

CSE 592 - Discussion 2

CSP

Seung Hyun Lee / Fall 26

Logistics

- HW 1 Due Sep 23rd, 11:59pm
- 4 Problems
 - Search
 - **CSP**
 - Propositional Logic
 - First-Order Logic

Constraint Satisfaction Problems (CSP)

- X: decisions. The variables name the decisions we need to make, such as the time of each meeting.
- D: options. Each variable has a domain, which is the set of values currently available for that decision.
- C: interactions. Constraints describe which combinations of choices are allowed, such as two meetings having different times.
- We will follow how these sets change: remove impossible choices, make a choice when needed, and check whether the decisions fit together.

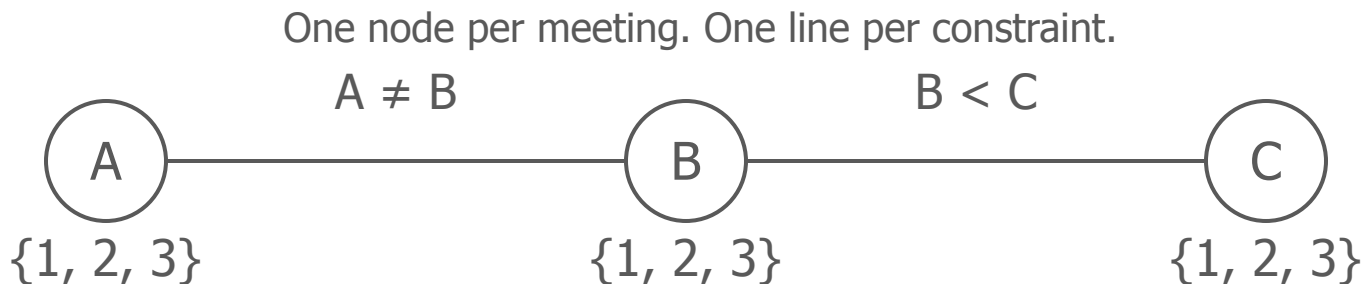
Variables and values

- A variable represents one decision that we still need to make. We will use a letter such as X to name that decision.
- For example, if we are scheduling a meeting, X represents its time slot. The numbers 1, 2, and 3 represent available time slots.
- Writing $X = 2$ means that we have chosen time slot 2 for this meeting. The letter names the decision, and the number records our choice.
- When a problem has several decisions, we create one variable for each decision. A solution will eventually give every variable a value.

Domains and constraints

- We write $D(X) = \{1, 2, 3\}$ to say that X can currently take any of these three values. The braces mean that we are listing a set.
- A constraint restricts the choices we can make together. If two meetings cannot share a time, we write $X \neq Y$, meaning X and Y must differ.
- If X must happen before Y , we write $X < Y$. The pair $X = 1$ and $Y = 3$ satisfies this rule, but $X = 3$ and $Y = 1$ violates it.
- Values can belong to their domains and still form an invalid combination. We must check both domain membership and the constraints.

A small scheduling problem



Example solution: $A = 3, B = 1, C = 2$

- A, B, and C are meeting times, each chosen from $\{1, 2, 3\}$. The connecting lines show the two constraints: $A \neq B$ and $B < C$.
- $A = 3, B = 1, C = 2$ works: 3 differs from 1, and 1 is less than 2. A and C have no direct constraint in this example.

Consistent and complete

- A consistent assignment violates none of the constraints that we can evaluate using its assigned variables. Some variables may still be unassigned.
- For the rules $A \neq B$ and $B < C$, assigning $A = 3$ and $B = 1$ is consistent. We still need a value for C , so this assignment is incomplete.
- A complete assignment gives every variable a value. $A = 1, B = 1, C = 2$ is complete, but it violates $A \neq B$.
- A solution must be both complete and consistent. A partial assignment can look acceptable now and still lead to a contradiction later.

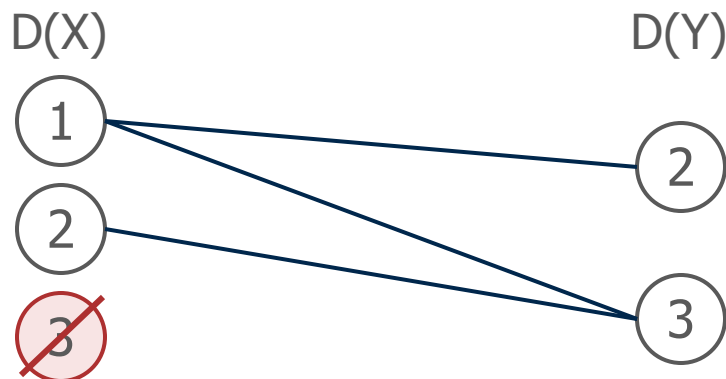
An assignment and a domain

- An assignment records a chosen value, such as $X = 2$. A domain records the values that remain possible, such as $D(X) = \{1, 2\}$.
- When we decide to try $X = 2$, we can represent that decision by restricting its domain to $\{2\}$ while exploring this search branch.
- A singleton domain contains exactly one value. For example, $D(Y) = \{3\}$ tells us that Y must be 3 if the current branch has a solution.
- A solver may keep Y unassigned until it explicitly selects Y . This distinction will matter when we compare forward checking with propagation.

Inference before guessing

- Inference removes values that cannot be part of a solution under our current assumptions. The solver can then search over fewer choices.
- For example, suppose $X < Y$, $D(X) = \{1, 2, 3\}$, and $D(Y) = \{2\}$. $X = 2$ and $X = 3$ cannot work because neither is less than 2.
- We can therefore replace $D(X)$ with $\{1\}$. We did not guess that X should be 1. The constraint and the current domain of Y forced this result.
- An empty domain means that this branch has no solution. Nonempty domains can still require more reasoning or search.

Support for a value



Constraint: $X < Y$

**$X = 3$ has no
supporting Y .**

Each line is a valid pair.

- A line joins two values that satisfy $X < Y$. $X = 1$ has support in $Y = 2$ or 3 . $X = 2$ has support in $Y = 3$.
- $X = 3$ has no supporting Y , so we remove it. One support is enough to keep a candidate, but that support can disappear later.

Arc consistency

- An arc is a directed check between two variables that share a constraint. We write (X, Y) when we check the values of X using the domain of Y .
- The arc (X, Y) is consistent if every value remaining in $D(X)$ has at least one supporting value in $D(Y)$.
- For $X < Y$ with $D(X) = \{1, 2\}$ and $D(Y) = \{2, 3\}$, $X = 1$ has support in $Y = 2$, and $X = 2$ has support in $Y = 3$.
- This check says that each value of X can satisfy this one constraint. It does not yet show that all variables can satisfy every constraint together.

The direction of a check

- The reverse arc (Y, X) checks values of Y using $D(X)$. We keep the same constraint $X < Y$ when we reverse the direction of the check.
- For example, let $D(X) = \{2\}$ and $D(Y) = \{1, 2, 3\}$. $Y = 1$ fails because $2 < 1$ is false. $Y = 2$ also fails because $2 < 2$ is false.
- $Y = 3$ has support in $X = 2$, so revising (Y, X) leaves $D(Y) = \{3\}$. Only the domain of the first variable in the arc can change.
- For a binary constraint, we normally check both directed arcs. Each direction asks whether the candidates on its own side still have support.

REVISE: one directed check

- $\text{REVISE}(X, Y)$ examines each value x in $D(X)$. For that value, it looks through $D(Y)$ for any value y that satisfies the constraint between X and Y .
- If it finds a supporting y , it keeps x . If it checks all of $D(Y)$ and finds none, it removes x from $D(X)$.
- The removal is safe: a complete solution containing x would need a compatible value for Y , and the current domain contains no such value.
- REVISE reports whether $D(X)$ changed. A change matters because another variable may have been relying on one of the values we just removed.

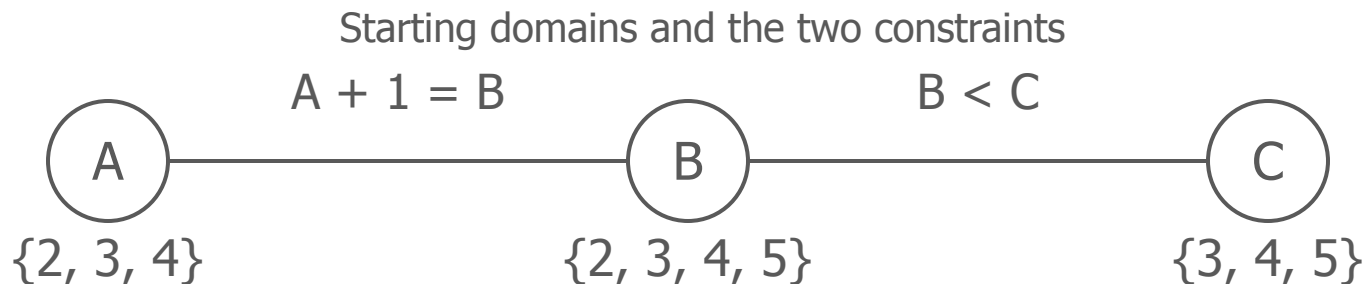
AC-3: repeated checks

- AC-3 repeats REVISE until there are no pending checks. It stores the directed arcs that still need attention in a queue, which is a list of pending work.
- For an initial pass over a binary CSP, we put both directions of every constraint into the queue. We then remove one arc and revise it.
- A check that changes nothing creates no new work. If a domain shrinks, we may need to recheck neighbors that relied on the removed values.
- An empty domain means failure. If the queue becomes empty without failure, all required checks have finished and the domains are arc-consistent.

Why an arc returns to the queue

- Suppose revising (X, Y) removes a value from $D(X)$. A different neighbor Z might have used that removed value as its only support in X .
- We therefore add (Z, X) to the queue. This direction checks the values of Z against the smaller domain of X .
- We do this for every neighbor of X except Y . The removed X values had no support in Y , so no Y value was relying on them.
- In the next example, we process arcs in first-in, first-out order. If an arc is already waiting in the queue, we keep a single pending copy.

AC-3 example: starting state

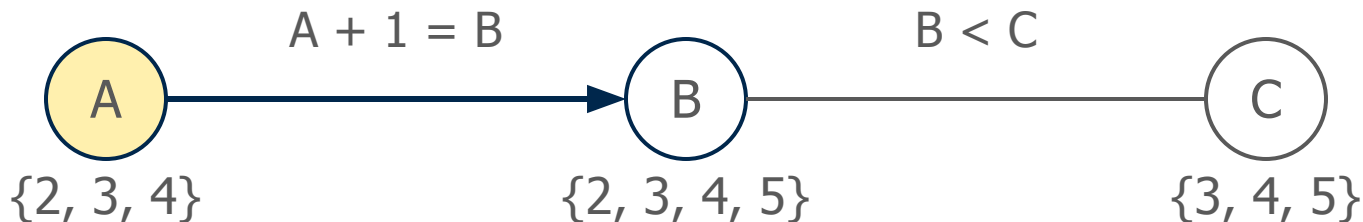


Pending queue: (A, B), (B, A), (B, C), (C, B)

- The domains below each node list the current choices. $A + 1 = B$ means B is exactly one greater than A. The other rule is $B < C$.
- We use the latest domains and process arcs in queue order. A change can require rechecking an earlier arc. We keep one pending copy.

AC-3: checking (A, B)

Blue arrow: directed check. Highlighted node: domain being checked.

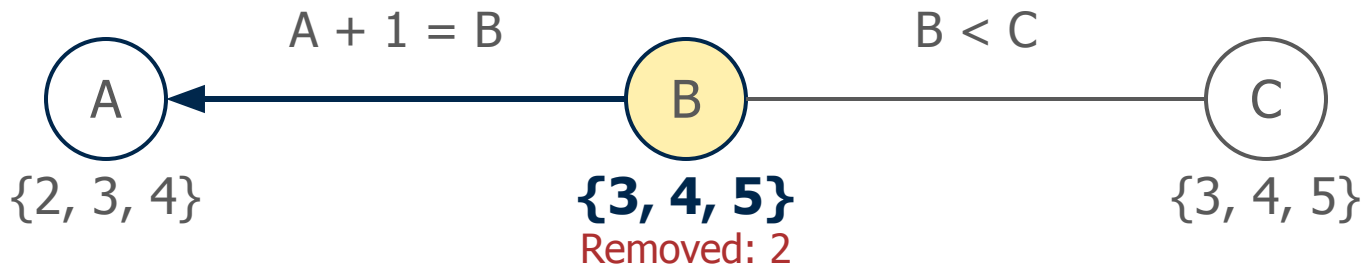


Pending queue: (B, A), (B, C), (C, B)

- The blue arrow marks the check (A, B): inspect A's candidates using $D(B)$. A = 2, 3, and 4 have support in B = 3, 4, and 5.
- Every value of A has support, so $D(A)$ stays $\{2, 3, 4\}$. This check does not inspect B's extra value, 2. With no change, we add no work.

AC-3: checking (B, A)

Blue arrow: directed check. Highlighted node: domain being checked.

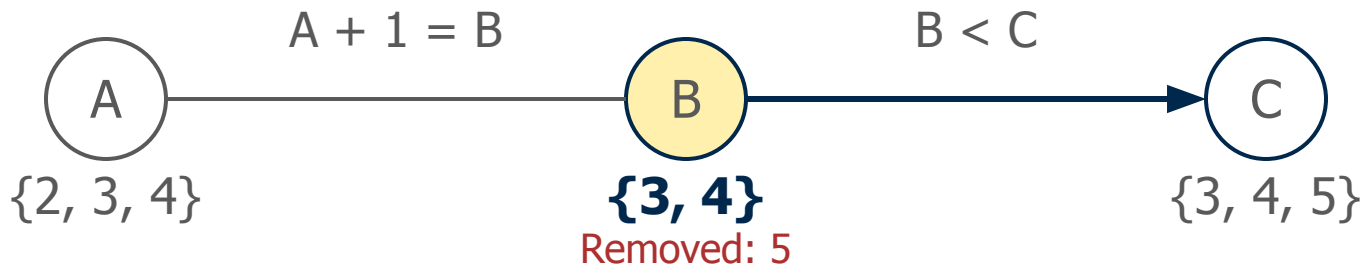


Pending queue: (B, C), (C, B)

- The reverse check inspects B using the same rule $A + 1 = B$. $B = 2$ needs $A = 1$, which is absent, so we remove 2.
- $B = 3, 4$, and 5 have support in $A = 2, 3$, and 4 . Since B shrank, C needs a check against B. The arc (C, B) is already waiting.

AC-3: checking (B, C)

Blue arrow: directed check. Highlighted node: domain being checked.

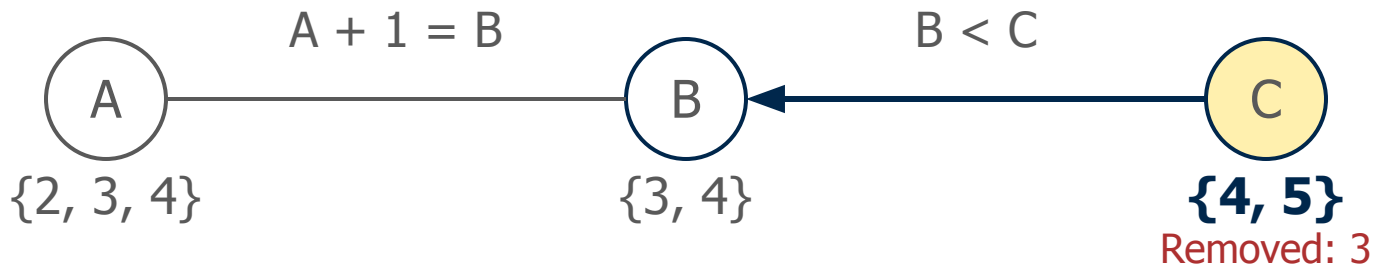


Pending queue: (C, B), (A, B)

- Using $B < C$, $B = 3$ has support in $C = 4$ or 5 , and $B = 4$ has support in $C = 5$. $B = 5$ needs C greater than 5 , so it disappears.
- The smaller domain of B may affect A : $A = 4$ had relied on $B = 5$. We add (A, B) after the already waiting check (C, B).

AC-3: checking (C, B)

Blue arrow: directed check. Highlighted node: domain being checked.

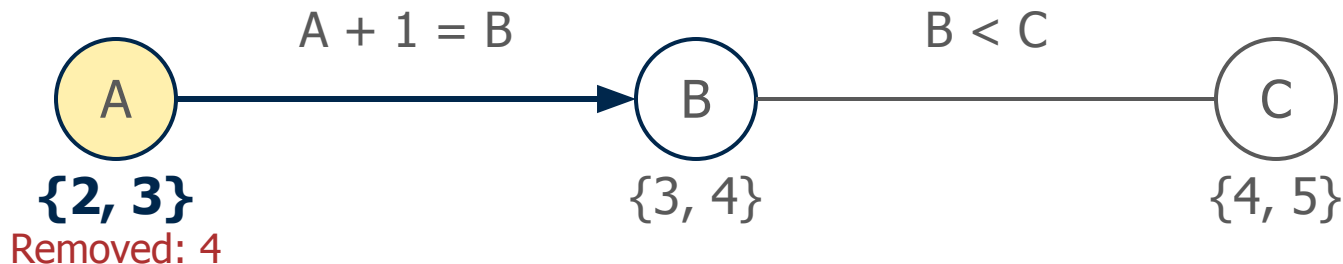


Pending queue: (A, B)

- This direction checks C but keeps the rule $B < C$. $C = 3$ needs B smaller than 3. The current $D(B) = \{3, 4\}$ offers no support, so 3 disappears.
- $C = 4$ has support in $B = 3$. $C = 5$ has support in $B = 3$ or 4. C has no neighbor besides B, so we add no further checks.

AC-3: checking (A, B) again

Blue arrow: directed check. Highlighted node: domain being checked.



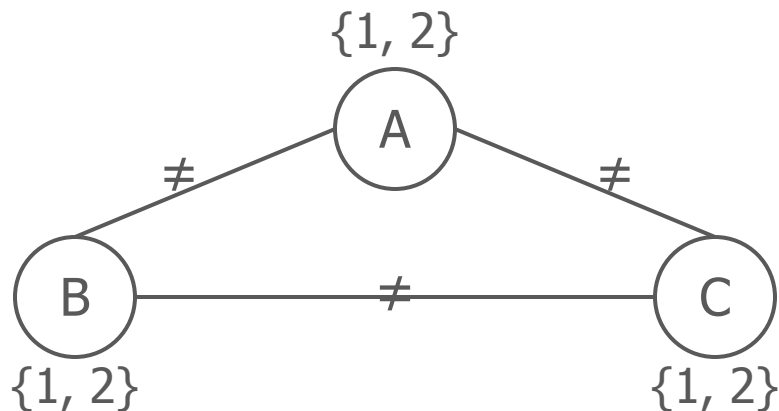
Pending queue: empty

- A = 2 still has support in B = 3, and A = 3 still has support in B = 4. A = 4 needs B = 5, which has now disappeared, so we remove 4.
- A has no other neighbor, so the queue becomes empty. A candidate can survive one check and disappear later if its supporting value is removed.

The result of propagation

- Our final domains are A in $\{2, 3\}$, B in $\{3, 4\}$, and C in $\{4, 5\}$. Every remaining candidate has support across each neighboring constraint.
- These domains still contain several values. To understand the result, we can check the complete assignments that satisfy $A + 1 = B$ and $B < C$.
- If $A = 2$, then B must be 3 and C can be 4 or 5. If $A = 3$, then B must be 4 and C must be 5.
- The solutions are therefore $(2, 3, 4)$, $(2, 3, 5)$, and $(3, 4, 5)$, in the order (A, B, C) . Propagation reduced the choices without choosing one solution.

A limit of arc consistency



Only two values
for three variables

- Every pair must use different values. Each candidate has local support: 1 can pair with 2, and 2 can pair with 1. All arcs are consistent.
- Yet three variables cannot all differ using only two values. The local supports cannot all hold together, so arc consistency does not guarantee a solution.

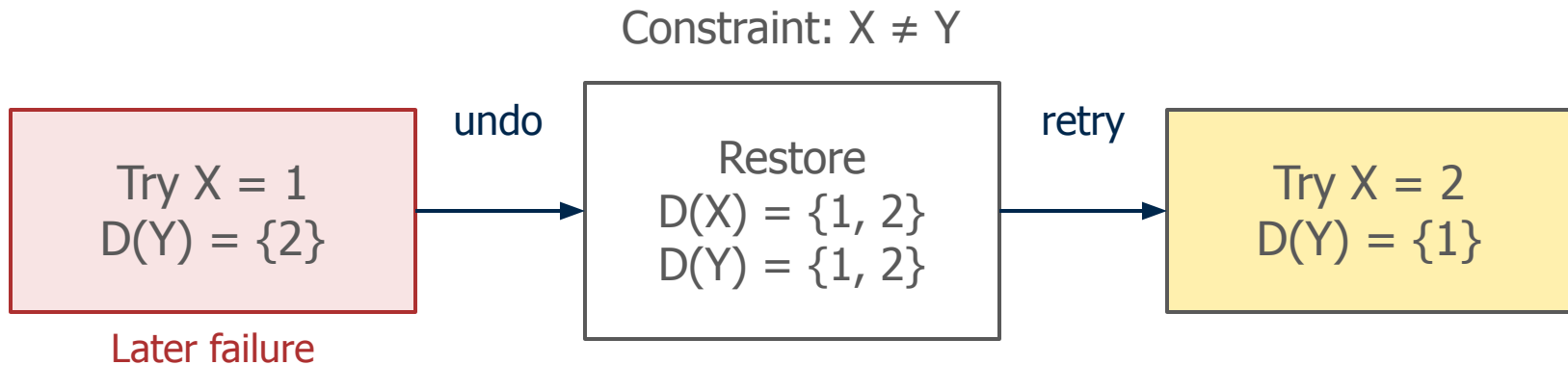
Finding the contradiction by search

- In that same three-variable problem, suppose we try $A = 1$. The constraints $A \neq B$ and $A \neq C$ then force both B and C to use 2.
- This creates a conflict with $B \neq C$. The assumption $A = 1$ therefore cannot lead to a solution.
- Trying $A = 2$ creates the symmetric problem: both B and C must become 1, and $B \neq C$ fails again.
- Both values of A have failed, so no solution exists. Trying a choice and following its consequences revealed what the initial arc checks missed.

Backtracking search

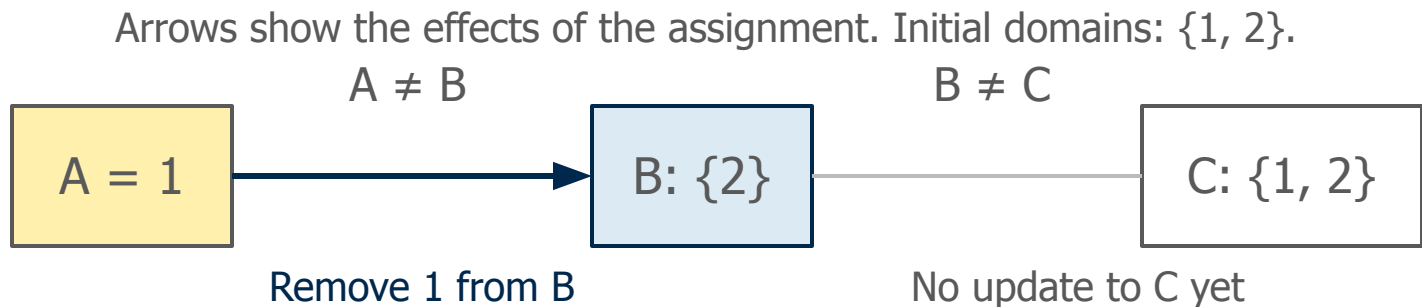
- Backtracking builds a solution one variable at a time. It selects an unassigned variable and temporarily tries one of the values in its current domain.
- We reject a choice if it conflicts with an existing assignment. Otherwise, we run inference and continue with the remaining variables.
- If a later step reaches a dead end, we return to the most recent decision that still has another value to try. This return is the backtracking step.
- We succeed when every variable has a value and all constraints hold. Failure means that we have exhausted all available branches.

Restoring a failed branch



- With $X \neq Y$ and both domains $\{1, 2\}$, trying $X = 1$ leaves Y with $\{2\}$. Suppose a separate constraint later makes this branch fail.
- We undo the assignment and its domain deletions before trying $X = 2$. Y must regain 1, because the new branch requires $Y = 1$.

Forward checking



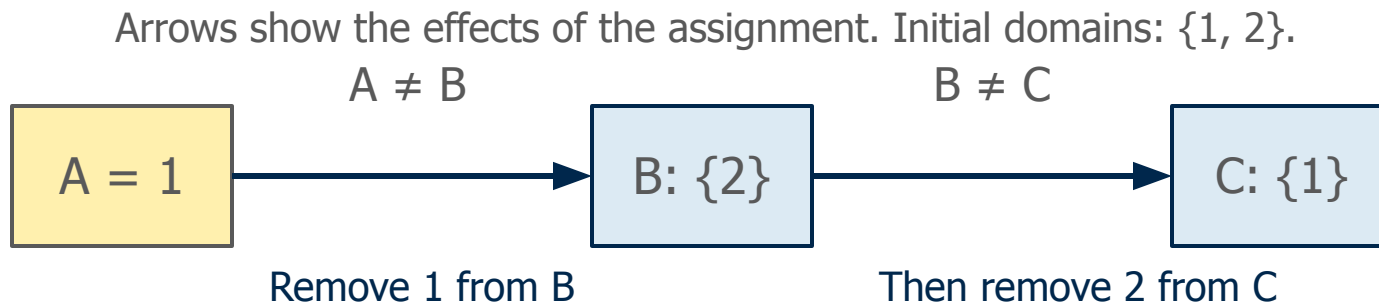
FC stops after updating the assigned variable's neighbors.

- All domains started as $\{1, 2\}$. After assigning $A = 1$, forward checking removes 1 from its neighbor B. B is now a singleton, $\{2\}$.
- C has no direct constraint with A, so C keeps $\{1, 2\}$. Ordinary forward checking stops here. A later assignment of B can trigger the next update.

A singleton under forward checking

- After $A = 1$, we have $D(B) = \{2\}$ and $D(C) = \{1, 2\}$. B has one possible value, but the search has not selected and assigned B yet.
- Ordinary forward checking stops after checking the new assignment's neighbors. A neighbor becoming a singleton does not trigger another update.
- If the search next selects B and assigns $B = 2$, that new assignment triggers forward checking on C. C then loses 2 and becomes $\{1\}$.
- The search still finds the consequence for C. It finds it after assigning B, during a new forward-checking step.

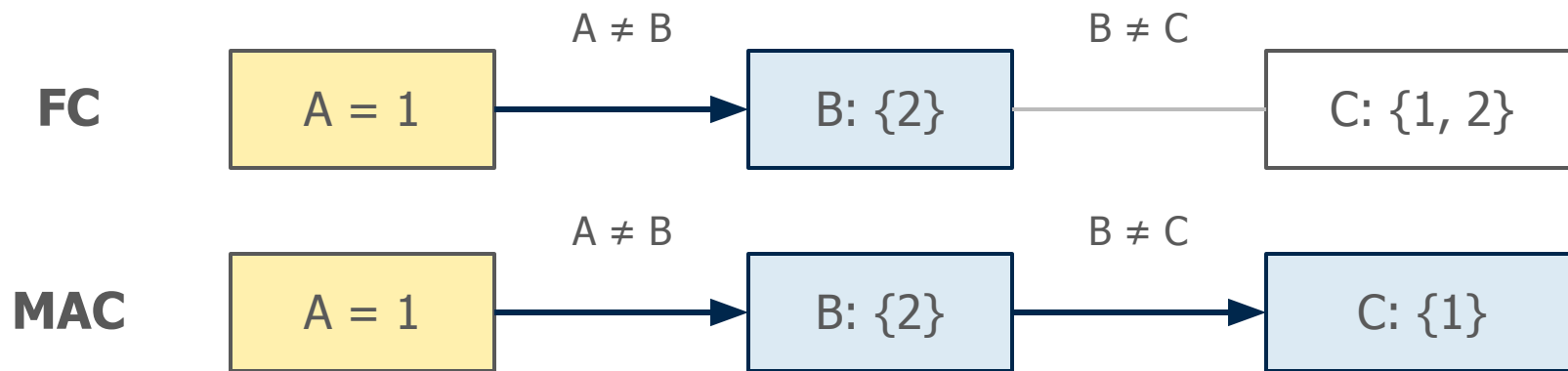
Maintaining arc consistency (MAC)



Propagation continues after B's domain changes.

- MAC maintains arc consistency during search. After $A = 1$ makes B's domain shrink to $\{2\}$, it also checks C against the smaller domain of B.
- $C = 2$ has no support under $B \neq C$, so C becomes $\{1\}$. MAC follows domain changes across several constraints before the next search decision.

FC and MAC on the same decision



- Both methods start from domains $\{1, 2\}$ and the choice $A = 1$. FC updates B. MAC also follows B's change to C before another assignment.
- MAC does more propagation at each step and may save later search. That extra work costs time, so stronger pruning is not always faster overall.

MAC after a choice in our example

- Return to $A + 1 = B$ and $B < C$, after AC-3 left $D(A) = \{2, 3\}$, $D(B) = \{3, 4\}$, and $D(C) = \{4, 5\}$.
- Suppose the search tries $C = 4$. We temporarily restrict $D(C)$ to $\{4\}$ and check which values of B still satisfy $B < C$.
- $B = 4$ no longer works because $4 < 4$ is false. Only $B = 3$ remains. This smaller domain then affects A through $A + 1 = B$.
- $A = 3$ would require $B = 4$, so it also disappears. We finish with $A = 2$, $B = 3$, $C = 4$, and these values satisfy both constraints.

Minimum Remaining Values (MRV)

- MRV is a rule for choosing the next unassigned variable. It selects a variable with the fewest values in its current domain.
- A variable with one remaining value has only one possible choice. A variable with many remaining values gives the search more alternatives to consider.
- A restricted variable can reveal a conflict sooner. If an unassigned variable has an empty domain, the branch has failed and we backtrack.
- The word current matters. We count the candidates left after the latest inference, and we leave already assigned variables out of the selection.

MRV: a concrete choice

All four variables are unassigned.



- These variables are all unassigned. MRV compares domain sizes: P has 3 candidates, Q has 2, and R and S each have 1.
- R and S tie. Using the example's order P, Q, R, S, we choose R and try 4. S's smaller digit, 1, does not give it priority.

Variable order and value order

- Variable ordering asks which unassigned variable to choose next. MRV answers that question by comparing current domain sizes.
- Value ordering asks which candidate to try first for the chosen variable. If $D(X) = \{1, 3\}$, the solver still needs an order for trying 1 and 3.
- The order affects how much search we perform. A useful order can find a solution or a contradiction after fewer attempted assignments.
- After a branch fails, we restore its changes and try other values. An ordering rule changes the order of trials, not which values are allowed.

Sudoku as a CSP

- In Sudoku, each cell is a variable and its value is the digit placed in that cell. A given digit starts with a singleton domain.
- In a standard 9×9 puzzle, a blank cell initially allows digits 1 through 9. Row, column, and 3×3 box rules restrict these candidates.
- Two cells are neighbors, or peers, when they share a row, column, or box. A value assigned to one cell rules out that same value in its unassigned peers.
- The goal is a complete grid that satisfies every rule. We will use a smaller 4×4 example so we can follow individual candidate changes clearly.

Peers and candidate removal

- For a cell, the relevant neighbors are the other cells in its row, the other cells in its column, and the other cells in its box.
- A cell can share both a row and a box with another cell. We still treat that other cell as one neighbor when we collect the set of peers.
- Assigning digit 3 removes 3 from each unassigned peer's domain. Other candidates stay unless another rule rules them out.
- A cell outside that row, column, and box keeps its domain in this step. Later changes may still reach it through other cells.

A 4×4 Sudoku example

1	.	.	4
.	4	1	.
2	.	4	.
4	.	.	1

- Each row, column, and shaded 2×2 box must contain digits 1 through 4 exactly once.
- A dot marks a blank cell. The printed numbers are fixed givens.
- Coordinates use (row, column), starting from 1. Cell (2, 1) begins row 2.
- We will first find candidates, then follow the effects of assigning one cell.

Candidates in the first row

	1	2	3	4
1	1	2, 3	2, 3	4
2	.	4	1	.
3	2	.	4	.
4	4	.	.	1

Row and column numbers start at 1.

- Row 1 already has 1 and 4, so both highlighted blanks can only use 2 or 3.
- Column 2 contains 4. The upper-left box contains 1 and 4. Both leave {2, 3} available.
- Column 3 and its box also allow {2, 3}.
- Neither highlighted cell is forced yet. We look for a cell with fewer candidates.

A forced cell: (2, 1)

	1	2	3	4
1	1	.	.	4
2	{3}	4	1	.
3	2	.	4	.
4	4	.	.	1

Row and column numbers start at 1.

- The yellow cell is (2, 1). Row 2 leaves {2, 3}.
- Column 1 contains 1, 2, and 4, so only 3 remains.
- The upper-left box allows 3 too. The combined domain is {3}.
- MRV has several singletons. With row-then-column ties, we choose (2, 1) and assign 3.

Forward checking after $(2, 1) = 3$

	1	2	3	4
1	1	{2}	2, 3	4
2	3	4	1	{2}
3	2	.	4	.
4	4	.	.	1

Row and column numbers start at 1.

- Yellow marks the new assignment $(2, 1) = 3$.
- The blue cells are unassigned peers. $(1, 2)$ shares its box, and $(2, 4)$ shares its row.
- Both lose 3, changing $\{2, 3\}$ to $\{2\}$.
- The outlined cell $(1, 3)$ shares no group with $(2, 1)$, so it keeps $\{2, 3\}$.

MRV after the update

	1	2	3	4
1	1	2	{3}	4
2	3	4	1	{2}
3	2	.	4	.
4	4	.	.	1

Row and column numbers start at 1.

- MRV next selects (1, 2), the earliest unassigned singleton. We assign 2 in yellow.
- The blue cell (1, 3) shares row 1. It loses 2 and becomes {3}.
- MRV now selects (1, 3) and assigns 3.
- The effect reaches this cell through another assignment and a new forward-checking step.

Completing the small grid

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

- Continuing the same process gives this completed grid. All original givens stay fixed.
- Each row, column, and 2×2 box contains 1, 2, 3, and 4 exactly once.
- The assignment is now complete and consistent, so it is a solution.
- This puzzle uses forced choices. Others may need guesses and backtracking.

How the pieces work together

- The CSP model tells us what the variables mean, which values they can use, and which combinations the constraints permit.
- Inference reduces domains. Forward checking updates neighbors of a new assignment. MAC also follows the effects of those domain changes.
- MRV then chooses an unassigned variable with few remaining candidates. The search tries a value, runs inference again, and repeats this process.
- An empty domain triggers backtracking and restoration of the earlier state. A complete, consistent assignment ends the search successfully.