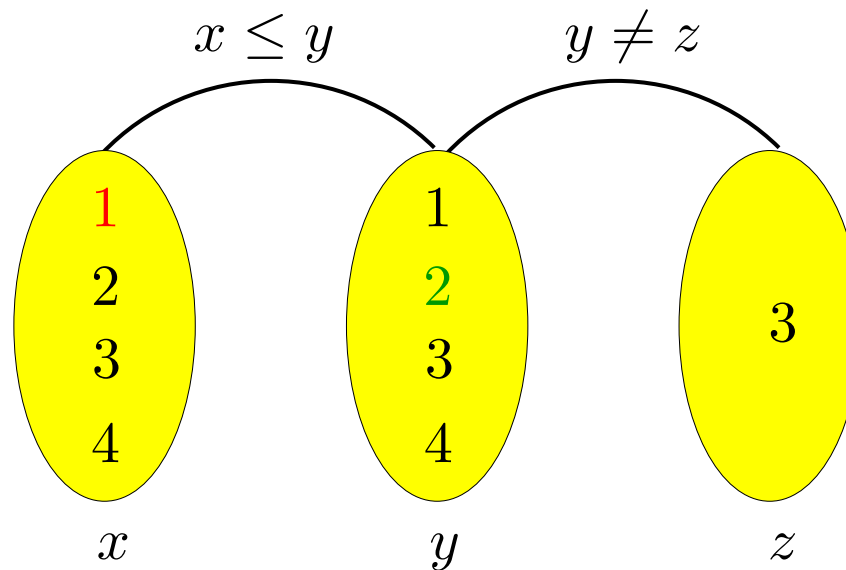
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(y, x)$
 - Finding support for $(y, 1)$: 1

Example of AC-3

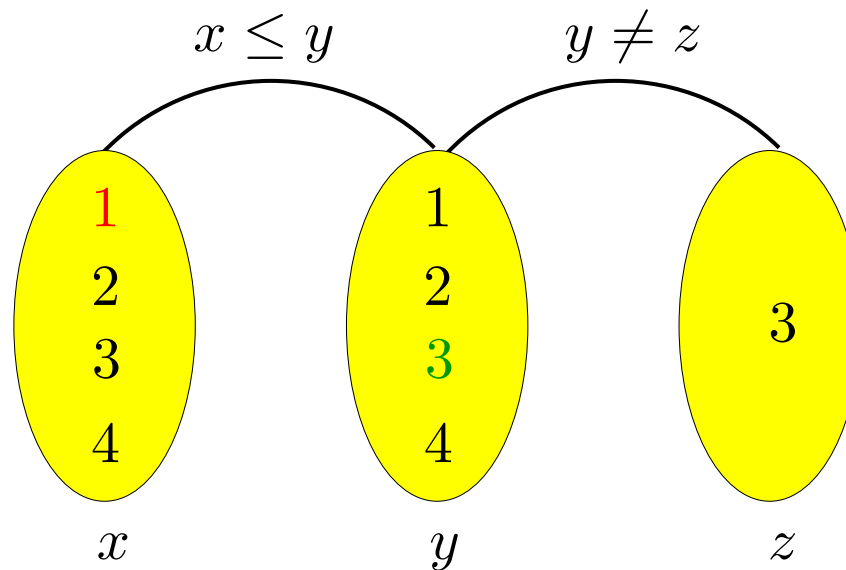
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(y, x)$
 - Finding support for $(y, 2)$: 1

Example of AC-3

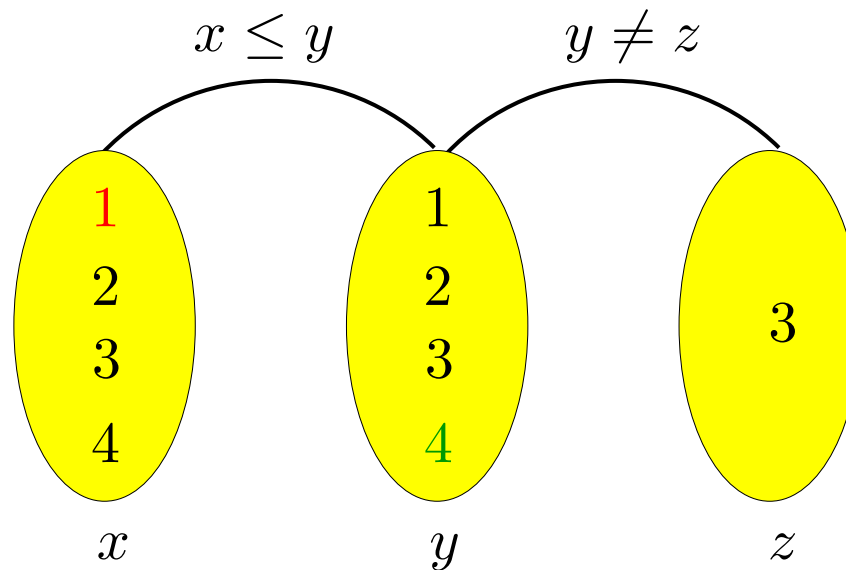
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(y, x)$
 - Finding support for $(y, 3)$: 1

Example of AC-3

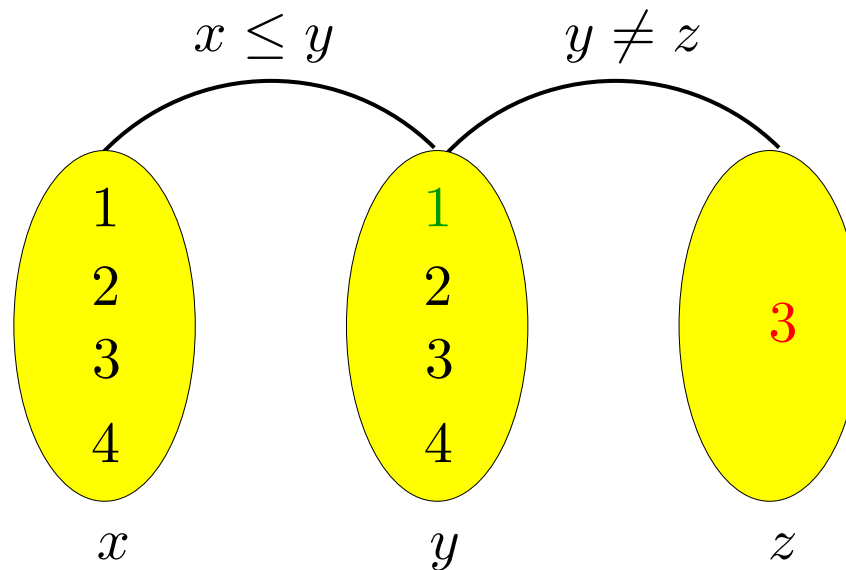
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(y, x)$
 - Finding support for $(y, 4)$: 1

Example of AC-3

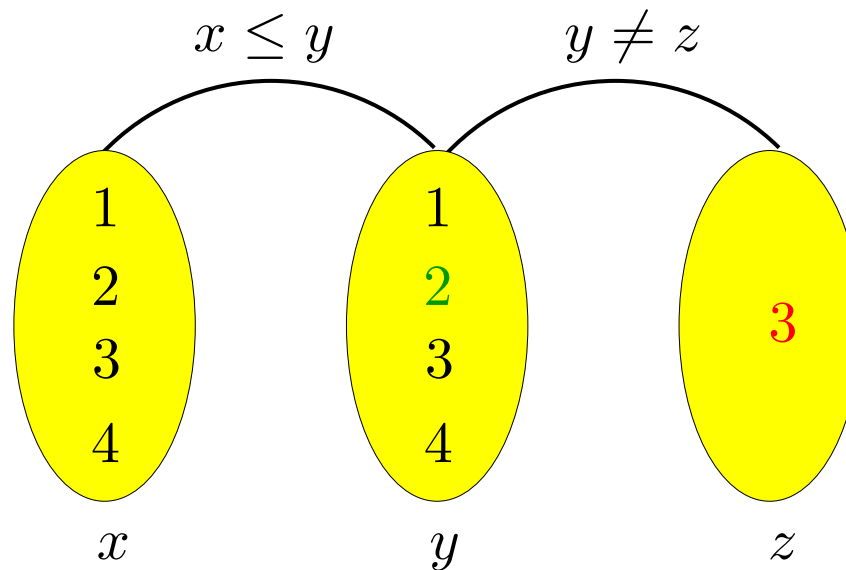
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(y, z)$
 - Finding support for $(y, 1)$: 1

Example of AC-3

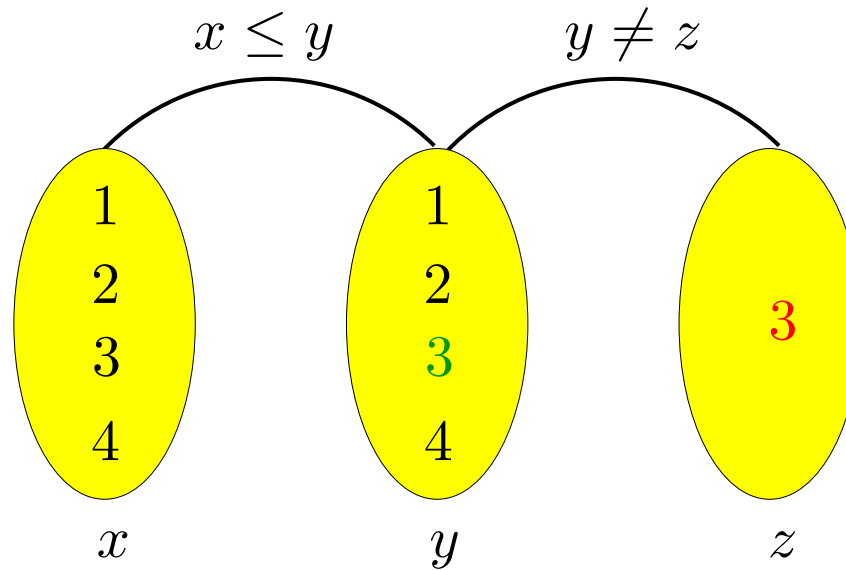
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(y, z)$
 - Finding support for $(y, 2)$: 1

Example of AC-3

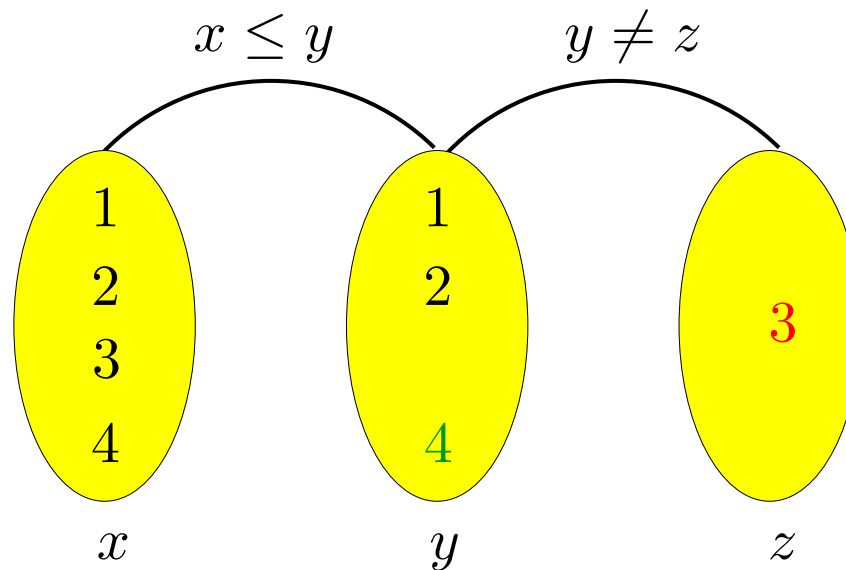
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(y, z)$
 - Finding support for $(y, 3)$: 1

Example of AC-3

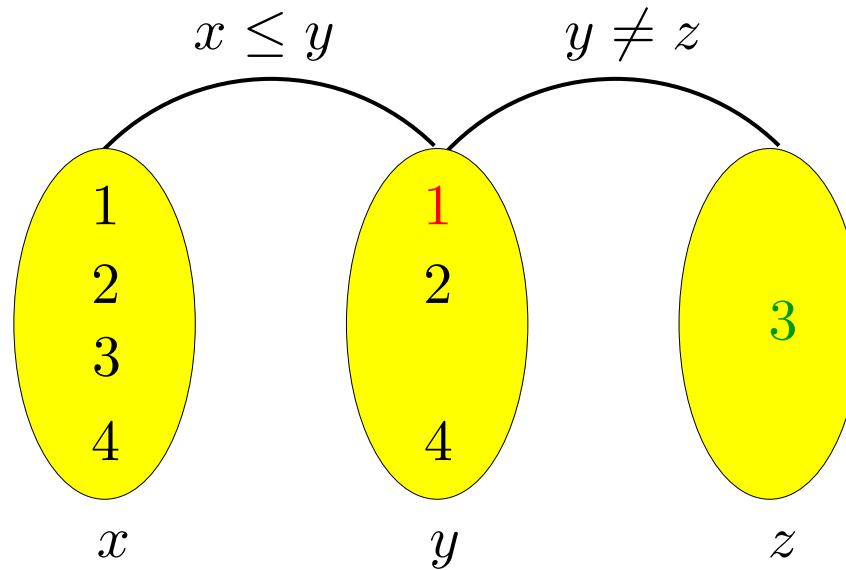
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(y, z)$
 - Finding support for $(y, 4)$: 1

Example of AC-3

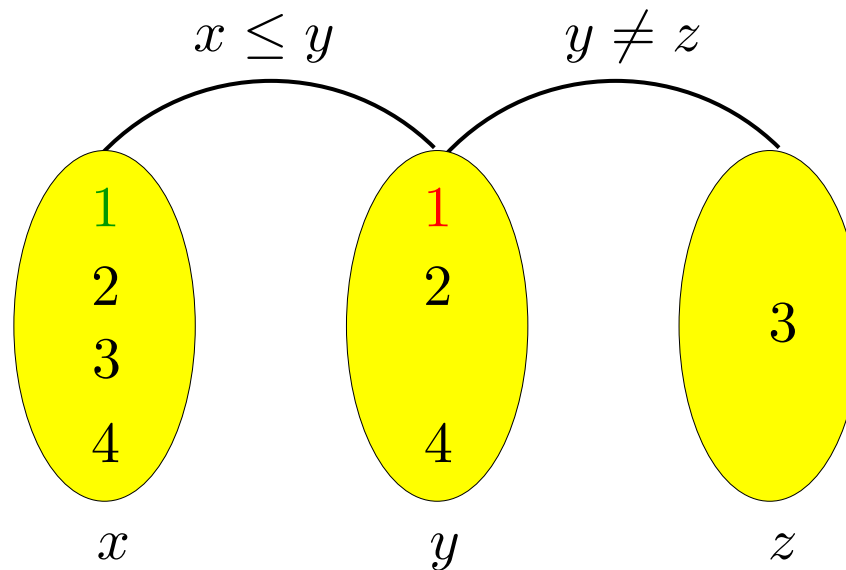
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(z, y)$
 - Finding support for $(z, 3)$: 1

Example of AC-3

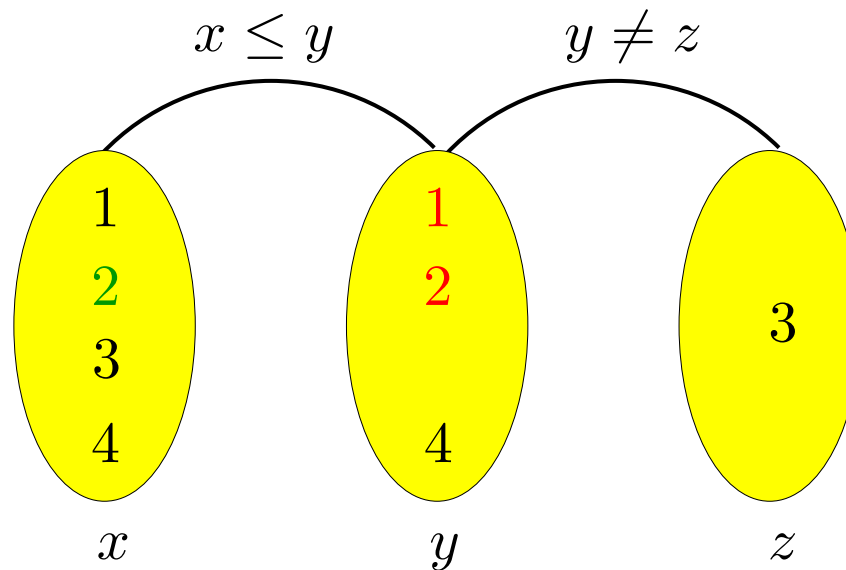
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(x, y)$
 - Finding support for $(x, 1)$: 1

Example of AC-3

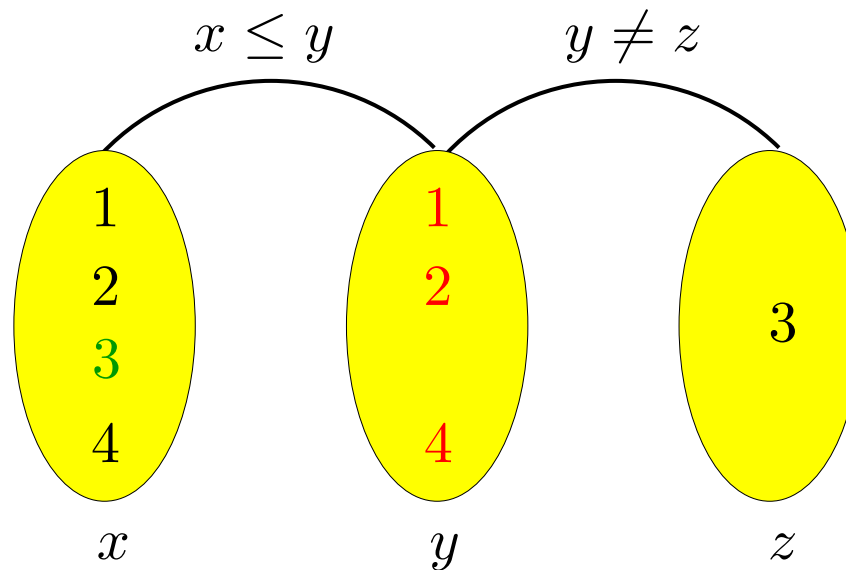
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(x, y)$
 - Finding support for $(x, 2)$: 2

Example of AC-3

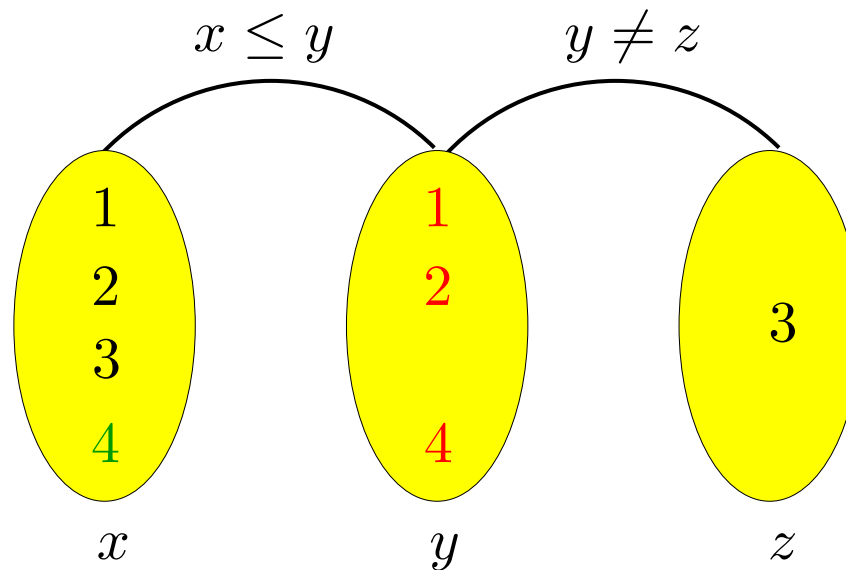
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(x, y)$
 - Finding support for $(x, 3)$: 3

Example of AC-3

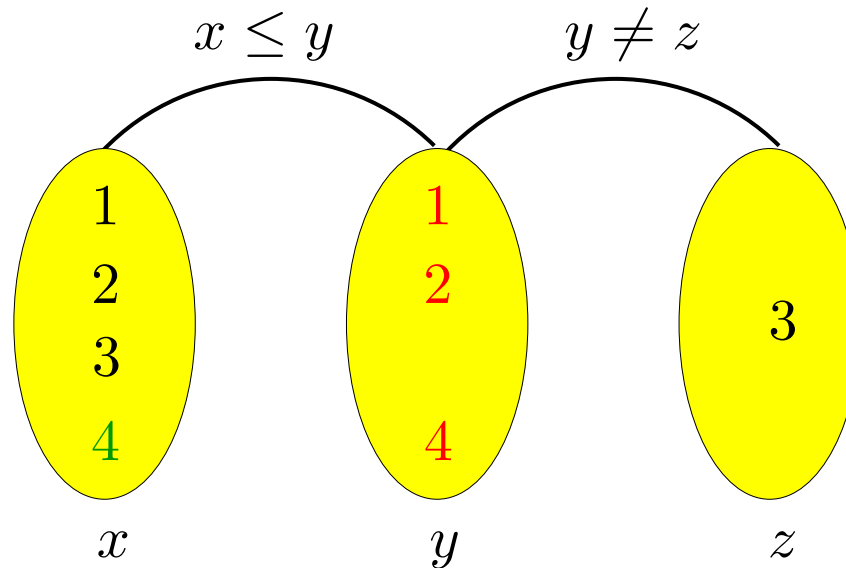
- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Let us count the number of constraint checks of $\text{Revise}(x, y)$
 - Finding support for $(x, 4)$: 3

Example of AC-3

- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$



- Altogether $10 + 4 + 4 + 1 + 9 = 28$ constraint checks
- From last 9, the only new check is $((x, 3), (y, 4))$!
- Still a lot of **work** is **repeated over and over** again

AC-4

- **AC-4** is an even more efficient algorithm. It uses:
 - ◆ $N[i, a, j]$ = “number of supports that $a \in d_i$ has in d_j ”
 - ◆ $S[j, b]$ = “set of pairs (i, a) supported by $b \in d_j$ ”
 - ◆ $(i, a) \in Q$ means that a has been pruned from d_i and its effect has not been updated on the N array yet

procedure AC-4(X, D, C)

// N is constructed full of 0's and S is constructed full of \emptyset 's

// Initialization phase

for each $c_{ij} \in C$, $a \in d_i$, $b \in d_j$ **such that** $c_{ij}(a, b)$ **do**

// Value b in d_j is a support for value $a \in d_i$

$N[i, a, j]++$

$S[j, b] := S[j, b] \cup \{(i, a)\}$

for each $c_{ij} \in C$, $a \in d_i$ **such that** $N[i, a, j] = 0$ **do**

remove a **from** d_i

$Q := Q \cup (i, a)$

...

AC-4

...

// Propagation phase

while $Q \neq \emptyset$ **do**

$(j, b) := \text{Fetch}(Q)$

for each $(i, a) \in S[j, b]$ **such that** $a \in d_i$ **do**

$N[i, a, j] \leftarrow -$

if $N[i, a, j] = 0$ **then**

remove a **from** d_i

$Q := Q \cup (i, a)$

- Time complexity of AC-4: $O(e \cdot m^2)$ (provable optimal!)
 - ◆ the initialization phase has cost $O(e \cdot m^2)$
 - ◆ the propagation phase has cost $O(e \cdot m^2)$
- Space complexity of AC-4: $O(e \cdot m^2)$

Example of AC-4

- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$
- During initialization, AC-4 performs all possible constraint checks for every value in each domain
 - ◆ For c_1 : $4 \cdot 4 = 16$ constraint checks
 - ◆ For c_2 : $4 \cdot 1 = 4$ constraint checks
 - ◆ In total 20 constraint checks

Example of AC-4

- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$
- After initialization:

$$N[x, 1, y] = 4 \quad N[y, 1, x] = 1 \quad N[y, 1, z] = 1 \quad N[z, 3, y] = 3$$

$$N[x, 2, y] = 3 \quad N[y, 2, x] = 2 \quad N[y, 2, z] = 1$$

$$N[x, 3, y] = 2 \quad N[y, 3, x] = 3 \quad N[y, 3, z] = 0$$

$$N[x, 4, y] = 1 \quad N[y, 4, x] = 4 \quad N[y, 4, z] = 1$$

$$S[x, 1] = \{(y, 1), (y, 2), (y, 3), (y, 4)\}$$

$$S[y, 1] = \{(z, 3), (x, 1)\}$$

$$S[x, 2] = \{(y, 2), (y, 3), (y, 4)\}$$

$$S[y, 2] = \{(z, 3), (x, 1), (x, 2)\}$$

$$S[x, 3] = \{(y, 3), (y, 4)\}$$

$$S[y, 3] = \{(x, 1), (x, 2), (x, 3)\}$$

$$S[x, 4] = \{(y, 4)\}$$

$$S[y, 4] = \{(z, 3), (x, 1), (x, 2), (x, 3), (x, 4)\}$$

$$S[z, 3] = \{(y, 1), (y, 2), (y, 4)\}$$

Example of AC-4

- The only counter equal to zero is $N[y, 3, z]$
- $(y, 3)$ is removed, propagation loop starts with $(y, 3)$ in Q
- When $(y, 3)$ is picked, traverse $S[y, 3] = \{(x, 1), (x, 2), (x, 3)\}$
 - ◆ $N[x, 1, y], N[x, 2, y], N[x, 3, y]$ are decremented
- No counter becomes zero, so nothing is added to Q
- Propagation did **not** require any **extra constraint check!**
- However
 - ◆ **Space complexity** is very **high**
 - ◆ The **initialization phase** can be **prohibitive**
(AC-4 has optimal worst-case complexity ...
but always reaches this worst case)

AC-6

- **Motivation:** keep the optimal worst-case of AC-4 but instead of counting **all** supports that a value has, just ensure that there is **at least one**
- AC-6 keeps the **smallest** support for each (x_i, a) on c_{ij}
- *Initialization phase:* cheaper than in AC-4
- *Propagation phase:* if value b of var x_j is removed
 - ◆ if it is not the **current** support of value a of var x_i :
no work due to constraint c_{ij} has to be done
 - ◆ if it is the **current** support of value a of var x_i :
a **new** support is sought,
but starting **from next value of b** in d_j instead of $\min\{d_j\}$
- The algorithm uses:

$S[j, b]$ = “set of (i, a) s.t. b is the **current** support of value a of x_i wrt. x_j ”

AC-6

procedure AC-6(X, D, C)

$Q := \emptyset; S[j, b] := \emptyset, \forall x_j \in X, \forall b \in d_j$

for each $x_i \in X, c_{ij} \in C, a \in d_i$ **do**

$b :=$ smallest value in d_j s.t. $c_{ij}(a, b)$

if $b \neq \perp$ **then** add (i, a) to $S[j, b]$

else remove a from d_i and add (i, a) to Q

while $Q \neq \emptyset$ **do**

$(j, b) := \text{Fetch}(Q)$

for each $(i, a) \in S[j, b]$ **such that** $a \in d_i$ **do**

$c :=$ smallest value $b' \in d_j$ s.t. $b' > b \wedge c_{ij}(a, b')$

if $c \neq \perp$ **then** add (i, a) to $S[j, c]$

else remove a from d_i and add (i, a) to Q

- Time complexity: $O(e \cdot m^2)$
- Space complexity: $O(e \cdot m)$

Example of AC-6

- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$
- In initialization, AC-6 performs the same number of constraint checks as AC-3: 10+4 on c_1 and 4+1 on c_2

$$S[x, 1] = \{(y, 1), (y, 2), (y, 3), (y, 4)\}$$

$$S[x, 2] = \{\}$$

$$S[x, 3] = \{\}$$

$$S[x, 4] = \{\}$$

$$S[y, 1] = \{(z, 3), (x, 1)\}$$

$$S[y, 2] = \{(x, 2)\}$$

$$S[y, 3] = \{(x, 3)\}$$

$$S[y, 4] = \{(x, 4)\}$$

$$S[z, 3] = \{(y, 1), (y, 2), (y, 4)\}$$

Example of AC-6

- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$
- In initialization, AC-6 performs the same number of constraint checks as AC-3: 10+4 on c_1 and 4+1 on c_2

$$S[x, 1] = \{(y, 1), (y, 2), (y, 3), (y, 4)\}$$

$$S[y, 1] = \{(z, 3), (x, 1)\}$$

$$S[x, 2] = \{\}$$

$$S[y, 2] = \{(x, 2)\}$$

$$S[x, 3] = \{\}$$

$$S[y, 3] = \{(x, 3)\}$$

$$S[x, 4] = \{\}$$

$$S[y, 4] = \{(x, 4)\}$$

$$S[z, 3] = \{(y, 1), (y, 2), (y, 4)\}$$

- Q contains $(y, 3)$, which has been removed

Example of AC-6

- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$
- In initialization, AC-6 performs the same number of constraint checks as AC-3: 10+4 on c_1 and 4+1 on c_2

$$S[x, 1] = \{(y, 1), (y, 2), (y, 3), (y, 4)\}$$

$$S[y, 1] = \{(z, 3), (x, 1)\}$$

$$S[x, 2] = \{\}$$

$$S[y, 2] = \{(x, 2)\}$$

$$S[x, 3] = \{\}$$

$$S[y, 3] = \{(x, 3)\}$$

$$S[x, 4] = \{\}$$

$$S[y, 4] = \{(x, 4)\}$$

$$S[z, 3] = \{(y, 1), (y, 2), (y, 4)\}$$

- When AC-6 enters propagation loop it pops $(y, 3)$ from Q , $S[y, 3] = \{(x, 3)\}$ is traversed and a new support **greater than 3** is sought for $(x, 3)$.

Example of AC-6

- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$
- In initialization, AC-6 performs the same number of constraint checks as AC-3: 10+4 on c_1 and 4+1 on c_2

$$S[x, 1] = \{(y, 1), (y, 2), (y, 3), (y, 4)\}$$

$$S[y, 1] = \{(z, 3), (x, 1)\}$$

$$S[x, 2] = \{\}$$

$$S[y, 2] = \{(x, 2)\}$$

$$S[x, 3] = \{\}$$

$$S[y, 3] = \{(x, 3)\}$$

$$S[x, 4] = \{\}$$

$$S[y, 4] = \{(x, 4)\}$$

$$S[z, 3] = \{(y, 1), (y, 2), (y, 4)\}$$

- Just 1 extra constraint check:
as $c_1(3, 4)$, we add $(x, 3)$ to $S[y, 4]$

Example of AC-6

- Let x, y, z be vars with domains $d_x = d_y = \{1, 2, 3, 4\}$, $d_z = \{3\}$, and $c_1 \equiv x \leq y$, $c_2 \equiv y \neq z$
- In initialization, AC-6 performs the same number of constraint checks as AC-3: 10+4 on c_1 and 4+1 on c_2

$$S[x, 1] = \{(y, 1), (y, 2), (y, 3), (y, 4)\}$$

$$S[y, 1] = \{(z, 3), (x, 1)\}$$

$$S[x, 2] = \{\}$$

$$S[y, 2] = \{(x, 2)\}$$

$$S[x, 3] = \{\}$$

$$S[y, 3] = \{(x, 3)\}$$

$$S[x, 4] = \{\}$$

$$S[y, 4] = \{(x, 4)\}$$

$$S[z, 3] = \{(y, 1), (y, 2), (y, 4)\}$$

- In total 20 constraint checks

Weaker than AC

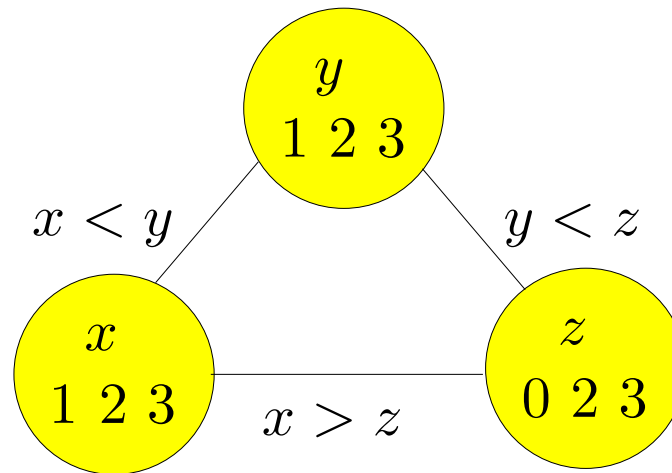
- Directional AC (DAC)
- Bounds Consistency (BC)

Directional Arc Consistency

- Given an order \prec among variables, each x_i only needs supports with respect to greater variables in the order
- Consider a CSP $P = (X, D, C)$.
Constraint $c_{ij} \in C$ is **directionally arc-consistent**
iff $x_i \prec x_j$ implies x_i is arc-consistent with respect to x_j
- The CSP P is **directionally arc-consistent (DAC)**
iff all its constraints are directionally arc-consistent
- DAC is weaker than AC but is enforced more efficiently in practice

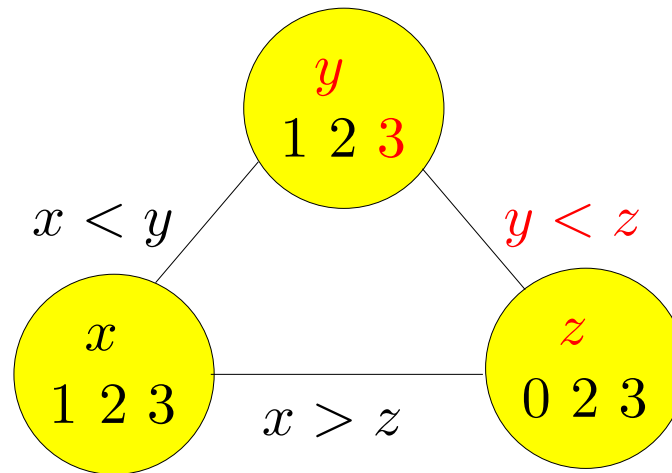
Directional Arc Consistency

- Consider CSP with vars $x \prec y \prec z$, domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



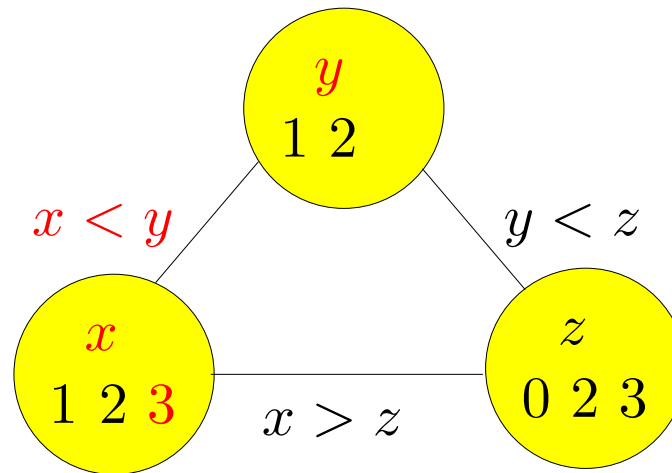
Directional Arc Consistency

- Consider CSP with vars $x \prec y \prec z$,
domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$,
and constraints $x < y$, $y < z$, $x > z$



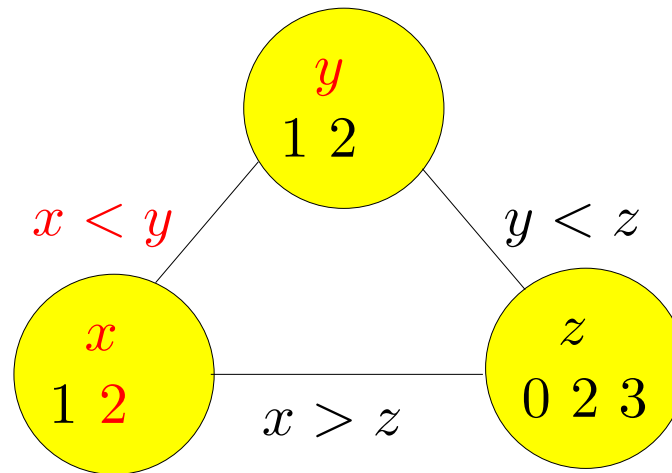
Directional Arc Consistency

- Consider CSP with vars $x \prec y \prec z$, domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



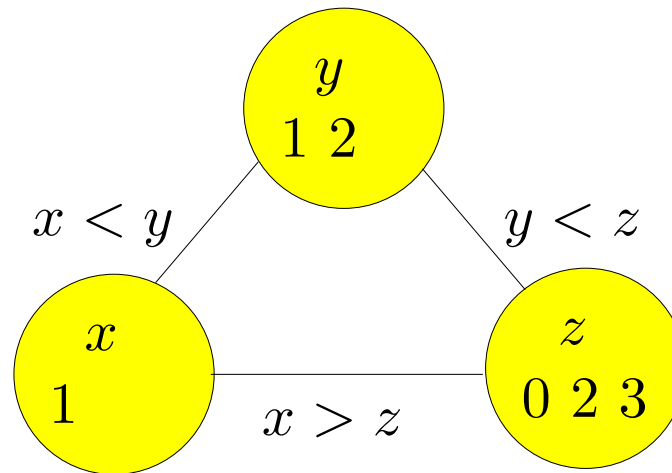
Directional Arc Consistency

- Consider CSP with vars $x \prec y \prec z$,
domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$,
and constraints $x < y$, $y < z$, $x > z$



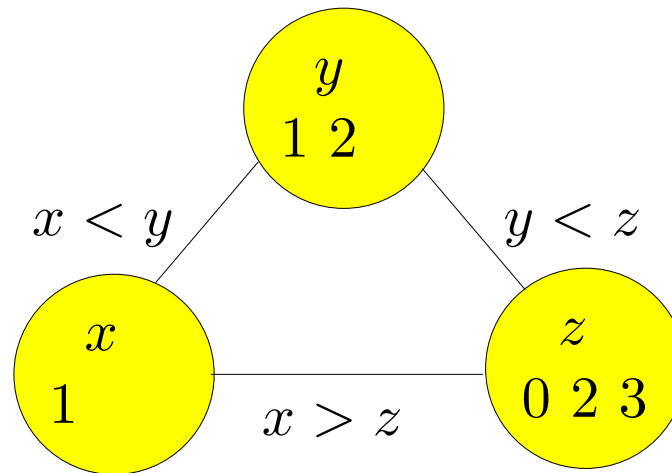
Directional Arc Consistency

- Consider CSP with vars $x \prec y \prec z$, domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



Directional Arc Consistency

- Consider CSP with vars $x \prec y \prec z$, domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



- Inconsistency is not detected by DAC!

DAC enforcing

```
procedure DAC( $X, D, C$ )  
  for each  $i := n - 1$  downto 1 do  
    for each  $c_{ij}$  s.t.  $x_i \prec x_j$  do Revise( $i, j$ )  
endprocedure
```

- Only one pass is required
- Once x_i is made arc-consistent with respect to $x_j \succ x_i$, removing values from $x_k \prec x_i$ does not destroy the arc-consistency of x_i wrt. x_j
- Time complexity: $O(e \cdot m^2)$

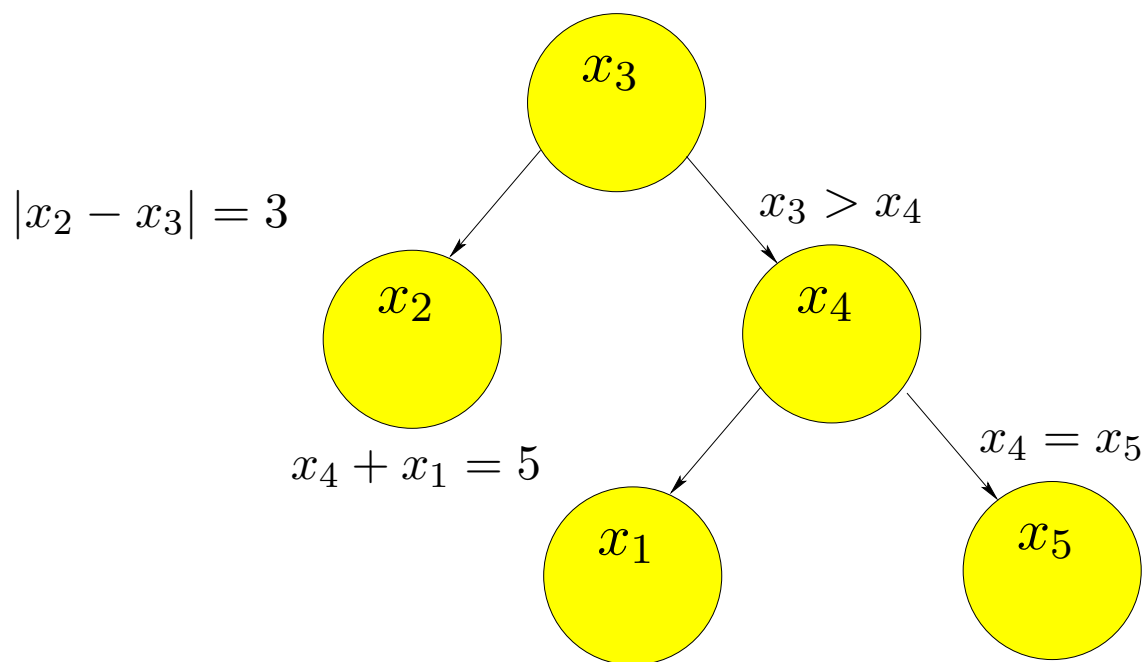
DAC and Acyclic CSP's

- A CSP is **acyclic** if its interaction graph has no loops
- **Theorem.** Acyclic CSP's can be solved in time $O(e \cdot m^2)$
- To prove this theorem we need some ingredients.
- Given an acyclic graph, i.e. a forest,
let us choose a root for each tree and orient edges away from the roots
- Given a directed acyclic graph $G = (V, E)$,
a **topological ordering** is a sequence of all vertices in V s.t.
if $(u, v) \in E$ then u comes before v in the sequence

DAC and Acyclic CSP's

- Consider a CSP with 5 integer vars with domain $[1, 5]$, and constraints $|x_2 - x_3| = 3$, $x_3 > x_4$, $x_4 + x_1 = 5$, $x_4 = x_5$

This CSP is **acyclic**, as its interaction graph is:



- ◆ $(x_3, x_4, x_5, x_1, x_2)$ is a topological ordering
- ◆ $(x_3, x_2, x_4, x_1, x_5)$ is a topological ordering
- ◆ $(x_3, x_2, x_1, x_4, x_5)$ is **not** a topological ordering

DAC and Acyclic CSP's

- **Theorem.** Acyclic CSP's can be solved in time $O(e \cdot m^2)$

Proof. Consider the following algorithm:

// The graph of the CSP (X, D, C) is a tree rooted at x_1

// (x_1, x_2, \dots, x_n) is a topological ordering

procedure AcyclicSolver(X, D, C)

$(X, D, C) := DAC(X, D, C)$ // enforce DAC wrt. the ordering

$a :=$ any element from d_1

$\mu := (x_1 \mapsto a)$

for each $i := 2$ **to** n **do**

 // Any non-root node x_i of the tree has a parent

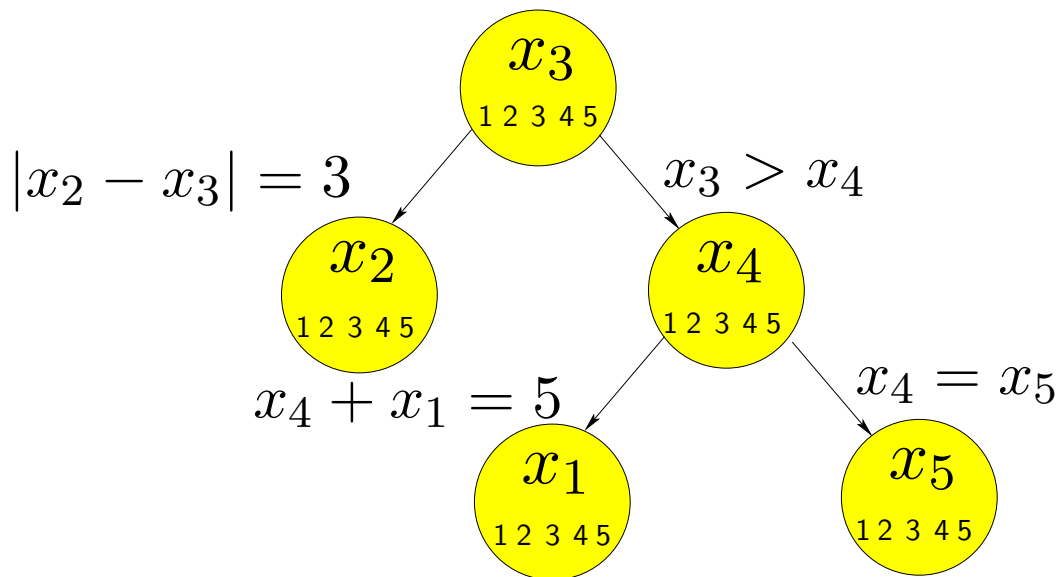
$x_j := \text{parent}(x_i)$ // x_j already assigned due to topological ordering

$v :=$ any support of $\mu(x_j)$ from d_i // \exists because DAC

$\mu := \mu \circ (x_i \mapsto v)$

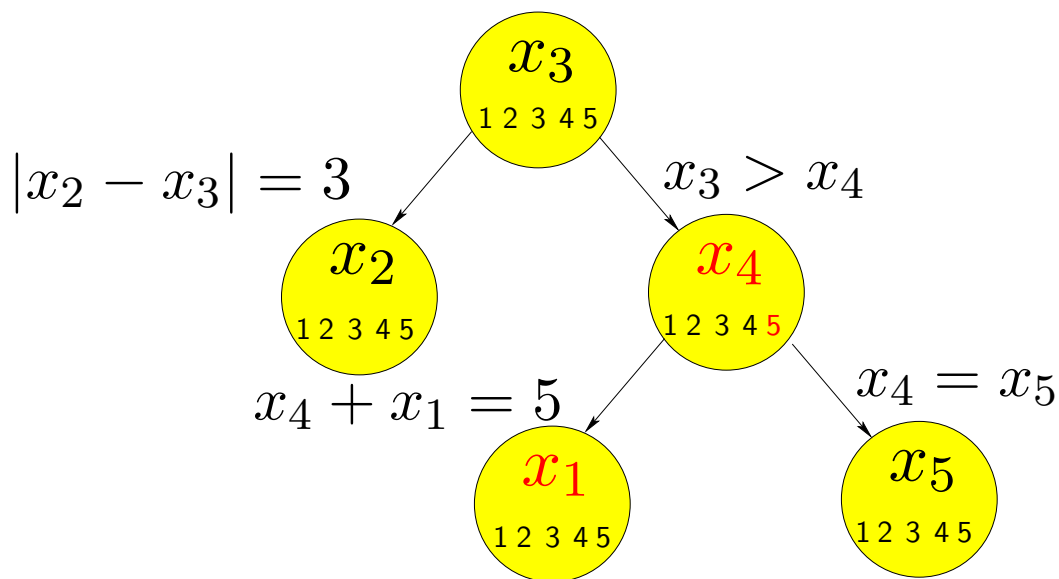
DAC and Acyclic CSP's

- Consider a CSP with 5 integer vars with domain $[1, 5]$, and constraints $|x_2 - x_3| = 3$, $x_3 > x_4$, $x_4 + x_1 = 5$, $x_4 = x_5$
- Let us take the topological ordering $(x_3, x_4, x_5, x_1, x_2)$
- First we enforce DAC



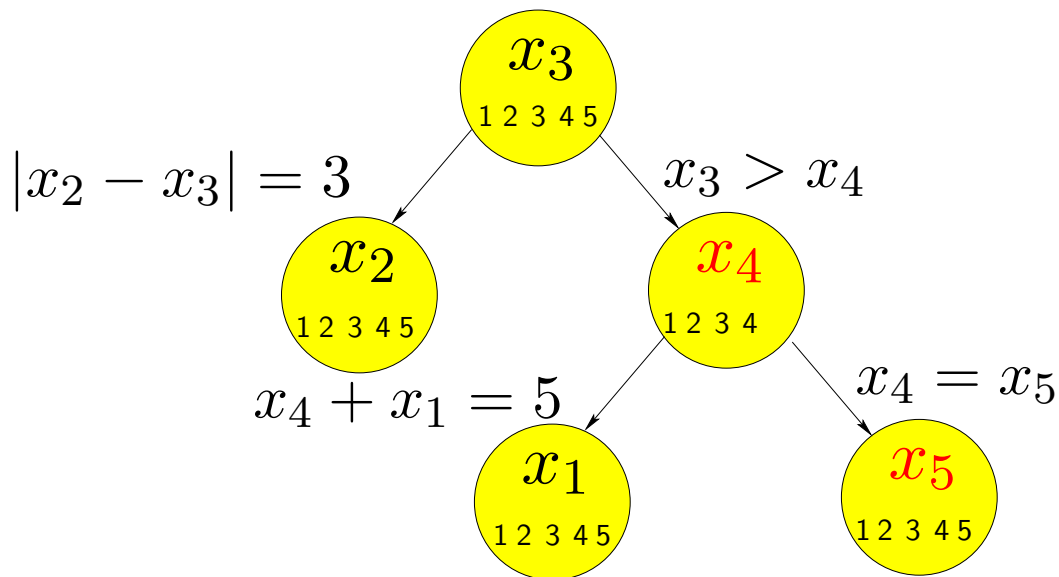
DAC and Acyclic CSP's

- Consider a CSP with 5 integer vars with domain $[1, 5]$, and constraints $|x_2 - x_3| = 3$, $x_3 > x_4$, $x_4 + x_1 = 5$, $x_4 = x_5$
- Let us take the topological ordering $(x_3, x_4, x_5, x_1, x_2)$
- First we enforce DAC



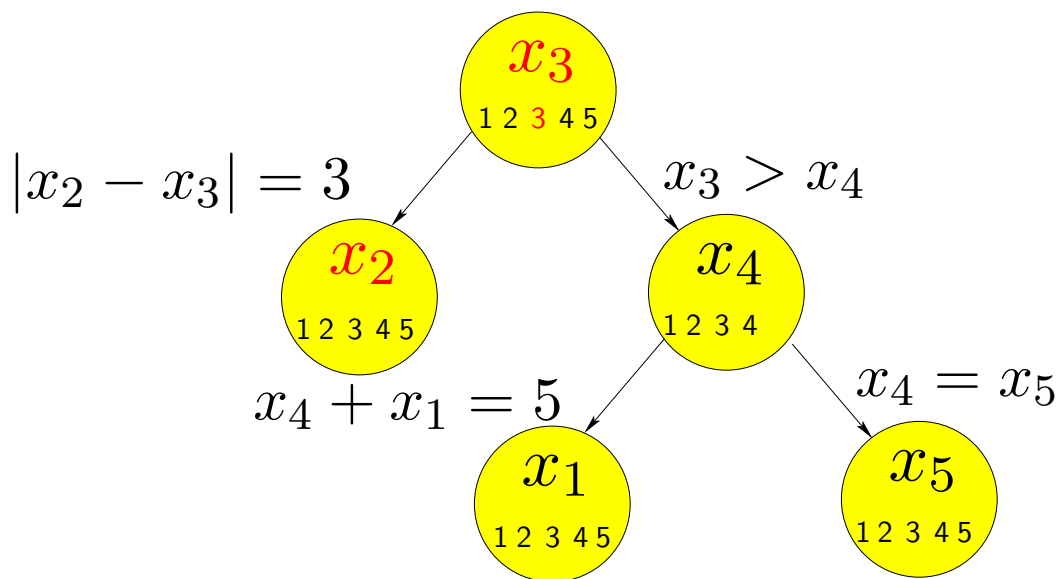
DAC and Acyclic CSP's

- Consider a CSP with 5 integer vars with domain $[1, 5]$, and constraints $|x_2 - x_3| = 3$, $x_3 > x_4$, $x_4 + x_1 = 5$, $x_4 = x_5$
- Let us take the topological ordering $(x_3, x_4, x_5, x_1, x_2)$
- First we enforce DAC



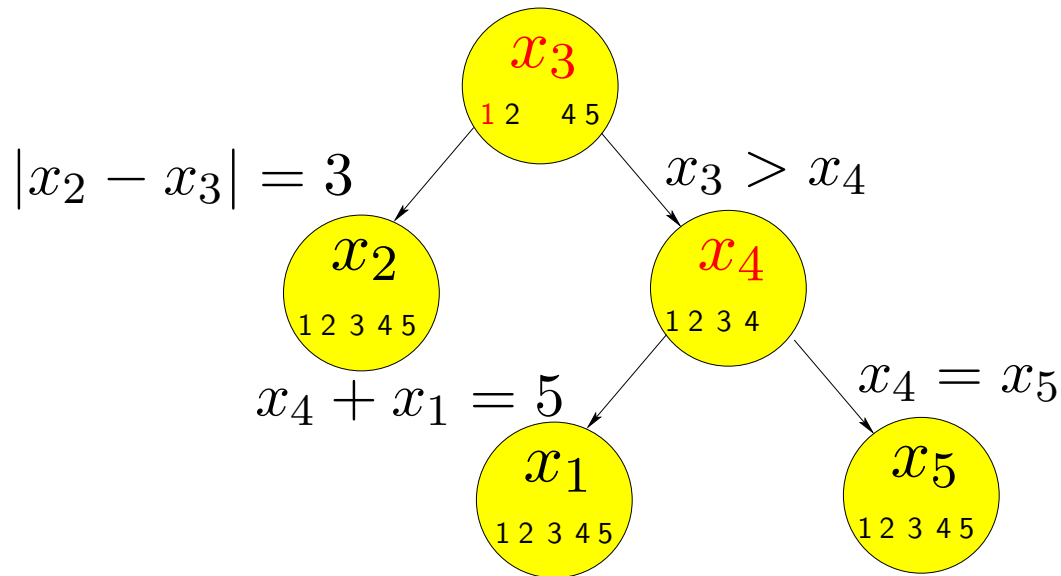
DAC and Acyclic CSP's

- Consider a CSP with 5 integer vars with domain $[1, 5]$, and constraints $|x_2 - x_3| = 3$, $x_3 > x_4$, $x_4 + x_1 = 5$, $x_4 = x_5$
- Let us take the topological ordering $(x_3, x_4, x_5, x_1, x_2)$
- First we enforce DAC



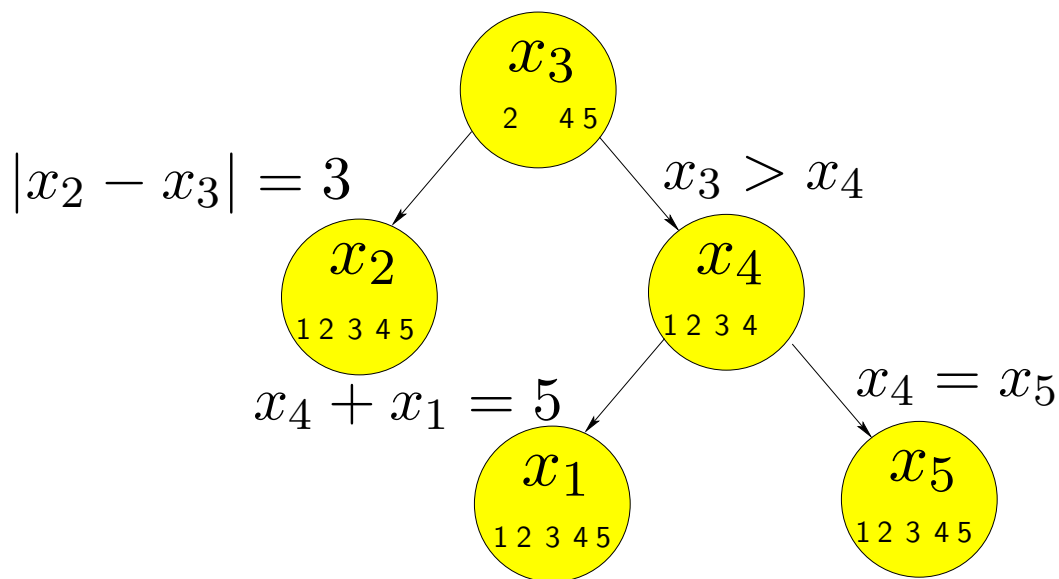
DAC and Acyclic CSP's

- Consider a CSP with 5 integer vars with domain $[1, 5]$, and constraints $|x_2 - x_3| = 3$, $x_3 > x_4$, $x_4 + x_1 = 5$, $x_4 = x_5$
- Let us take the topological ordering $(x_3, x_4, x_5, x_1, x_2)$
- First we enforce DAC



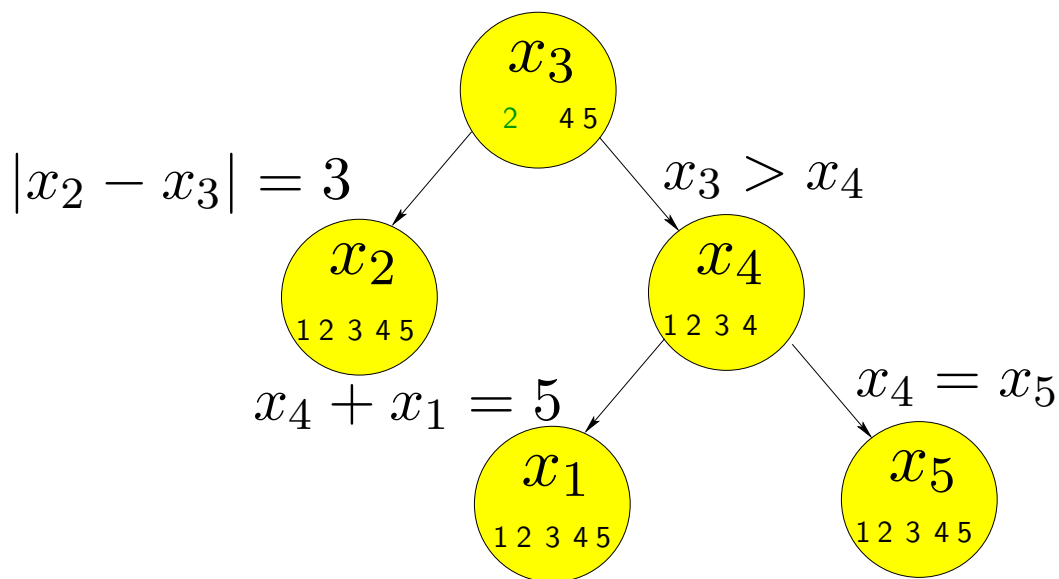
DAC and Acyclic CSP's

- Consider a CSP with 5 integer vars with domain $[1, 5]$, and constraints $|x_2 - x_3| = 3$, $x_3 > x_4$, $x_4 + x_1 = 5$, $x_4 = x_5$
- Let us take the topological ordering $(x_3, x_4, x_5, x_1, x_2)$
- First we enforce DAC



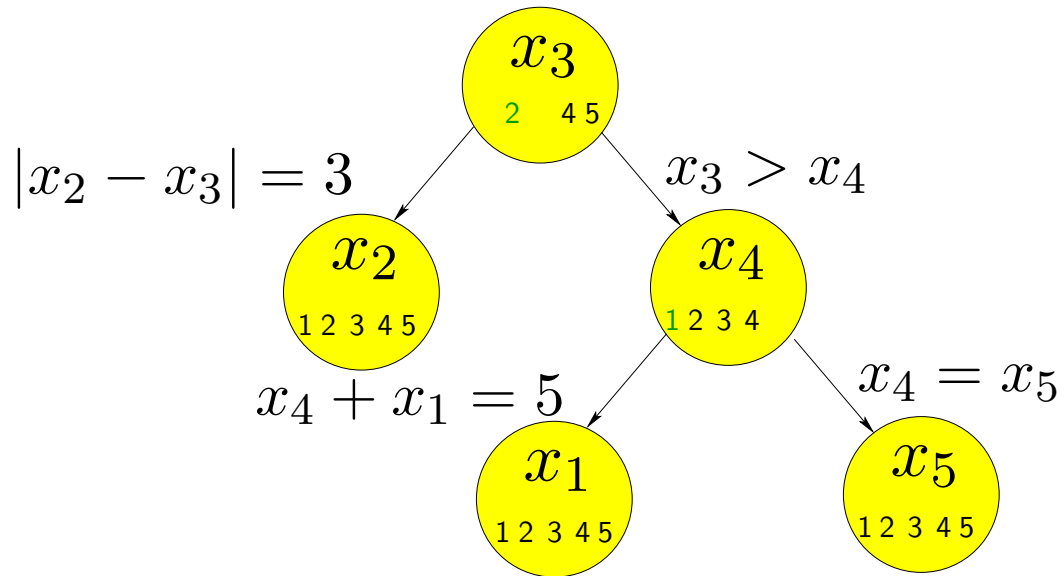
DAC and Acyclic CSP's

- Consider a CSP with 5 integer vars with domain $[1, 5]$, and constraints $|x_2 - x_3| = 3$, $x_3 > x_4$, $x_4 + x_1 = 5$, $x_4 = x_5$
- Let us take the topological ordering $(x_3, x_4, x_5, x_1, x_2)$
- Second we build the assignment



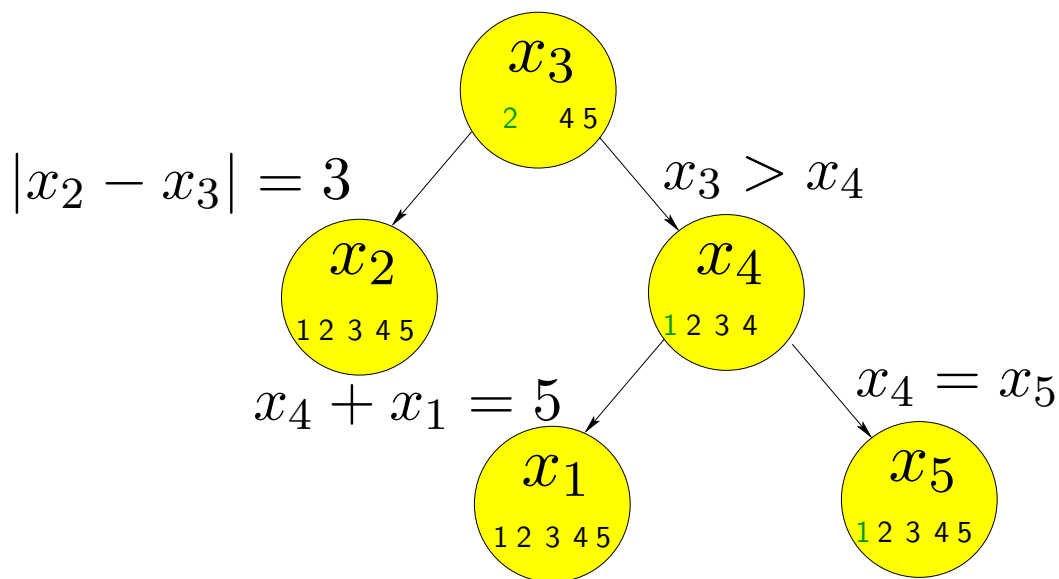
DAC and Acyclic CSP's

- Consider a CSP with 5 integer vars with domain $[1, 5]$, and constraints $|x_2 - x_3| = 3$, $x_3 > x_4$, $x_4 + x_1 = 5$, $x_4 = x_5$
- Let us take the topological ordering $(x_3, x_4, x_5, x_1, x_2)$
- Second we build the assignment



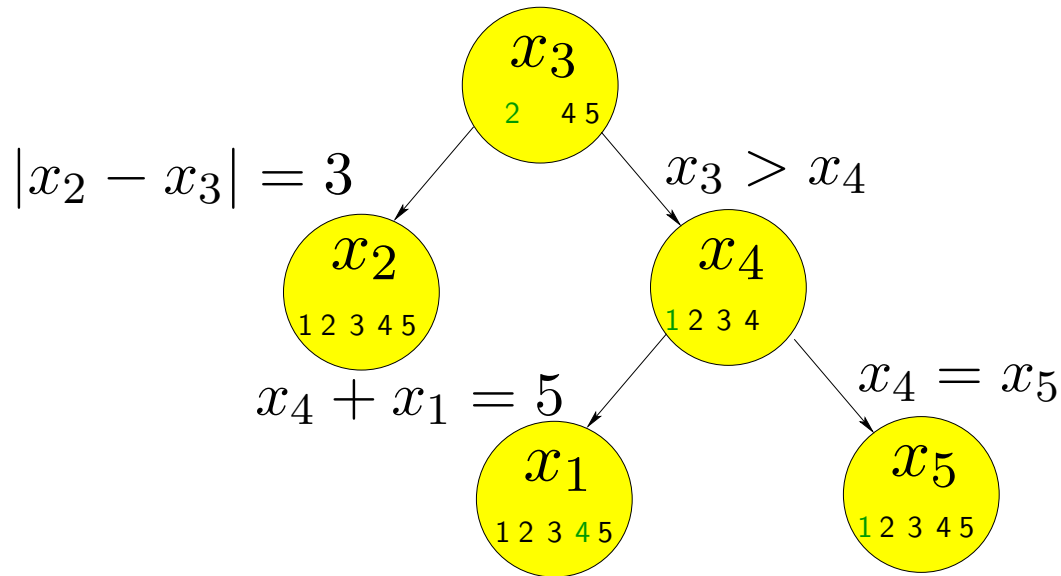
DAC and Acyclic CSP's

- Consider a CSP with 5 integer vars with domain $[1, 5]$, and constraints $|x_2 - x_3| = 3$, $x_3 > x_4$, $x_4 + x_1 = 5$, $x_4 = x_5$
- Let us take the topological ordering $(x_3, x_4, x_5, x_1, x_2)$
- Second we build the assignment



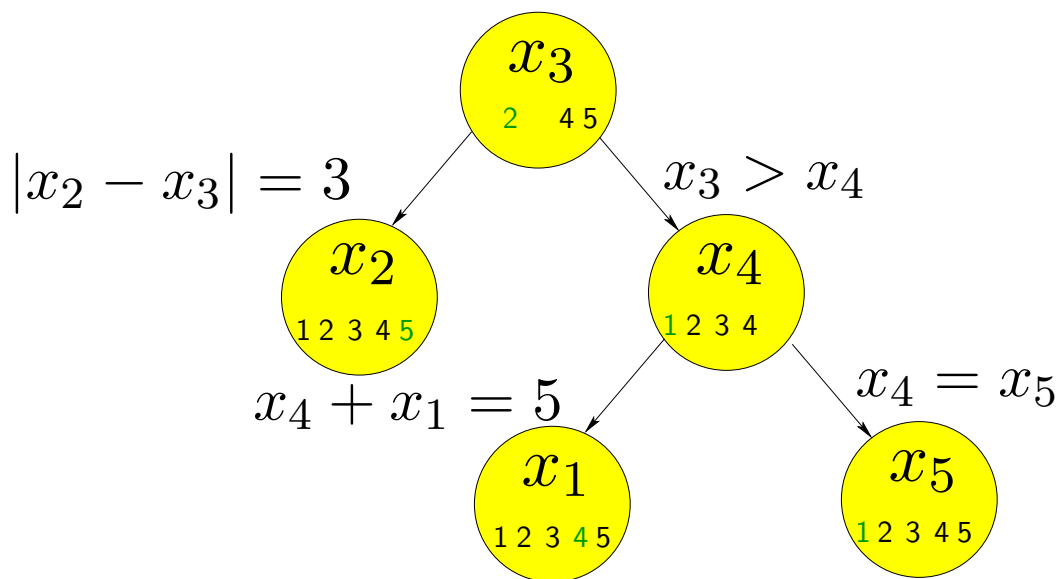
DAC and Acyclic CSP's

- Consider a CSP with 5 integer vars with domain $[1, 5]$, and constraints $|x_2 - x_3| = 3$, $x_3 > x_4$, $x_4 + x_1 = 5$, $x_4 = x_5$
- Let us take the topological ordering $(x_3, x_4, x_5, x_1, x_2)$
- Second we build the assignment



DAC and Acyclic CSP's

- Consider a CSP with 5 integer vars with domain $[1, 5]$, and constraints $|x_2 - x_3| = 3$, $x_3 > x_4$, $x_4 + x_1 = 5$, $x_4 = x_5$
- Let us take the topological ordering $(x_3, x_4, x_5, x_1, x_2)$
- Second we build the assignment

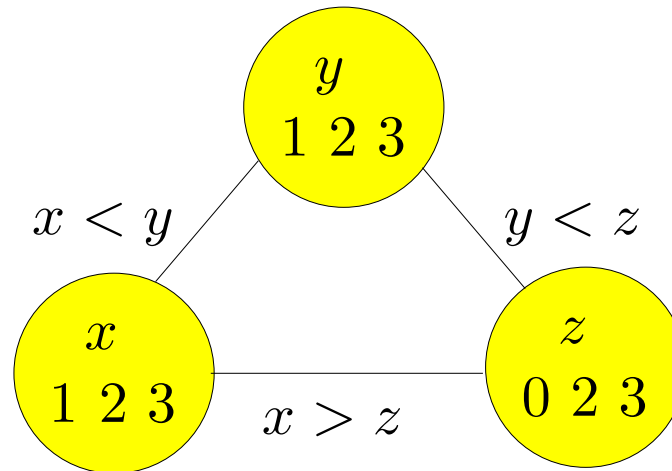


Bounds Consistency

- AC may be too costly when domains are very large
- Idea: Establish an order on domain values and require supports only for the extreme values (i.e., $\min\{d_i\}$ and $\max\{d_i\}$)
- Consider a CSP (X, D, C) with ordered domains
 - ◆ Variable $x_i \in X$ is bounds-consistent wrt. x_j iff $\min\{d_i\}$ and $\max\{d_i\}$ have a support in x_j
 - ◆ Constraint $c_{ij} \in C$ is bounds-consistent iff x_i is bounds-consistent wrt. x_j , and x_j wrt. x_i
 - ◆ The CSP is bounds-consistent iff all its constraints are bounds-consistent
- Notation: BC means bounds-consistent
- BC weaker than AC, but can be enforced more efficiently in practice

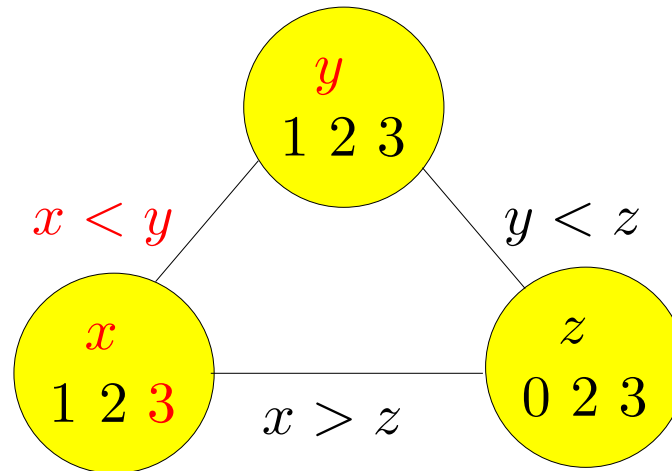
Bounds Consistency

- Let $x, y \in X$ be integer variables with domains $[1, 10]$.
Constraint $|x - y| > 5$ is BC (but not AC nor DAC)
- Consider CSP with vars x, y, z , domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



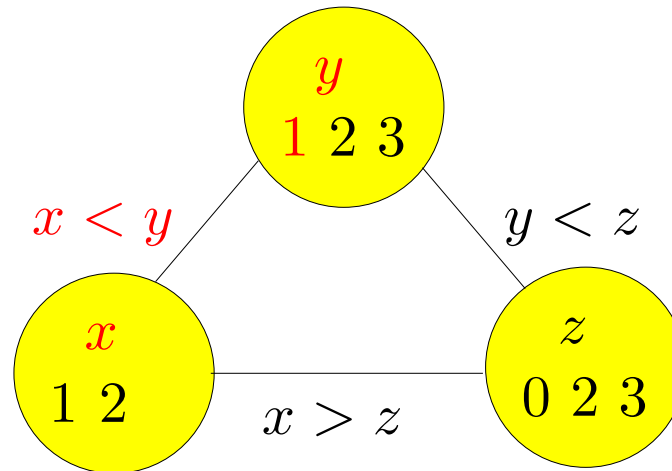
Bounds Consistency

- Let $x, y \in X$ be integer variables with domains $[1, 10]$.
Constraint $|x - y| > 5$ is BC (but not AC nor DAC)
- Consider CSP with vars x, y, z , domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



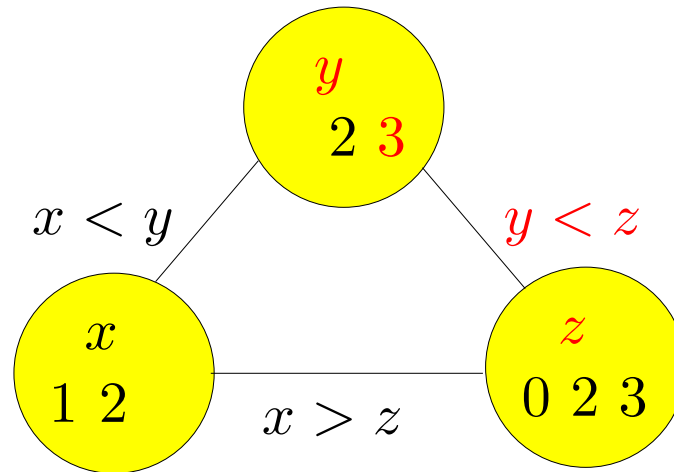
Bounds Consistency

- Let $x, y \in X$ be integer variables with domains $[1, 10]$. Constraint $|x - y| > 5$ is BC (but not AC nor DAC)
- Consider CSP with vars x, y, z , domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



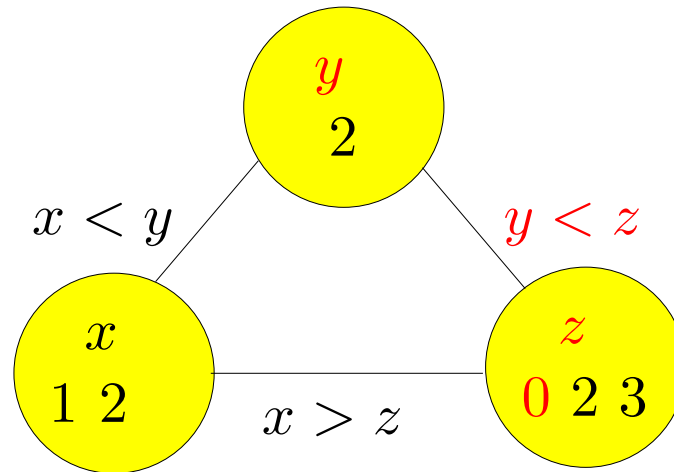
Bounds Consistency

- Let $x, y \in X$ be integer variables with domains $[1, 10]$. Constraint $|x - y| > 5$ is BC (but not AC nor DAC)
- Consider CSP with vars x, y, z , domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



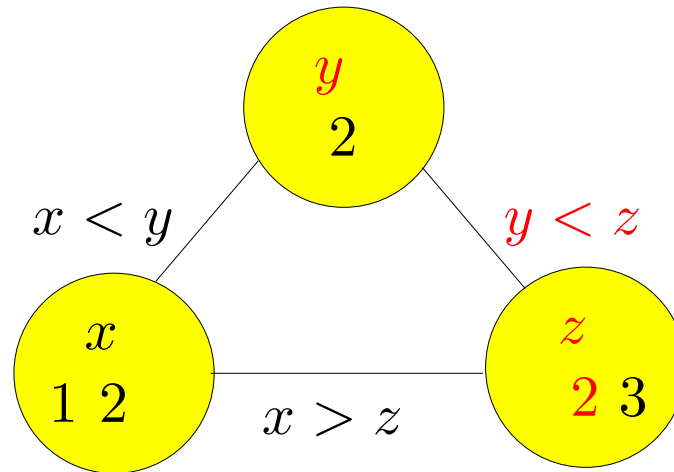
Bounds Consistency

- Let $x, y \in X$ be integer variables with domains $[1, 10]$.
Constraint $|x - y| > 5$ is BC (but not AC nor DAC)
- Consider CSP with vars x, y, z , domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



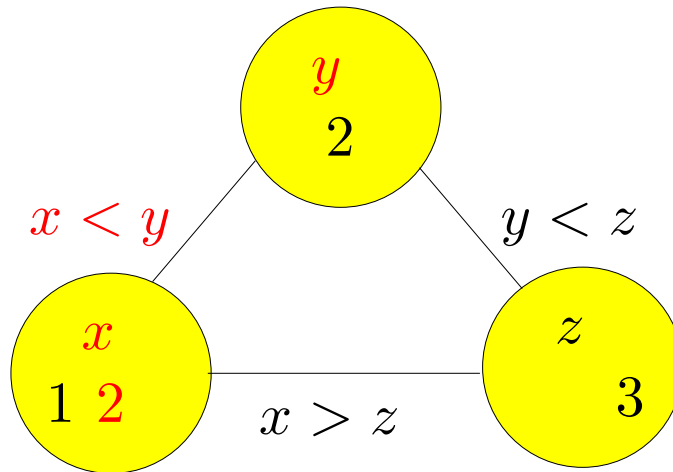
Bounds Consistency

- Let $x, y \in X$ be integer variables with domains $[1, 10]$. Constraint $|x - y| > 5$ is BC (but not AC nor DAC)
- Consider CSP with vars x, y, z , domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



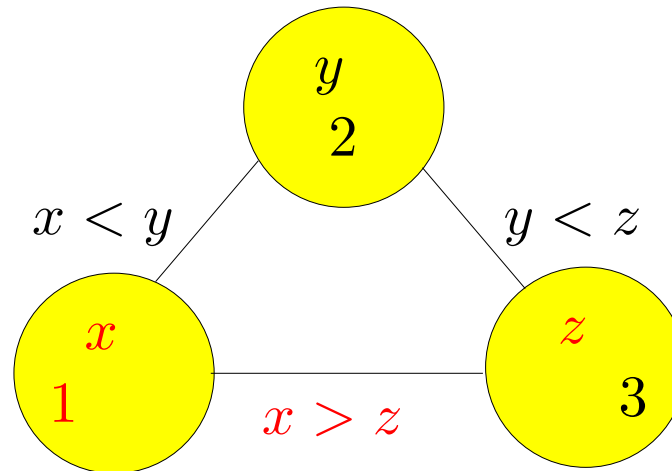
Bounds Consistency

- Let $x, y \in X$ be integer variables with domains $[1, 10]$.
Constraint $|x - y| > 5$ is BC (but not AC nor DAC)
- Consider CSP with vars x, y, z , domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



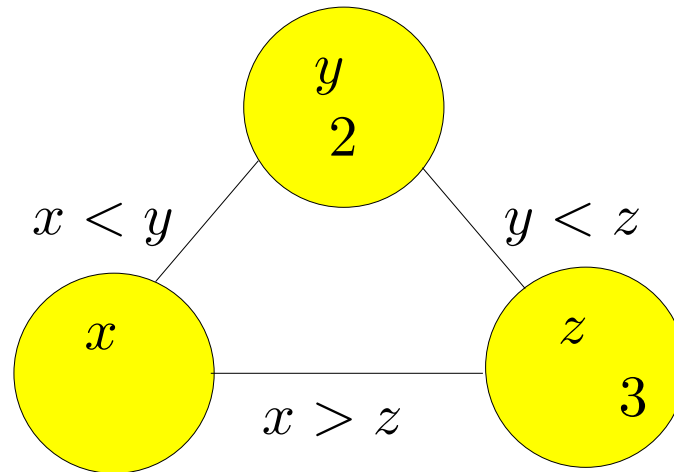
Bounds Consistency

- Let $x, y \in X$ be integer variables with domains $[1, 10]$. Constraint $|x - y| > 5$ is BC (but not AC nor DAC)
- Consider CSP with vars x, y, z , domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



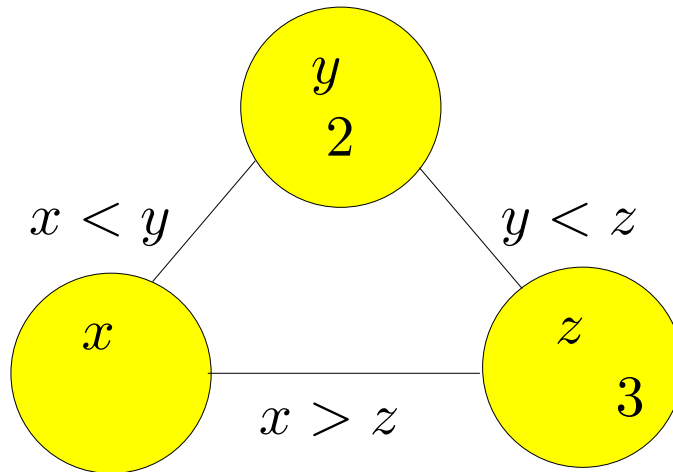
Bounds Consistency

- Let $x, y \in X$ be integer variables with domains $[1, 10]$.
Constraint $|x - y| > 5$ is BC (but not AC nor DAC)
- Consider CSP with vars x, y, z , domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



Bounds Consistency

- Let $x, y \in X$ be integer variables with domains $[1, 10]$. Constraint $|x - y| > 5$ is BC (but not AC nor DAC)
- Consider CSP with vars x, y, z , domains $d_x = d_y = \{1, 2, 3\}$ and $d_z = \{0, 2, 3\}$, and constraints $x < y$, $y < z$, $x > z$



- The examples show that DAC and BC are incomparable (and both weaker than AC)

BC-3: ReviseBounds(i, j)

- Natural adaptation of AC-3 to bounds consistency: **BC-3**
- Based on function **ReviseBounds(i, j)**,
which removes values from the extremes of d_i without support in d_j
Returns **true** if some value is removed

```
function ReviseBounds( $i, j$ )  
    change := false  
    while  $|d_i| \neq \emptyset \wedge (\forall b \in d_j \neg c_{ij}(\min\{d_i\}, b))$  do  
        change := true  
        remove  $\min\{d_i\}$  from  $d_i$   
    while  $|d_i| \neq \emptyset \wedge \forall b \in d_j \neg c_{ij}(\max\{d_i\}, b)$  do  
        change := true  
        remove  $\max\{d_i\}$  from  $d_i$   
    return change
```

- The time complexity of **ReviseBounds(i, j)** is $O(|d_i| \cdot |d_j|)$

BC-3

// $(i, j) \in Q$ means

“cannot guarantee $\min\{d_i\}$, $\max\{d_i\}$ have support in d_j ”

procedure BC3(X, D, C)

$Q := \{(i, j), (j, i) \mid c_{ij} \in C\}$ // each constraint is added twice

while $Q \neq \emptyset$ **do**

$(i, j) := \text{Fetch}(Q)$

if ReviseBounds(i, j) **then**

$Q := Q \cup \{(k, i) \mid c_{ki} \in C, k \neq j\}$

- Space complexity: $O(e)$
- Time complexity: $O(e \cdot m^3)$
- Same asymptotic costs of AC-3, but BC-3 is more efficient in practice

Stronger than AC

- Singleton AC (SAC)
- Neighborhood Inverse Consistency (NIC)

Singleton AC

- Let $AC(P)$ denote the CSP resulting from enforcing AC on P
- If $AC(P[x_i \rightarrow a])$ has an empty domain, then $P[x_i \rightarrow a]$ does not have any solution, and $a \in d_i$ is unfeasible in P
- A CSP $P = (X, D, C)$ is **singleton arc-consistent (SAC)** iff $\forall d_i \in D, \forall a \in d_i$, problem $AC(P[x_i \rightarrow a])$ has no empty domains

Singleton AC

- Let us enforce SAC in the 4-queens problem

Singleton AC

- Let us enforce SAC in the 4-queens problem

<i>Q</i>	<i>X</i>	<i>X</i>	<i>X</i>
<i>X</i>	<i>X</i>		
<i>X</i>		<i>X</i>	
<i>X</i>			<i>X</i>

Singleton AC

- Let us enforce SAC in the 4-queens problem

<i>Q</i>	<i>X</i>	<i>X</i>	<i>X</i>
<i>X</i>	<i>X</i>		
<i>X</i>	<i>Q</i>	<i>X</i>	
<i>X</i>	<i>X</i>	<i>X</i>	<i>X</i>

Singleton AC

- Let us enforce SAC in the 4-queens problem

<i>Q</i>	<i>X</i>	<i>X</i>	<i>X</i>
<i>X</i>	<i>X</i>	<i>X</i>	<i>X</i>
<i>X</i>		<i>X</i>	<i>Q</i>
<i>X</i>			<i>X</i>

Singleton AC

- Let us enforce SAC in the 4-queens problem

<i>X</i>	<i>X</i>	<i>X</i>	
<i>X</i>	<i>Q</i>	<i>X</i>	<i>X</i>
<i>X</i>	<i>X</i>	<i>X</i>	
	<i>X</i>		<i>X</i>

Singleton AC

- Let us enforce SAC in the 4-queens problem

<i>X</i>	<i>X</i>	<i>X</i>	<i>Q</i>
<i>X</i>	<i>Q</i>	<i>X</i>	<i>X</i>
<i>X</i>	<i>X</i>	<i>X</i>	<i>X</i>
	<i>X</i>		<i>X</i>

Singleton AC

- Let us enforce SAC in the 4-queens problem

<i>X</i>			<i>X</i>
	<i>X</i>	<i>X</i>	
	<i>X</i>	<i>X</i>	
<i>X</i>			<i>X</i>

by symmetry

Enforcing SAC

```
procedure SAC( $P$ )  
   $P := AC(P)$   
  repeat  
     $\text{change} := \text{false}$   
    for each  $d_i \in D, a \in d_i$  do  
      if  $AC(P[x_i \rightarrow a])$  has an empty domain then  
         $\text{change} := \text{true}$   
        remove  $a$  from  $d_i$   
    if  $\text{change}$  then  
       $P := AC(P)$   
  until  $\neg \text{change}$ 
```

- Complexity: $O(e \cdot n^2 \cdot m^4)$

Neighborhood Inverse Consistency

- Let $P = (X, D, C)$ be a CSP.
- The **neighborhood** of $x_i \in X$, noted N_i , is the set of vars containing x_i and all x_j such that $c_{ij} \in C$
- The **projection** of P on N_i , noted $P[N_i]$, is the problem obtained from P by taking all variables in N_i and all constraints c such that $\text{scope}(c) \subseteq N_i$
- If $a \in d_i$ is unfeasible in $P[N_i]$, then so is in P
- A CSP $P = (X, D, C)$ is **neighborhood inverse consistent (NIC)** iff for every $x_i \in X$, $a \in d_i$, a is feasible in $P[N_i]$

Enforcing NIC

```
procedure NIC( $P$ )  
   $P := AC(P)$   
  repeat  
    change := false  
    for each  $d_i \in D, a \in d_i$  do  
      if  $SOL(P[N_i][x_i \rightarrow a]) = \emptyset$  then  
        change := true  
        remove  $a$  from  $d_i$   
    endfunction  
  until  $\neg$  change
```

- Complexity: $O(g^2 \cdot m^g \cdot n^2 \cdot m^2)$,
where g is the degree of the interaction graph
(i.e, the max number of neighbors that some vertex has)