Foundations of Artificial Intelligence

D5. Constraint Satisfaction Problems: Arc Consistency

Malte Helmert

University of Basel

April 13, 2026

Constraint Satisfaction Problems: Overview

Chapter overview: constraint satisfaction problems

- D1–D2. Introduction
- D3–D6. Basic Algorithms
  - D3. Backtracking
  - D4. Inference and Forward Checking
  - D5. Arc Consistency
  - D6. Path Consistency
- D7–D8. Problem Structure

Arc Consistency

Arc Consistency: Motivation

- forward checking removes values from variables
  that don't work together with values
  that we already assigned to other variables
- can take this idea further:
  for all pairs of variables v and v', remove values of v
  that cannot work together with any value of v'
- can prune more values than forward checking
  $\rightsquigarrow$ more inference $\rightsquigarrow$ less search

$\rightsquigarrow$ arc consistency

Arc Consistency: Definition

Definition (Arc Consistent)

Let $\mathcal{C} = \langle V, \mathrm{dom}, (R_{uv}) \rangle$ be a constraint network.

1. The variable $v \in V$ is arc consistent
   with respect to another variable $v' \in V$,
   if for every value $d \in \mathrm{dom}(v)$
   there exists a value $d' \in \mathrm{dom}(v')$ with $\langle d, d' \rangle \in R_{vv'}$.
2. The constraint network $\mathcal{C}$ is arc consistent,
   if every variable $v \in V$ is arc consistent
   with respect to every other variable $v' \in V$.

German: kantenkonsistent

remarks:

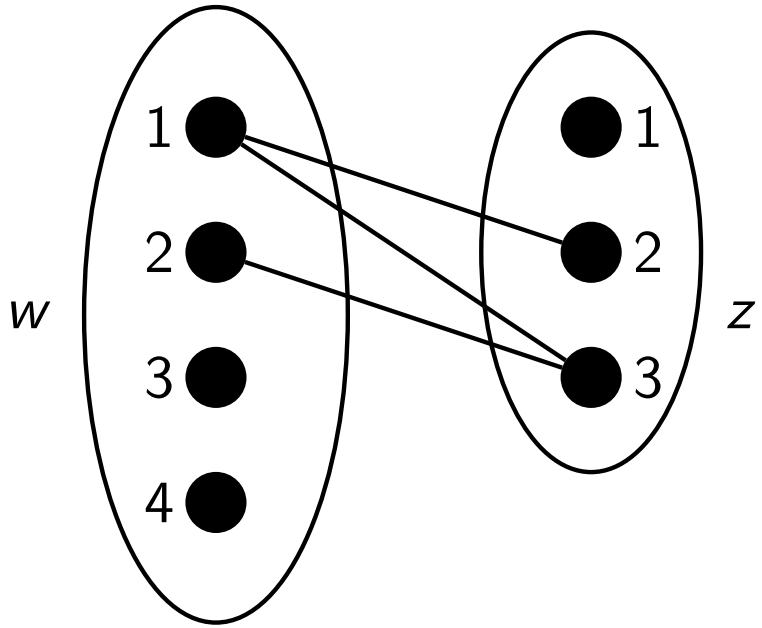
- definition for variable pair is not symmetrical
- v always arc consistent with respect to v'
  if the constraint between v and v' is trivial

Arc Consistency: Example

Running Example

Consider variables w and z from our running example:

- $\text{dom}(w) = \{1, 2, 3, 4\}$
- $\text{dom}(z) = \{1, 2, 3\}$
- $R_{wz} = \{\langle 1, 2\rangle, \langle 1, 3\rangle, \langle 2, 3\rangle\}$



Arc consistency
of w with respect to z and
of z with respect to w
is violated.

Enforcing Arc Consistency

- Enforcing arc consistency, i.e., removing values from $\mathrm{dom}(v)$ that violate the arc consistency of v with respect to v', is a correct inference method. (Why?)
- more powerful than forward checking (Why?)

Enforcing Arc Consistency

- Enforcing arc consistency, i.e., removing values from $\mathrm{dom}(v)$ that violate the arc consistency of v with respect to v', is a correct inference method. (Why?)
- more powerful than forward checking (Why?)
  - $\rightsquigarrow$ Forward checking is a special case: enforcing arc consistency of all variables with respect to the just assigned variable corresponds to forward checking.

We will next consider algorithms that enforce arc consistency.

Revise

Processing Variable Pairs: revise

function revise($\mathcal{C}, v, v'$):

$\langle V, \mathrm{dom}, (R_{uv}) \rangle := \mathcal{C}$

for each $d \in \mathrm{dom}(v)$:

  if there is no $d' \in \mathrm{dom}(v')$ with $\langle d, d' \rangle \in R_{vv'}$:

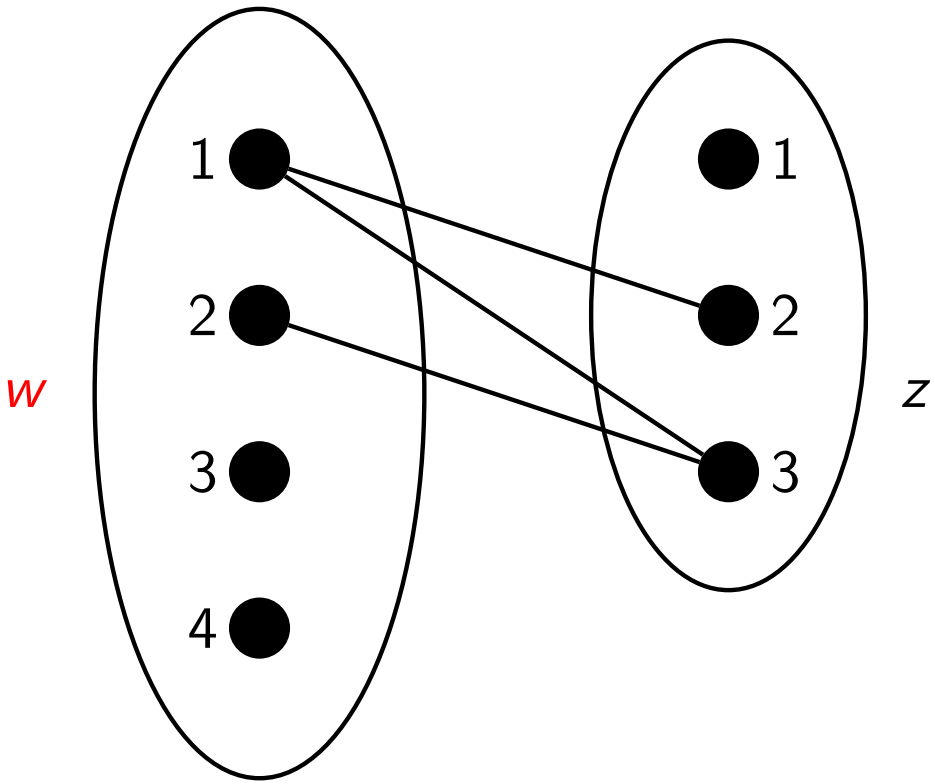
    remove d from $\mathrm{dom}(v)$

input: constraint network $\mathcal{C}$ and two variables v, v' of $\mathcal{C}$

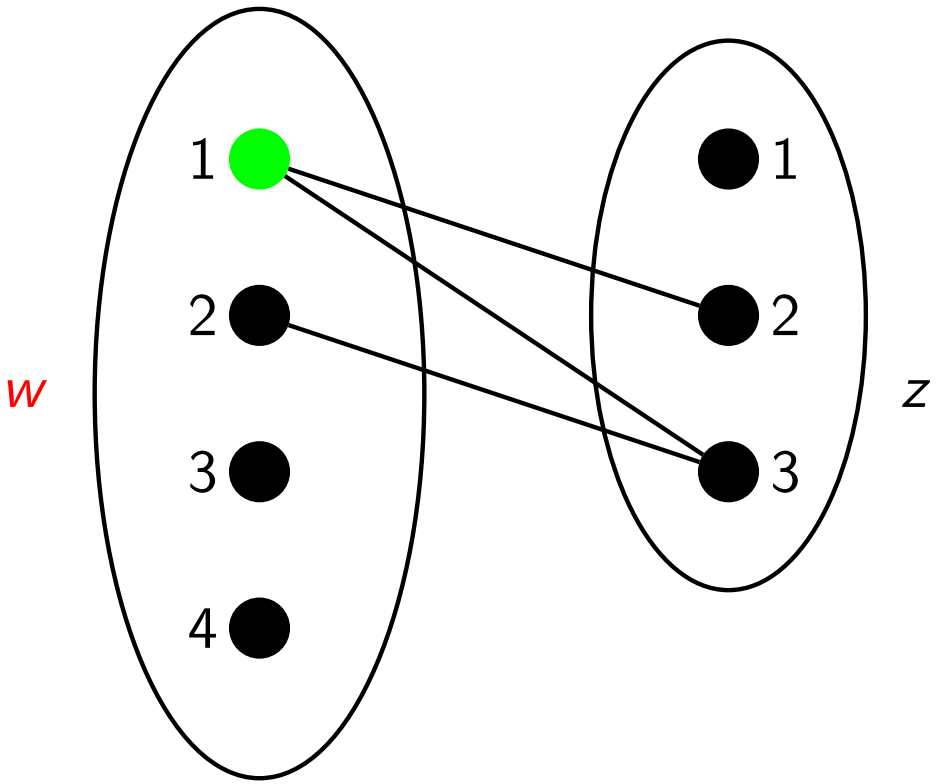
effect: v arc consistent with respect to v'.
All violating values in $\mathrm{dom}(v)$ are removed.

time complexity: $O(k^2)$, where k is maximal domain size

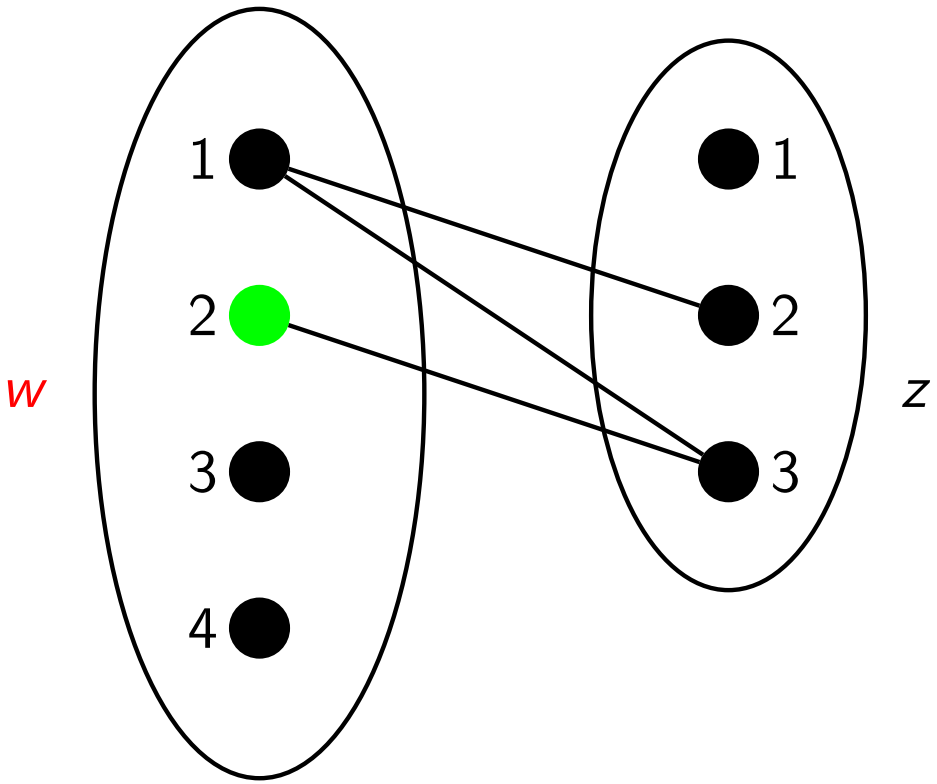
revise($\mathcal{C}, w, z$) in Running Example



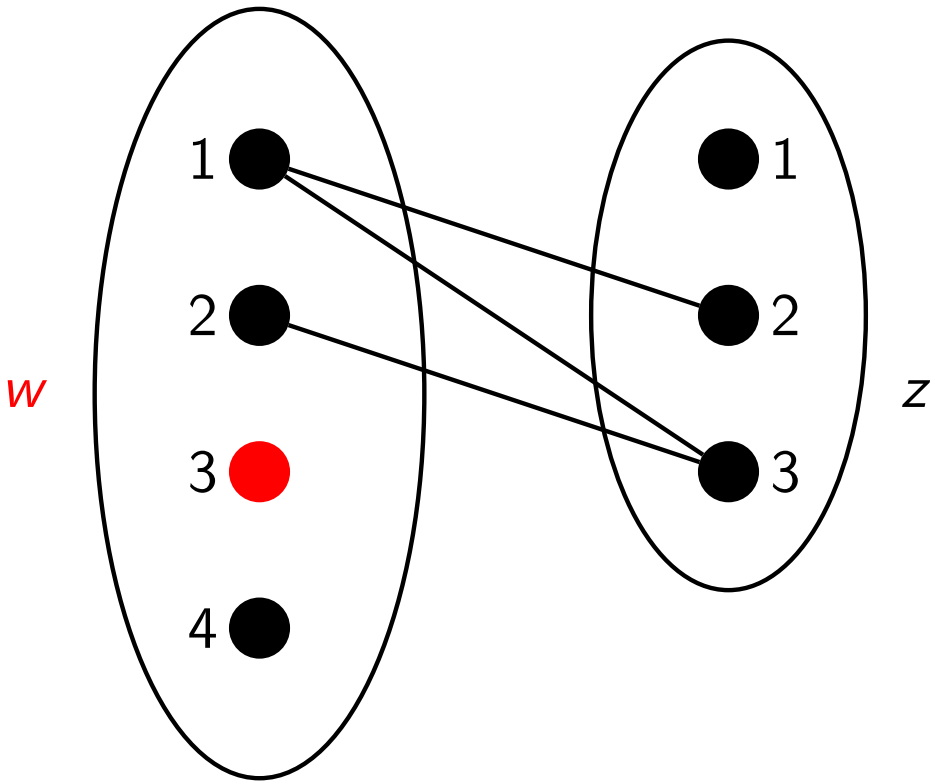
revise($\mathcal{C}, w, z$) in Running Example



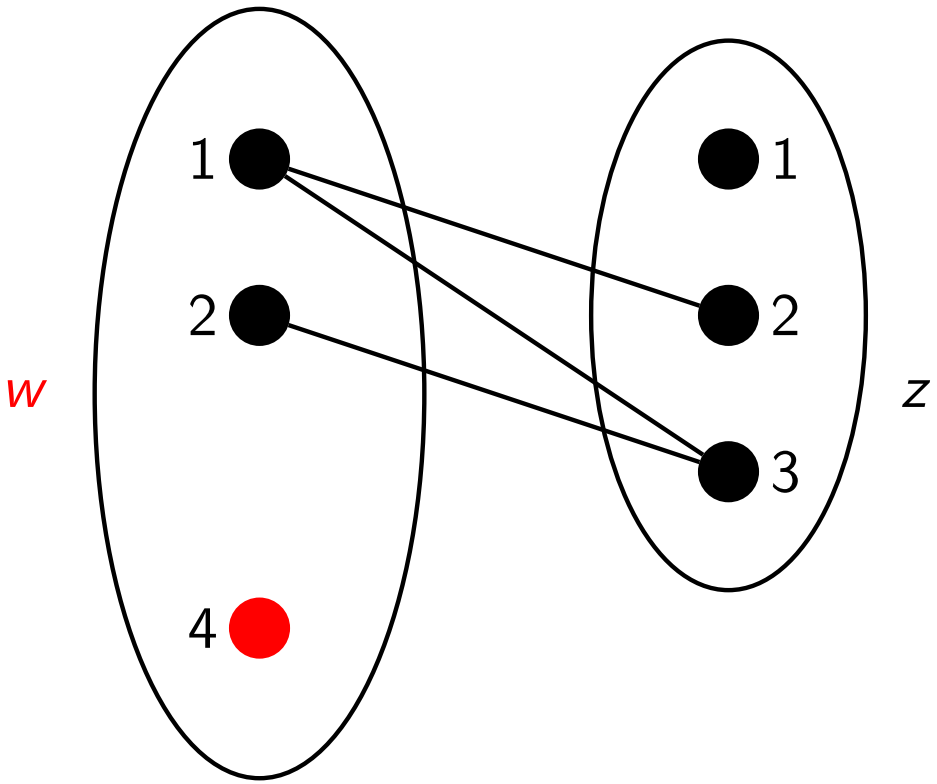
revise($\mathcal{C}, w, z$) in Running Example



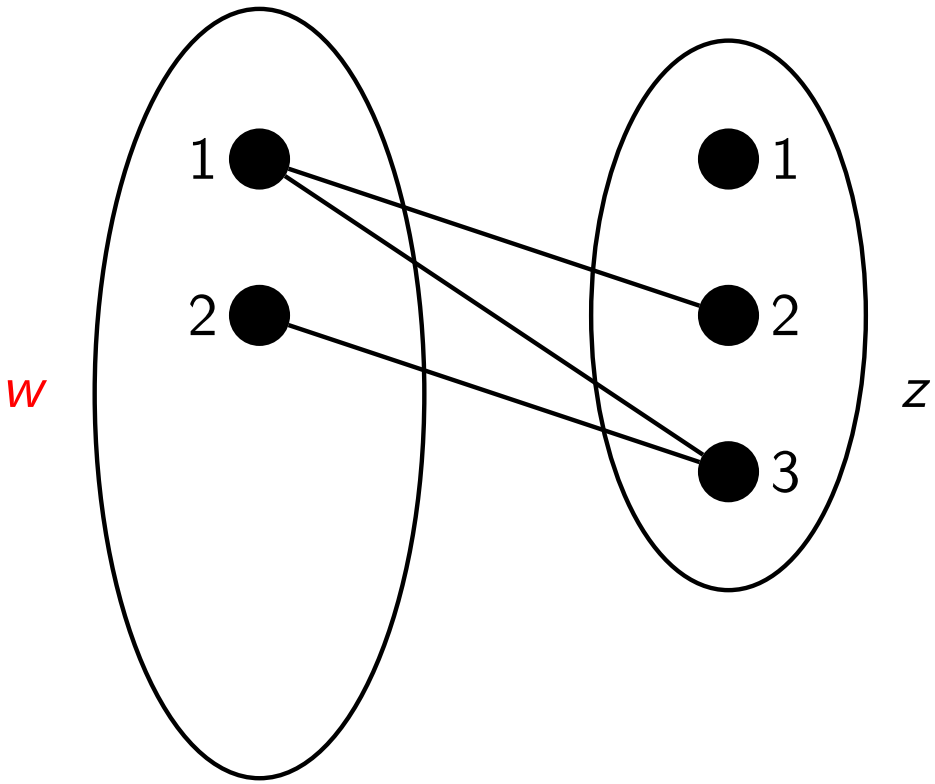
revise($\mathcal{C}, w, z$) in Running Example



revise($\mathcal{C}, w, z$) in Running Example



revise($\mathcal{C}, w, z$) in Running Example



Naive Algorithm: AC-1

Enforcing Arc Consistency: AC-1

function AC-1($\mathcal{C}$):

$\langle V, \mathrm{dom}, (R_{uv}) \rangle := \mathcal{C}$

repeat

  for each nontrivial constraint R_{uv}:

    revise($\mathcal{C}, u, v$)

    revise($\mathcal{C}, v, u$)

until no domain has changed in this iteration

input: constraint network $\mathcal{C}$

effect: transforms $\mathcal{C}$ into equivalent arc consistent network

time complexity: ?

Enforcing Arc Consistency: AC-1

function AC-1($\mathcal{C}$):

$\langle V, \mathrm{dom}, (R_{uv}) \rangle := \mathcal{C}$

repeat

  for each nontrivial constraint R_{uv}:

    revise($\mathcal{C}, u, v$)

    revise($\mathcal{C}, v, u$)

until no domain has changed in this iteration

input: constraint network $\mathcal{C}$

effect: transforms $\mathcal{C}$ into equivalent arc consistent network

time complexity: $O(n \cdot e \cdot k^3)$, with n variables,
e nontrivial constraints and maximal domain size k

AC-1: Discussion

- AC-1 does the job, but is rather inefficient.
- Drawback: Variable pairs are often checked again and again although their domains have remained unchanged.
- These (redundant) checks can be saved.

$\rightsquigarrow$ more efficient algorithm: AC-3

Efficient Algorithm: AC-3

Enforcing Arc Consistency: AC-3

idea: store potentially inconsistent variable pairs in a queue

function AC-3($\mathcal{C}$):

$\langle V, \mathrm{dom}, (R_{uv}) \rangle := \mathcal{C}$

$queue := \emptyset$

for each nontrivial constraint R_{uv}:

    insert $\langle u, v \rangle$ into *queue*

    insert $\langle v, u \rangle$ into *queue*

while $queue \neq \emptyset$:

    remove an arbitrary element $\langle u, v \rangle$ from *queue*

    revise($\mathcal{C}, u, v$)

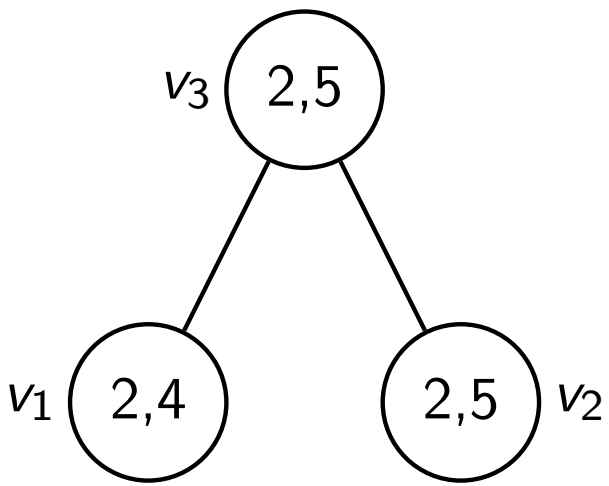
    if dom(u) changed in the call to revise:

        for each $w \in V \setminus \{u, v\}$ where R_{wu} is nontrivial:

            insert $\langle w, u \rangle$ into *queue*

AC-3: Example

Consider a constraint network with three variables v_1, v_2, v_3 with $\mathrm{dom}(v_1) = \{2, 4\}$ and $\mathrm{dom}(v_2) = \mathrm{dom}(v_3) = \{2, 5\}$ and the constraints $v_3|v_1$ and $v_3|v_2$ ("divides exactly").

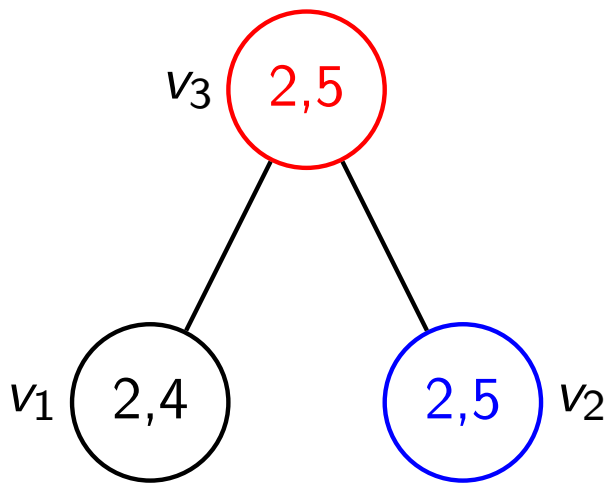


Queue

$\langle v_1, v_3 \rangle$
$\langle v_3, v_1 \rangle$
$\langle v_2, v_3 \rangle$
$\langle v_3, v_2 \rangle$

AC-3: Example

Consider a constraint network with three variables v_1, v_2, v_3 with $\mathrm{dom}(v_1) = \{2, 4\}$ and $\mathrm{dom}(v_2) = \mathrm{dom}(v_3) = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides exactly").

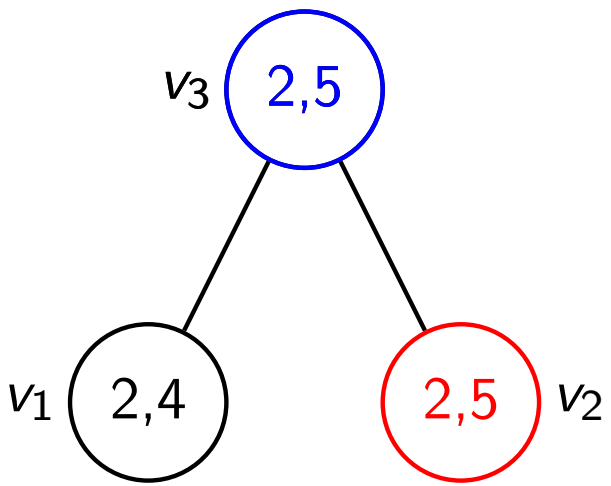


Queue

$\langle v_1, v_3 \rangle$
$\langle v_3, v_1 \rangle$
$\langle v_2, v_3 \rangle$
$\langle v_3, v_2 \rangle$

AC-3: Example

Consider a constraint network with three variables v_1, v_2, v_3 with $\mathrm{dom}(v_1) = \{2, 4\}$ and $\mathrm{dom}(v_2) = \mathrm{dom}(v_3) = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides exactly").

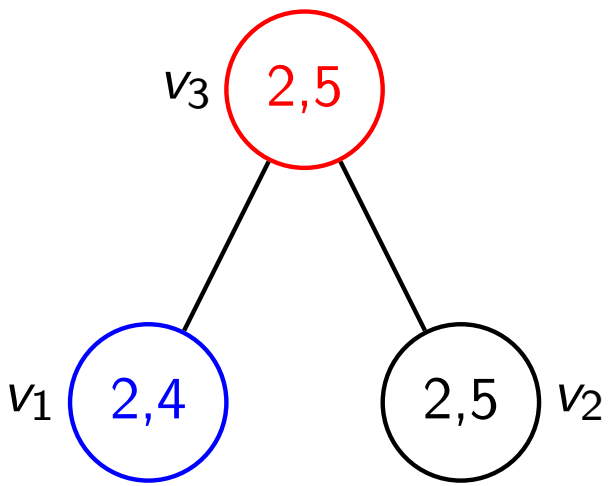


Queue

$\langle v_1, v_3 \rangle$
$\langle v_3, v_1 \rangle$
$\langle v_2, v_3 \rangle$

AC-3: Example

Consider a constraint network with three variables v_1, v_2, v_3 with $\mathrm{dom}(v_1) = \{2, 4\}$ and $\mathrm{dom}(v_2) = \mathrm{dom}(v_3) = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides exactly").

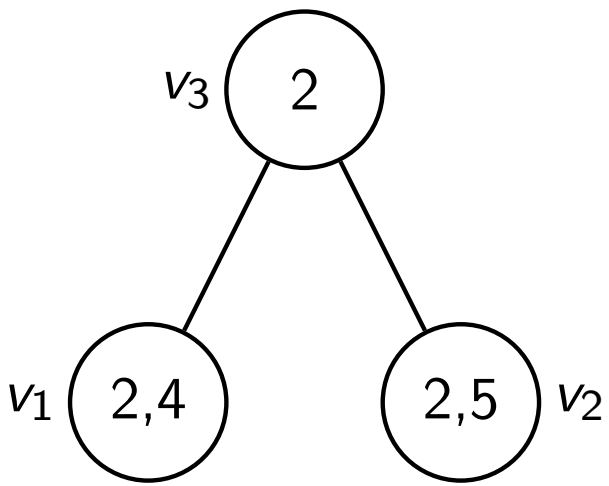


Queue

$\langle v_1, v_3 \rangle$
$\langle v_3, v_1 \rangle$

AC-3: Example

Consider a constraint network with three variables v_1, v_2, v_3 with $\mathrm{dom}(v_1) = \{2, 4\}$ and $\mathrm{dom}(v_2) = \mathrm{dom}(v_3) = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides exactly").

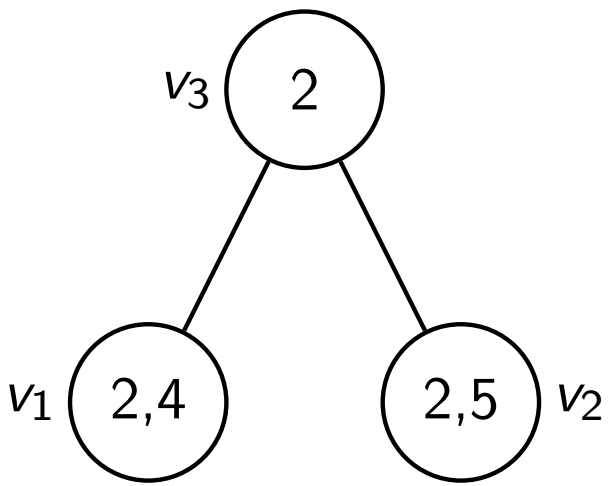


Queue

$\langle v_1, v_3 \rangle$

AC-3: Example

Consider a constraint network with three variables v_1, v_2, v_3 with $\mathrm{dom}(v_1) = \{2, 4\}$ and $\mathrm{dom}(v_2) = \mathrm{dom}(v_3) = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides exactly").



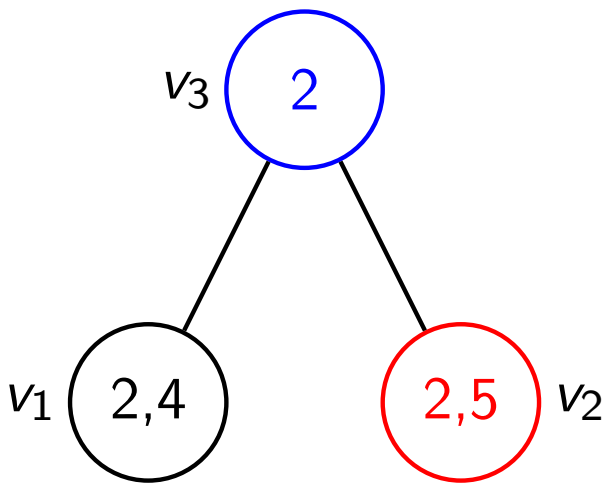
Queue

$\langle v_1, v_3 \rangle$

$\langle v_2, v_3 \rangle$

AC-3: Example

Consider a constraint network with three variables v_1, v_2, v_3 with $\mathrm{dom}(v_1) = \{2, 4\}$ and $\mathrm{dom}(v_2) = \mathrm{dom}(v_3) = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides exactly").



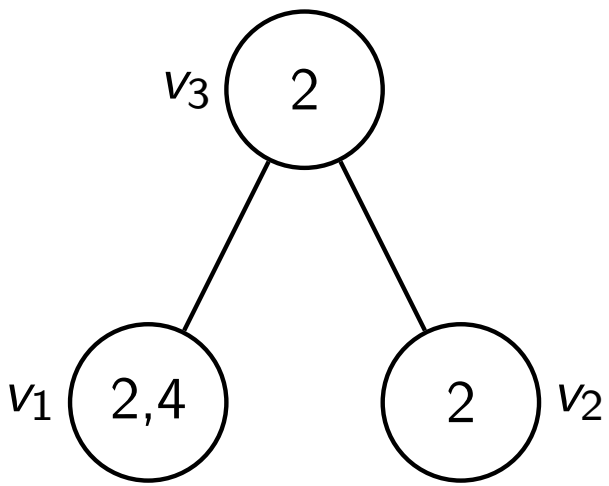
Queue

$\langle v_1, v_3 \rangle$

$\langle v_2, v_3 \rangle$

AC-3: Example

Consider a constraint network with three variables v_1, v_2, v_3 with $\mathrm{dom}(v_1) = \{2, 4\}$ and $\mathrm{dom}(v_2) = \mathrm{dom}(v_3) = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides exactly").

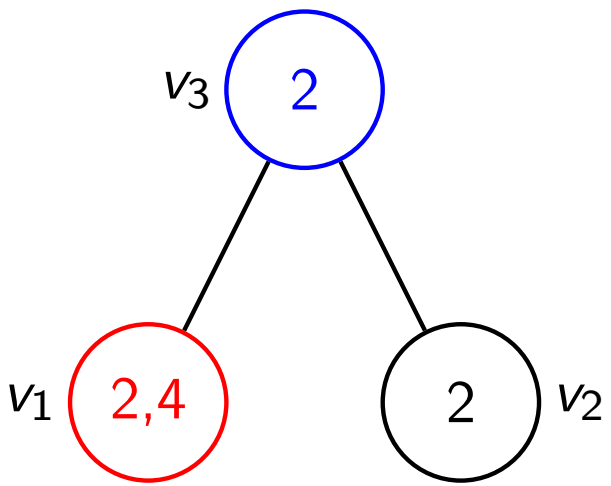


Queue

$\langle v_1, v_3 \rangle$

AC-3: Example

Consider a constraint network with three variables v_1, v_2, v_3 with $\mathrm{dom}(v_1) = \{2, 4\}$ and $\mathrm{dom}(v_2) = \mathrm{dom}(v_3) = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides exactly").

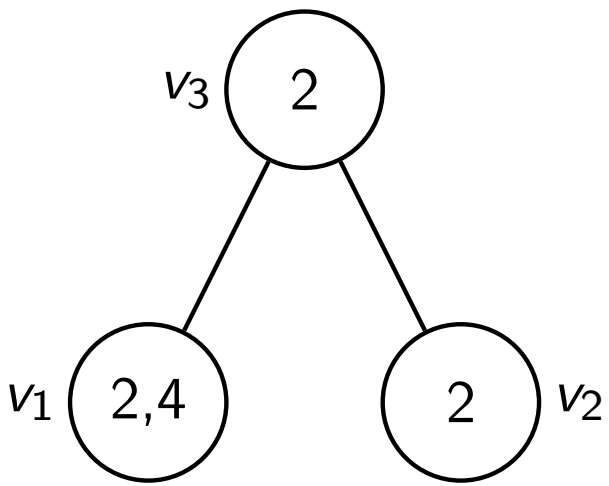


Queue

$\langle v_1, v_3 \rangle$

AC-3: Example

Consider a constraint network with three variables v_1, v_2, v_3 with $\mathrm{dom}(v_1) = \{2, 4\}$ and $\mathrm{dom}(v_2) = \mathrm{dom}(v_3) = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides exactly").



Queue

AC-3: Discussion

- *queue* can be an arbitrary data structure
  that supports insert and remove operations
  (the order of removal does not affect the result)

$\rightsquigarrow$ use data structure with fast insertion and removal, e.g., stack

- AC-3 has the same effect as AC-1:
  it enforces arc consistency
- proof idea: invariant of the **while** loop:
  If $\langle u, v \rangle \notin \mathit{queue}$, then u is arc consistent with respect to v

AC-3: Time Complexity

Proposition (time complexity of AC-3)

Let $\mathcal{C}$ be a constraint network with e nontrivial constraints and maximal domain size k.

The time complexity of AC-3 *is* $O(e \cdot k^3)$.

AC-3: Time Complexity (Proof)

Proof.

Consider a pair $\langle u, v \rangle$ such that there exists a nontrivial constraint R_{uv} or R_{vu}. (There are at most $2e$ of such pairs.)

AC-3: Time Complexity (Proof)

Proof.

Consider a pair $\langle u, v \rangle$ such that there exists a nontrivial constraint R_{uv} or R_{vu}. (There are at most $2e$ of such pairs.)

Every time this pair is inserted to the queue (except for the first time) the domain of the second variable has just been reduced.

AC-3: Time Complexity (Proof)

Proof.

Consider a pair $\langle u, v \rangle$ such that there exists a nontrivial constraint R_{uv} or R_{vu}. (There are at most $2e$ of such pairs.)

Every time this pair is inserted to the queue (except for the first time) the domain of the second variable has just been reduced.

This can happen at most k times.

AC-3: Time Complexity (Proof)

Proof.

Consider a pair $\langle u, v \rangle$ such that there exists a nontrivial constraint R_{uv} or R_{vu}. (There are at most $2e$ of such pairs.)

Every time this pair is inserted to the queue (except for the first time) the domain of the second variable has just been reduced.

This can happen at most k times.

Hence every pair $\langle u, v \rangle$ is inserted into the queue
at most $k + 1$ times $\rightsquigarrow$ at most $O(ek)$ insert operations in total.

AC-3: Time Complexity (Proof)

Proof.

Consider a pair $\langle u, v \rangle$ such that there exists a nontrivial constraint R_{uv} or R_{vu}. (There are at most $2e$ of such pairs.)

Every time this pair is inserted to the queue (except for the first time) the domain of the second variable has just been reduced.

This can happen at most k times.

Hence every pair $\langle u, v \rangle$ is inserted into the queue at most $k + 1$ times $\rightsquigarrow$ at most $O(ek)$ insert operations in total.

This bounds the number of **while** iterations by $O(ek)$, giving an overall time complexity of $O(ek) \cdot O(k^2) = O(ek^3)$. □

Summary

Summary

- arc consistency extends forward-checking
- more expensive and more powerful:
  - iteratively remove values without a suitable "partner value" for another variable until fixed point reached
- basic operation: revise makes one variable arc-consistent w.r.t. another variable
- efficient implementation of AC-3: $O(ek^3)$
  with e: #nontrivial constraints, k: size of domain