A constraint \( C(V_1, V_2, V_3, ..., V_n) \) is GAC wrt a variable \( V_i \) iff for every domain value of \( V_i \), there exist domain values for \( V_1, V_2, ..., V_{i-1}, V_{i+1}, ..., V_n \) that satisfy \( C(V_1, V_2, V_3, ..., V_n) \).

\( C(V_1, V_2, V_3, ..., V_n) \) is GAC iff it is GAC with respect to all variables in its scope.

A CSP is GAC if and only if all of its constraints are GAC.
Say we find a value $d$ of variable $V_i$ that is **not consistent** wrt a constraint: that is, there is **no assignments** to the other variables that satisfy the constraint when $V_i = d$:

- $d$ is said to be **Arc Inconsistent**.
- We can **remove** $d$ from the domain of $V_i$ as this value cannot lead to a solution (much like Forward Checking, but more powerful).

**Example:** $C(X, Y) : X > Y$

$Dom[X] = \{1, 5, 11\}$, $Dom[Y] = \{3, 8, 15\}$

$X = 1$ is arc inconsistent

$Y = 15$ is arc inconsistent
Pruning the domain of a variable to make a constraint GAC can make a different constraint no longer GAC.

Example: $C_1(X, Y) : X > Y$, $C_2(Y, Z) : Y > Z$

$$\text{Dom}[X] = \{1, 5, 11\}, \text{Dom}[Y] = \{3, 8, 15\}, \text{Dom}[Z] = \{4, 6\}$$

- To make $C_1$ GAC we must prune 1 from $\text{Dom}[X]$
- To make $C_2$ GAC we must prune 3 from $\text{Dom}[Y]$
- Now $C_1$ is no longer GAC since $X=5$ is arc inconsistent

$\Rightarrow$ must remove 5 from $\text{Dom}[X]$

Need to re-achieve GAC for some constraints whenever a domain value is pruned.
GAC: Considerations

- All constraints must be **GAC at every node** of the search space. This is accomplished by removing from the domains of the variables all **arc inconsistent values**:
  - Every time we assign a value to a variable \( V \), we check all constraints over \( V \) and prune arc inconsistent values from the current domain of the other variables of the constraints.

- Removing a value from a variable domain may trigger **further inconsistency**. We have to repeat the procedure until everything is consistent:
  - Have a queue of constraints that need to be made GAC.
  - Constraints are added (back) to the queue if the domain of one of their variables is changed.
  - The procedure stops when the queue is empty.

- After backtracking from the current assignment the values that were pruned (as a result of that assignment) must be restored. Some **bookkeeping** needs to be done to remember which values were pruned by which assignment.
GAC: Map Coloring Example

\[ C_1(SA, WA) : SA \neq WA, \quad C_2(NT, WA) : NT \neq WA, \quad C_3(SA, NT) : SA \neq NT \]
\[ C_4(SA, Q) : SA \neq Q, \quad C_5(SA, NSW) : SA \neq NSW, \quad C_6(SA, V) : SA \neq V \]
\[ C_7(NT, Q) : NT \neq Q, \quad C_8(Q, NSW) : Q \neq NSW, \quad C_9(NSW, V) : NSW \neq V \]

**Value Assignments:** \( WA = R \)
Then, for \( SA \) and \( NT \), \( R \) becomes arc inconsistent wrt \( C_1 \) and \( C_2 \).

**Current Domains:**

\[
\text{Dom}[SA] = \{R, G, B\} \quad \text{Dom}[NT] = \{R, G, B\} \\
\text{Dom}[Q] = \{R, G, B\} \quad \text{Dom}[NSW] = \{R, G, B\} \\
\text{Dom}[V] = \{R, G, B\} \quad \text{Dom}[T] = \{R, G, B\}
\]
GAC: Map Coloring Example

\[ C_1(SA, WA) : SA \neq WA, \quad C_2(NT, WA) : NT \neq WA, \quad C_3(SA, NT) : SA \neq NT \]
\[ C_4(SA, Q) : SA \neq Q, \quad C_5(SA, NSW) : SA \neq NSW, \quad C_6(SA, V) : SA \neq V \]
\[ C_7(NT, Q) : NT \neq Q, \quad C_8(Q, NSW) : Q \neq NSW, \quad C_9(NSW, V) : NSW \neq V \]

**Value Assignments:** \( WA = R, Q = G \)
Then, for \( SA, NT \) and \( NSW \), \( G \) becomes arc inconsistent wrt \( C_4, C_7, \) and \( C_8 \).

**Current Domains:**
\[ \text{Dom}[SA] = \{G, B\} \quad \text{Dom}[NT] = \{G, B\} \]
\[ \text{Dom}[Q] = \{R, G, B\} \quad \text{Dom}[NSW] = \{R, G, B\} \]
\[ \text{Dom}[V] = \{R, G, B\} \quad \text{Dom}[T] = \{R, G, B\} \]

\[ \text{Dom}[SA] = \{G, B\} \quad \text{Dom}[NT] = \{G, B\} \]
\[ \text{Dom}[Q] = \{R, G, B\} \quad \text{Dom}[NSW] = \{R, G, B\} \]
\[ \text{Dom}[V] = \{R, G, B\} \quad \text{Dom}[T] = \{R, G, B\} \]

\[ SA = R \]

\[ C_3 \]
\[ C_4 \]
\[ C_5 \]
\[ C_6 \]
\[ C_7 \]
\[ C_8 \]
\[ C_9 \]

must remove \( B \) from \( \text{Dom}[SA] \) to make \( C_3 \) GAC => \( \text{Dom}[SA] = \{G\} \) DWO!
Value Assignments: $Q_1 = 1$
Then $Q_2 = 1, Q_2 = 2, Q_3 = 1, Q_3 = 3, Q_4 = 1, Q_4 = 4$
become arc inconsistent.

Current Domains:

$\text{Dom}[Q_2] = \{1, 2, 3, 4\}$ $\text{Dom}[Q_3] = \{1, 2, 3, 4\}$ $\text{Dom}[Q_4] = \{1, 2, 3, 4\}$

Put all constraints on the queue

$Q_2 = 3$; no consistent $Q_3$ value $\Rightarrow \text{Dom}[Q_2] = \{3, 4\}$
$Q_3 = 4$: no consistent $Q_2$ value $\Rightarrow \text{Dom}[Q_3] = \{2, 4\}$

$Q_3 = 2$: no consistent $Q_4$ value $\Rightarrow \text{Dom}[Q_3] = \emptyset$

DWO
Value Assignments: $Q_1 = 2$
Then $Q_2 = 1$, $Q_2 = 2$, $Q_2 = 3$, $Q_3 = 2$, $Q_3 = 4$, $Q_4 = 2$ become arc inconsistent.

Current Domains:

- $Dom[Q_2] = \{1, 2, 3, 4\}$
- $Dom[Q_3] = \{1, 2, 3, 4\}$
- $Dom[Q_4] = \{1, 2, 3, 4\}$

$Q_3 = 3$: no consistent $Q_2$ Value $\Rightarrow Dom[Q_3] = \{1, 3\}$

$\Rightarrow$ Put all constraints on the queue
\( Q_4 = 1 \): no consistent \( Q_3 \) value \( \Rightarrow \text{Dom}[Q_4] = \{9, 3, 47\} \)

\( Q_4 = 4 \): no consistent \( Q_2 \) value \( \Rightarrow \text{Dom}[Q_4] = \{8, 47\} \)

\( \Rightarrow \text{Dom}[Q_2] = \{47\} \quad \text{Dom}[Q_3] = \{17\} \)

\( \text{Dom}[Q_4] = \{3\} \)

Now search no longer has to branch since only one value left for each variable. It just walks down to a solution assigning each variable in turn.
GAC-Based Propagation

- **Plain Backtracking** check a constraint only when it has zero unassigned variables.

- **Forward checking** checks a constraint only when it has one unassigned variables.

- **GAC** checks all constraints, leading to much more pruning in general.
  - Even at the root before any variables have been assigned, we can get some pruning by making the constraints GAC consistent.
  - Checking for consistency can be done as a pre-processing step, or it can be directly integrated into a search algorithm.
  - If we apply arc consistency propagation during search the search tree’s size will typically be much reduced in size.
  - **Note:** GAC enforce does NOT find a solution! (why?) To find a solution we must use do search while enforcing GAC.
def GAC_Enforce()
// GAC-Queue contains all constraints one of whose variables has
// had its domain reduced. At the root of the search tree we can
// first run GAC_Enforce with all constraints on GAC-Queue
1. while GACQueue not empty
2. C = GACQueue.extract()
3. for V := each member of scope(C)
4. for d := CurDom[V]
5. Find an assignment A for all other variables in scope(C)
such that C(A ∪ V=d) is True
6. if A not found
7. CurDom[V] = CurDom[V] - d  # remove d from the domain of V
8. if CurDom[V] == {}  # DWO for V
9. empty GACQueue
10. return DWO  # return immediately
11. else
12. push all constraints C’ such that V ∈ scope(C’)
    and C’ ∉ GACQueue on to GACQueue
13. return TRUE  # loop exited without DWO
def GAC(Level):
1. if all Variables assigned
2. PRINT Value of each Variable
3. EXIT or RETURN  # EXIT for only one solution
               # RETURN for more solutions
4. V := PickUnassignedVariable()
5. Assigned[V] := TRUE
6. for d := each member of CurDom(V)
7.   Value[V] := d
8. Prune all values other than from CurDom[V]
9. for each constraint C whose scope contains V
10.   Put C on GACQueue
11. if (GAC_Enforce() != DWO)
12.   GAC(Level+1)  # all constraints were
13. RestoreAllValuesPrunedByFCCheck()
14. Assigned[V] := FALSE  # UNDO as we have tried all of V’s values
15. RETURN
When all constraints are GAC three outcomes are possible:

1. Each domain has a **single value**.

2. At least one domain is **empty**.

3. Some domains have **more than one value**.
   Need to solve this new CSP (usually) **simpler** problem: same constraints, domains have been reduced
GAC: Complexity

• **BT worst-case running time:** $O(d^N)$, where $d$ is the max size of a variable domain, and $N$ is the number of variables.

• **Worst-case complexity** of arc consistency procedure on a problem with $N$ variables, $c$ binary constraints, and $d$ be the max size of a variable domain:
  
  – How often will we prune the domain of variable $V$?
  
  – How many constraints will be put on the queue when pruning domain of a variable $V$?
  
  – Sum of degrees of all variables:
  
  – Overall, how many constraints will be put on the queue?
  
  – Checking consistency of each constraint:
  
  – **Overall Complexity:**
• For CSP with higher-order constraints:
  
  – Checking consistency of a constraint $C$ with arity $k$ (i.e., $|\text{scope}(C)| = k$):
  
  – It can be shown that the Overall Complexity is:

More readings:

A **support** for a value assignment $V = d$ in a constraint $C$ is an **assignment** $A$ to all of the other variables in $\text{scope}(C)$ s.t. $A \cup \{V = d\}$ satisfies $C$.

A constraint $C$ is **GAC** if for **every** variable $V_i$ in its scope, **every** value $d_i \in CurDomain(V_i)$ has a **support** in $C$. 
• Smarter implementations keep track of **supports** to avoid having to search though all possible assignments to the other variables for a satisfying assignment.

• Rather than search for a satisfying assignment to $C$ containing $V = d$, they check if the **current support** is still valid.

• Also they take advantage that a support for $V = d$, e.g. $\{V = d, X = a, Y = b, Z = c\}$ is also a support for $X = a, Y = b$, and $Z = c$.

• Another key development in practice is that for some constraints this computation can be done in polynomial time.

  **Example:** Ideas from graph matching theory are used to find support for variables in $\text{All} − \text{diff}(V_1, .., V_n)$ in **polynomial time**.

The special purpose algorithms for achieving GAC on particular types of constraints are very important in practice.