

# Foundations of Artificial Intelligence

## D3. Constraint Satisfaction Problems: Backtracking

Malte Helmert

University of Basel

April 9, 2025

# Constraint Satisfaction Problems: Overview

## Chapter overview: constraint satisfaction problems

- D1–D2. Introduction
- D3–D5. Basic Algorithms
  - D3. Backtracking
  - D4. Arc Consistency
  - D5. Path Consistency
- D6–D7. Problem Structure

# CSP Algorithms

# CSP Algorithms

In the following chapters, we consider **algorithms for solving** constraint networks.

basic concepts:

- **search**: check partial assignments systematically
- **backtracking**: discard inconsistent partial assignments
- **inference**: derive equivalent, but tighter constraints to reduce the size of the search space

# Naive Backtracking

# Naive Backtracking (= Without Inference)

```
function NaiveBacktracking( $\mathcal{C}, \alpha$ ):
```

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

```
  if  $\alpha$  is inconsistent with  $\mathcal{C}$ :  
    return inconsistent
```

```
  if  $\alpha$  is a total assignment:  
    return  $\alpha$ 
```

```
  select some variable  $v$  for which  $\alpha$  is not defined
```

```
  for each  $d \in \text{dom}(v)$  in some order:
```

```
     $\alpha' := \alpha \cup \{v \mapsto d\}$ 
```

```
     $\alpha'' := \text{NaiveBacktracking}(\mathcal{C}, \alpha')$ 
```

```
    if  $\alpha'' \neq \text{inconsistent}$ :  
      return  $\alpha''$ 
```

```
  return inconsistent
```

**input:** constraint network  $\mathcal{C}$  and partial assignment  $\alpha$  for  $\mathcal{C}$   
(first invocation: empty assignment  $\alpha = \emptyset$ )

**result:** solution of  $\mathcal{C}$  or **inconsistent**

# Running Example

## Full Formal Model of Running Example

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

- variables:

$$V = \{w, x, y, z\}$$

- domains:

$$\text{dom}(w) = \text{dom}(y) = \{1, 2, 3, 4\}$$

$$\text{dom}(x) = \text{dom}(z) = \{1, 2, 3\}$$

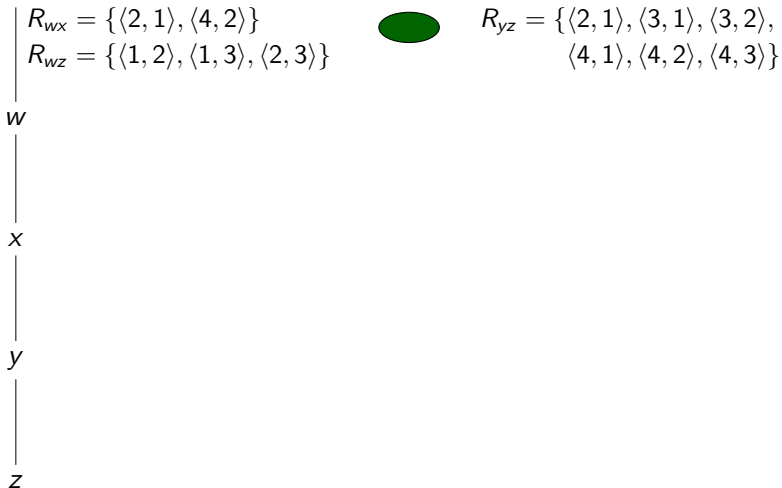
- constraints:

$$R_{wx} = \{\langle 2, 1 \rangle, \langle 4, 2 \rangle\}$$

$$R_{wz} = \{\langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 2, 3 \rangle\}$$

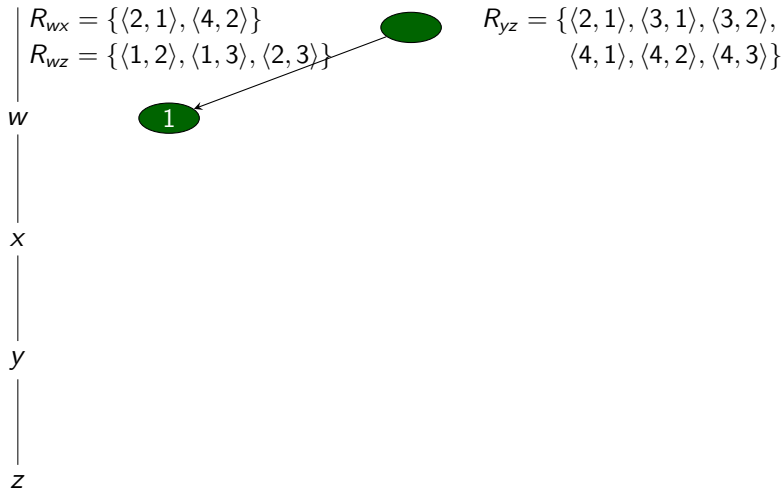
$$R_{yz} = \{\langle 2, 1 \rangle, \langle 3, 1 \rangle, \langle 3, 2 \rangle, \\ \langle 4, 1 \rangle, \langle 4, 2 \rangle, \langle 4, 3 \rangle\}$$

# Running Example: Search Tree

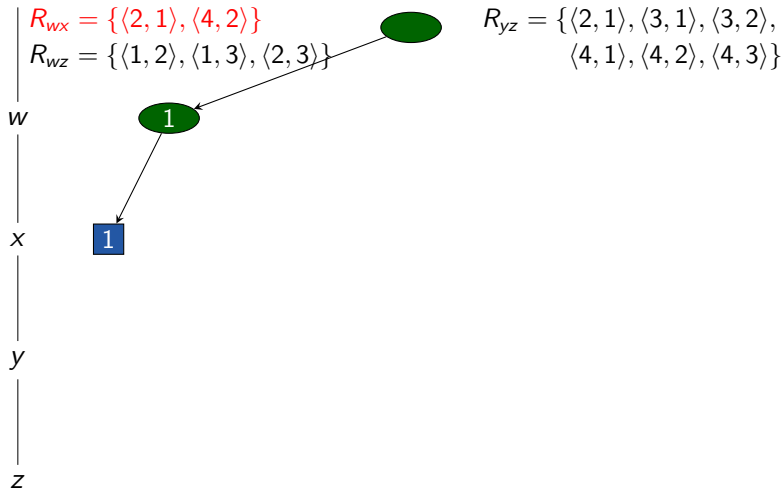




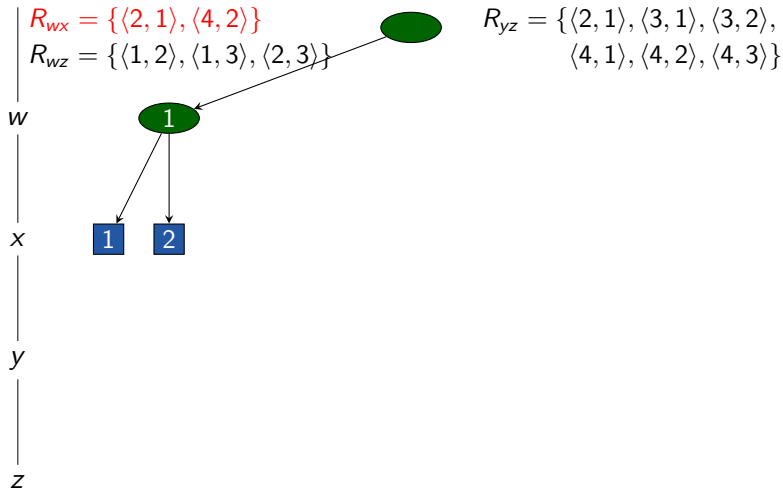
# Running Example: Search Tree



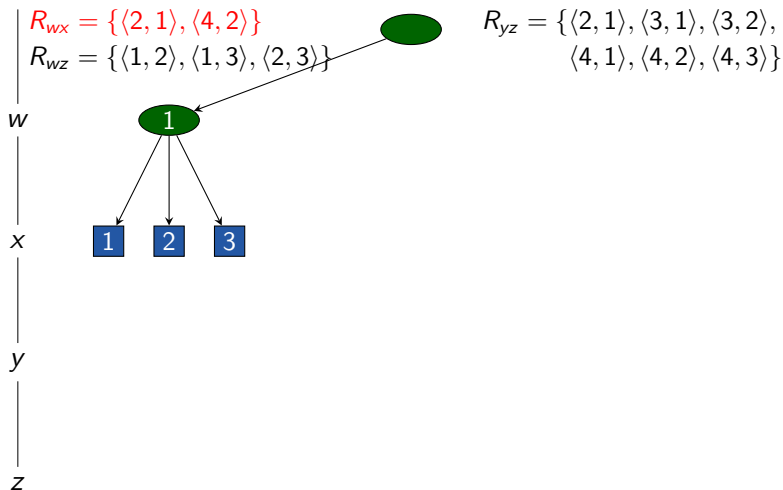
# Running Example: Search Tree



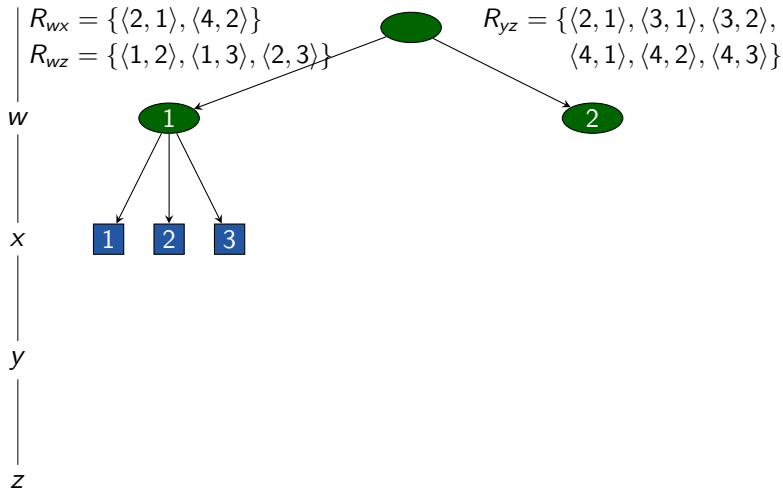
# Running Example: Search Tree



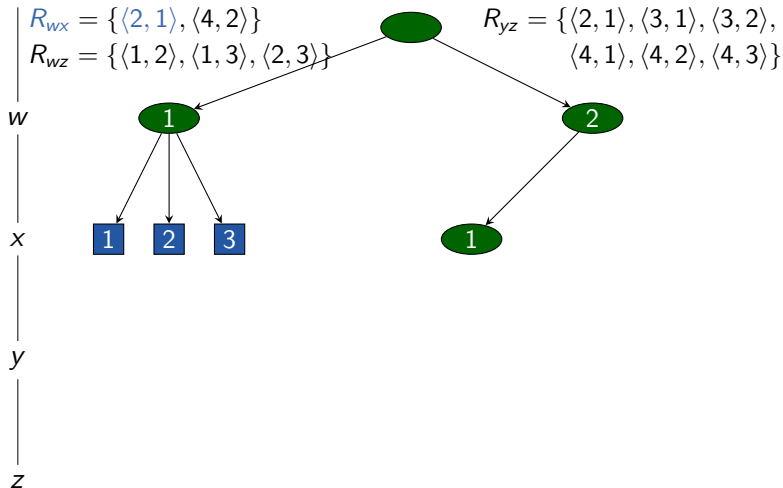
# Running Example: Search Tree



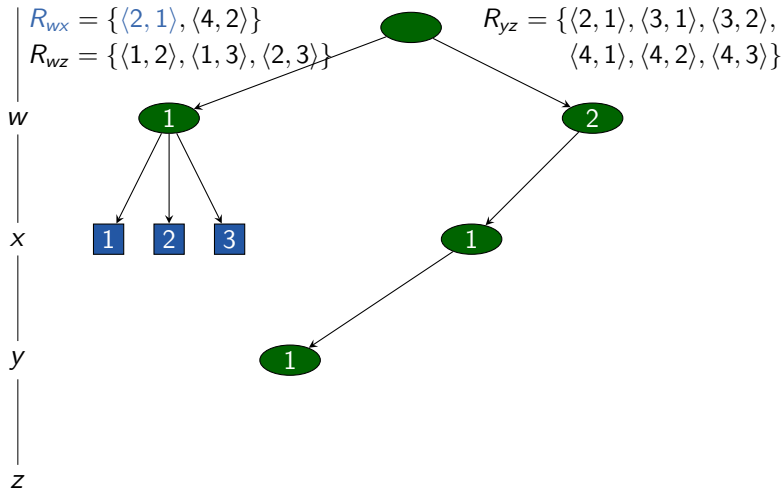
# Running Example: Search Tree



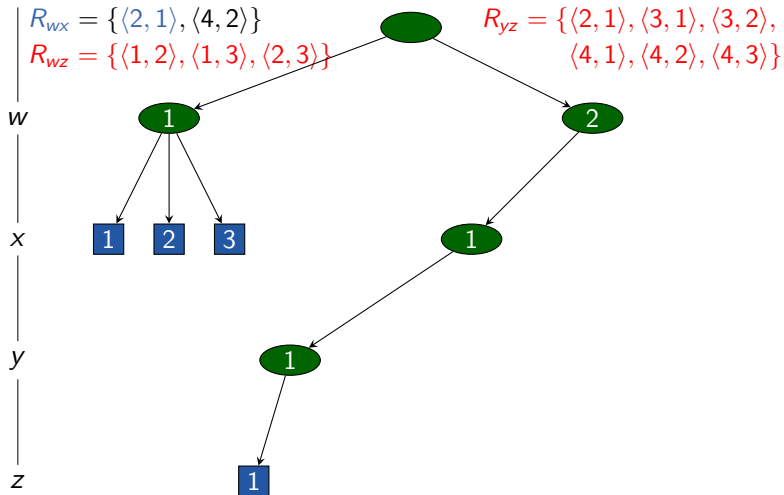
# Running Example: Search Tree



# Running Example: Search Tree

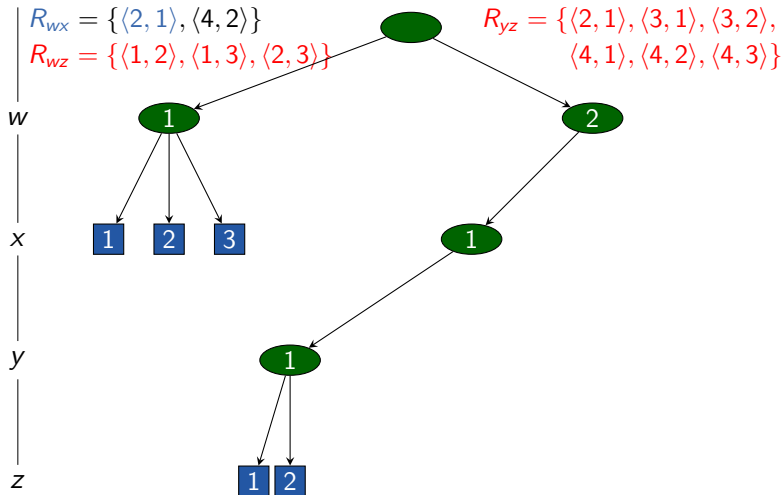


# Running Example: Search Tree

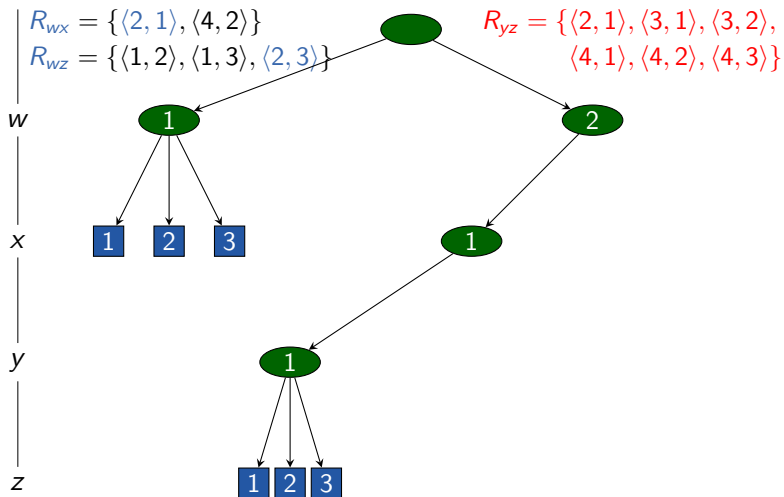




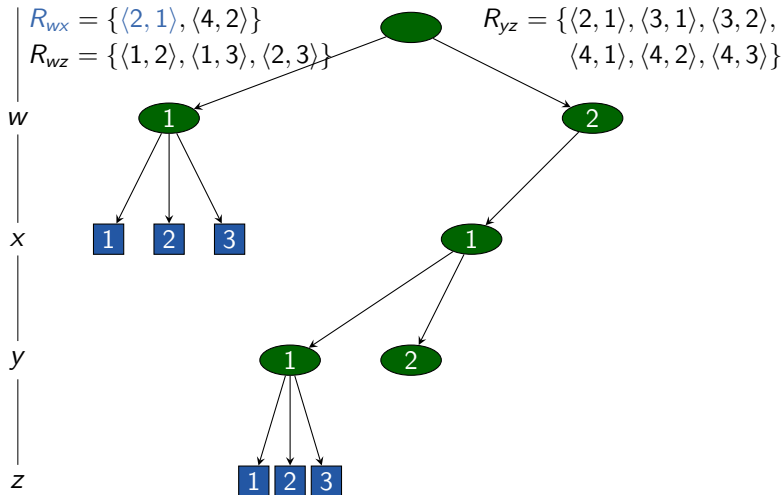
# Running Example: Search Tree



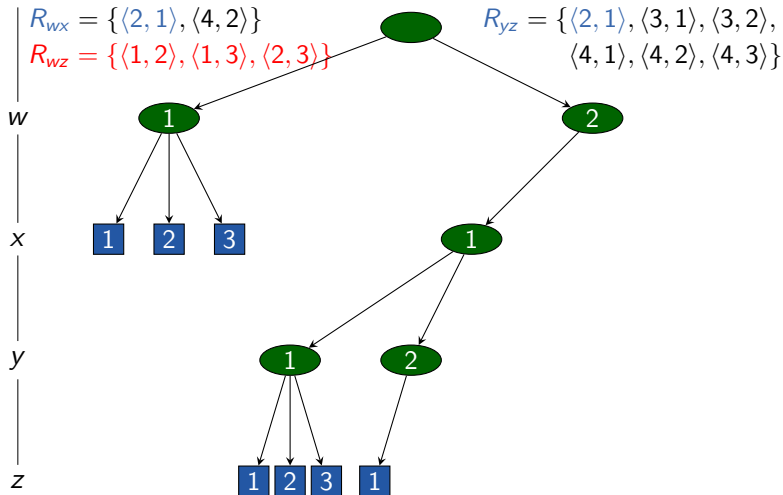
## Running Example: Search Tree



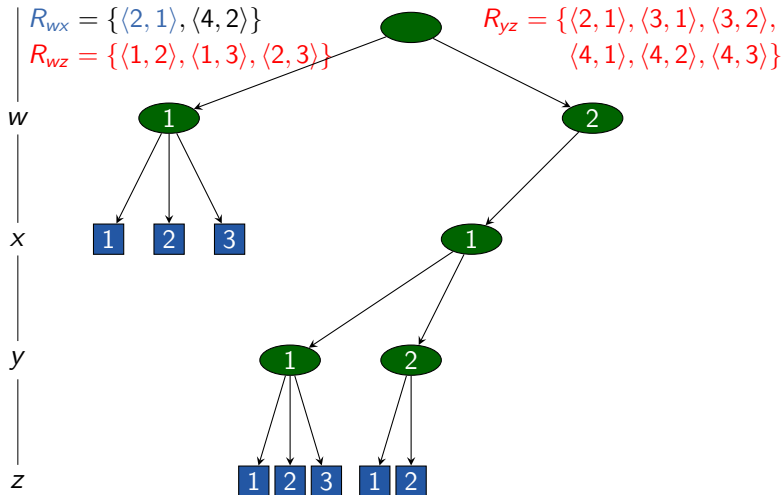
# Running Example: Search Tree



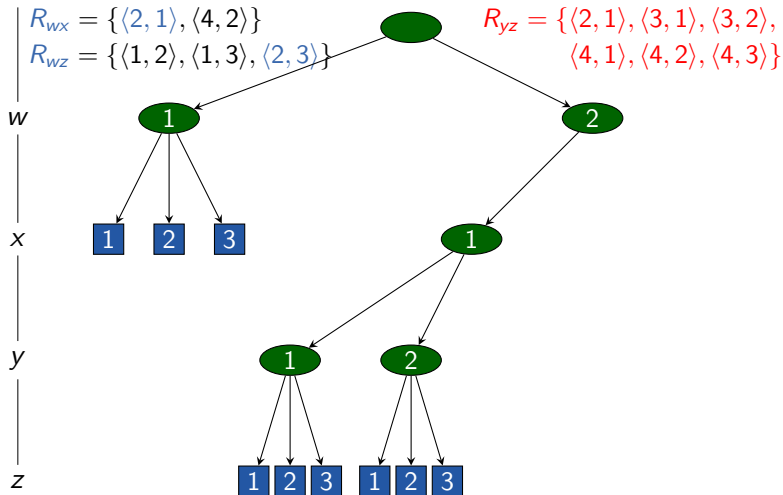
# Running Example: Search Tree



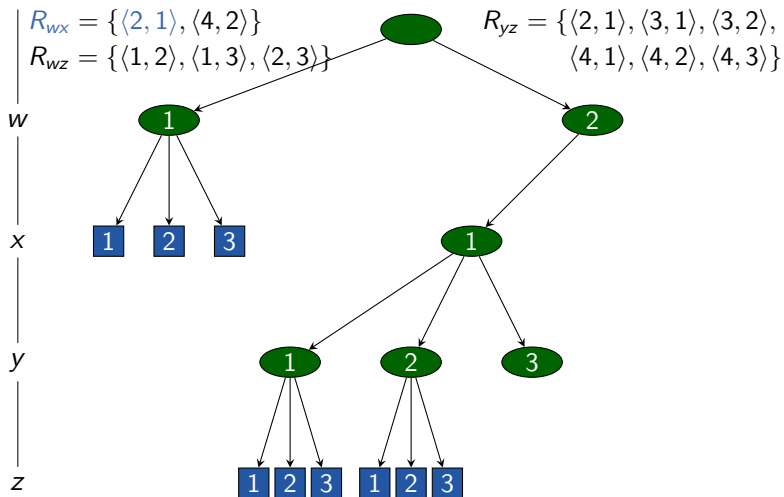
# Running Example: Search Tree



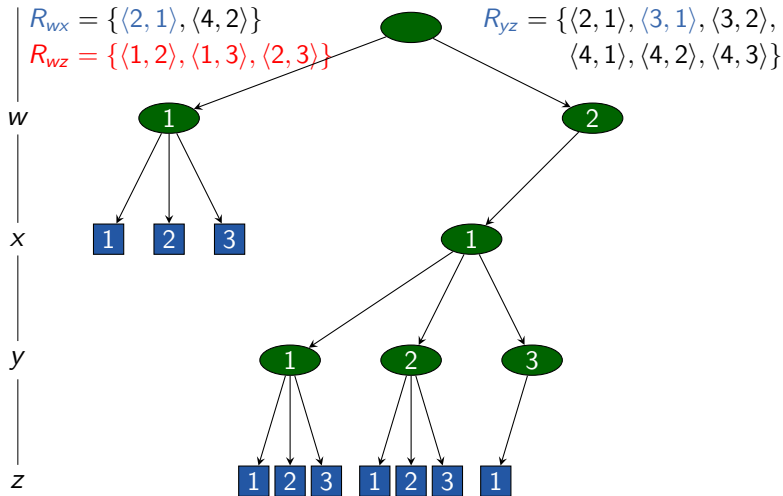
# Running Example: Search Tree



## Running Example: Search Tree

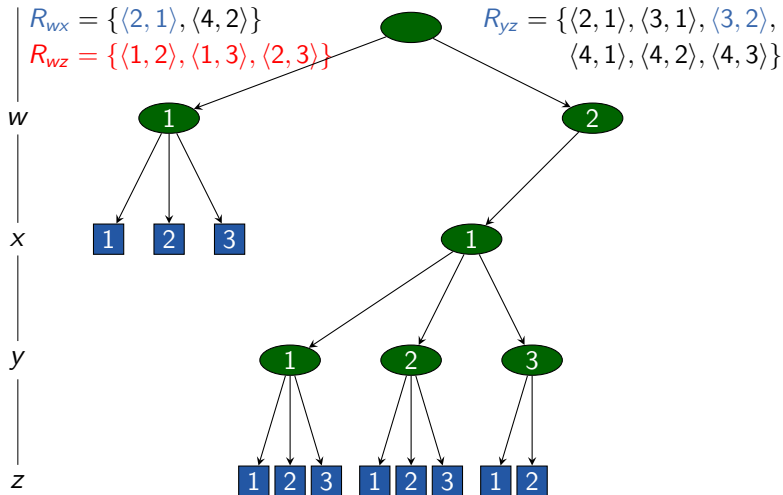


## Running Example: Search Tree

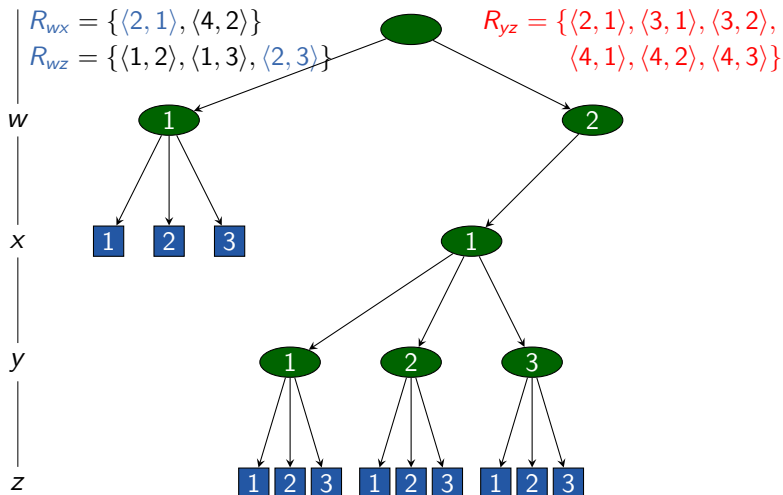




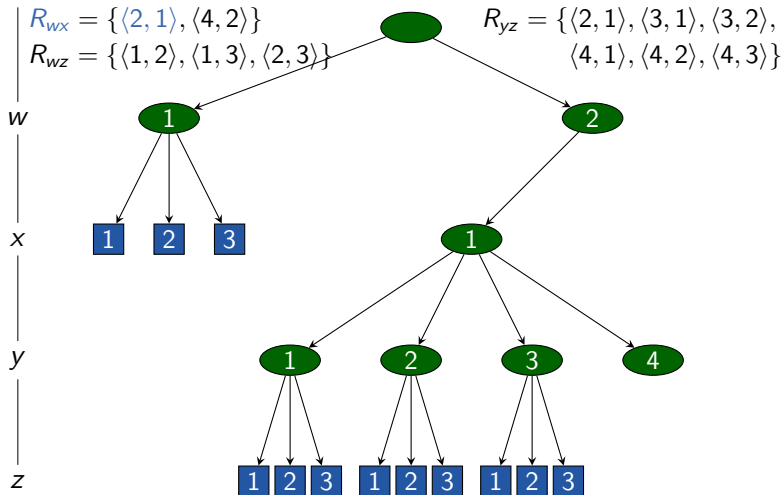
# Running Example: Search Tree



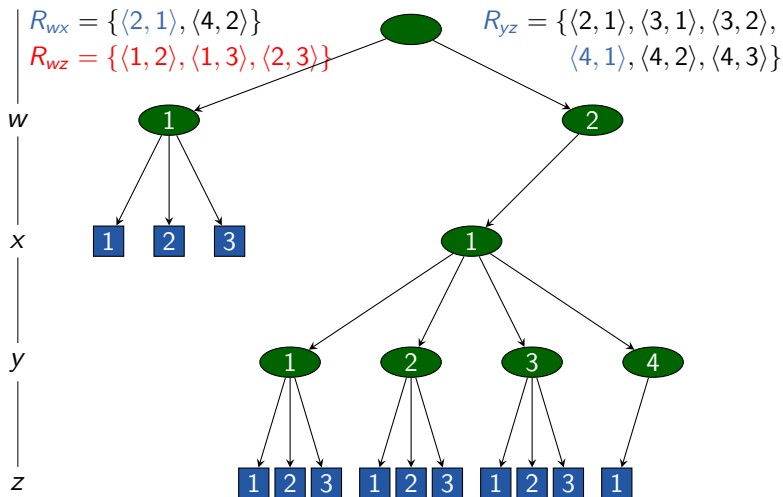
# Running Example: Search Tree



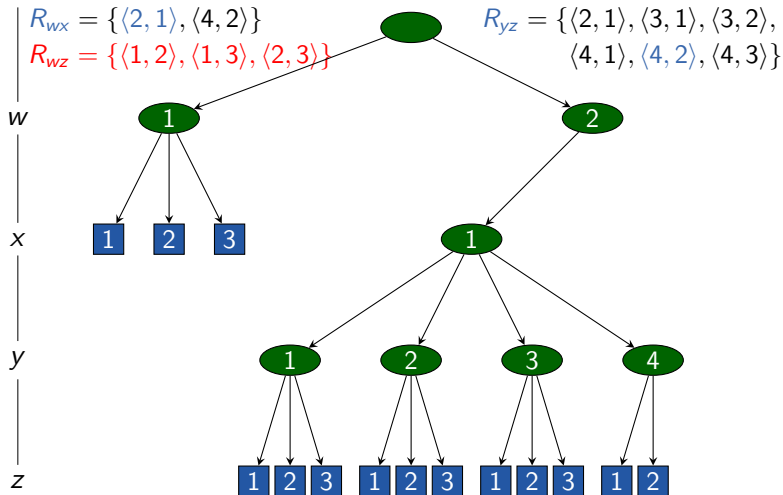
# Running Example: Search Tree



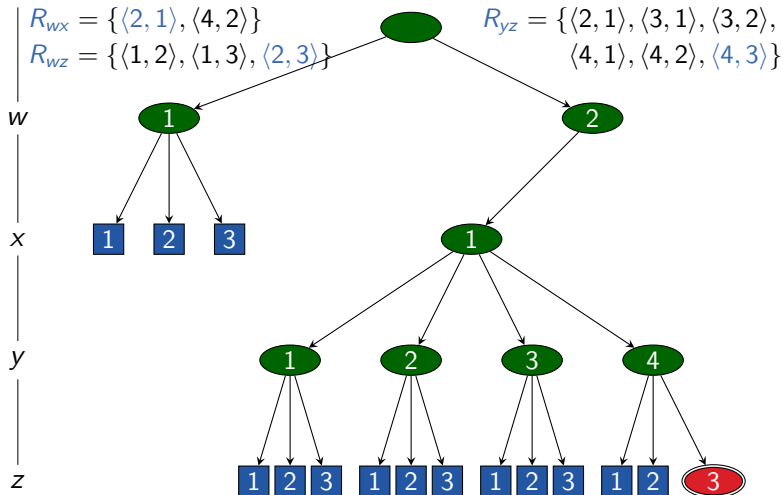
# Running Example: Search Tree



# Running Example: Search Tree



# Running Example: Search Tree



# Is This a New Algorithm?

We have already seen this algorithm:

**Backtracking corresponds to depth-first search** (Chapter B8)

with the following state space:

- **states**: partial assignments
- **initial state**: empty assignment  $\emptyset$
- **goal states**: consistent total assignments
- **actions**:  $assign_{v,d}$  assigns value  $d \in \text{dom}(v)$  to variable  $v$
- **action costs**: all 0 (all solutions are of equal quality)
- **transitions**:
  - for each **non-total consistent** assignment  $\alpha$ ,  
choose variable  $v = \text{select}(\alpha)$  that is unassigned in  $\alpha$
  - transition  $\alpha \xrightarrow{assign_{v,d}} \alpha \cup \{v \mapsto d\}$  for each  $d \in \text{dom}(v)$

# Why Depth-First Search?

Depth-first search is particularly well-suited for CSPs:

- path length **bounded** (by the number of variables)
- solutions located at **the same depth** (lowest search layer)
- state space is directed **tree**, initial state is the root  
     $\rightsquigarrow$  **no duplicates** (Why?)

Hence none of the problematic cases for depth-first search occurs.



# Naive Backtracking: Discussion

- Naive backtracking often has to exhaustively explore **similar** search paths (i.e., partial assignments that are identical except for a few variables).
  - “Critical” variables are not recognized and hence considered for assignment (too) late.
  - Decisions that necessarily lead to constraint violations are only recognized when all variables involved in the constraint have been assigned.
- ⇒ more intelligence by **focusing on critical decisions** and by **inference** of consequences of previous decisions

# Variable and Value Orders

# Naive Backtracking

**function** NaiveBacktracking( $\mathcal{C}, \alpha$ ):

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

**if**  $\alpha$  is inconsistent with  $\mathcal{C}$ :

**return inconsistent**

**if**  $\alpha$  is a total assignment:

**return**  $\alpha$

select **some variable**  $v$  for which  $\alpha$  is not defined

**for each**  $d \in \text{dom}(v)$  **in some order**:

$\alpha' := \alpha \cup \{v \mapsto d\}$

$\alpha'' := \text{NaiveBacktracking}(\mathcal{C}, \alpha')$

**if**  $\alpha'' \neq \text{inconsistent}$ :

**return**  $\alpha''$

**return inconsistent**

# Variable Orders

- Backtracking does not specify in which order **variables** are considered for assignment.
- Such orders can strongly influence the search space size and hence the search performance.  
    ~> **example:** exercises
- Eventually we have to assign all variables  
    ~> prefer critical assignments (**fail early**)

**German:** Variablenordnung

# Value Orders

- Backtracking does not specify in which order the **values** of the selected variable  $v$  are considered.
- This is not as important because it **does not matter** in subtrees without a solution. (Why not?)
- **If** there is a solution in the subtree, then ideally a value that leads to a solution should be chosen.  
     $\rightsquigarrow$  prefer promising assignments

German: Werteordnung

# Static vs. Dynamic Orders

we distinguish:

- **static** orders (fixed prior to search)
- **dynamic** orders (selected variable or value order depends on the search state)

comparison:

- dynamic orders obviously more powerful
- static orders  $\rightsquigarrow$  no computational overhead during search

The following ordering criteria can be used statically, but are more effective combined with inference ( $\rightsquigarrow$  later) and used dynamically.

# Variable Orders

two common variable ordering criteria:

- **minimum remaining values:**  
prefer variables that have small **domains**
  - **intuition:** few subtrees  $\rightsquigarrow$  smaller tree
  - **extreme case:** only **one** value  $\rightsquigarrow$  forced assignment
- **most constraining variable:**  
prefer variables contained in **many** nontrivial constraints
  - **intuition:** constraints tested early  
 $\rightsquigarrow$  inconsistencies recognized early  $\rightsquigarrow$  smaller tree

**combination:** use minimum remaining values criterion,  
then most constraining variable criterion to break ties

# Value Orders

## Definition (conflict)

Let  $\mathcal{C} = \langle V, \text{dom}, (R_{uv}) \rangle$  be a constraint network.  
For variables  $v \neq v'$  and values  $d \in \text{dom}(v)$ ,  $d' \in \text{dom}(v')$ ,  
the assignment  $v \mapsto d$  is **in conflict** with  $v' \mapsto d'$  if  $\langle d, d' \rangle \notin R_{vv'}$ .

value ordering criterion for partial assignment  $\alpha$   
and selected variable  $v$ :

- **minimum conflicts:** prefer values  $d \in \text{dom}(v)$   
such that  $v \mapsto d$  causes as few conflicts as possible  
with variables that are unassigned in  $\alpha$



# Summary

# Summary: Backtracking

basic search algorithm for constraint networks: **backtracking**

- extends the (initially empty) partial assignment step by step until an **inconsistency** or a **solution** is found
- is a form of **depth-first search**
- depth-first search particularly well-suited because state space is directed tree and all solutions at same (known) depth

# Summary: Variable and Value Orders

- **Variable orders** influence the performance of backtracking significantly.
  - goal: **critical** decisions as early as possible
- **Value orders** influence the performance of backtracking on **solvable** constraint networks significantly.
  - goal: **most promising** assignments first