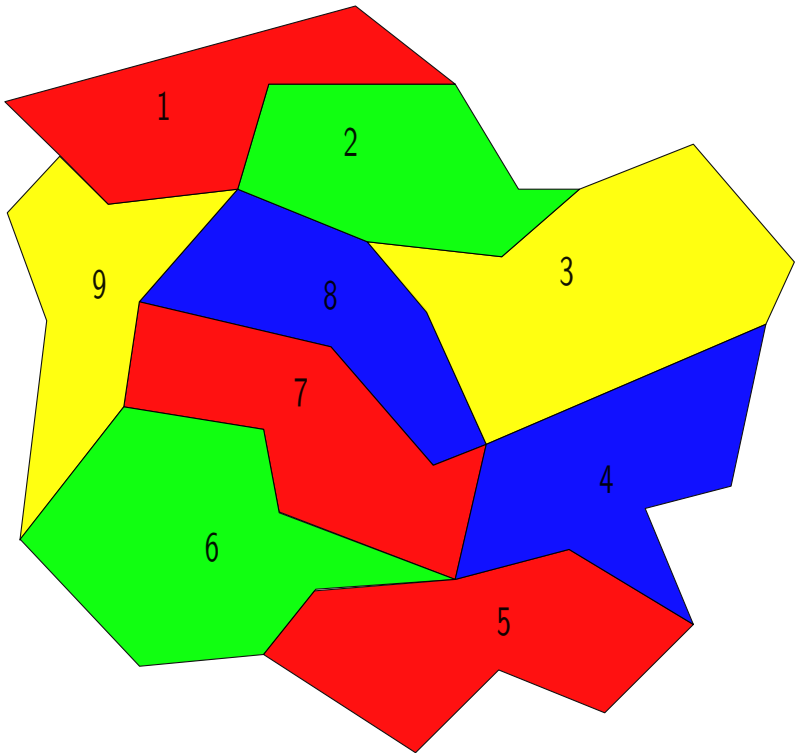
Constraints Satisfaction

- Components of the state:
  - Variables
  - Domains (Allowed values for the variables)
  - Binary constraints among variables
- **Goal:** To find a state that satisfies all the constraints (Set of values for the variables that satisfies the constraints)
- Examples:
  - Map coloring, Crosswords, 8-queens
  - Allocation/distribution of resources (Scheduling of tasks of a process, location of gas stations, mobile antennas, ...)
  - Automatic reasoning and logic
  - Image recognition and artificial vision
  - Task planning
  - ...

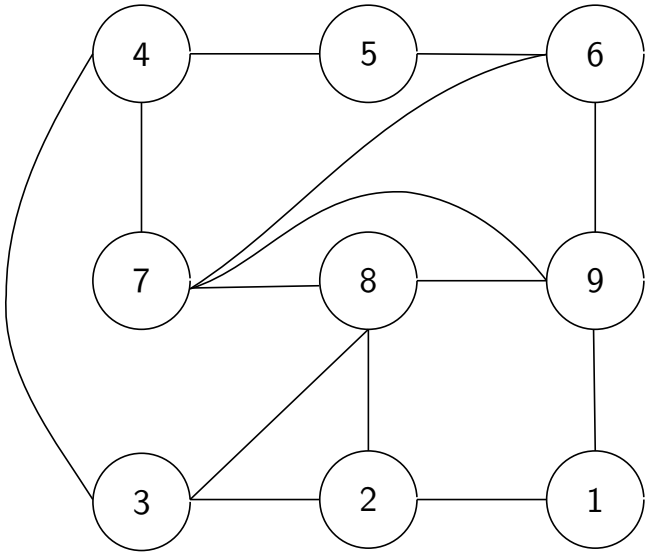
Representation

- State = Graph of constraints
  - Variables = Labels of the vertices
  - Domains = Values of the vertices
  - Constrains = directed and labeled edges among vertices
- Example: Map coloring



Domains={Red,Green,Blue,Yellow}

Constraint := Inequality



Algorithms

- **Generate and test:** Very inefficient
- **Blind search**
  - Depth-first search with chronological backtracking
- **Constraint propagation**
  - Before the search
  - During the search

Depth-first search with chronological backtracking

- Depth first search over the set of variables
- Exhaustive strategy to assign the values to the variables
  - Test the constraints after each instantiation of value to a variable
  - If any constraint is violated, perform a backtrack to the last valid instantiation
- The search is performed in the space of partial solutions
- Chronological Backtracking: types of variables (past, current, future)

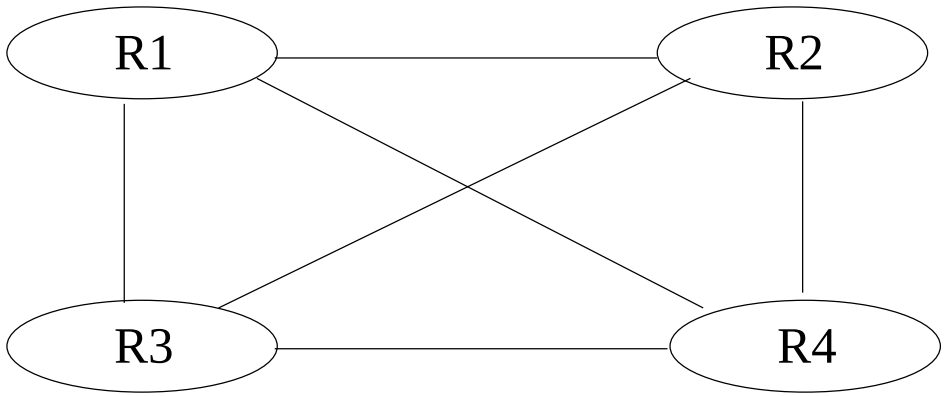
Chronological Backtracking Algorithm

Function: chronological backtracking(vfuture, solution)

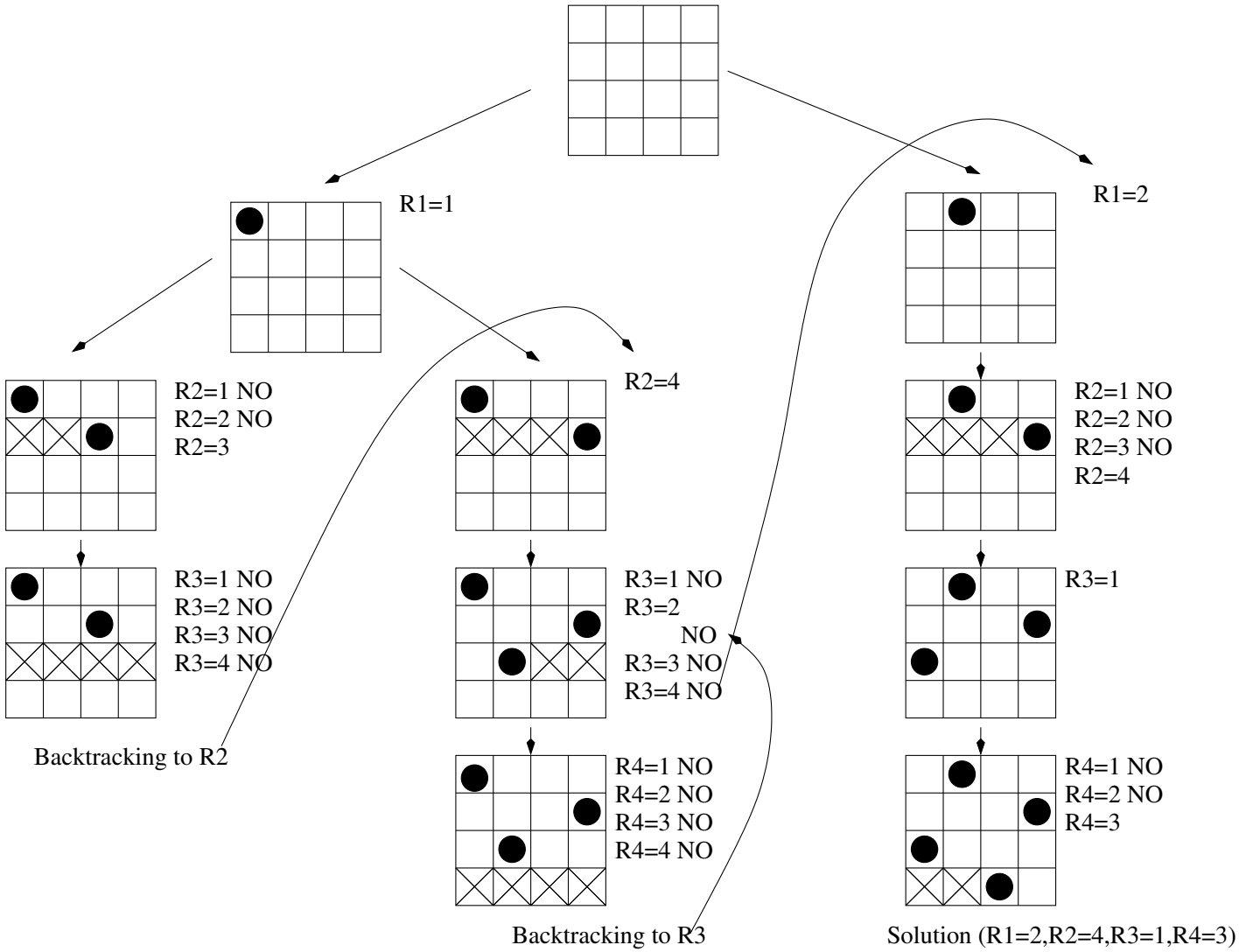
```
if vfuture.is_empty?() then
    return solution
else
    vcurrent ← vfuture.first()
    vfuture.delete_first()
    foreach v ∈ vcurrent.values() do
        vcurrent.assign(v)
        solution.add(vcurrent)
        if solution.valid() then
            solution ← chronological backtracking(vfuture,solution)
            if not solution.is_fail?() then
                return solution
            else
                solution.delete(vcurrent)
        else
            solution.delete(vcurrent)
    return solution.fail()
```

Example: 4-queens

- Assign 4 queens, 1 to each row of a 4x4 board, the queens must not attack each other
  - Variables: R_1, ... , R_4 (queens)
  - Domains: [1 .. 4] for each R_i (column)
  - Constraints: R_i does not attach R_j
- Graph of Constraints:

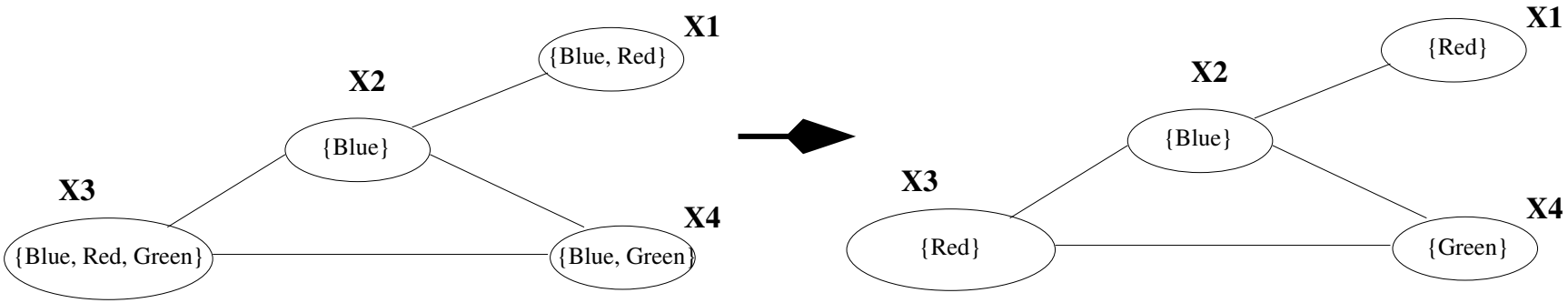


4-queens solved by chronological backtracking



Propagation of constraints

A set of constraints can induce others that were implicit. The propagation of constraints is the process of making these constraints explicit



The goal is to reduce the search space. The propagation can be performed:

1. As a preprocess (prune the parts of the search space where there are no solutions)
2. During the search: Pruning the search space using the information from the search (Forward Checking)

Propagation of constraints

It is a two steps cycle:

1. Constraints are propagated:
   A constraint usually is not independent from the rest (constraints involve many variables, variables appear in many constraints)
   The propagation may use inference rules.
2. Results are analyzed:
   1. The solution has been found
   2. There is no solution
   3. Continue the search: Heuristic search in the rest of search space

Properties of the graph of constraints

- Some properties of the graph of constraints could be used to reduce the size of the search space
  - k-consistency: Pruning of values that are not possible for a group of k variables
  - Arc consistency (2-consistency): The values that are impossible for pairs of variables are eliminated
  - Path consistency (3-consistency): The values that are impossible for trios of variables are eliminated
  - ...
- A problem that has a graph of constraints that satisfies k-consistency (2, 3, ...) reduces the number of backtrackings during the search

Arc-consistency preprocess

- A CSP is arc-consistent if for each pair of variables (X_i, X_j) and any value v_k from D_i there is a value v_l from D_j that satisfies the constraints. We are looking for the values from X_i that are consistent with the constraints on the edge of this variable
- We want that all variables are arc-consistent for all the edges they have. In other words, the domains of all the variables must be consistent with all the constraints

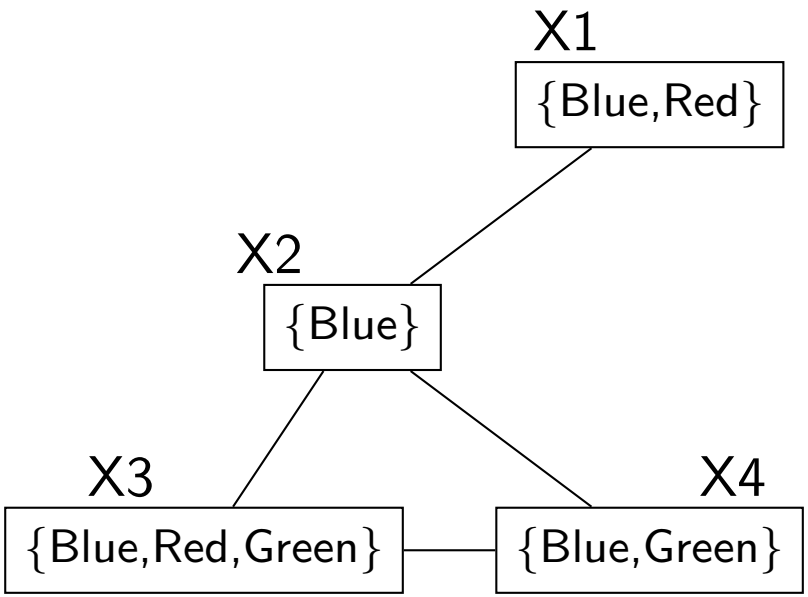
Arc-consistency algorithm

If a CSP is not arc-consistent we can transform it using this algorithm:

Algorithm: Arc consistency

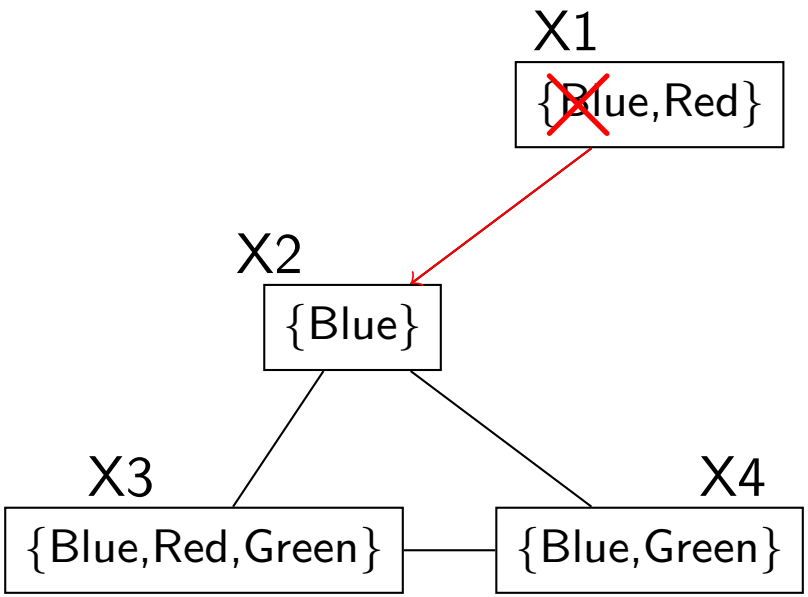
R $\leftarrow$ set of edges of the problem `/* both sides */`
while *the domains of the variables are modified* **do**
    r $\leftarrow$ get_arc(R)
    `/*` r_i `is the variable origin of the edge */`
    `/*` r_j `is the variable end of the edge */`
    foreach $v \in$ *the domain of* r_i **do**
        if *v does not have a value in* r_j *domain that satisfies r* **then**
            delete v from r_i domain
            add all edges that end in r_i except $(r_j \rightarrow r_i)$
        end
    end
end

Arc-consistency example



Initial list of edges:
(X1,X2), (X2,X1), (X2,X3), (X3,X2),
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

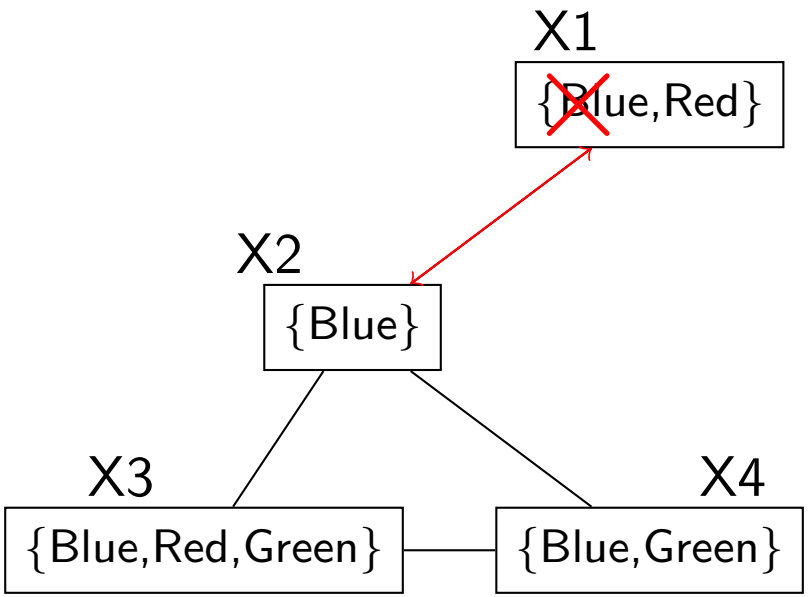
Arc-consistency example



1. $X_1 - X_2 \rightarrow$ Delete Blue from X_1

Initial list of edges:
(X1,X2), (X2,X1), (X2,X3), (X3,X2),
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

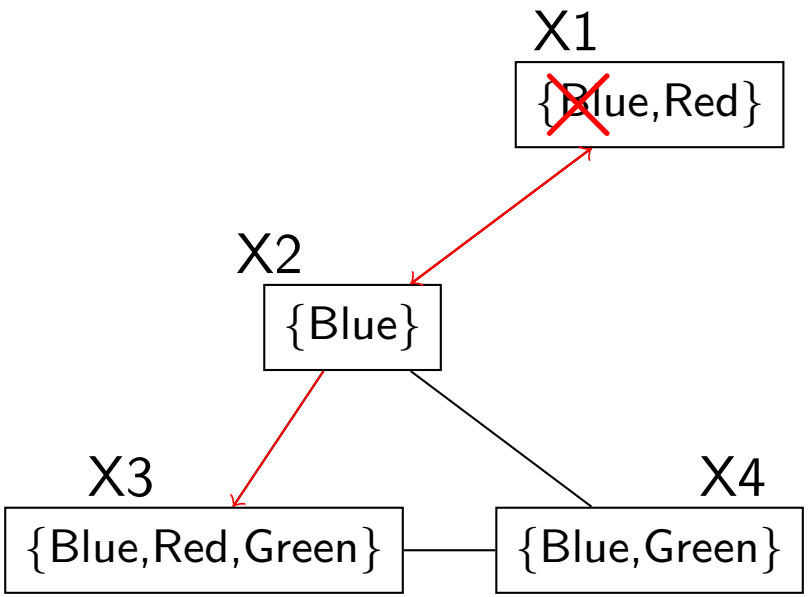
Arc-consistency example



1. $X_1 - X_2 \rightarrow$ Delete Blue from X_1
2. $X_2 - X_1 \rightarrow$ All consistent

Initial list of edges:
(X1,X2), (X2,X1), (X2,X3), (X3,X2),
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

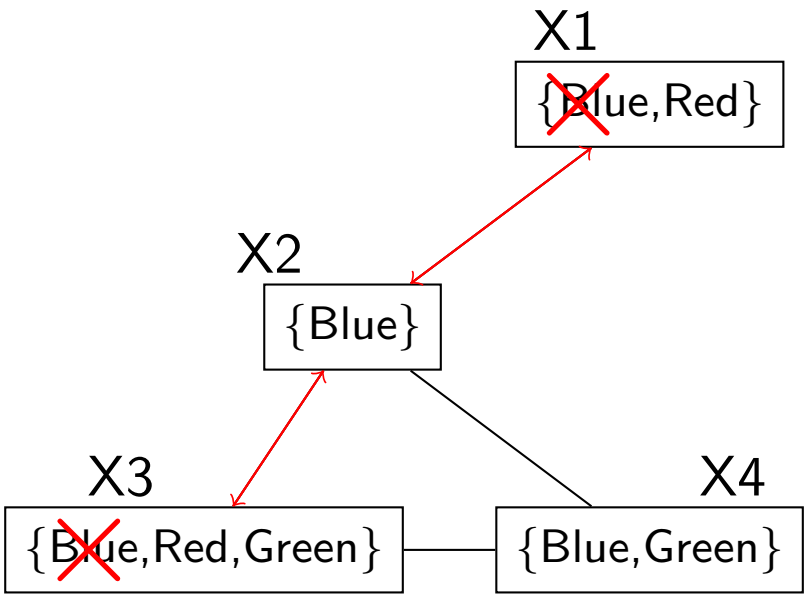
Arc-consistency example



1. $X_1 - X_2 \rightarrow$ Delete Blue from X_1
2. $X_2 - X_1 \rightarrow$ All consistent
3. $X_2 - X_3 \rightarrow$ All consistent

Initial list of edges:
(X1,X2), (X2,X1), (X2,X3), (X3,X2),
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

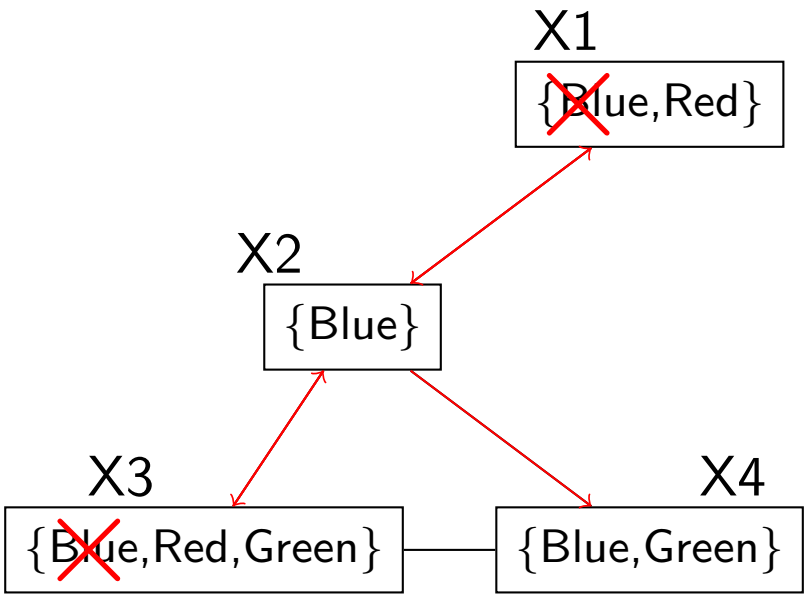
Arc-consistency example



1. $X_1 - X_2 \rightarrow$ Delete Blue from X_1
2. $X_2 - X_1 \rightarrow$ All consistent
3. $X_2 - X_3 \rightarrow$ All consistent
4. $X_3 - X_2 \rightarrow$Delete Blue from X_3, we should add $X_4 - X_3$ but it is in already

Initial list of edges:
(X1,X2), (X2,X1), (X2,X3), (X3,X2),
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

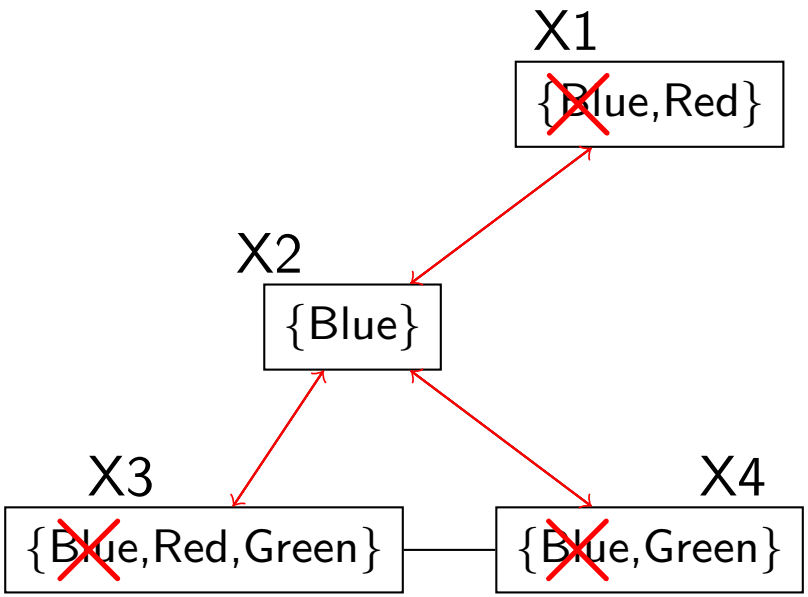
Arc-consistency example



1. $X_1 - X_2 \rightarrow$ Delete Blue from X_1
2. $X_2 - X_1 \rightarrow$ All consistent
3. $X_2 - X_3 \rightarrow$ All consistent
4. $X_3 - X_2 \rightarrow$Delete Blue from X_3, we should add $X_4 - X_3$ but it is in already
5. $X_2 - X_4 \rightarrow$ All consistent

Initial list of edges:
(X1,X2), (X2,X1), (X2,X3), (X3,X2),
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

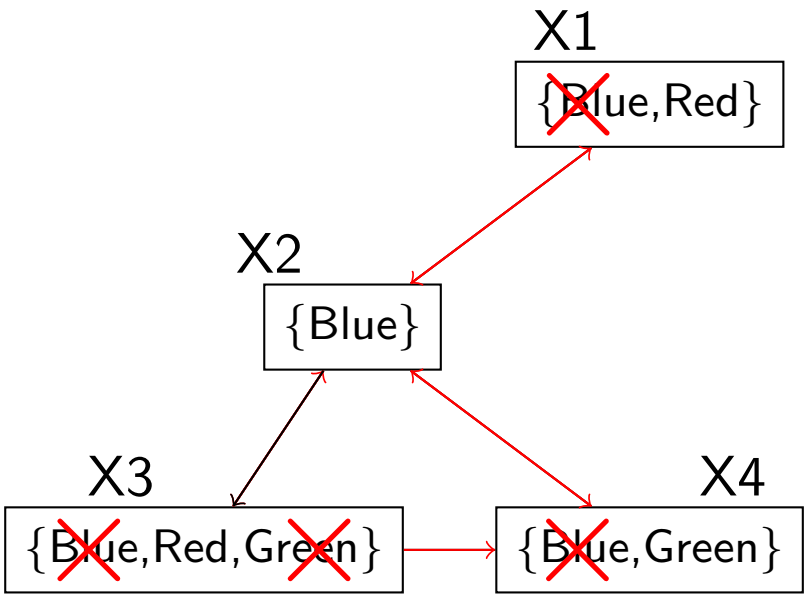
Arc-consistency example



1. $X_1 - X_2 \rightarrow$ Delete Blue from X_1
2. $X_2 - X_1 \rightarrow$ All consistent
3. $X_2 - X_3 \rightarrow$ All consistent
4. $X_3 - X_2 \rightarrow$Delete Blue from X_3, we should add $X_4 - X_3$ but it is in already
5. $X_2 - X_4 \rightarrow$ All consistent
6. $X_4 - X_2 \rightarrow$ Delete Blue from X_4, we should add $X_3 - X_4$ but it is in already

Initial list of edges:
(X1,X2), (X2,X1), (X2,X3), (X3,X2),
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

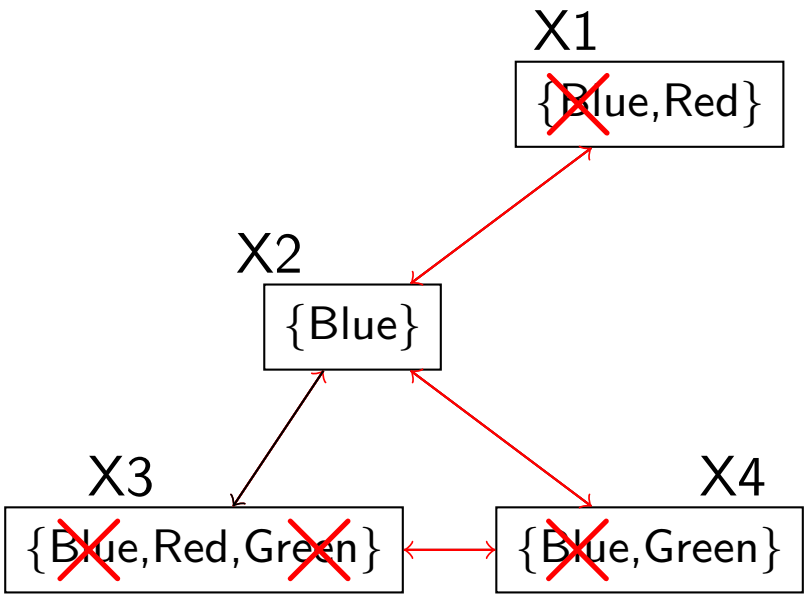
Arc-consistency example



Initial list of edges:
(X1,X2), (X2,X1), (X2,X3), (X3,X2), (X2,X4), (X4,X2), (X3,X4), (X4,X3)

1. $X_1 - X_2 \rightarrow$ Delete Blue from X_1
2. $X_2 - X_1 \rightarrow$ All consistent
3. $X_2 - X_3 \rightarrow$ All consistent
4. $X_3 - X_2 \rightarrow$Delete Blue from X_3, we should add $X_4 - X_3$ but it is in already
5. $X_2 - X_4 \rightarrow$ All consistent
6. $X_4 - X_2 \rightarrow$ Delete Blue from X_4, we should add $X_3 - X_4$ but it is in already
7. $X_3 - X_4 \rightarrow$ Delete Blue from X_3, we add $X_2 - X_3$

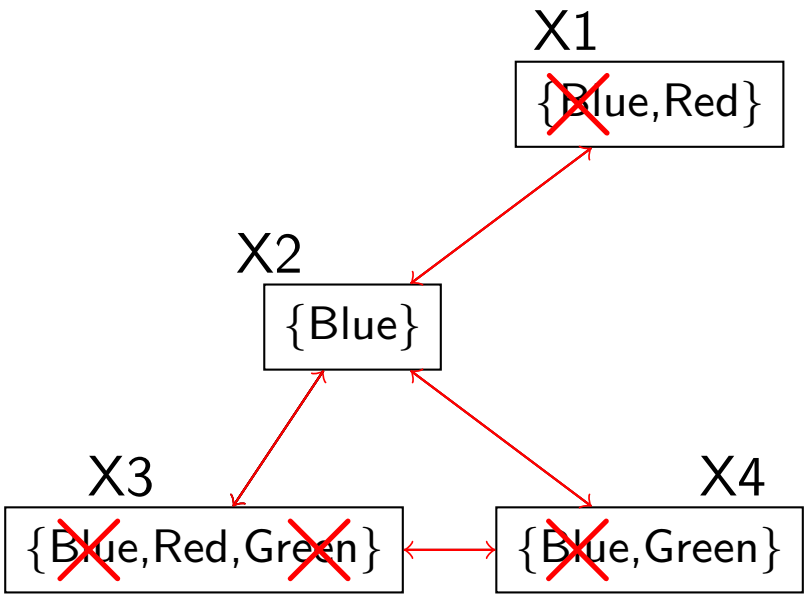
Arc-consistency example



Initial list of edges:
(X1,X2), (X2,X1), (X2,X3), (X3,X2), (X2,X4), (X4,X2), (X3,X4), (X4,X3)

1. $X_1 - X_2 \rightarrow$ Delete Blue from X_1
2. $X_2 - X_1 \rightarrow$ All consistent
3. $X_2 - X_3 \rightarrow$ All consistent
4. $X_3 - X_2 \rightarrow$Delete Blue from X_3, we should add $X_4 - X_3$ but it is in already
5. $X_2 - X_4 \rightarrow$ All consistent
6. $X_4 - X_2 \rightarrow$ Delete Blue from X_4, we should add $X_3 - X_4$ but it is in already
7. $X_3 - X_4 \rightarrow$ Delete Blue from X_3, we add $X_2 - X_3$
8. $X_4 - X_3 \rightarrow$ All consistent

Arc-consistency example



Initial list of edges:
(X1,X2), (X2,X1), (X2,X3), (X3,X2), (X2,X4), (X4,X2), (X3,X4), (X4,X3)

1. $X_1 - X_2 \rightarrow$ Delete Blue from X_1
2. $X_2 - X_1 \rightarrow$ All consistent
3. $X_2 - X_3 \rightarrow$ All consistent
4. $X_3 - X_2 \rightarrow$Delete Blue from X_3, we should add $X_4 - X_3$ but it is in already
5. $X_2 - X_4 \rightarrow$ All consistent
6. $X_4 - X_2 \rightarrow$ Delete Blue from X_4, we should add $X_3 - X_4$ but it is in already
7. $X_3 - X_4 \rightarrow$ Delete Blue from X_3, we add $X_2 - X_3$
8. $X_4 - X_3 \rightarrow$ All consistent
9. $X_2 - X_3 \rightarrow$ All consistent

Propagation during the search (forward checking)

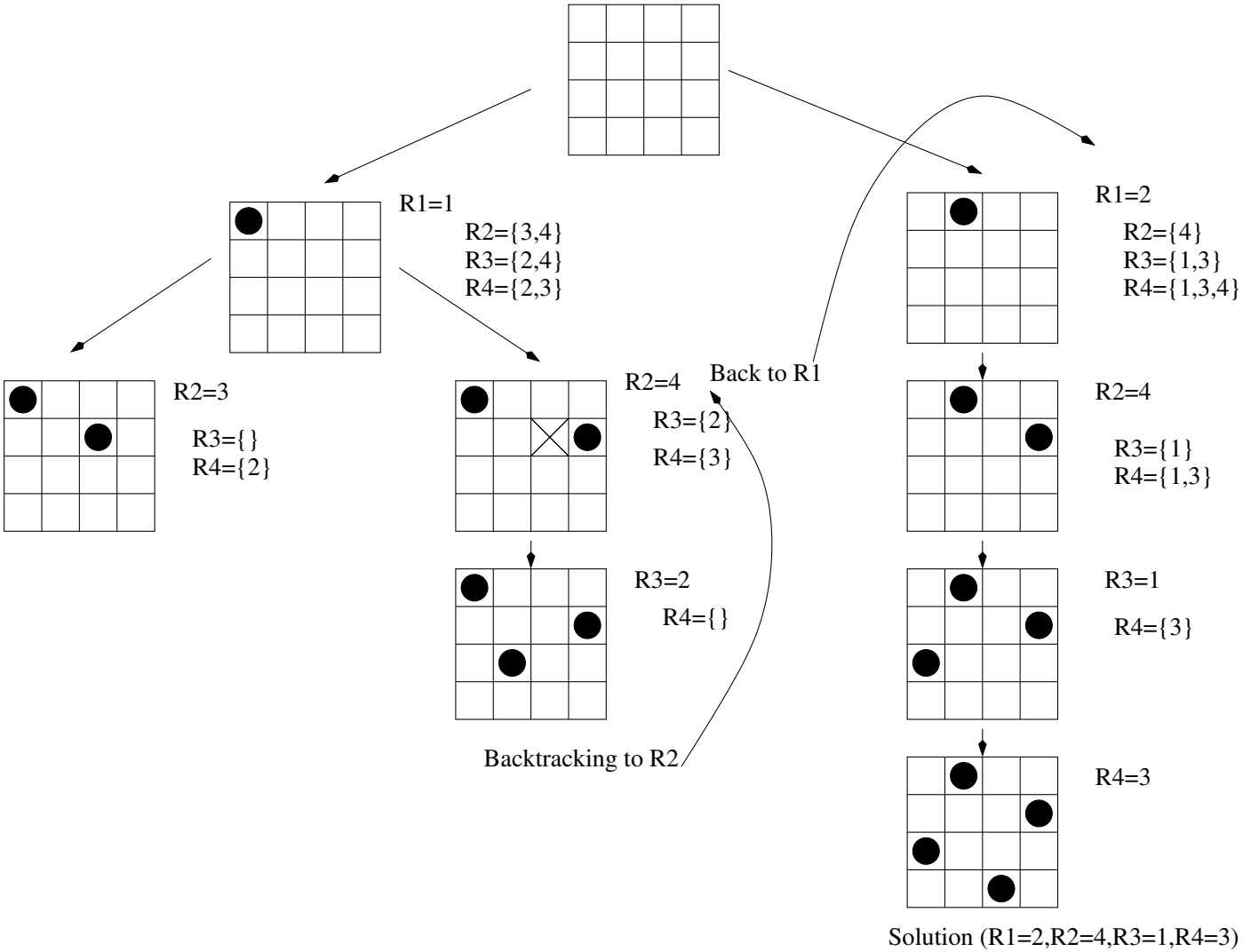
- Variation of the chronological backtracking depth-first algorithm (a propagation of constraints is performed after each variable instantiation)
- **Anticipation:** Detect and prune no solution paths as soon as possible
  - Assign a value and check the constraints with the future variables that have an edge from the current variable
  - Delete all the values that are not consistent from the domains of the future variables
- This es equivalent to make arc-consistent the current variable and the future variables each search iteration
- Efficiency depends on the problem (the cost of each iteration is increased)

Forward Checking Algorithm

Function: forward checking (vfuture, solution)

```
if vfuture.is_empty?() then
    return solution
else
    vcurrent ← vfuture.first()
    vfuture.delete_first()
    foreach v ∈ vcurrent.values() do
        vcurrent.assign(v)
        solution.add(vcurrent)
        vfuture.propagate_constraint(vcurrent) /* forward checking */
        if not vfuture.any_empty_domain?() then
            solution ← forward checking(vfuture,solution)
            if not solution.is_fail?()) then
                return solution
            else
                solution.delete(vcurrent)
        else
            solution.delete(vcurrent)
    return solution.fail()
```

Example: 4-queens solved by forward checking



Additional heuristics

Backtracking search can be improved

- Checking more restrictive consistency properties (more temporal cost)
  - Making all the problem arc-consistent each iteration (MAC Algorithm)
- Choosing the order to explore the variables
  - When?
    - Before the search (always the same order)
    - During the search (dynamic order)
  - What order?
    - First the variables with more constraints
    - First the variables with less values
  - Variable reordering can reduce the search time several orders of magnitude in some problems