A4M33MAS - Multiagent Systems
Distributed Constraint Satisfaction

Michal Pechoucek & Michal Jakob
Department of Computer Science
Czech Technical University in Prague

In parts based on Multi-agent Constraint Programming, Boi Faltings, Laboratoire d'Intelligence Artificielle, EPFL

Constraint Satisfaction Problem

Given $< X, D, C >$ where:

- $X = \{x_1, .., x_n\}$ is a set of n variables.
- $D = \{d_1, .., d_n\}$ is a set of n domains.
- $C = \{c_1, .., c_m\}$ is a set of m constraints.

Find solution $= (x_1 = v_1 \in d_1, ..., x_n = v_n \in d_n)$ such that for all constraints, value combinations are allowed by relations.

$=$ Assignment

$C =$ represented as a list of Boolean predicate on 1 ... n variables in X and their values from D, so that $\mathcal{P}(\mathrm{X}, D) \rightarrow \{0,1\}$

Multi-agent Constraint Satisfaction

Given $< X, D, C, A >$ where:

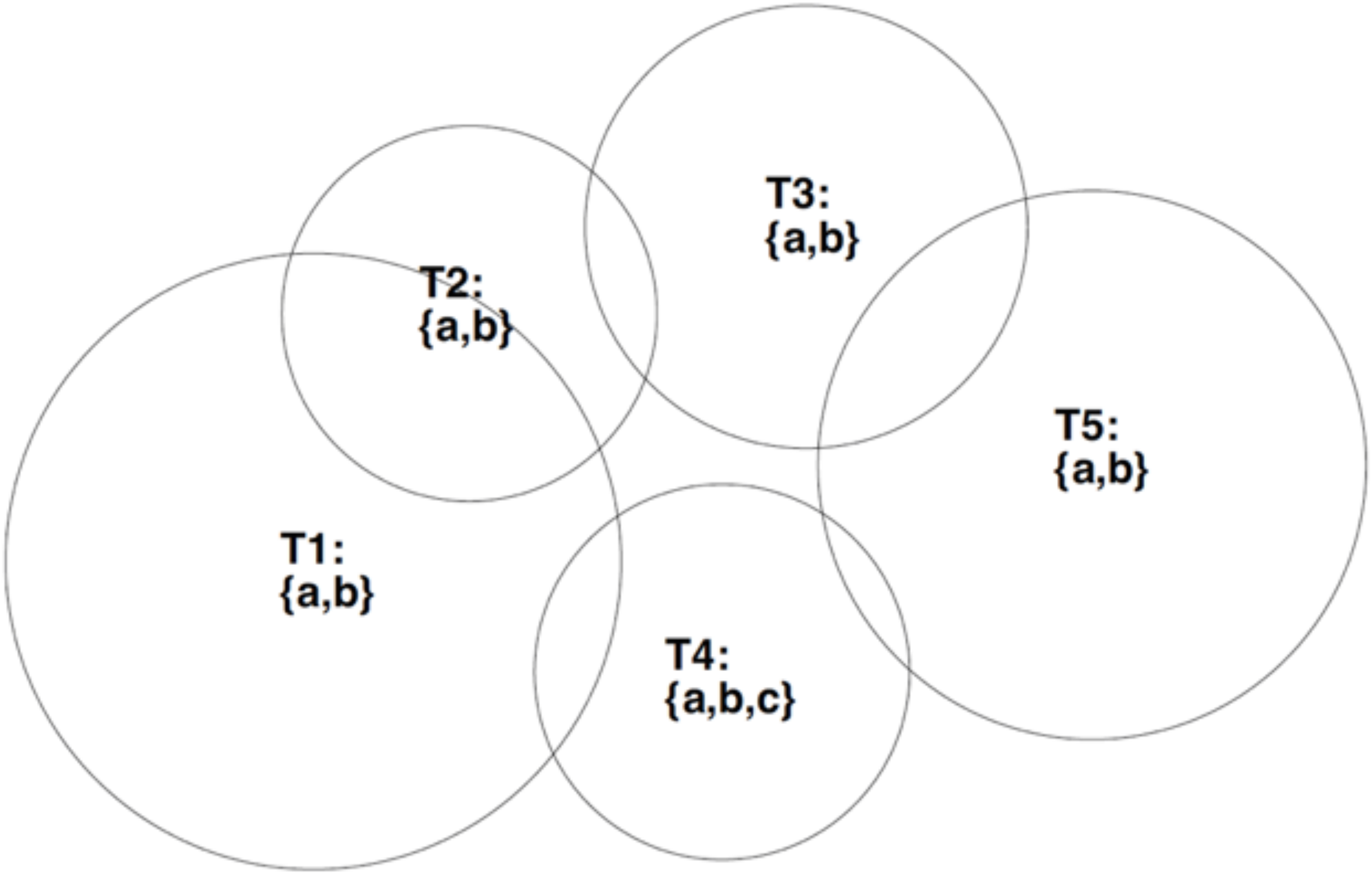
- $X = \{x_1, .., x_n\}$ is a set of n variables.
- $D = \{d_1, .., d_n\}$ is a set of n domains.
- $C = \{c_1, .., c_m\}$ is a set of m constraints.
- $A = \{a_1, .., a_n\}$ is a set of n agents, not necessarily all different.

Find solution $= (x_1 = v_1 \in d_1, ..., x_n = v_n \in d_n)$ such that for all constraints, value combinations are allowed by relations.

$=$ Assignment

$C =$ represented as a list of Boolean predicate on 1 … n variables in X and their values from D, so that $\mathcal{P}(\mathrm{X}, D) \rightarrow \{0,1\}$

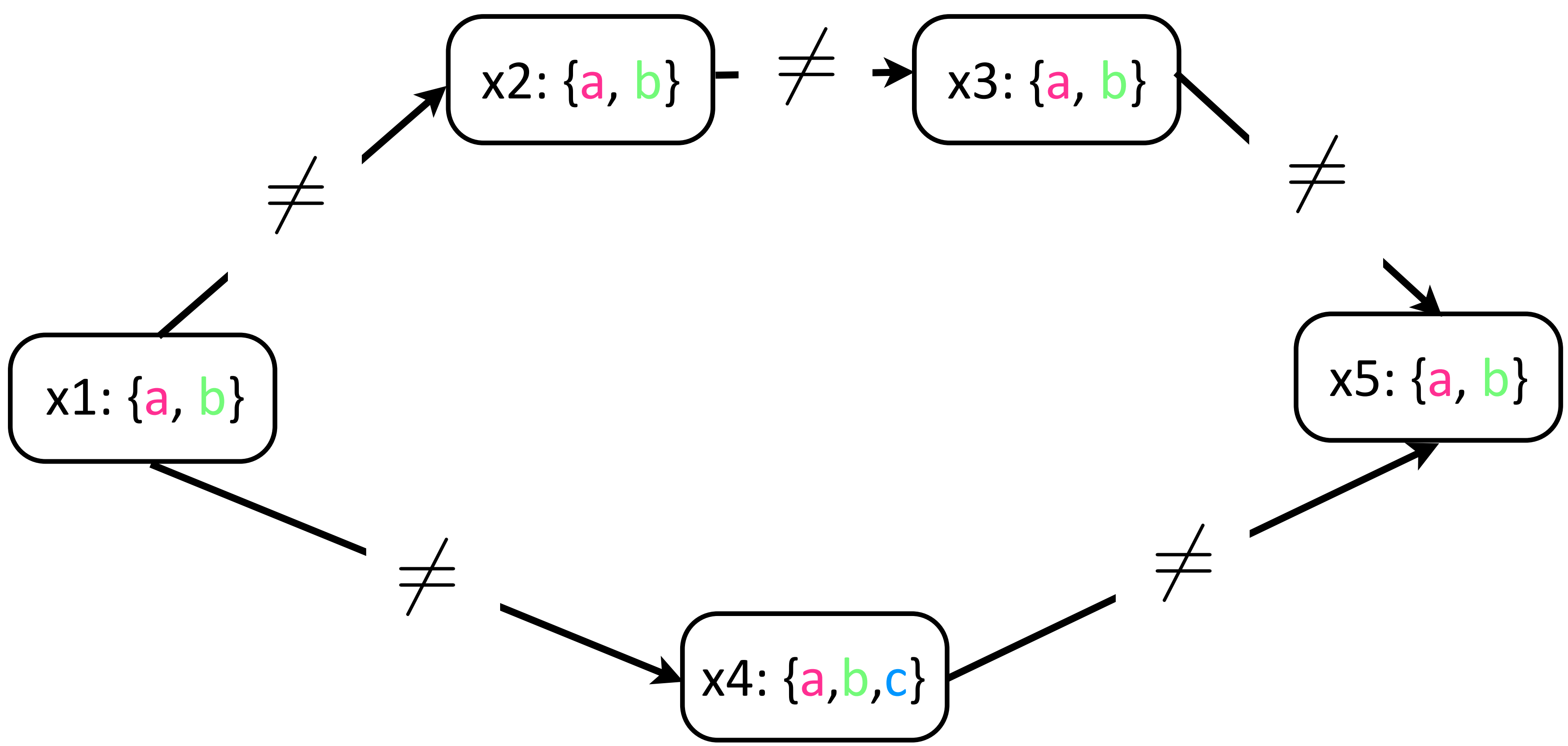
Example



Example

CSP model:

- Variables = choice of frequency
- Domains = frequency bands
- Constraints = inequalities between overlapping ranges
- Agents control transmitters



Multi-agent Constraint Satisfaction

Given $< X, D, C, A >$ where:

- $X = \{x_1, .., x_n\}$ is a set of n variables.
- $D = \{d_1, .., d_n\}$ is a set of n domains.
- $C = \{c_1, .., c_m\}$ is a set of m constraints.
- $A = \{a_1, .., a_n\}$ is a set of n agents, not necessarily all different.

Find solution $= (x_1 = v_1 \in d_1, ..., x_n = v_n \in d_n)$ such that for all constraints, value combinations are allowed by relations.

$=$ Assignment

$C =$ represented as a list of Boolean predicate on 1 ... n variables in X and their values from D, so that $\mathcal{P}(\mathrm{X}, D) \rightarrow \{0,1\}$

Multiagent Constraint Optimization

Given $< X, D, C, A >$ where:

- $X = \{x_1, .., x_n\}$ is a set of n variables.
- $D = \{d_1, .., d_n\}$ is a set of n domains.
- $C = \{c_1, .., c_m\}$ is a set of m constraints.
- $A = \{a_1, .., a_n\}$ is a set of n agents, not necessarily all different.

Find solution $= (x_1 = v_1 \in d_1, ..., x_n = v_n \in d_n)$ such that for all the overall cost of the assignment is minimized

$$\text{Cost}\ (\{v_1, .., v_n\}) = \sum_{\forall c_i \in C} c_i(\{v_1, .., v_n\})$$

$C =$ represented as a list of cost functions on 1 ... n variables in X and their values from D, so that $\mathcal{P}(X,D) \rightarrow \mathbf{R}$

Solving CSP

- Importance of CSP: large theory and tools for computing solutions
- 2 common methods:
  - backtrack search: assign one variable at a time, backtrack when no assignment without satisfying constraints.
  - local search: start with random assignment, make local changes to reduce number of constraint violations.

Multiagent/Distributed CSP & COP

- Problem is distributed in a network of *agents*.
- Each variable *belongs* to one agent who is responsible for setting its value (typically these are connected to complex local subproblems).
- Constraints are known to all agents with variables in it.
- Distributed $\neq$ parallel: distribution of variables to agents cannot be chosen to optimize performance.

- WHY?
  - Real world problems are distributed, no agreement on a common model.
  - Costly to formalize constraints and preferences for all possible cases.
  - No trusted third party, privacy concerns.
  - but generally not efficiency!

Multiagent/Distributed CSP & COP

- Top-down approaches:
  - **Pruning algorithms:** used mainly as a preprocessing step
    - *Filtering, Hyper-resolution*
  - **Search algorithms:**
    - *Chronological (Synchronous) Backtracking,*
    - *Asynchronous Backtracking, ADOPT*
- Bottom-up approaches:
    - *Distributed breakout*

Multiagent/Distributed CSP & COP

- Top-down approaches:
  - **Pruning algorithms:** used mainly as a preprocessing step
    - *Filtering, Hyper-resolution*
  - **Search algorithms:**
    - *Chronological (Synchronous) Backtracking,*
      - A few agents are active, most are waiting
      - Active agents take decisions with updated information
      - Low degree of concurrency / poor robustness
      - Algorithms: direct extensions of centralized ones
    - *Asynchronous Backtracking, ADOPT*
      - All agents are active simultaneously
      - Information is less updated, obsolescence appears
      - High degree of concurrency / robust approaches
      - Algorithms: new approaches

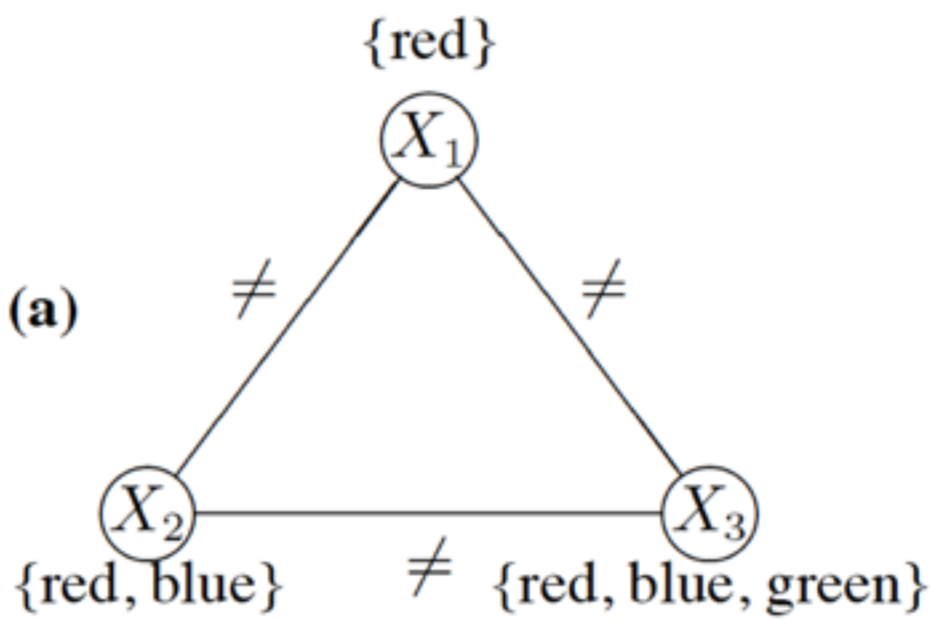
Domain Pruning Algorithms

- Filtering algorithm:
  - For each node x_i repeatadly execute **Revise**(x_i, x_j) with each neighbour x_j.

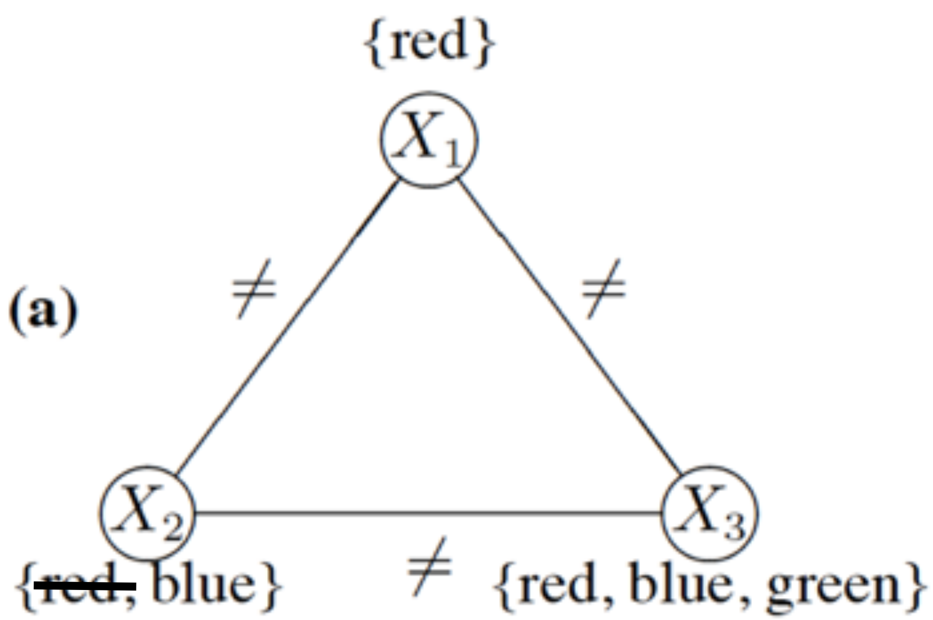
procedure Revise(x_i, x_j)
forall $v_i \in D_i$ **do**
  if *there is no value* $v_j \in D_j$ *such that* v_i *is consistent with* v_j **then**
    delete v_i from D_i

- Filtering terminates when no further elimination happens:
  - The solution is found if there is one value for each variable only
  - If there is an empty set assigned for one of the variables, -> no solution
  - If there is non-singleton set for one variable, the result is nonconlusive

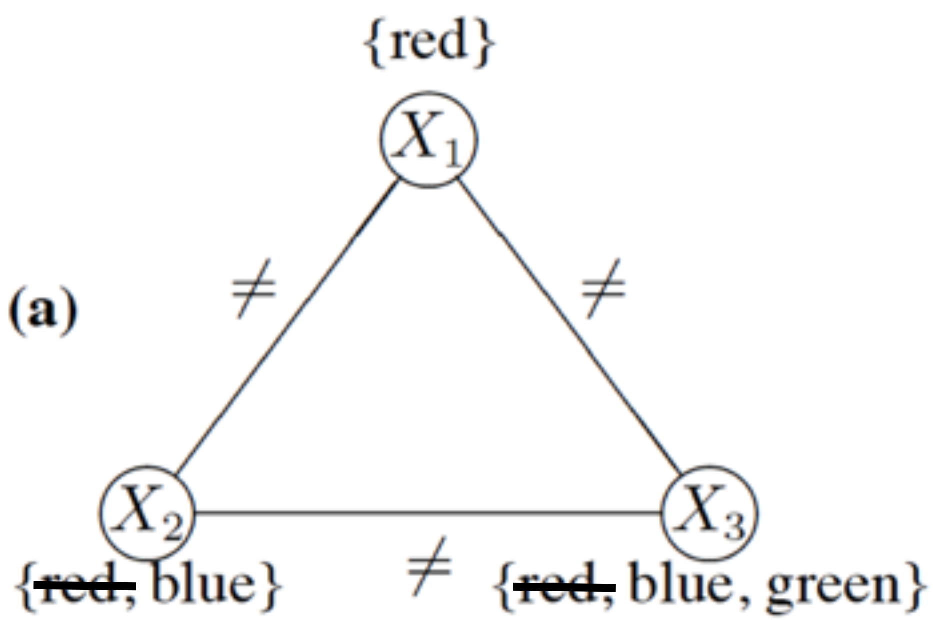
Domain Pruning Algorithms



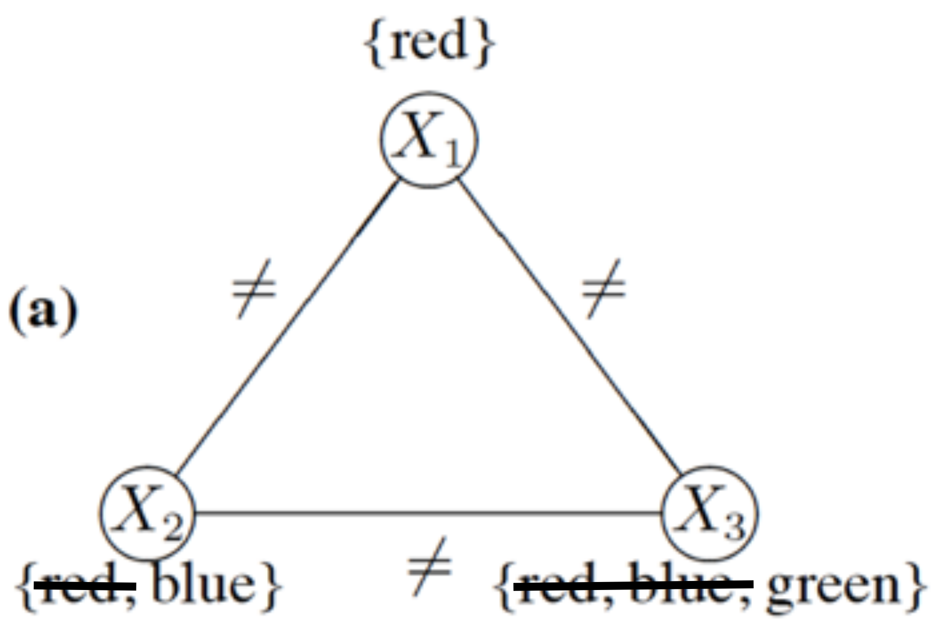
Domain Pruning Algorithms



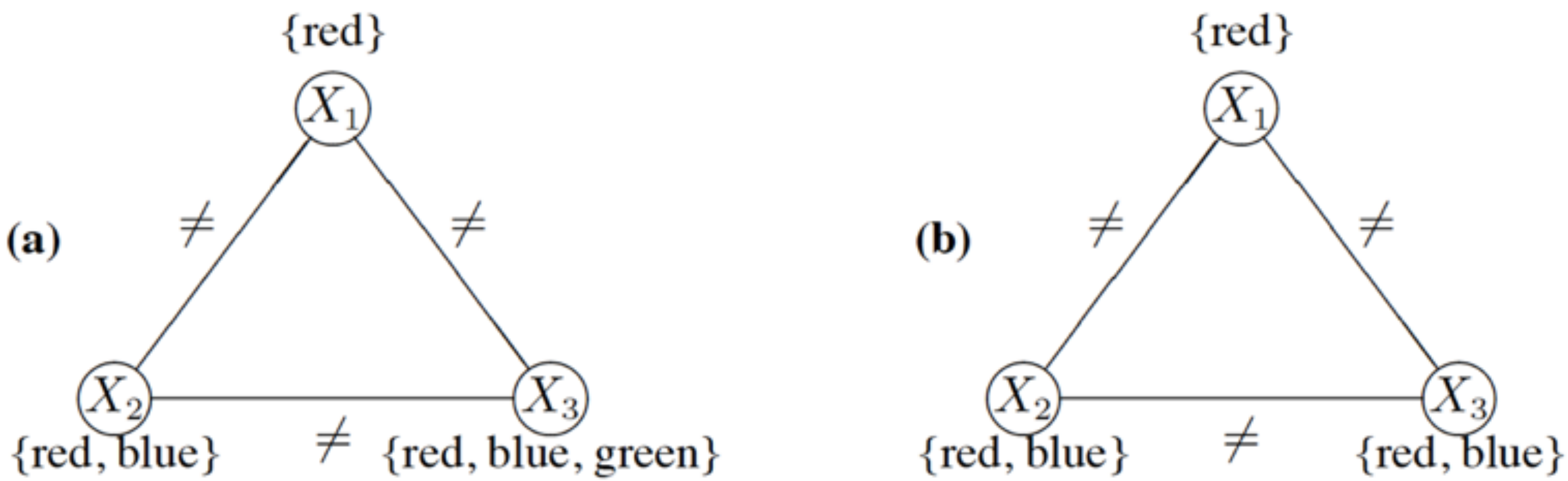
Domain Pruning Algorithms



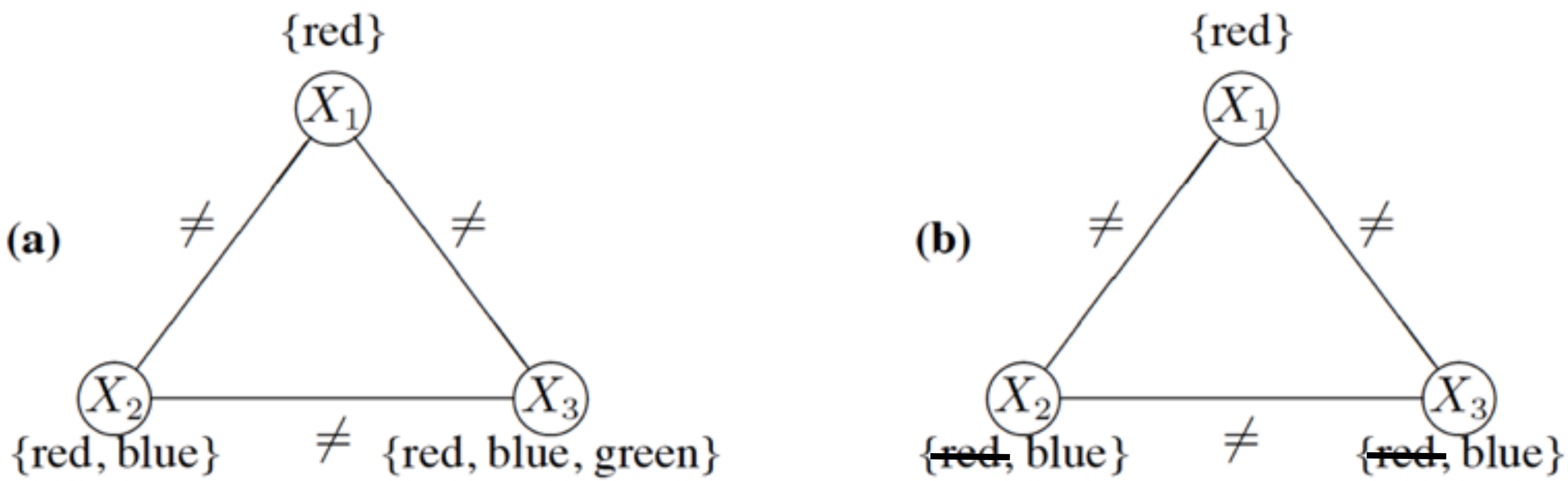
Domain Pruning Algorithms



Domain Pruning Algorithms

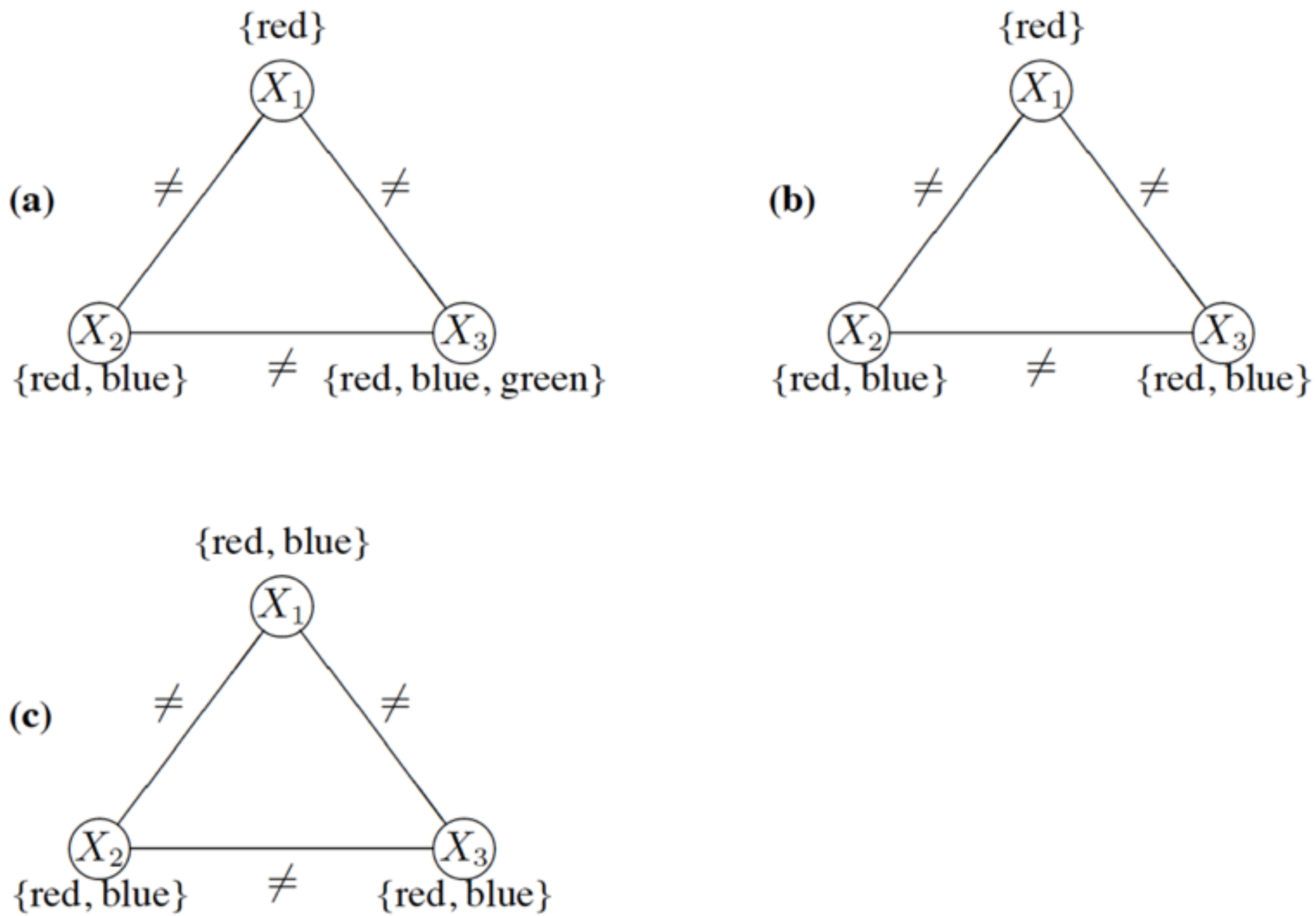


Domain Pruning Algorithms

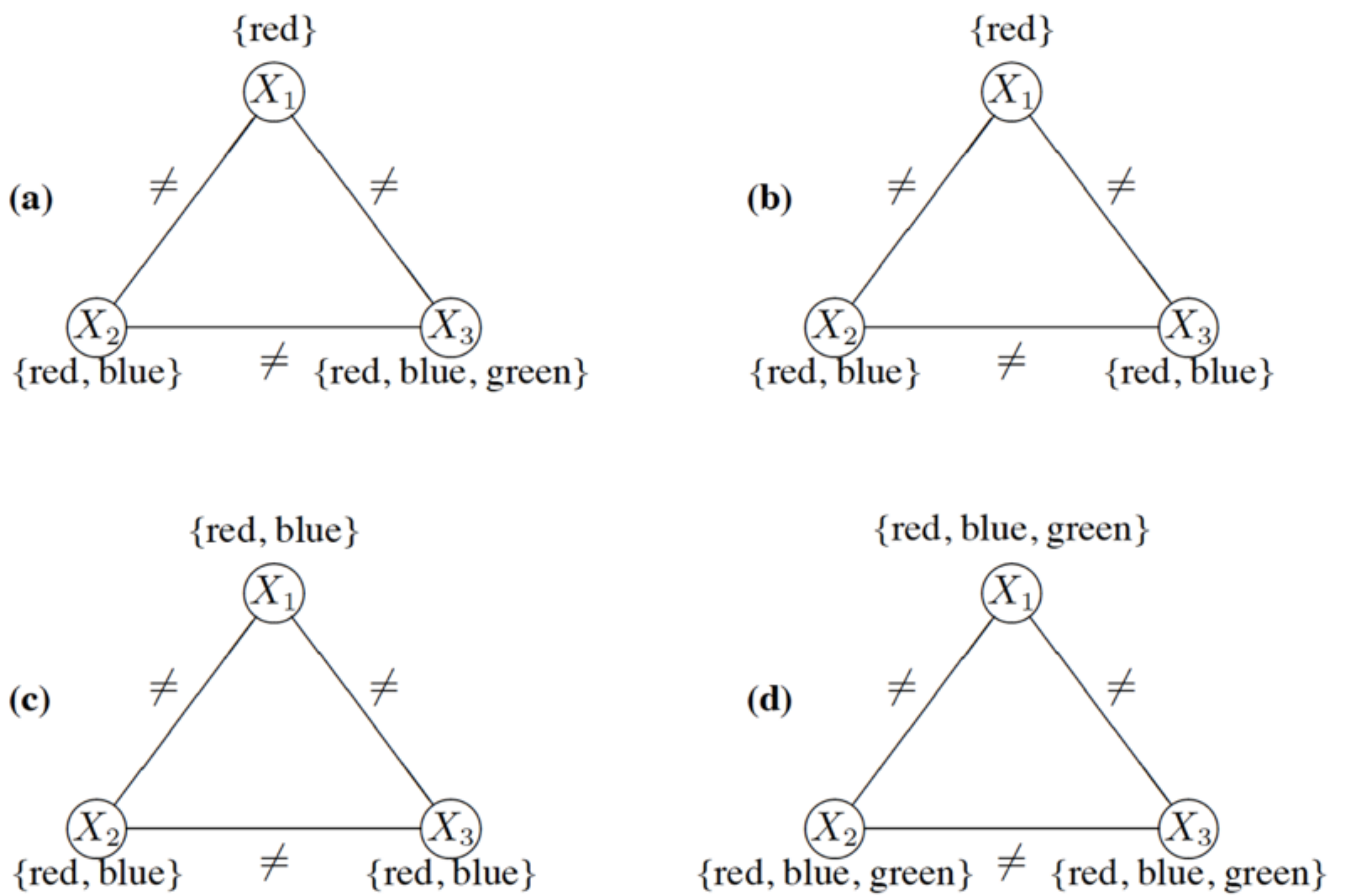


Domain Pruning Algorithms

01



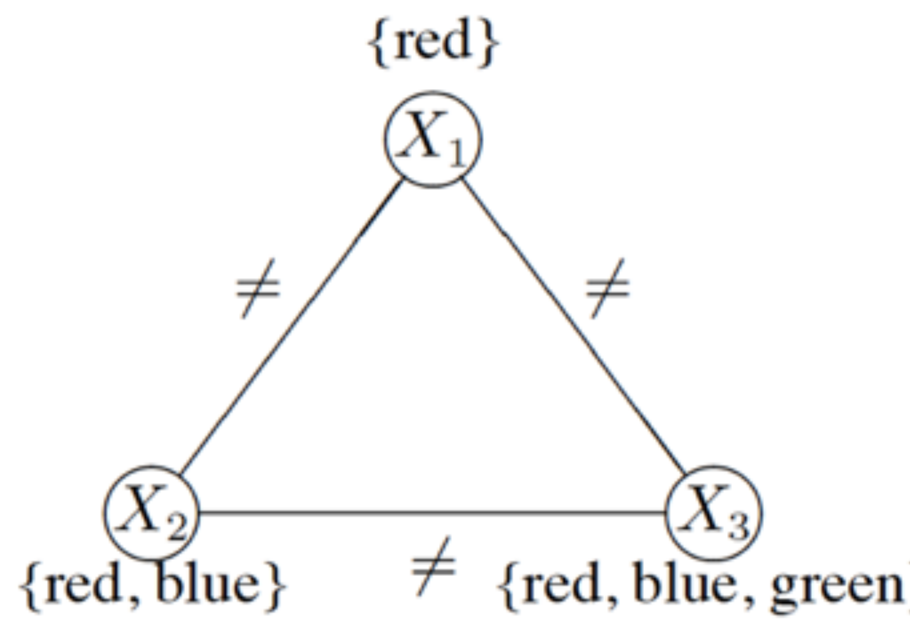
Domain Pruning Algorithms



Filtering based on hyper-resolution

- Works with the concept of forbidden combinations: NOGOOD (NG)
  - example: $\neg(x_1 = red \wedge x_2 = red)$
- Unit resolution:

$$\frac{\begin{array}{l} x_1 = red \\ \neg(x_1 = red \wedge x_2 = red) \end{array}}{\neg(x_2 = red)}$$



- Hyper-resolution:

$$\frac{\begin{array}{l} A_1 \vee A_2 \vee \cdots \vee A_m \\ \neg(A_1 \wedge A_{1,1} \wedge A_{1,2} \wedge \cdots) \\ \neg(A_2 \wedge A_{2,1} \wedge A_{2,2} \wedge \cdots) \\ \vdots \\ \neg(A_m \wedge A_{m,1} \wedge A_{m,2} \wedge \cdots) \end{array}}{\neg(A_{1,1} \wedge \cdots \wedge A_{2,1} \wedge \cdots \wedge A_{m,1} \wedge \cdots)}$$

Filtering based on hyper-resolution

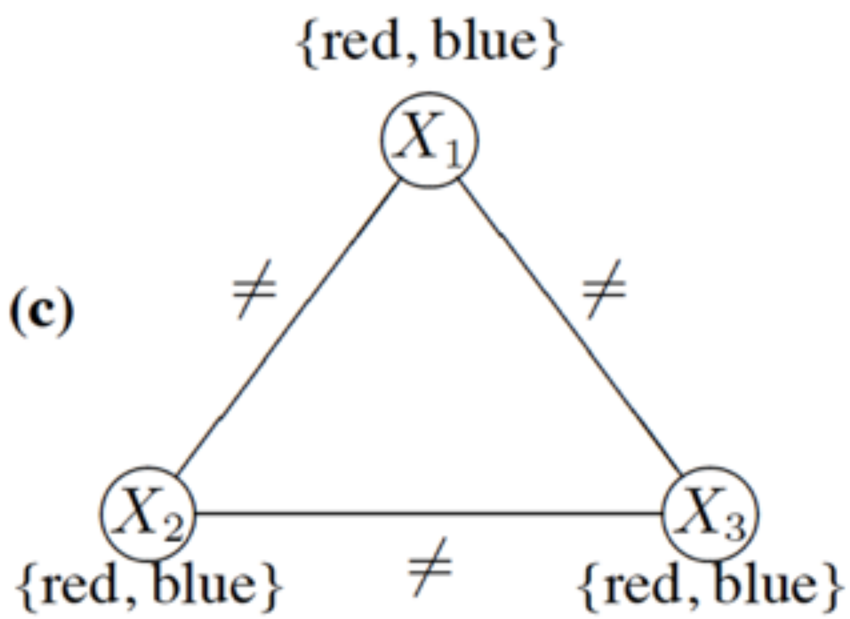
- Each agent repeatedly generates new constraints for his neighbors, notifies them of these new constraints, and prunes his own domain based on new constraints passed to him by his neighbors.

procedure ReviseHR(NG_i, NG_j^*)
repeat
    $NG_i \leftarrow NG_i \bigcup NG_j^*$
    let NG_i^* denote the set of new Nogoods that i can derive from NG_i and his domain using hyper-resolution
    if NG_i^* *is nonempty* **then**
        $NG_i \leftarrow NG_i \bigcup NG_i^*$
        send the Nogoods NG_i^* to all neighbors of i
        if $\{\} \in NG_i^*$ **then**
            stop
until *there is no change in i's set of Nogoods NG_i*

Filtering based on hyper-resolution

- As hyper-resolution is sound and complete for propositional logic it gave a rise to an efficient while complete distributed CSP algorithm.
- The algorithm is guaranteed to converge in the sense that after sending and receiving a finite number of messages, each agent will stop sending messages and generating Nogoods.

Filtering based on hyper-resolution



NOGOODS:

$$\{x_1 = red, x_2 = red\}, \{x_1 = red, x_3 = red\}$$
$$\{x_1 = blue, x_2 = blue\}, \{x_1 = blue, x_3 = blue\}$$

$$\begin{array}{l} x_1 = red \vee x_1 = blue \\ \neg(x_1 = red \wedge x_2 = red) \\ \neg(x_1 = blue \wedge x_3 = blue) \\ \hline \neg(x_2 = red \wedge x_3 = blue) \end{array}$$

NOGOODS:

$$\{x_2 = red, x_3 = blue\}\ \{x_2 = blue, x_3 = red\}.$$

$$\begin{array}{l} x_2 = red \vee x_2 = blue \\ \neg(x_2 = red \wedge x_3 = blue) \\ \neg(x_2 = blue \wedge x_3 = blue) \\ \hline \neg(x_3 = blue) \end{array}$$

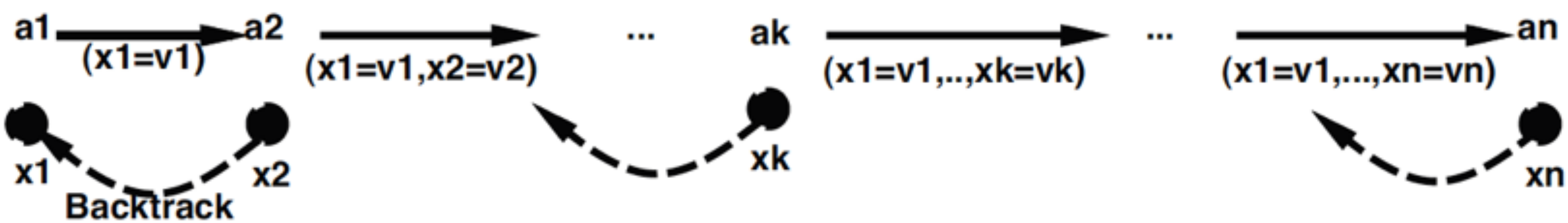
similarly:

$$\neg(x_3 = red)$$

Chronological Backtracking

Agents agree on an variable order and repeat:

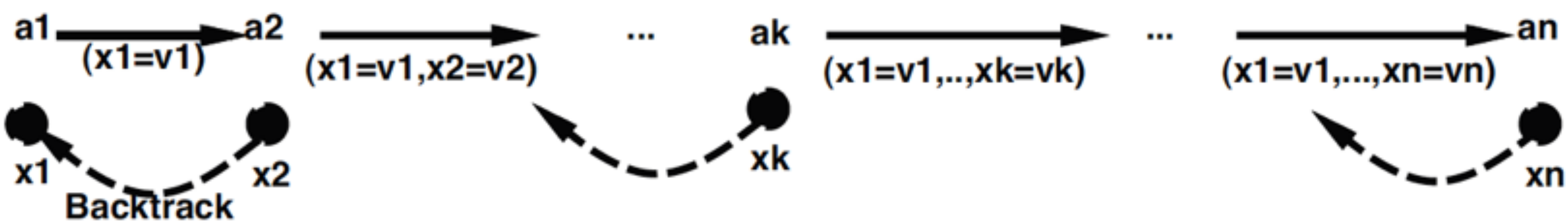
1. send partial solution up to x_{k-1} to k-th agent.
2. k-th agent generates the next extension to this partial solution.
3. if solution cannot be extended consistently, $k \leftarrow k - 1$.
4. if solution can be extended consistently, $k \leftarrow k + 1$.
5. if $k < 1$, stop: unsolvable.
6. if $k > n$, assigment = solution.



Chronological Backtracking

Agents agree on an variable order and repeat:

1. send partial solution up to x_{k-1} to k-th agent.
2. k-th agent generates the next extension to this partial solution.
3. if solution cannot be extended consistently, $k \leftarrow k - 1$.
4. if solution can be extended consistently, $k \leftarrow k + 1$.
5. if $k < 1$, stop: unsolvable.
6. if $k > n$, assigment $=$ solution.



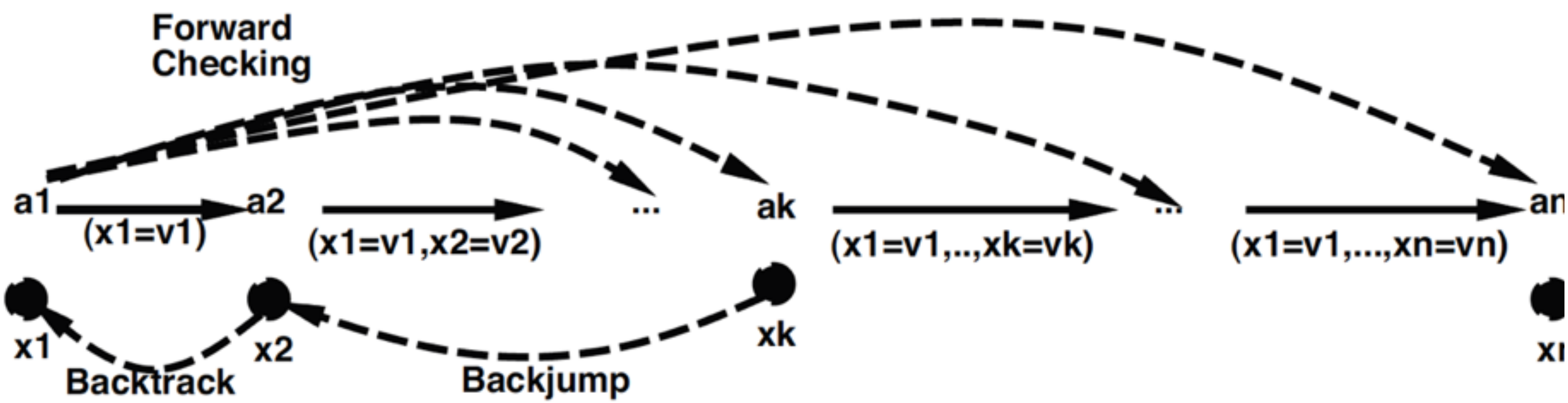
Towards optimization: Synchronous branch-bounds

- Extend synchronous backtracking to optimization
  - every constraint contributes a cost.
  - upper bound = lowest cost of full assignment found so far.
  - partial assignment extended while cost < upper bound.
  - result = solution with lowest cost

Improvements

Synchronous backtracking allows common CSP heuristics:

- forward checking: send partial solution to all higher agents.
- dynamic variable ordering: select next variable according to domain size.
- backjumping: reduce k to last variable involved in conflict.



Improvements

Distributed forward checking:

- $A(x_k)$ sends $(x_1 = v_1, .., x_k = v_k)$ to all $A(x_j)$, $j > k$
- $A(x_j)$ removes inconsistent values and initiates backtrack at x_k whenever domain becomes empty

Can be done aynchronously (asynchronous forward checking)

Dynamic variable ordering:

- $A(x_j)$ sends back size of remaining domain for x_j
- $A(x_k)$ chooses smallest one to be x_{k+1}

Backjumping:

reduce k to last variable involved in current conflict.

Performance metrics

- non-concurrent constraint checks (NCCC): longest chain of constraint checks with serial dependency (ignores message delivery time).
- concurrent time: (simulated) time taken in parallel execution.
- wall clock time (time taken by the simulator).
- number of messages (ignores computation time and size of messages).
- amount of information exchanged (ignores computation time).

Asynchronous backtracking (ABT)

- Assumptions:
  - Agents communicate by sending messages, agent send messages to others, iff it knows their identifiers (directed communication/no broadcasting)
  - The delay transmitting a message is finite but random, for any pair of agents, messages are delivered in the order they were sent
  - Agents know only the constraints in which they are involved
  - Each agent owns a single variable, constraints are binary
  - Asynchronous algorithm: Agents work in parallel without synchronization.
    - *all agents active, take a value and inform*
    - *no agent has to wait for other agents*
  - Global priority ordering among variables, and agents (to avoid cycles)
  - Constraints are directed: from higher-priority to lower-priority agents
- ABT plays in asynchronous distributed context the same role as backtracking in centralized

ABT: Core principles

- Higher priority agent (j) informs the lower one (k) of its assignment
- Lower priority agent (k) evaluates the constraint with its own assignment
  - If permitted: no action
  - else: look for a value consistent with j
    - *If it exists k takes that value*
    - *else the agent view of k becomes a* NOGOOD (constraint) & *backtrack*

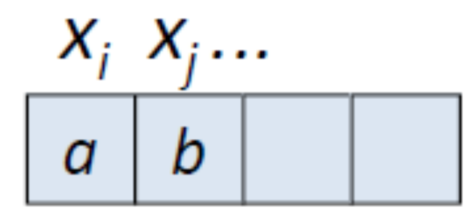
- NOGOOD: conjunction of (variable, value) pairs of higher priority agents, which removes a value of the current one
  - are required to ensure systematic traversal of search space in asynchronous, distributed context

How ABT operates?

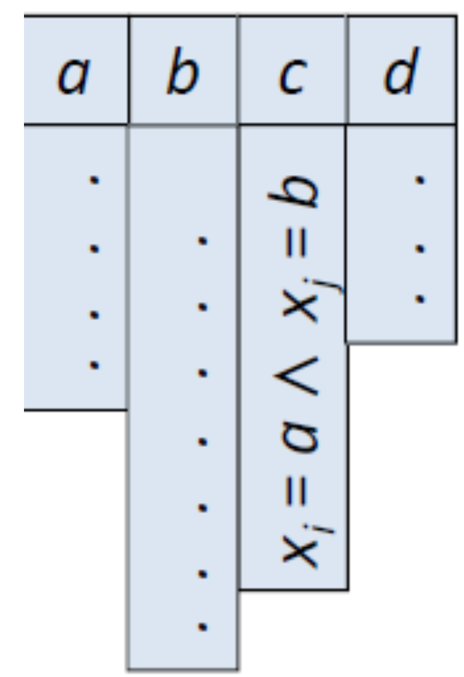
- ABT agents: asynchronous action; spontaneous assignment
- **Assignment**: *j* takes value *a*, *j* informs lower priority agents
- **Backtrack**: *k* has no consistent values with higher-priority agents, *k* resolves nogoods and sends a backtrack message
- **New links**: *j* receives a nogood mentioning *i*, unconnected with *j*; *j* asks *i* to set up a link
- **Stop:** "no solution" detected by an agent, stop
- **Solution:** when agents are silent for a while (quiescence), every constraint is satisfied => solution; detected by specialized algorithms

ABT Data Structures

- **AgentView** (current assignment context):
  - values of higher-priority constrained agents



- **NOGOOD store**: each removed value has justifying NOGOOD
  - stored NOGOOD must be active wrt to AgentView



ABT message passing

when *received* (***Ok?***, (A_j, d_j)) **do**

  add (A_j, d_j) to *agent_view*

  check_agent_view

when *received* (***Nogood***, nogood) **do**

  add *nogood* to Nogood list

  forall $(A_k, d_k) \in$ nogood, *if* A_k *is not a neighbor of* A_i **do**

    add (A_k, d_k) to *agent_view*

    request A_k to add A_i as a neighbor

  check_agent_view

procedure check_agent_view

when agent_view *and* current_value *are inconsistent* **do**

  if *no value in* D_i *is consistent with* agent_view **then**

    backtrack

  else

    select $d \in D_i$ consistent with *agent_view*

    current_value $\leftarrow d$

    send (**ok?**, (A_i, d)) to lower-priority neighbors

ABT message passing

procedure backtrack

nogood ← some inconsistent set, using hyper-resolution or similar procedure

if nogood *is the empty set* **then**

  broadcast to other agents that there is no solution

  terminate this algorithm

else

  select $(A_j, d_j) \in$ *nogood* where A_j has the lowest priority in *nogood*

  send (**Nogood**, *nogood*) to A_j

  remove (A_j, d_j) from *agent_view*

  check_agent_view

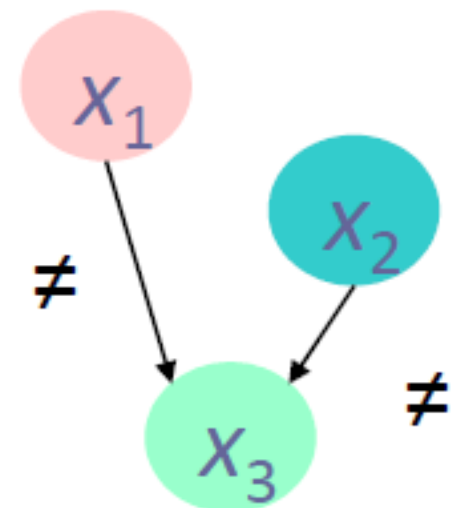
Example

Variables x_1, x_2, x_3; $D_1 = \{b, a\}$, $D_2 = \{a\}$, $D_3 = \{a, b\}$

3 agents, lex ordered:

2 difference constraints: c_{13} and c_{23}

Constraint graph:

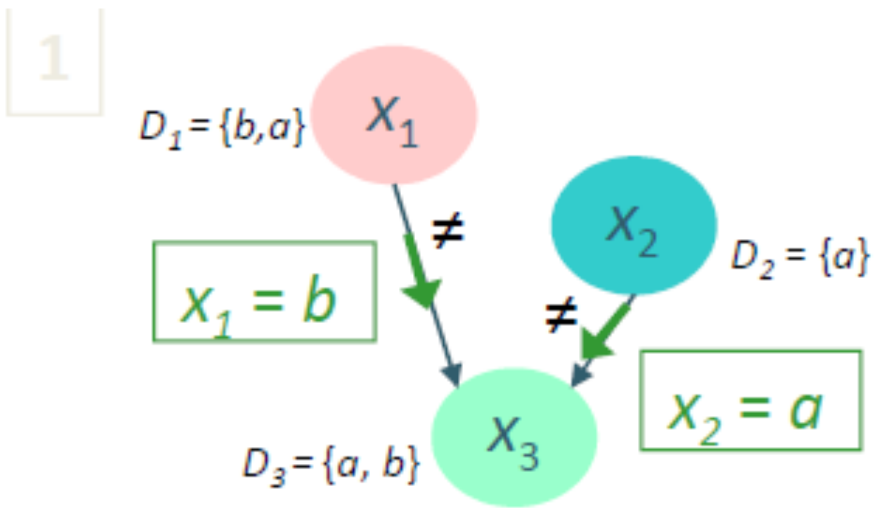


Value-sending agents: x_1 and x_2

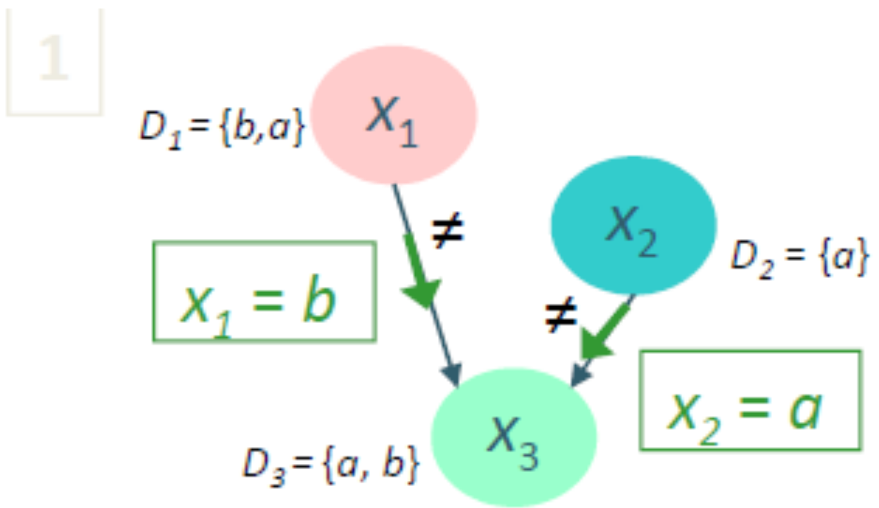
Constraint-evaluating agent: x_3

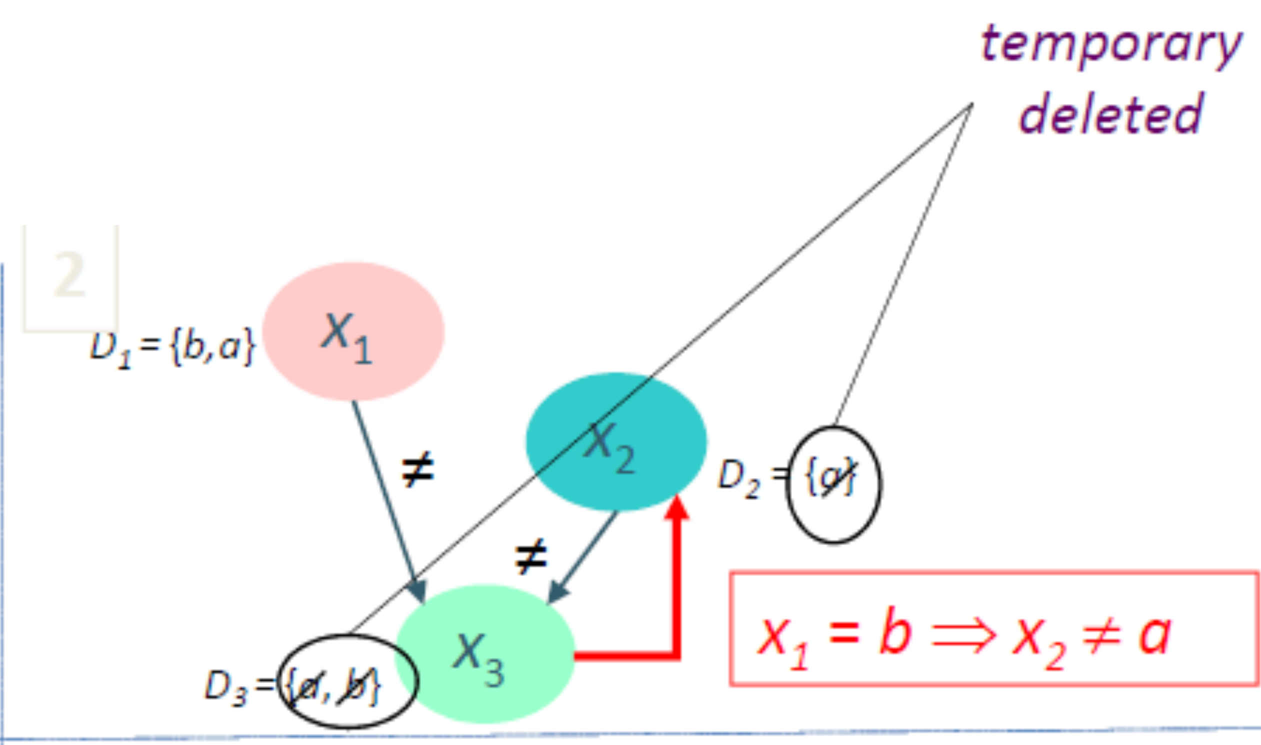
Each agent *checks* constraints of incoming links: $Agent_1$ and $Agent_2$ check nothing, $Agent_3$ checks c_{13} and c_{23}

Example

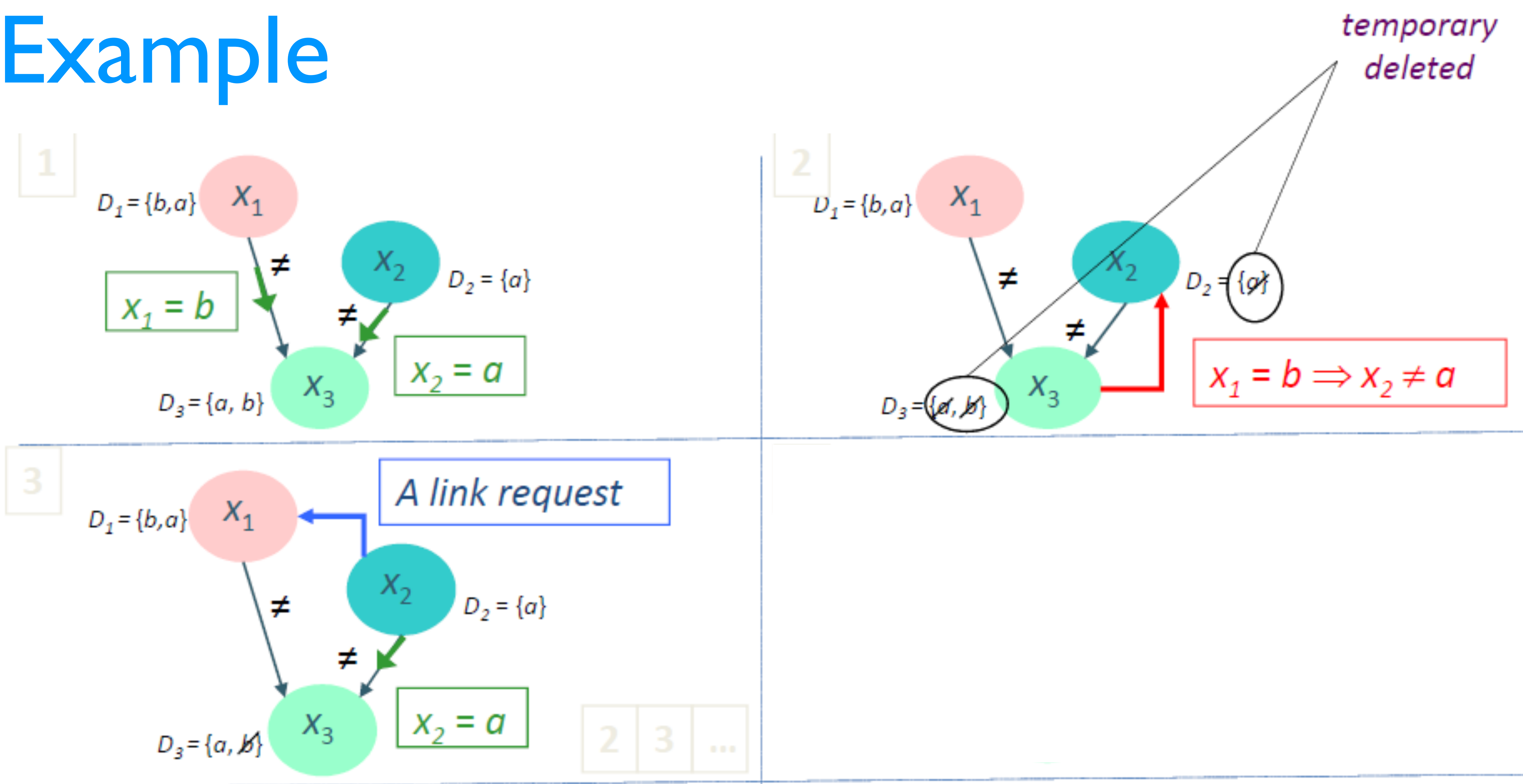


Example

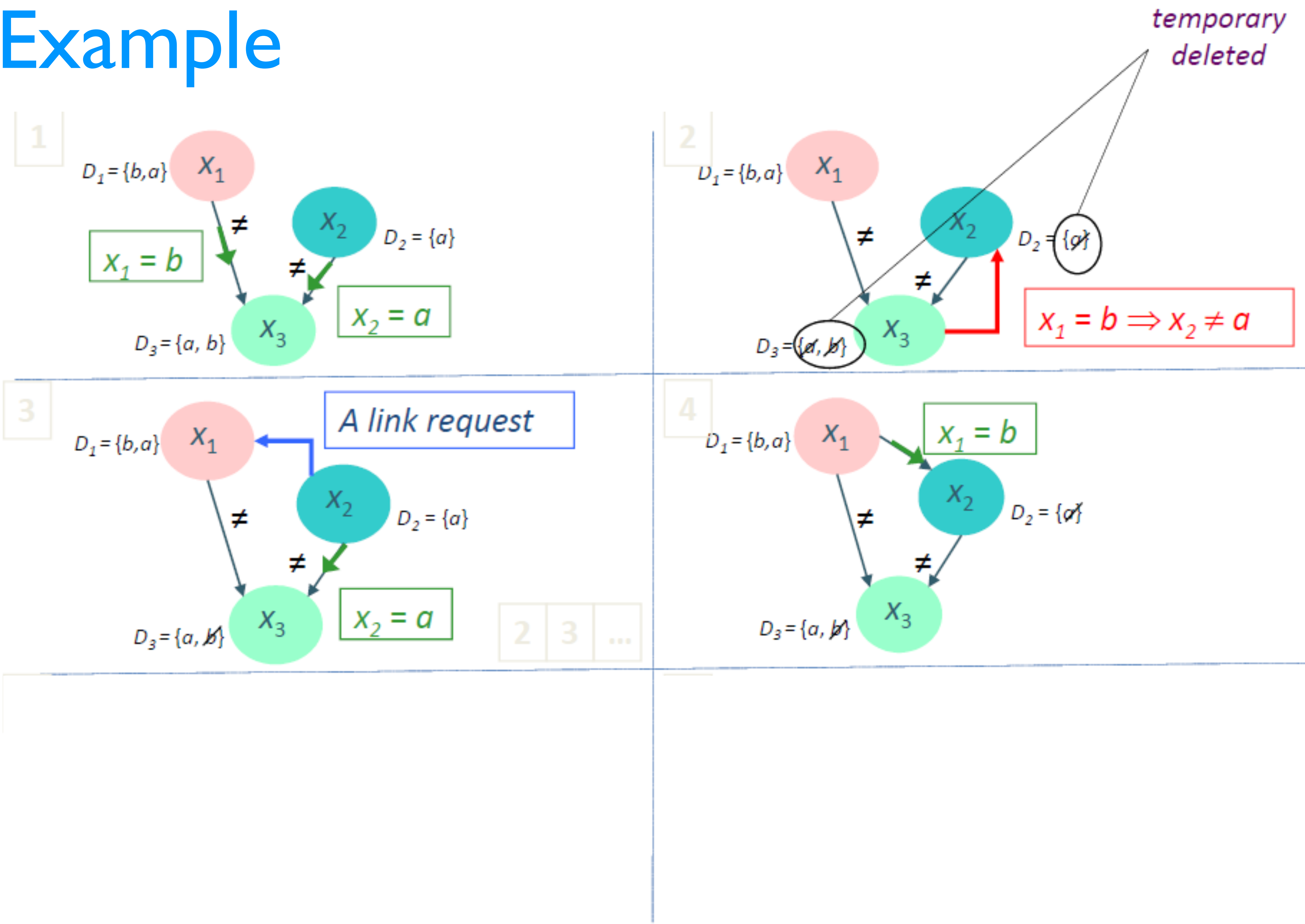

1
$D_1 = \{b,a\}$
X_1
X_2
$D_2 = \{a\}$
$x_1 = b$
$x_2 = a$
X_3
$D_3 = \{a, b\}$


temporary deleted
2
$D_1 = \{b,a\}$
X_1
X_2
$D_2 = \{\cancel{a}\}$
X_3
$D_3 = \{\cancel{a}, \cancel{b}\}$
$x_1 = b \Rightarrow x_2 \neq a$

Example

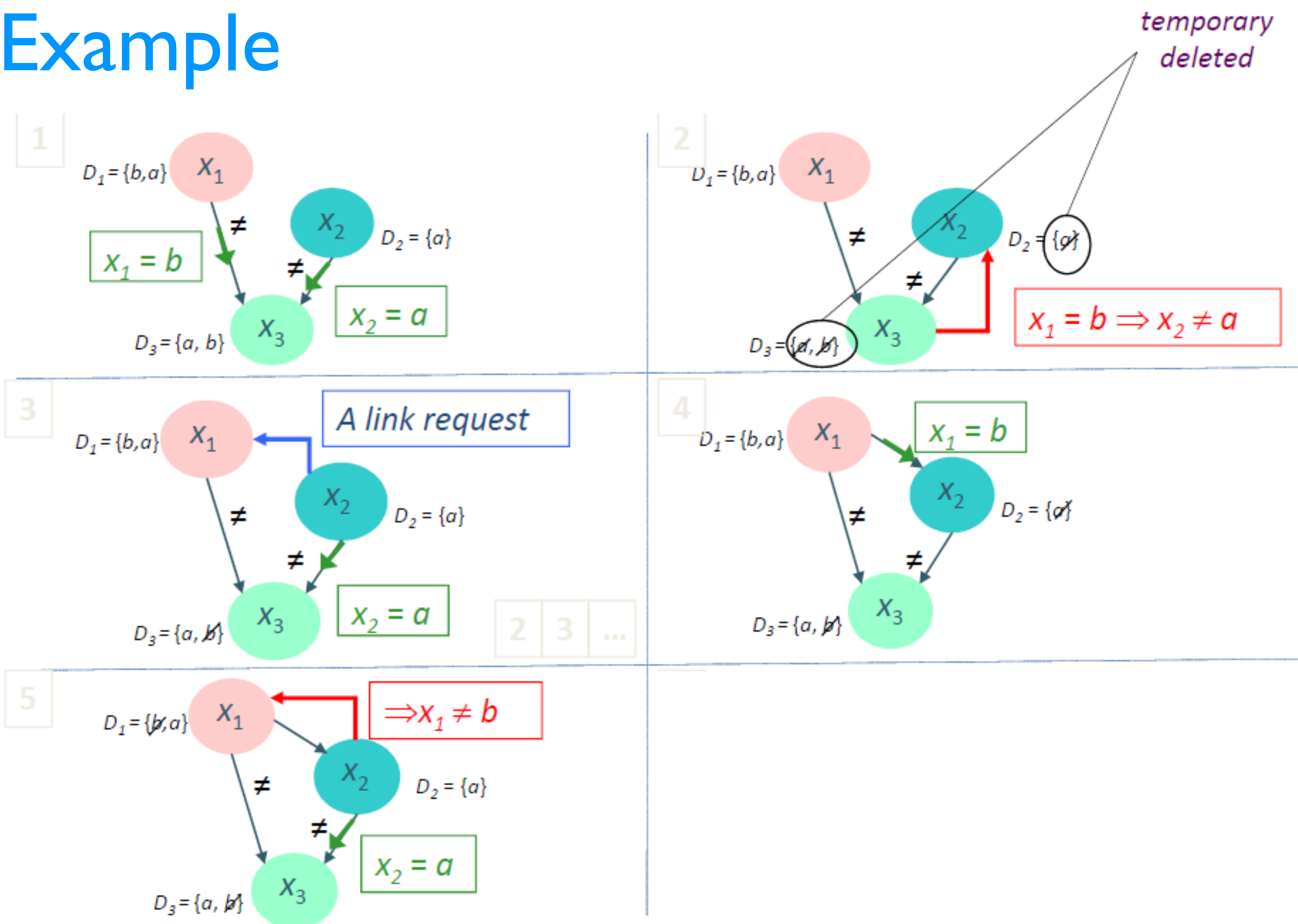

temporary
deleted
1
$D_1 = \{b, a\}$
x_1
x_2
x_3
$D_2 = \{a\}$
$D_3 = \{a, b\}$
$x_1 = b$
$x_2 = a$
2
$D_1 = \{b, a\}$
$D_2 = \{\cancel{a}\}$
$D_3 = \{\cancel{a}, \cancel{b}\}$
$x_1 = b \Rightarrow x_2 \neq a$
3
A link request
$D_1 = \{b, a\}$
$D_2 = \{a\}$
$D_3 = \{a, \cancel{b}\}$
$x_2 = a$
2
3
...

Example



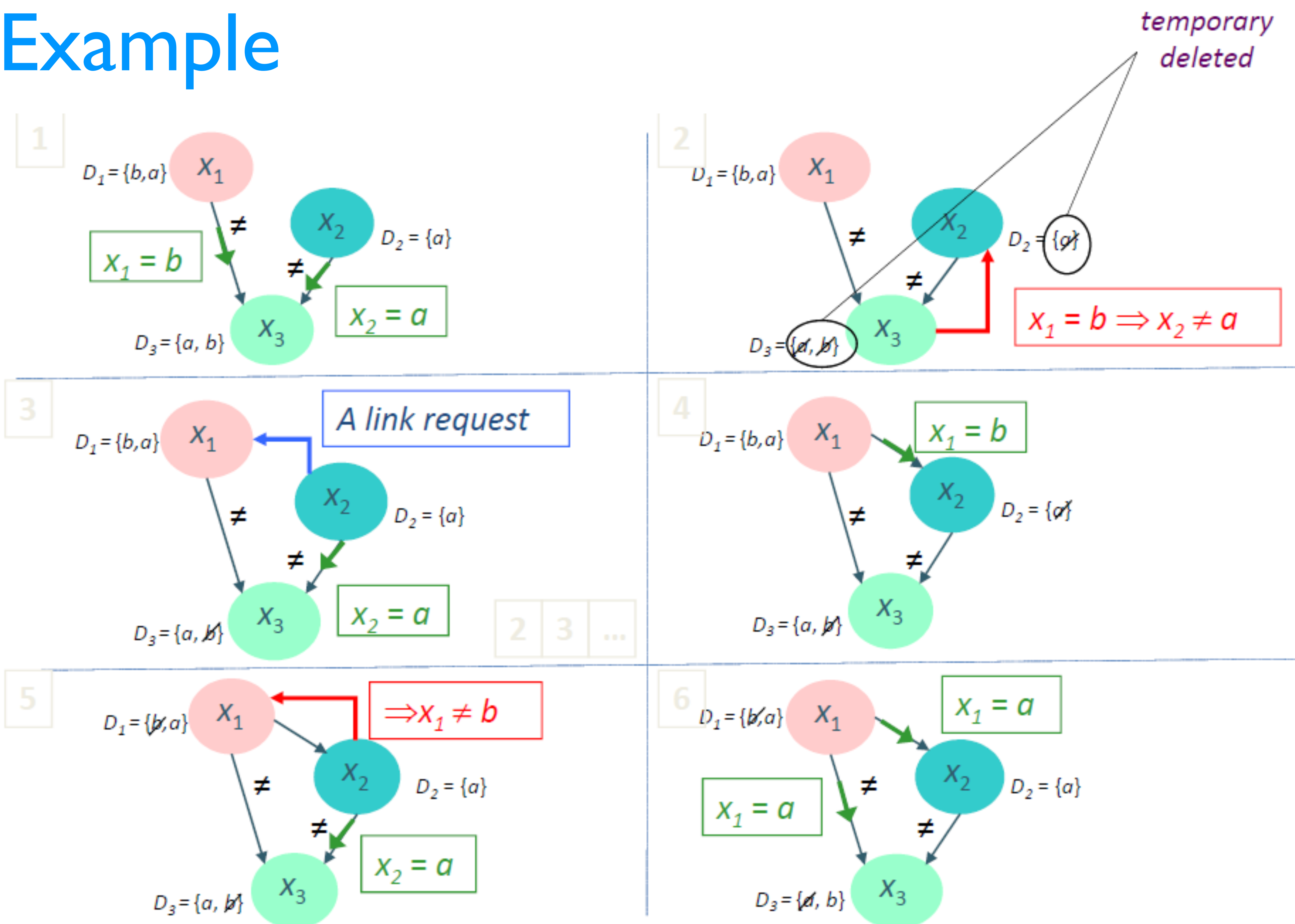
temporary deleted
1
D1 = {b,a}
X1
D2 = {a}
X2
x1 = b
≠
≠
x2 = a
D3 = {a, b}
X3
2
D1 = {b,a}
X1
X2
D2 = {a}
D3 = {a, b}
X3
x1 = b ⇒ x2 ≠ a
3
D1 = {b,a}
X1
A link request
X2
D2 = {a}
D3 = {a, b}
X3
x2 = a
2
3
...
4
D1 = {b,a}
X1
x1 = b
X2
D2 = {a}
D3 = {a, b}
X3

Example

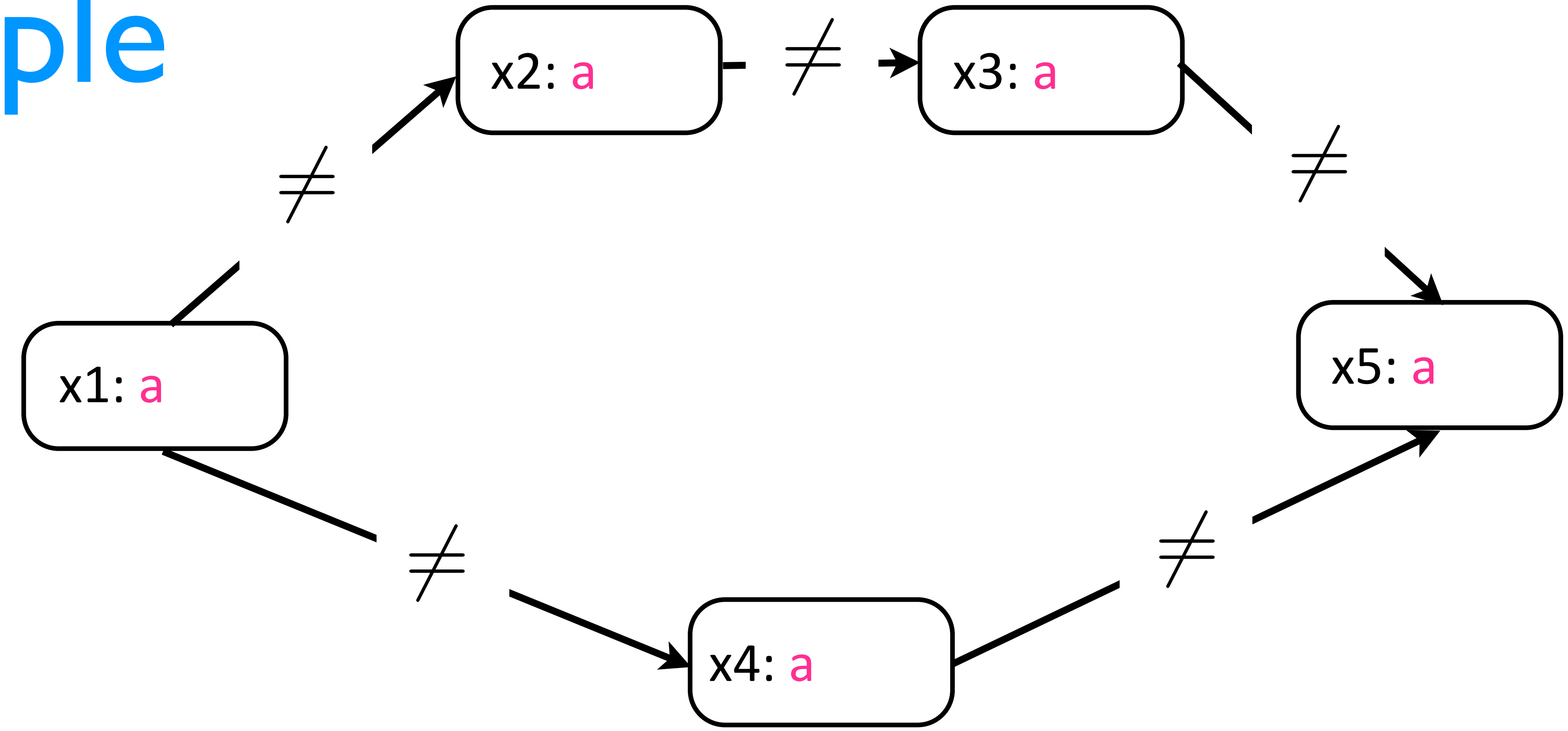


temporary
deleted
1
$D_1 = \{b,a\}$
X_1
X_2
$D_2 = \{a\}$
$x_1 = b$
$x_2 = a$
X_3
$D_3 = \{a, b\}$
2
$D_1 = \{b,a\}$
$D_2 = \{\cancel{a}\}$
$D_3 = \{\cancel{a}, \cancel{b}\}$
$x_1 = b \Rightarrow x_2 \neq a$
3
A link request
$D_1 = \{b,a\}$
$D_2 = \{a\}$
$x_2 = a$
$D_3 = \{a, \cancel{b}\}$
2 3 ...
4
$x_1 = b$
$D_1 = \{b,a\}$
$D_2 = \{\cancel{a}\}$
$D_3 = \{a, \cancel{b}\}$
5
$\Rightarrow x_1 \neq b$
$D_1 = \{\cancel{b},a\}$
$D_2 = \{a\}$
$x_2 = a$
$D_3 = \{a, \cancel{b}\}$

Example

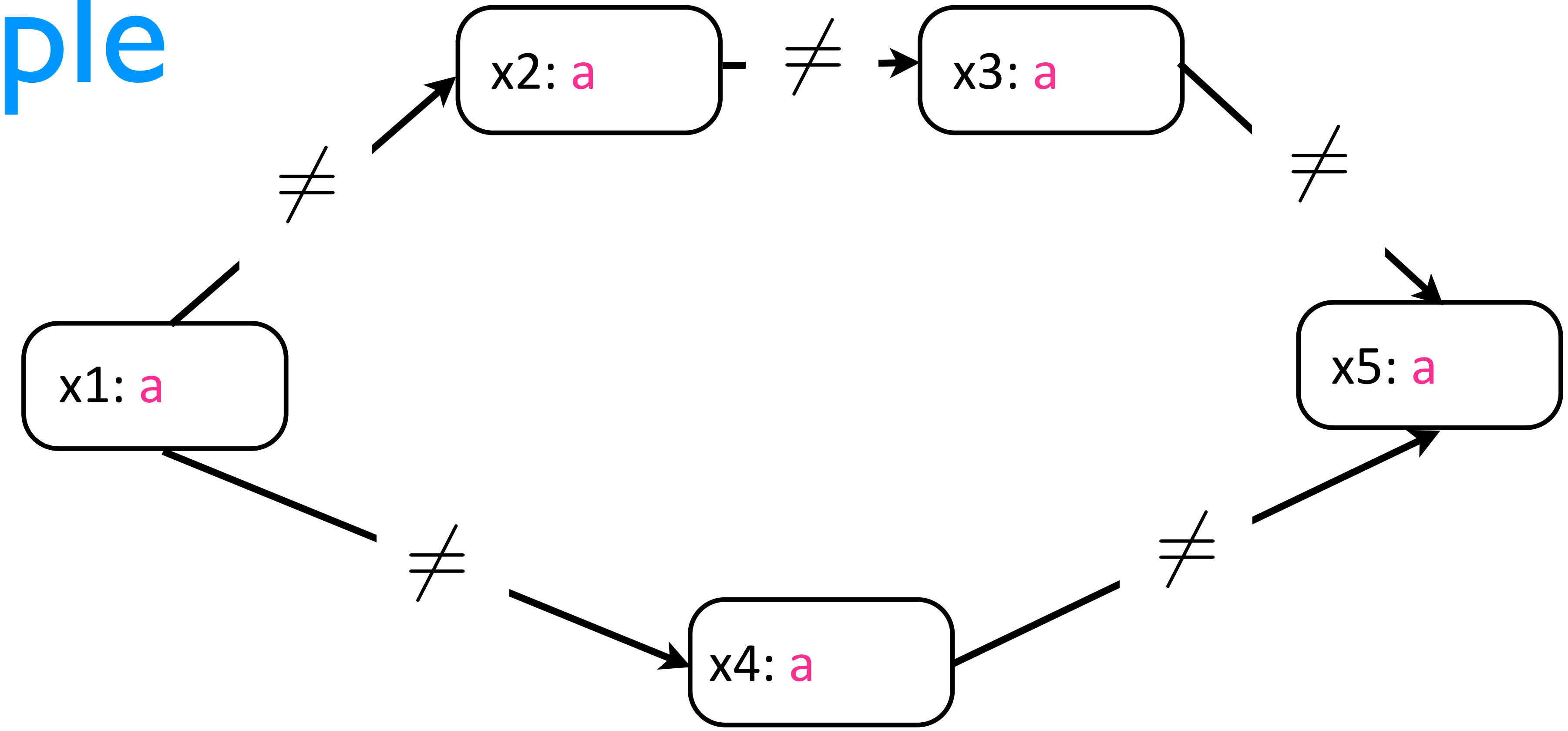

temporary
deleted
1
$D_1 = \{b, a\}$
X_1
X_2
$D_2 = \{a\}$
X_3
$D_3 = \{a, b\}$
$x_1 = b$
$x_2 = a$
2
$x_1 = b \Rightarrow x_2 \neq a$
3
A link request
4
5
$\Rightarrow x_1 \neq b$
6
$x_1 = a$
2
3
...

Example



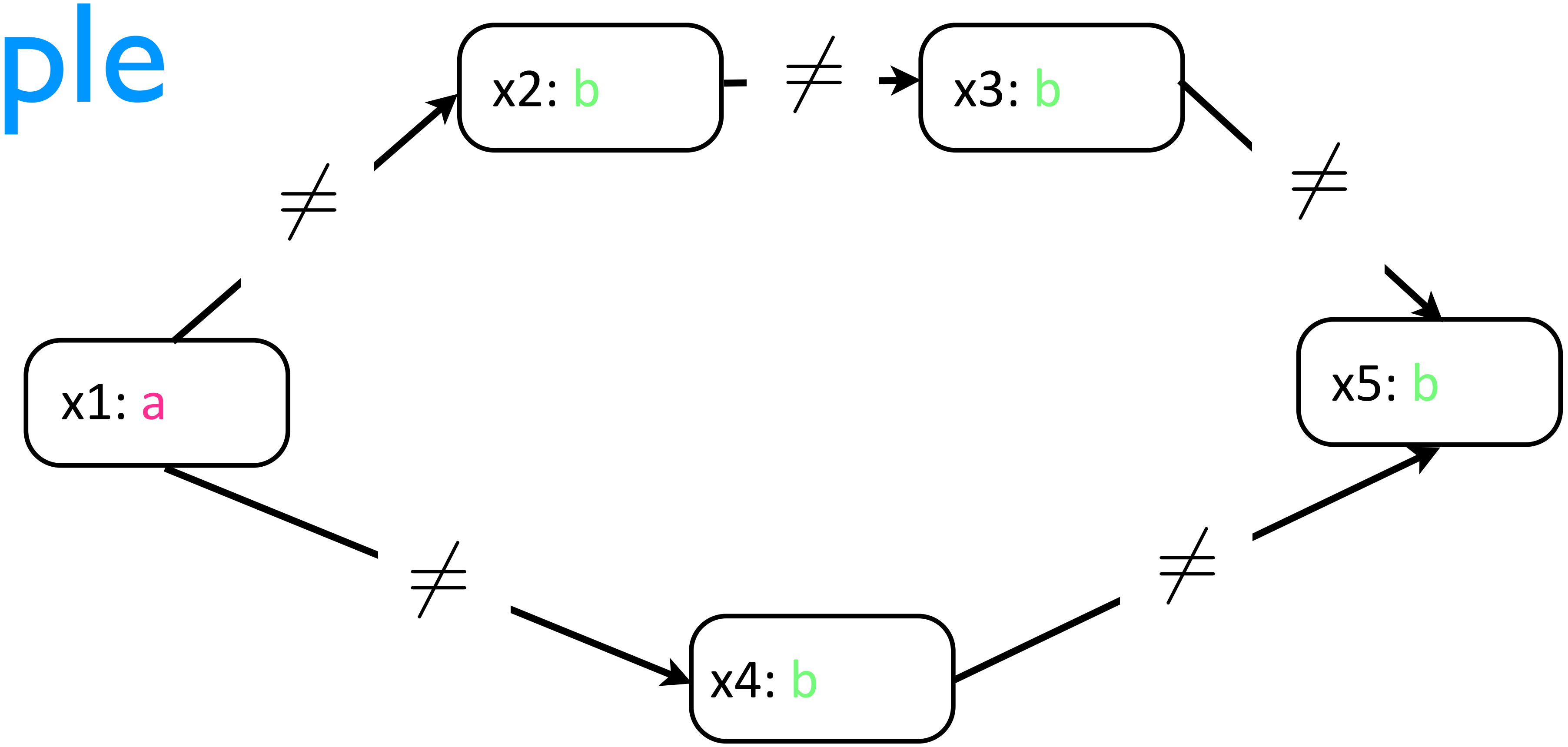
	message(s)	action
a_2	OK(x_1=a)	
a_3	OK(x_2=a)	
a_4	OK(x_1=a)	
a_5	OK(x_3=a)	
	OK(x_4=a)	

Example



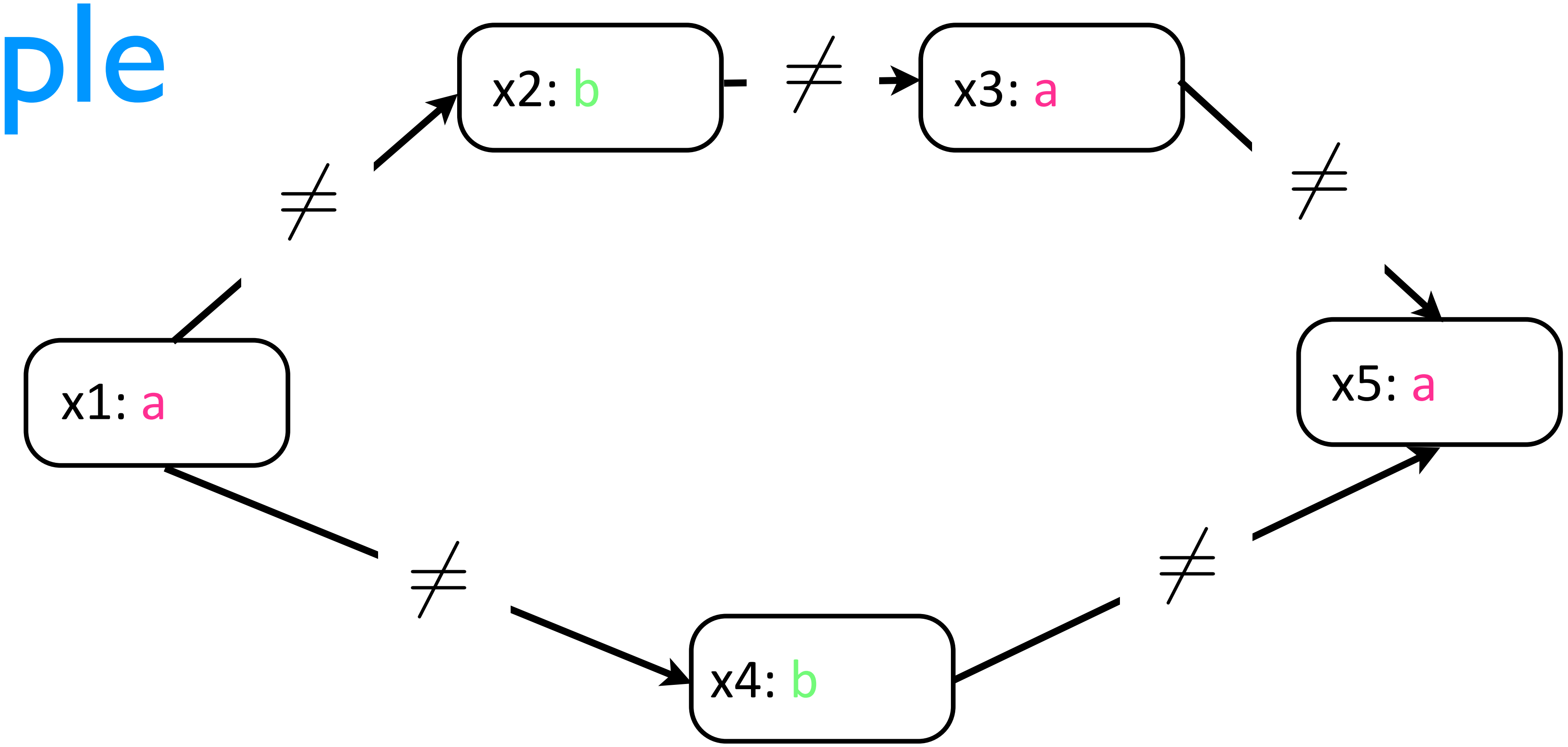
	message(s)	action
a_2	OK(x_1=a)	$x_2 \leftarrow$ b
a_3	OK(x_2=a)	$x_3 \leftarrow$ b
a_4	OK(x_1=a)	$x_4 \leftarrow$ b
a_5	OK(x_3=a)	$x_5 \leftarrow$ b
	OK(x_4=a)	

Example



	message(s)	action
a_3	$\mathtt{OK}(x_2\mathtt{=b})$	$x_3 \leftarrow \mathsf{a}$
a_5	$\mathtt{OK}(x_3\mathtt{=b})$	$x_5 \leftarrow \mathsf{a}$
	$\mathtt{OK}(x_4\mathtt{=b})$	

Example



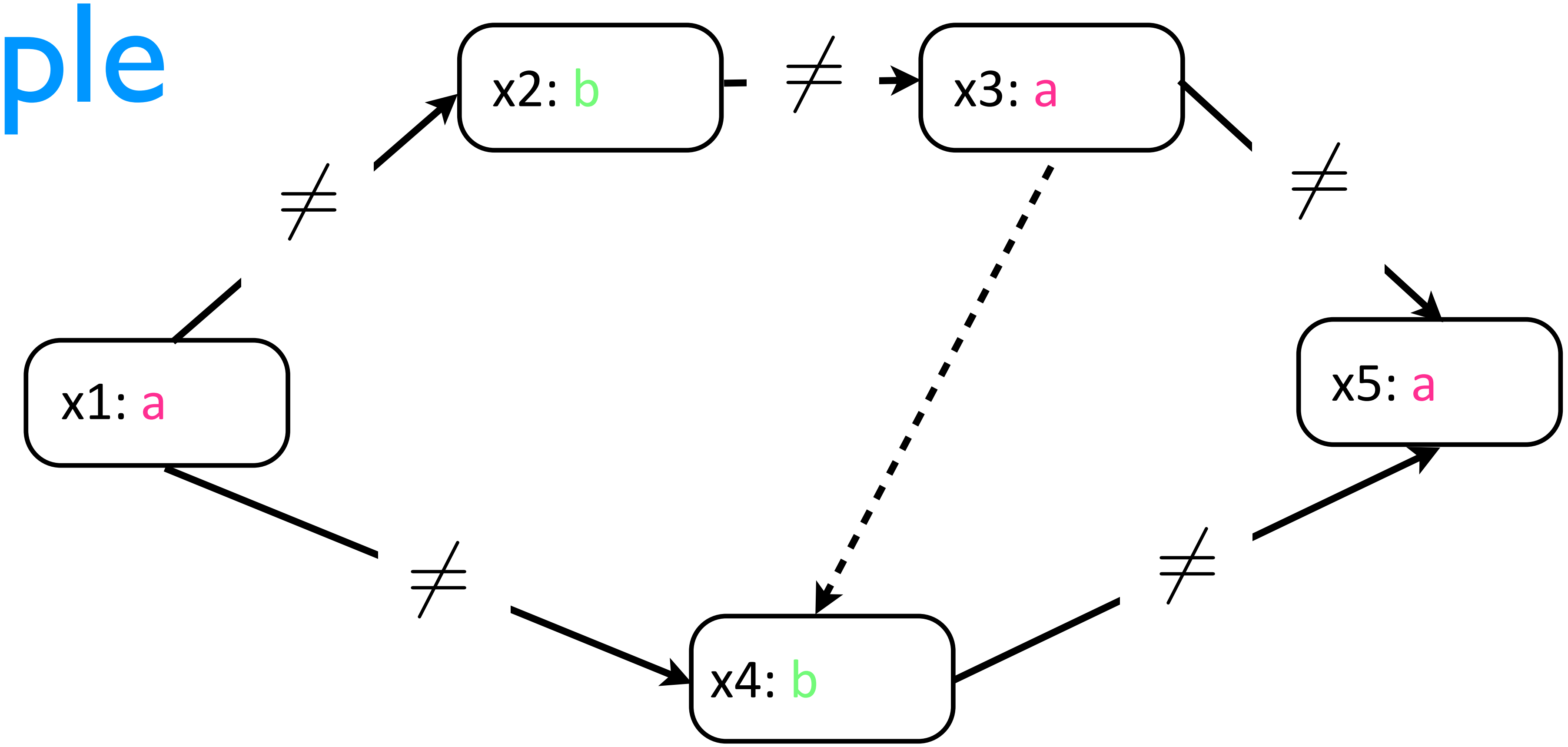
	message(s)	action
a_5	OK(x_3=a)	inconsistent!

$$x_3 = \mathrm{a} \Rightarrow x_5 \neq \mathrm{a}$$
$$x_4 = \mathrm{b} \Rightarrow x_5 \neq \mathrm{b}$$

a_5 sends a nogood to a_4:

v = b, cond $= (x_3 = \mathrm{a})$, tag $= x_5$ cost = 1

Example



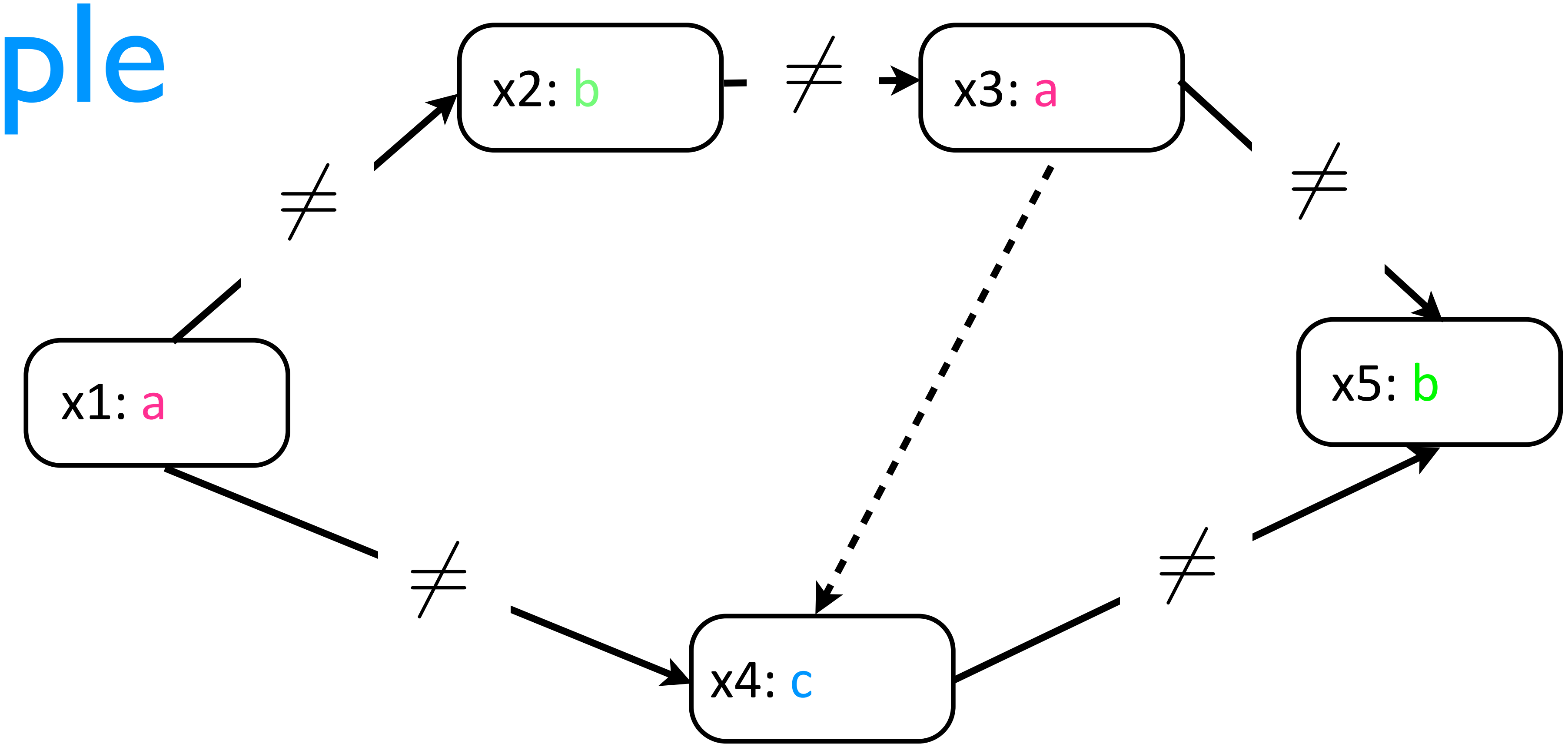
	message(s)	action
a_5	OK(x_3=a)	inconsistent!

$$x_3 = \mathrm{a} \Rightarrow x_5 \neq \mathrm{a}$$
$$x_4 = \mathrm{b} \Rightarrow x_5 \neq \mathrm{b}$$

a_5 sends a nogood to a_4:

v = b, cond $= (x_3 = \mathrm{a})$, tag $= x_5$ cost = 1

Example



	message(s)	action
a_5	OK(x_3=a)	inconsistent!

$$x_3 = \mathrm{a} \Rightarrow x_5 \neq \mathrm{a}$$
$$x_4 = \mathrm{b} \Rightarrow x_5 \neq \mathrm{b}$$

a_5 sends a nogood to a_4:

v = b, cond $= (x_3 = \mathrm{a})$, tag $= x_5$ cost = 1

Example

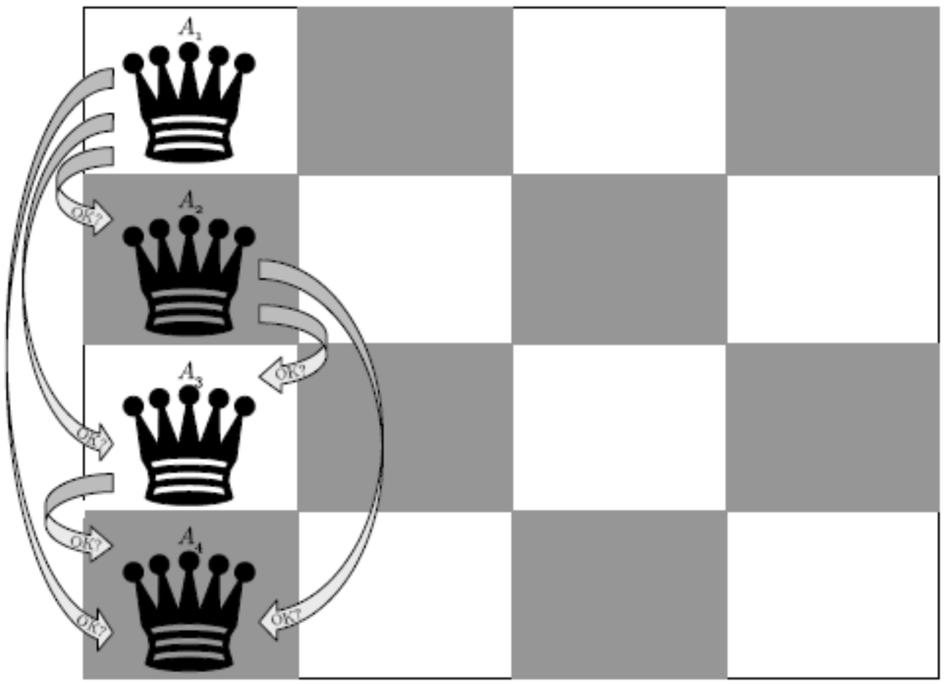


Figure 1.6: Cycle 1 of ABT for four queens. All agents are active.

Example

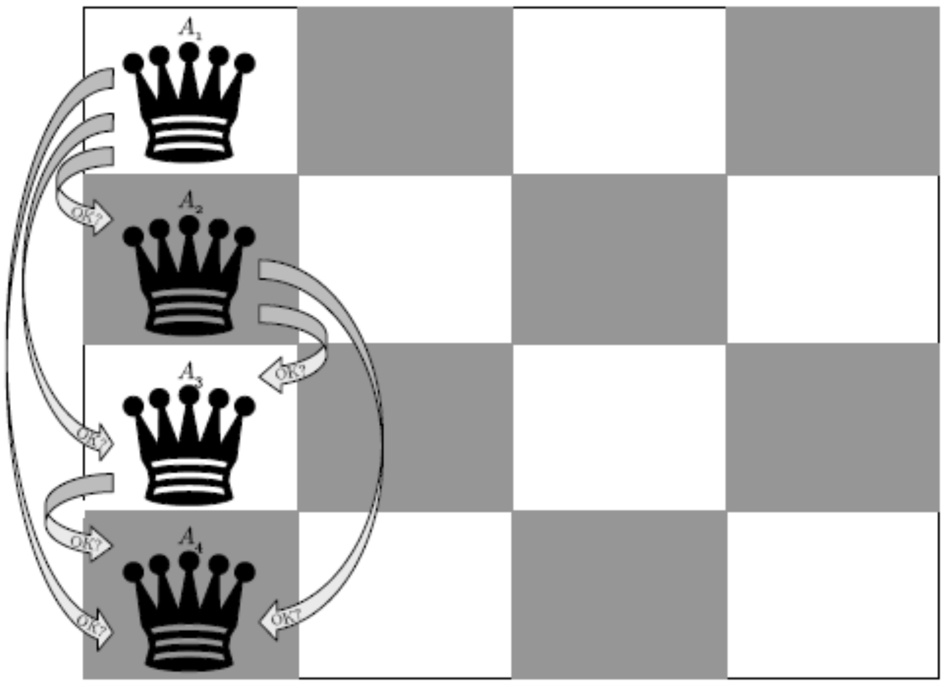


Figure 1.6: Cycle 1 of ABT for four queens. All agents are active.

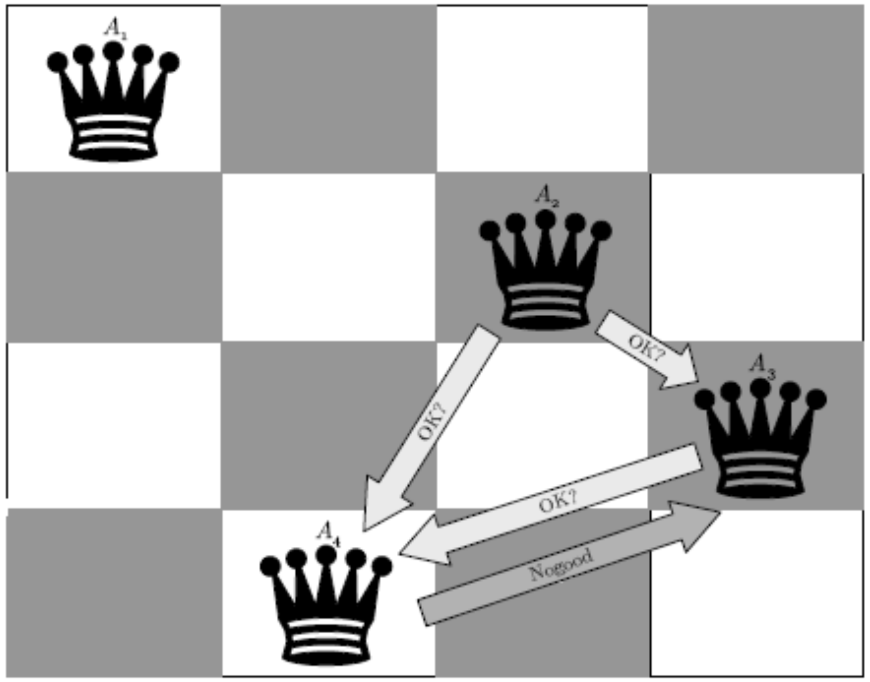


Figure 1.7: Cycle 2 of ABT for four queens. A_2, A_3 and A_4 are active. The Nogood message is $A_1 = 1 \wedge A_2 = 1 \rightarrow A_3 \neq 1$.

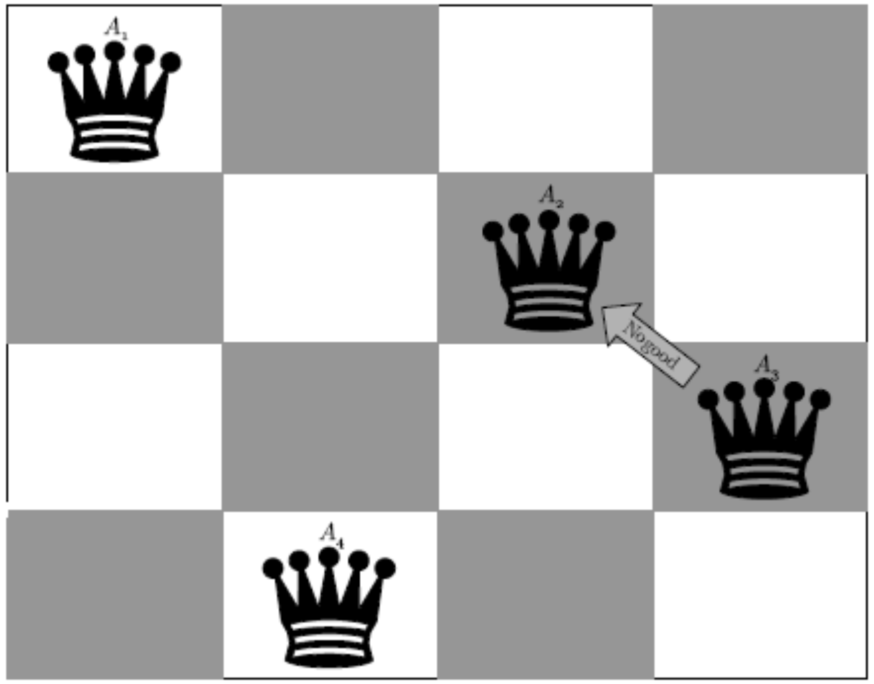


Figure 1.8: Cycle 3. Only A_3 is active. The Nogood message is $A_1 = 1 \rightarrow A_2 \neq 3$.

Example

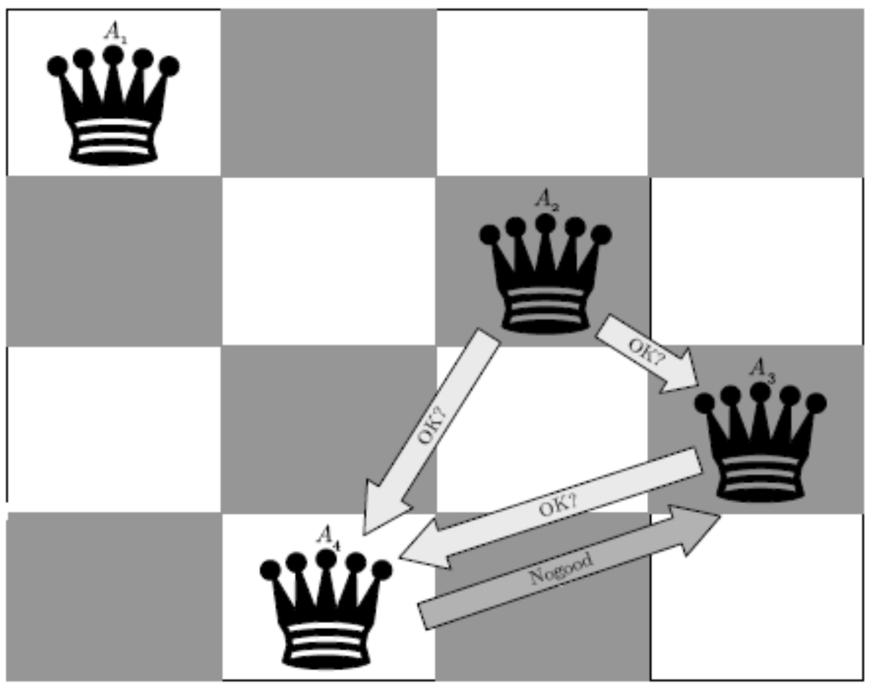


Figure 1.7: Cycle 2 of ABT for four queens. A_2, A_3 and A_4 are active. The Nogood message is $A_1 = 1 \wedge A_2 = 1 \rightarrow A_3 \neq 1$.

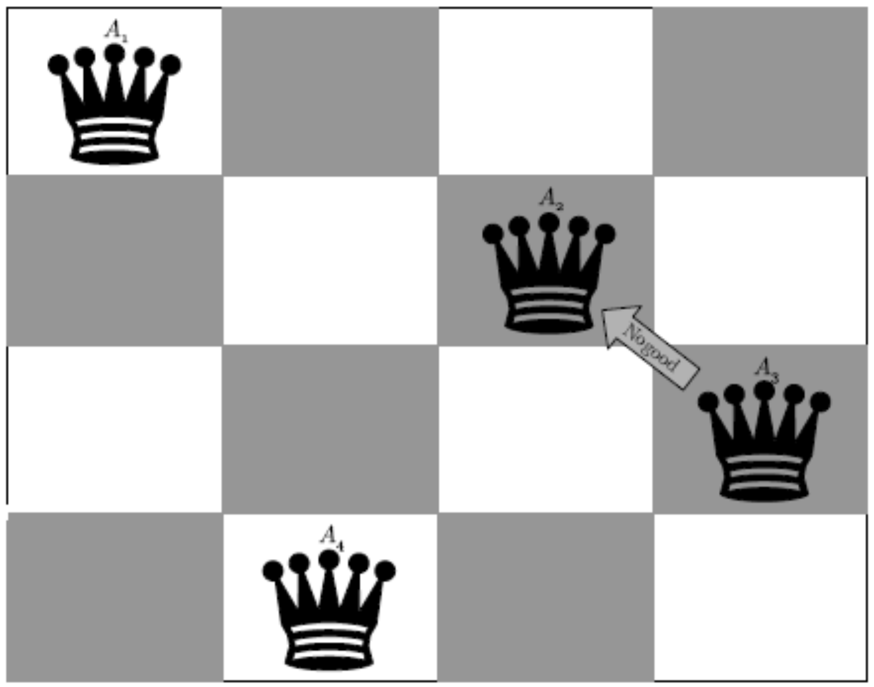


Figure 1.8: Cycle 3. Only A_3 is active. The Nogood message is $A_1 = 1 \rightarrow A_2 \neq 3$.

Example

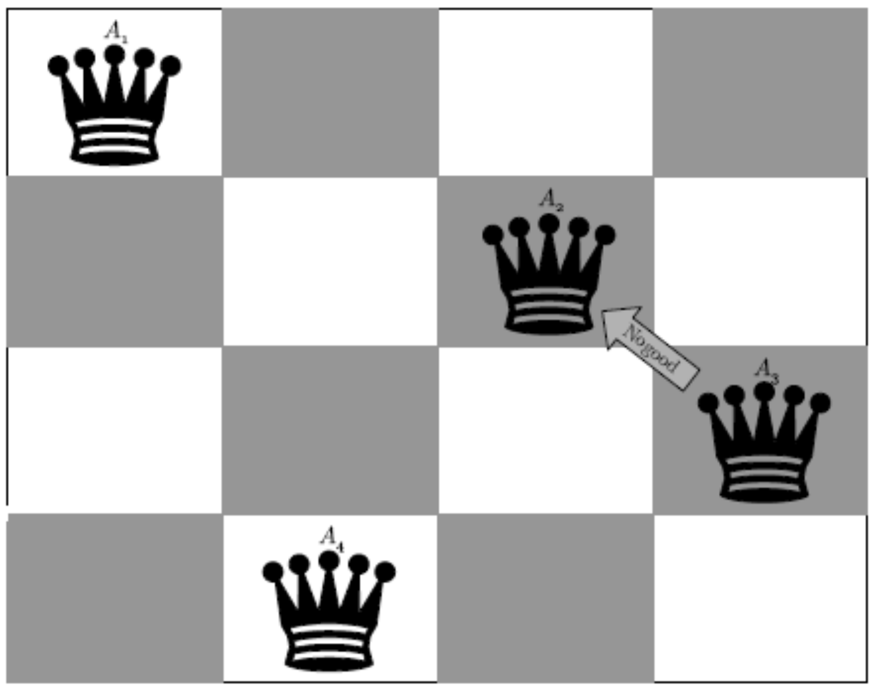


Figure 1.8: Cycle 3. Only A_3 is active. The Nogood message is $A_1 = 1 \rightarrow A_2 \neq 3$.

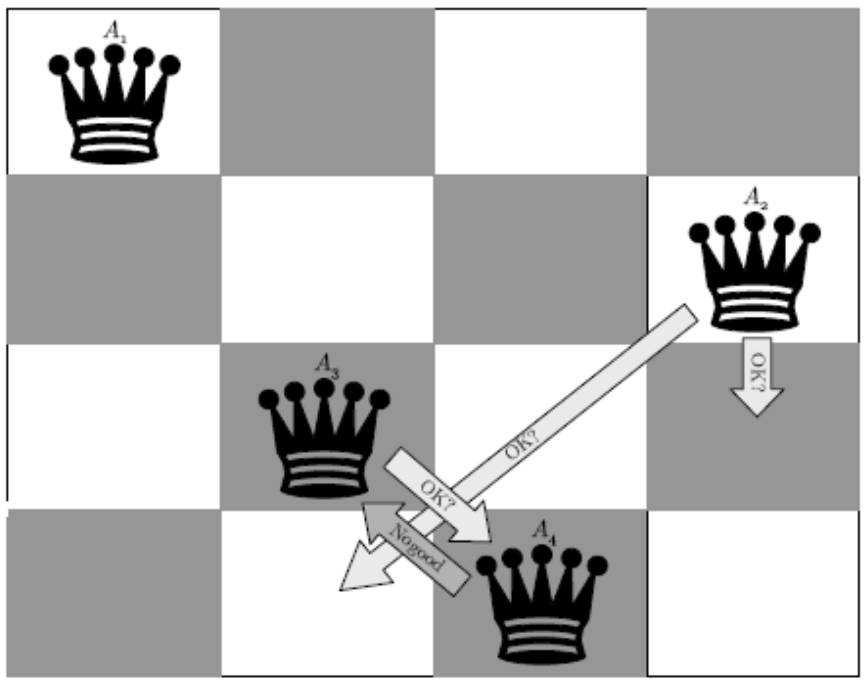


Figure 1.9: Cycles 4 and 5. A_2, A_3 and A_4 are active. The Nogood message is $A_1 = 1 \wedge A_2 = 4 \rightarrow A_3 \neq 4$.

Example

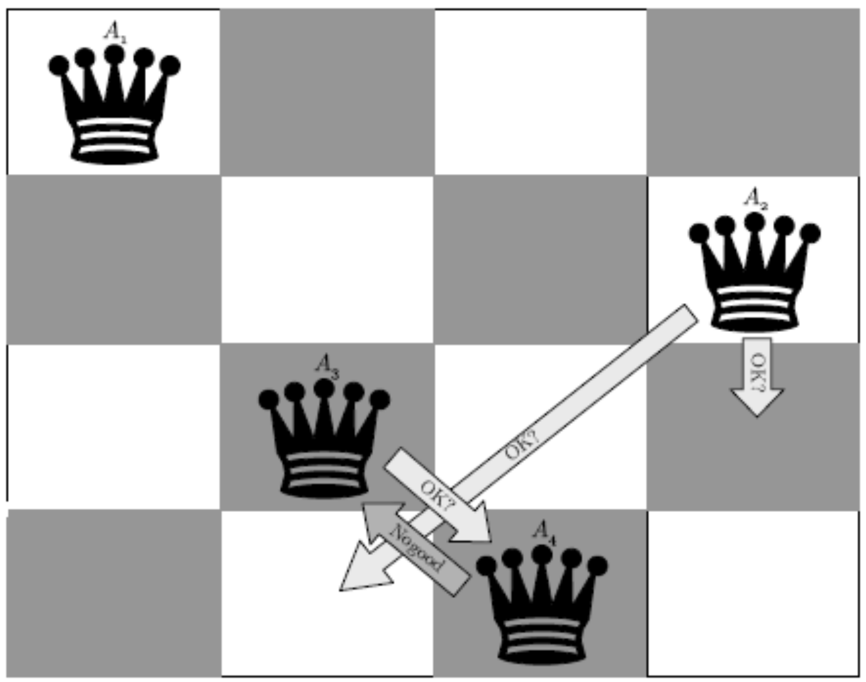


Figure 1.9: Cycles 4 and 5. A_2, A_3 and A_4 are active. The Nogood message is $A_1 = 1 \wedge A_2 = 4 \rightarrow A_3 \neq 4$.

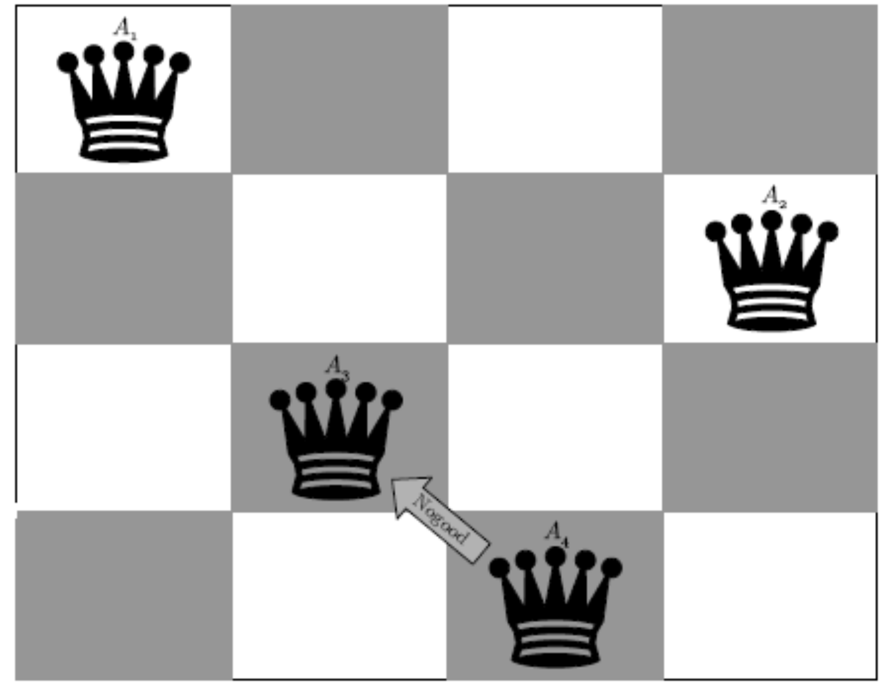


Figure 1.10: Cycle 6. Only A_4 is active. The Nogood message is $A_1 = 1 \wedge A_2 = 4 \rightarrow A_3 \neq 2$.

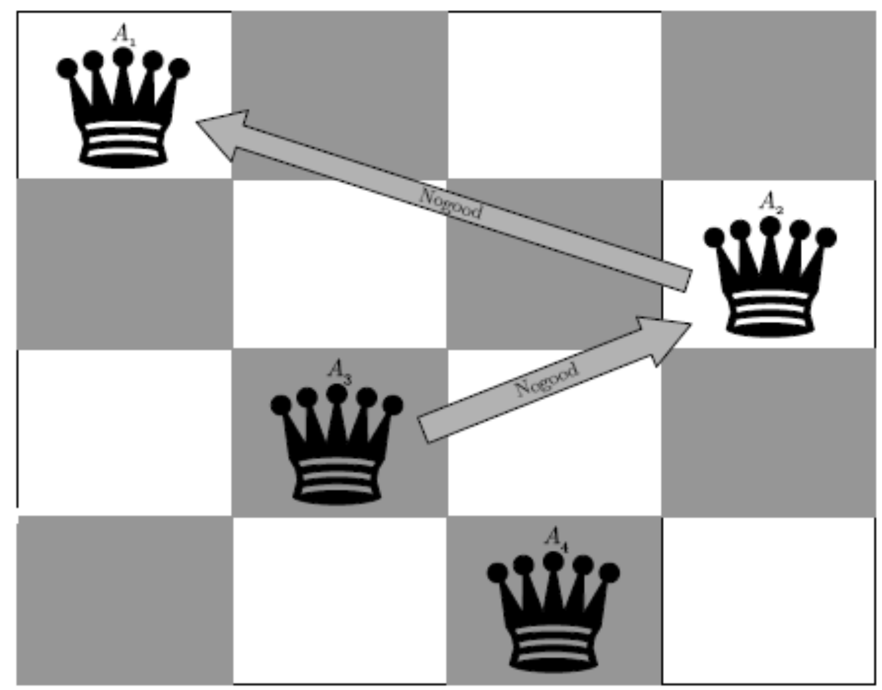


Figure 1.11: Cycles 7 and 8. A_3 is active in the first cycle and A_2 is active in the second. The Nogood messages are $A_1 = 1 \rightarrow A_2 \neq 4$ and $A_1 \neq 1$.

Example

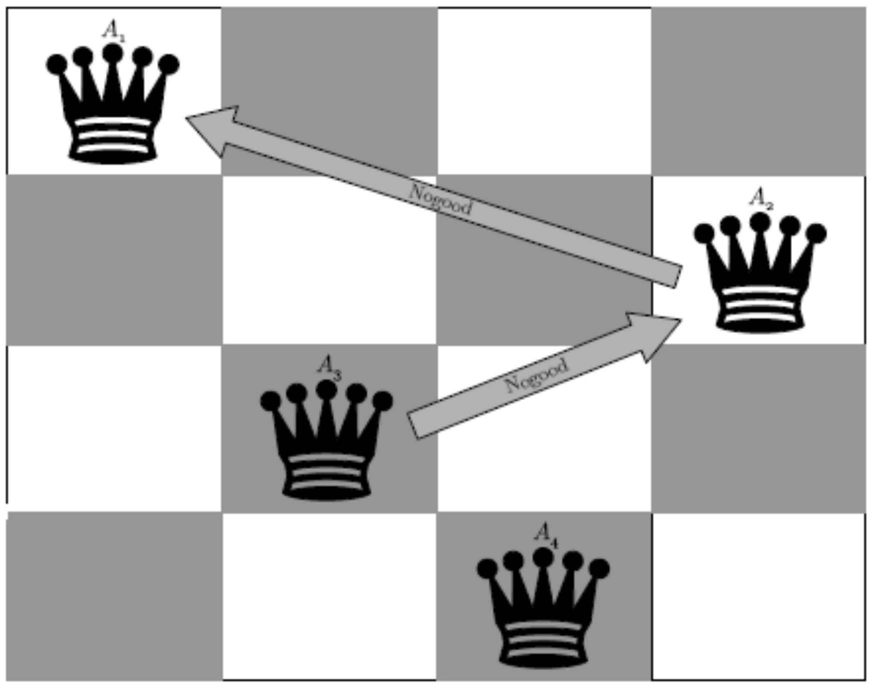


Figure 1.11: Cycles 7 and 8. A_3 is active in the first cycle and A_2 is active in the second. The Nogood messages are $A_1 = 1 \rightarrow A_2 \neq 4$ and $A_1 \neq 1$.

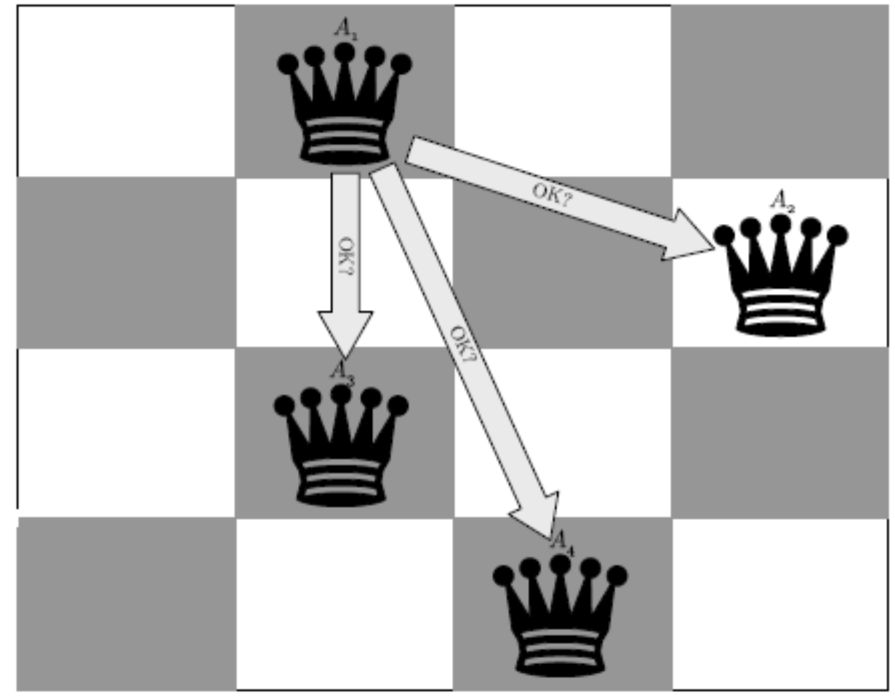


Figure 1.12: Cycle 9. Only A_1 is active.

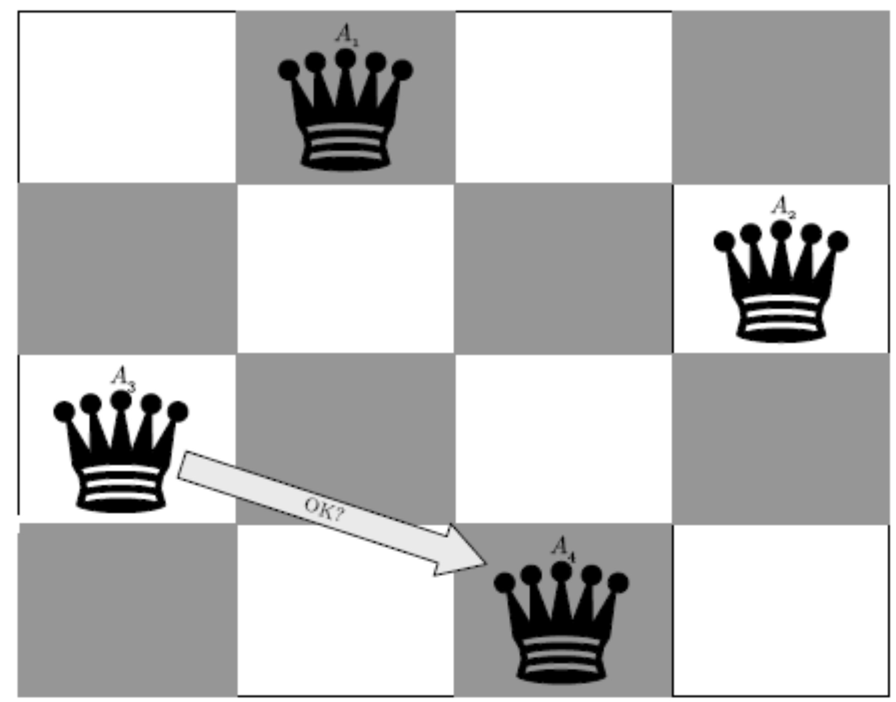


Figure 1.13: Cycle 10. Only A_3 is active.