

# Artificial Intelligence

## AI-2002

### Lecture 9

Mahzaib Younas  
Lecturer Department of Computer Science  
FAST NUCES CFD

# Constraint Satisfaction Problems

- A CSP consists of **variables with constraints** on them. It contains
  - Finite *set of variables*  $X_1, X_2, \dots, X_n$
  - *Nonempty domain* of possible values for each variable  $D_1, D_2, \dots, D_d$
  - Finite *set of constraints*  $C_1, C_2, \dots, C_m$
  - Each constraint  $C_i$  limits the values that variables can take, e.g.,  $X_1 \neq X_2$
- A **state** is defined as an **assignment of values** to some or all variables.

# Constraint Satisfaction Problems

## Assignment:

- A state of the problem is defined by assigning values to some or all of the variables,  $\{X_i = v_i, X_j = v_j\}$ .

## Consistent assignment:

- If the assignment *does not violate the constraints*.

## Complete assignment:

- An assignment is **complete** *when every variable is assigned a value*.

## Commutative:

- Variable assignments are **commutative**
  - e.g. [ step 1: WA = red; step 2: NT = green ] equivalent to [ step 1: NT = green; step 2: WA = red ]

# Constraint Satisfaction Problems

## Solution to CSP:

- A *solution* to a CSP is a complete assignment that satisfies all constraints.
- Some CSPs require a solution that maximizes an *objective function*.

## Domain:

- Each variable  $X_i$  has a **nonempty domain**  $D_i$  of possible values.
  - e.g. Color is assigned to a variable  $X_i$ . Domain  $D_i$  may be set of possible colors like {R, G, B}.

# Map-Colouring

- **Variables:**

$WA, NT, Q, NSW, V, SA, T$

- **Domains:**

$D_i = \text{red; green; blue}$

- **Constraints:**

adjacent regions must have different colours

- e.g.,  $WA \neq NT$
- $(WA; NT) \in [(\text{red}; \text{green}); (\text{red}; \text{blue}); (\text{green}; \text{red}); (\text{green}; \text{blue}) \dots]$



# Map-Colouring



**Solutions** are **complete** and **consistent** assignments,

- e.g., WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

# Sudoku

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

- **Variables:** empty cells
- **Domains:** numbers between 1 to 9
- **Constraints:** rows, columns, boxes contain all different numbers

# N-Queens

- **Variables:**

$Q_i$

- **Domains:**

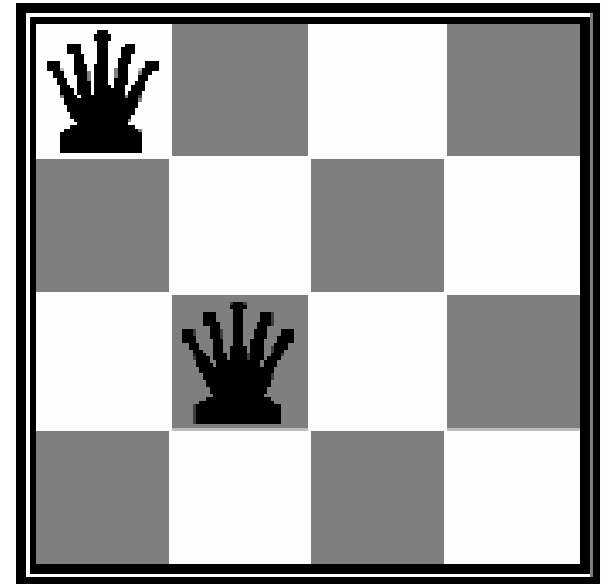
$D_i = \{1, 2, 3, 4\}$

- **Constraints:**

- Queen can NOT be in same row
- Queen can NOT be in same column
- Queen can NOT be in same diagonal

- **Valid values** for  $(Q_1, Q_2)$  are:

- (1,3) (1,4) (2,4) (3,1) (4,1) (4,2)



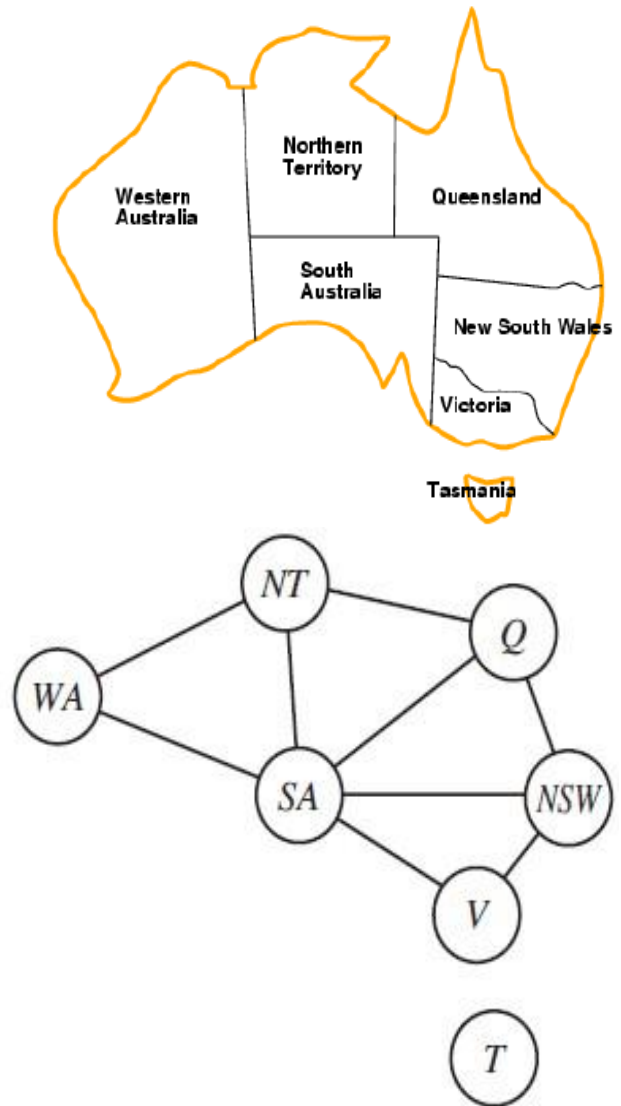
$Q_1 = 1 \quad Q_2 = 3$



# Constraint Satisfaction Problems

## Constraint Graph:

- Constraint Satisfaction Problem (CSP) can be visualized as a *constrained graph*.
  - The **nodes** of the graph correspond to *variables* of the problem
  - The **arcs** correspond to the *constraints*.



# Constraint Satisfaction Problems

## Finite Domain:

- The simplest kind of CSP involves variables that have **domains that are limited or restricted**.
  - Map coloring problems are of this kind.

## Boolean CSP:

- Finite-domain CSPs include Boolean CSPs, *whose variables can either be true or false*.

## Continuous Domain:

- Domain in which there is a sequence of assignment to the variables.
  - The scheduling experiments via telescope requires very precise timings of observation

# Constraint Satisfaction Problems

## Constraint

## Language:

- With **infinite domains**, it is *NO longer possible to describe constraints* by enumerating all combinations of values.
- Instead, a **Constraint Language** is used in which **set of rules are specified**.
  - If  $job_1$ , which takes 5 days, must precede  $job_3$ , then a constraint language of algebraic inequalities such as  $start\ job_1 + 5 \leq start\ job_3$  will be required.

# Constraint Satisfaction Problems

## Types of Constraints

### Unary Constraint:

- The simplest type of constraint, which **restricts the value of a single variable**, is called Unary Constraint.
  - e.g.  $SA \neq \text{Green}$

### Binary Constraint:

- It relates **two variables** or **involves pair of variables**.
  - e.g.  $SA \neq \text{NSW}$

### Constraint Hypergraph:

- Higher order constraints involve **three** or **more variables**. A Constraint Hypergraph represents these constraints.
  - e.g., crypt arithmetic column constraints

# Crypt-arithmetic

- **Variables:**
  - D, E, M, N, O, R, S, Y
- **Domains:**
  - $D_i = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- **Constraints:**
  - $M \neq 0, S \neq 0$
  - $D \neq E, D \neq M, D \neq N$
  - $Y = D + E$  OR  $Y = D + E - 10$

S E N D  
+ M O R E  

---

M O N E Y

# Crypt-arithmetic

- **Variables:**

- D, E, M, N, O, R, S, Y

- **Domains:**

- $D_i = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

- **Constraints:**

- $M \neq 0, S \neq 0$
- $D \neq E, D \neq M, D \neq N$
- $Y = D + E$  OR  $Y = D + E - 10$

$$\begin{array}{rcccccc} & S(9) & E(5) & N(6) & D(7) & & \\ + & M(1) & O(0) & R(8) & E(5) & & \\ \hline M(1) & O(0) & N(6) & E(5) & Y(2) & & \end{array}$$

# Constraint Satisfaction Problems

## Linear Constraint:

- Constraint in which variable appears only in *linear form* is called **Linear Constraint**.
- Linear Constraints are solvable.

## Non-Linear Constraint:

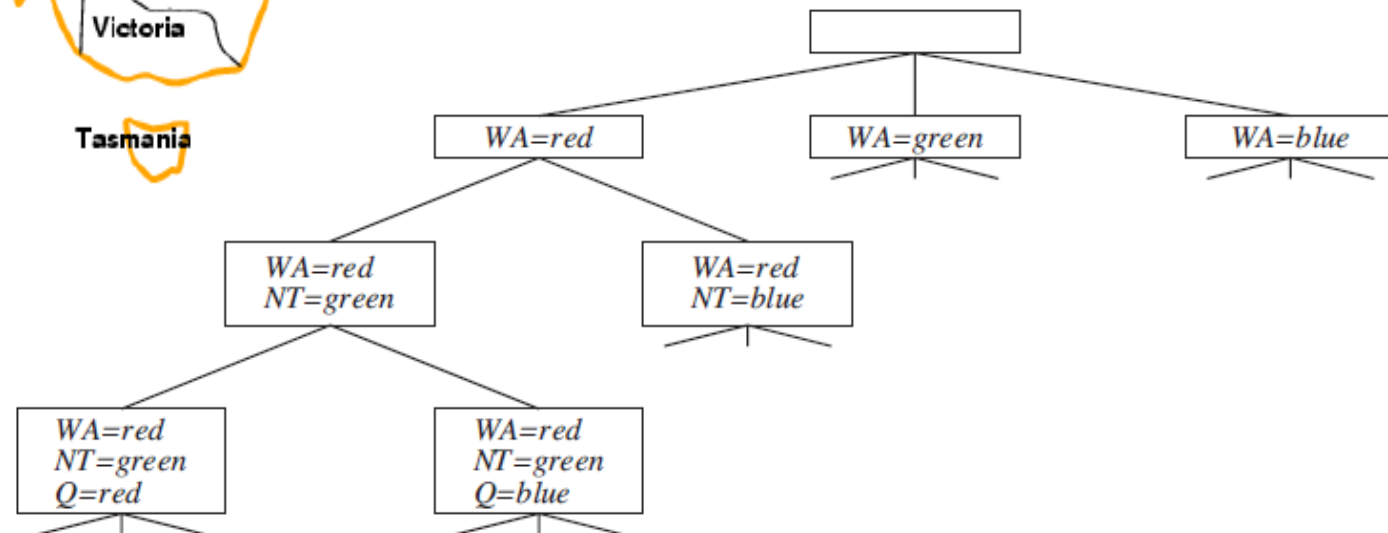
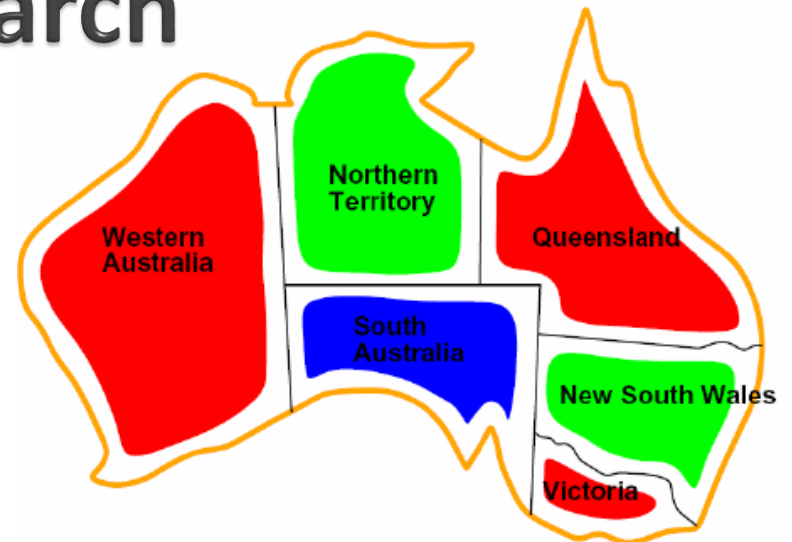
- Constraint in which variables appear in *non-linear form* is called **Non-linear Constraint**.
- Non-linear Constraints are undecidable.

# CSP as a Standard Search

- A CSP can easily expressed as a standard search problem,
  - **Initial State:** *the empty assignment {}*
  - **Successor function:** *Assign value to unassigned variable provided that there is no conflict*
  - **Goal test:** *the current assignment is complete*
  - **Path cost:** *a constant cost for every step*
- Solution is found at depth  $n$ , for  $n$  variables
  - Hence **depth first search** can be used
  - Only need to consider **assignments to a single variable** at each node



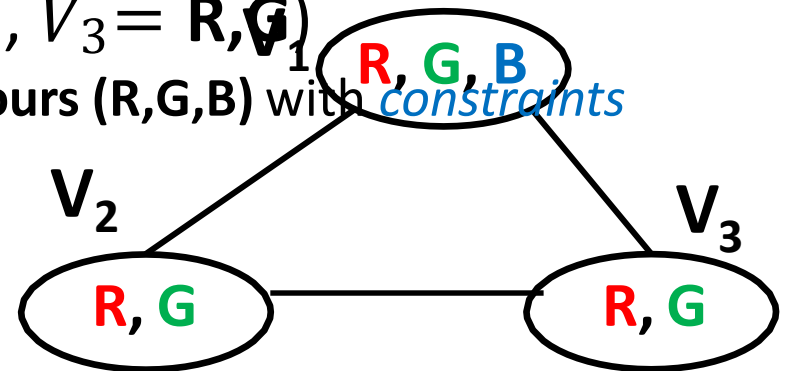
# CSP as a Standard Search



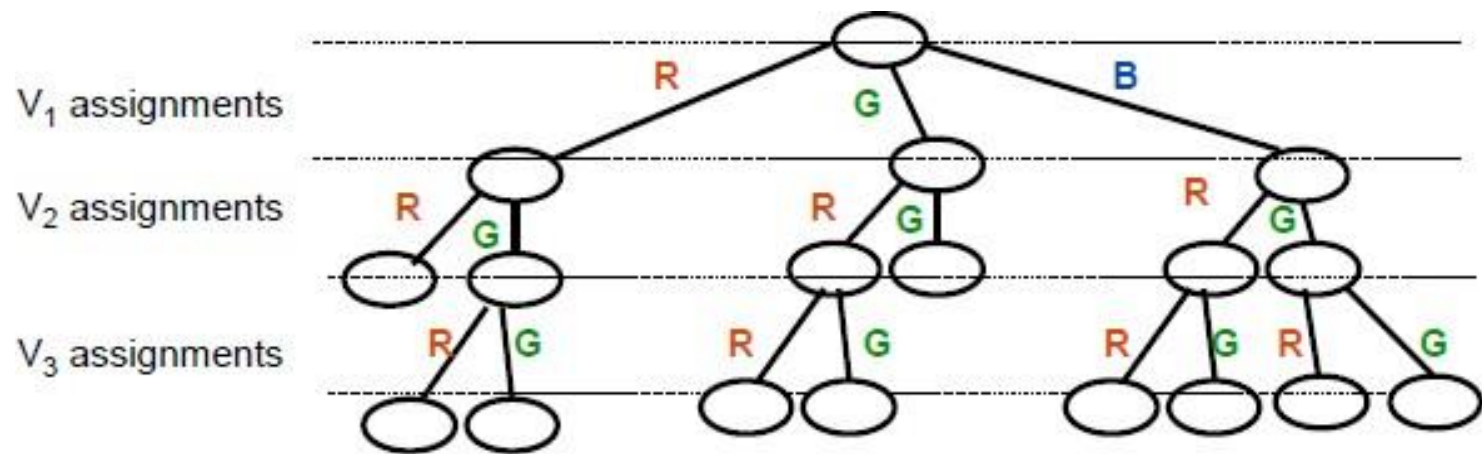
# CSP as a Standard Search

variables.

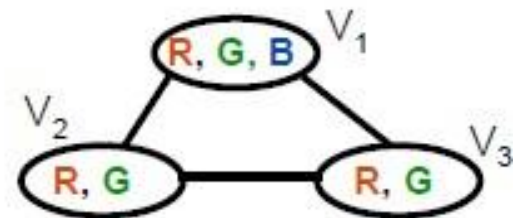
- **State:** assignment to  $k$
- **Successor:** The successor of a state is obtained by assigning a value to variable, keeping others Unchanged
- **Start state:** ( $V_1 = \mathbf{R,G,B}$ ,  $V_2 = \mathbf{R,G}$ ,  $V_3 = \mathbf{R,G}$ )
- **Goal state:** All variables assigned colours (R,G,B) with *constraints satisfied*.



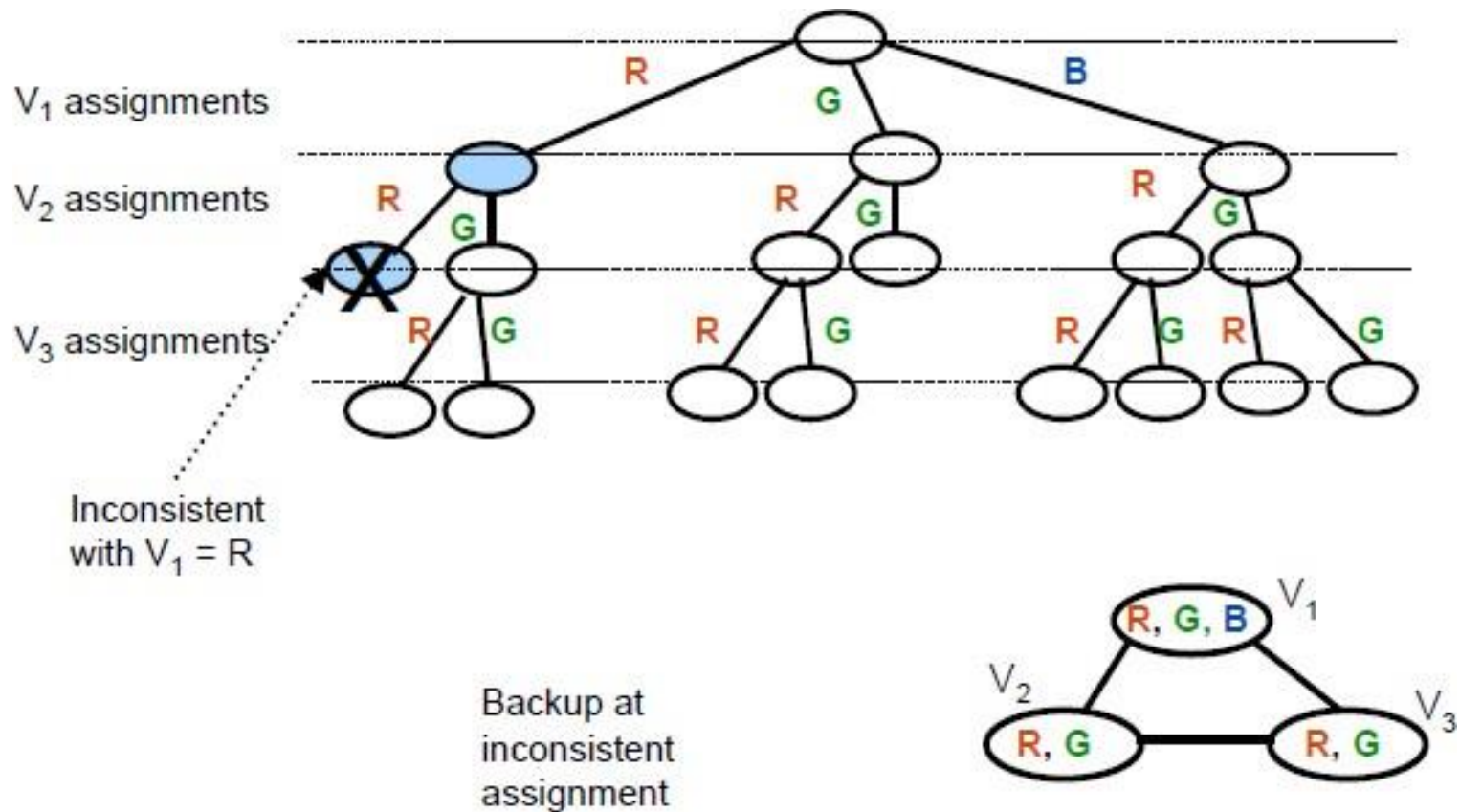
# CSP as a Standard Search



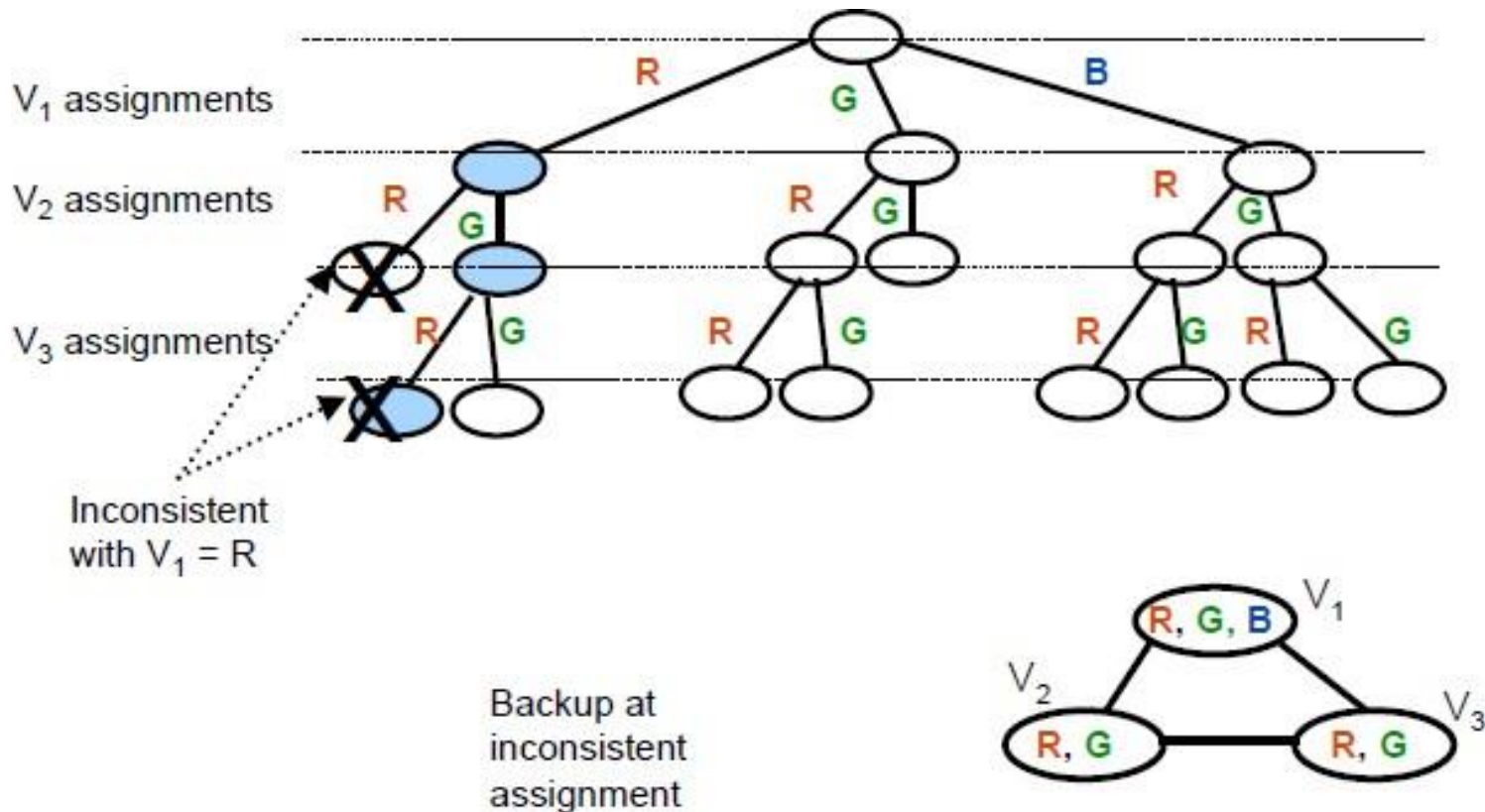
**Depth First Search**  
can be performed



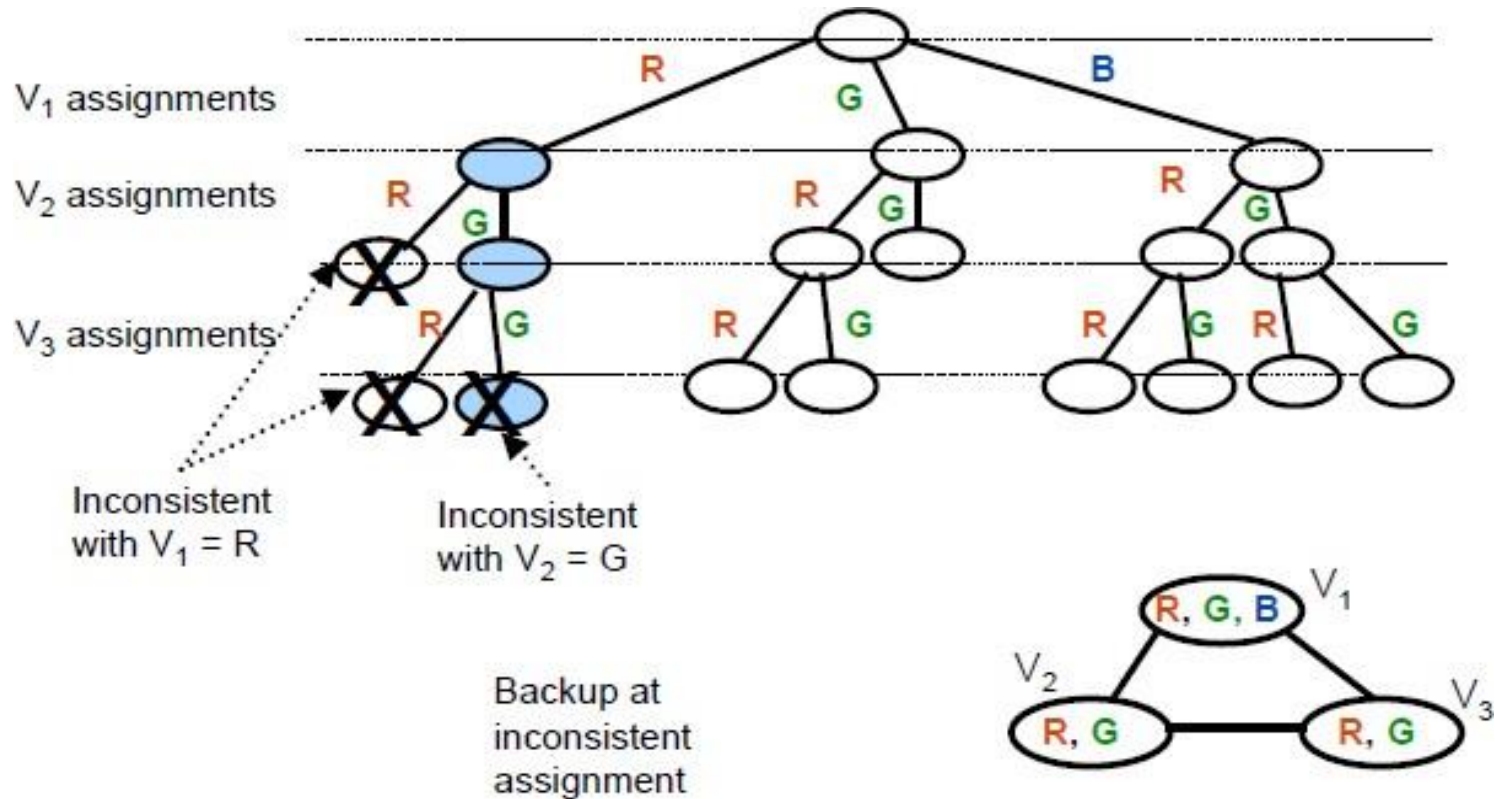
# CSP as a Standard Search



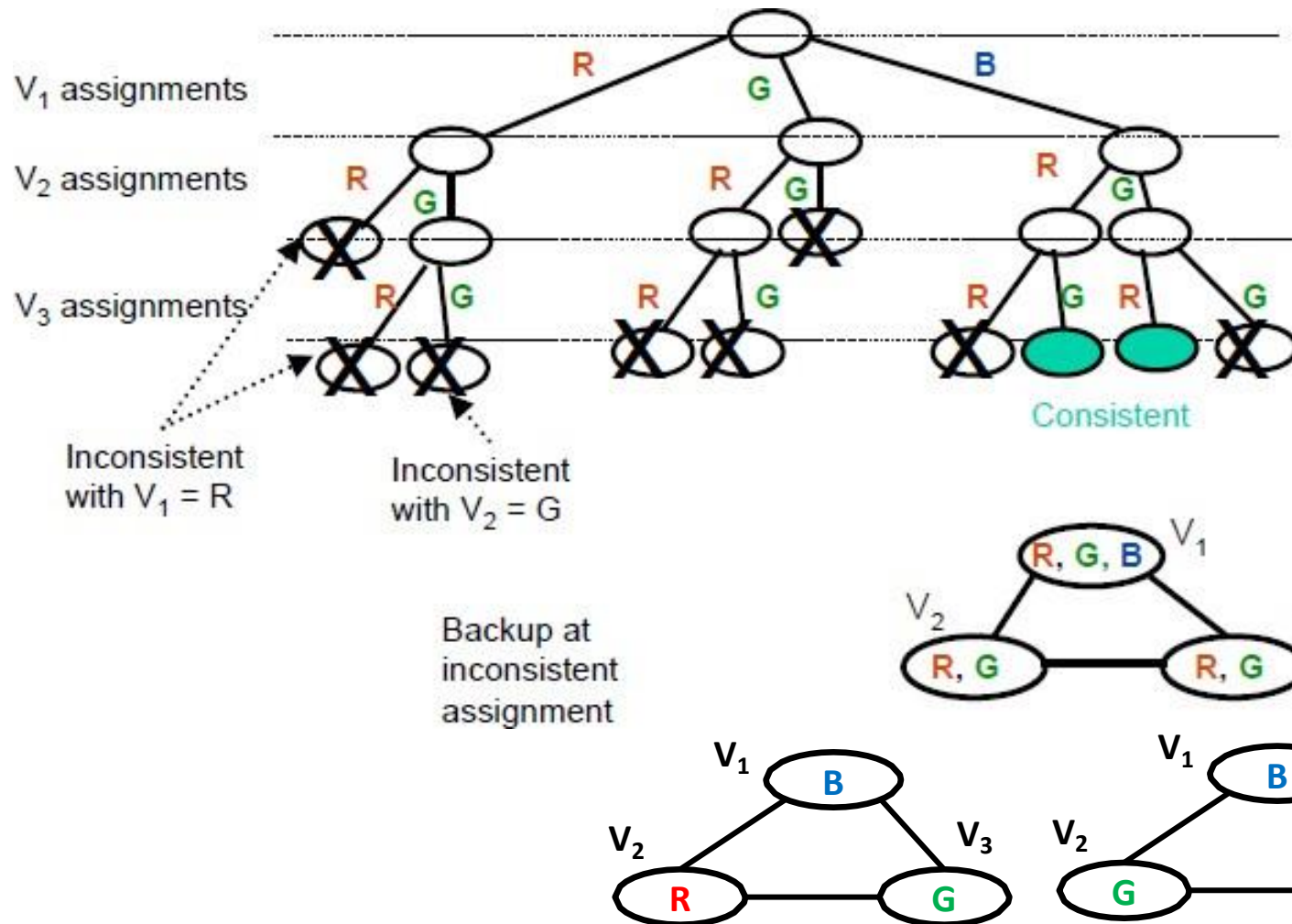
# CSP as a Standard Search



# CSP as a Standard Search



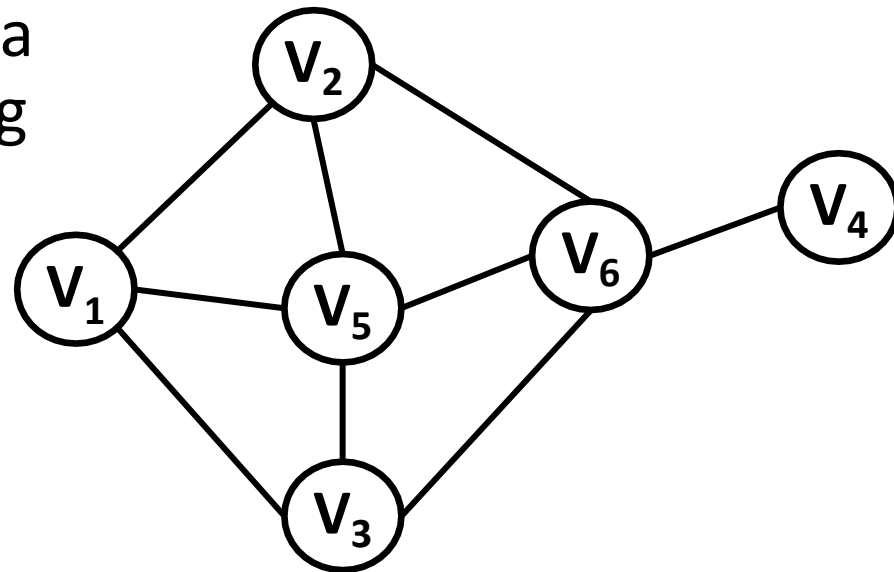
# CSP as a Standard Search





# CSP as a Standard Search

- **State**: assignment to  $k$  variables.
- **Successor**: The successor of a state is obtained by assigning a value to variable, keeping others unchanged.
- **Start state**: ( $V_1 = ?$ ,  $V_2 = ?$ ,  $V_3 = ?$ ,  $V_4 = ?$ ,  $V_5 = ?$ ,  $V_6 = ?$ )
- **Goal state**: All variables assigned colours (R,G,B) with constraints satisfied.

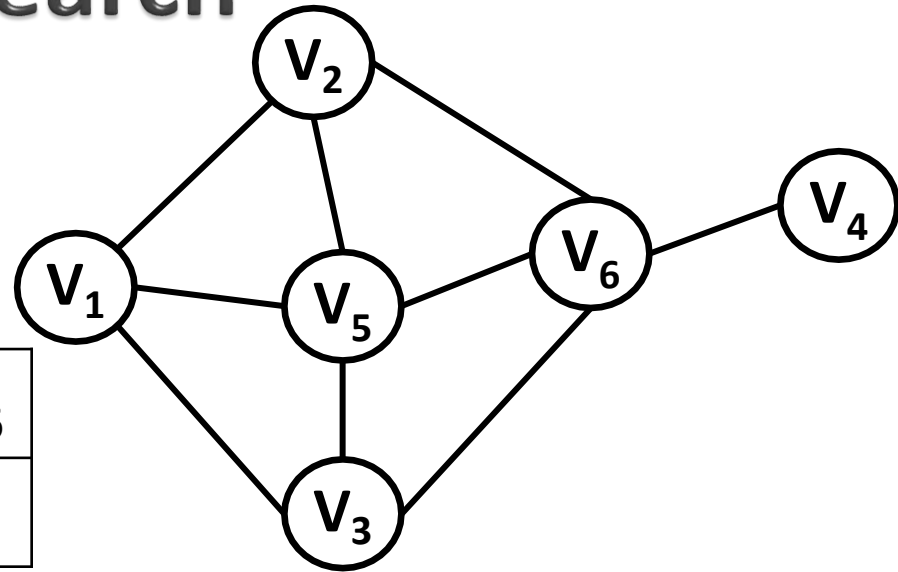




# CSP as a Standard Search

## Depth First Search

can be performed



$V_1$	$V_2$	$V_3$	$V_4$	$V_5$	$V_6$
?	?	?	?	?	?

$V_1$	$V_2$	$V_3$	$V_4$	$V_5$	$V_6$
B	?	?	?	?	?

$V_1$	$V_2$	$V_3$	$V_4$	$V_5$	$V_6$
G	?	?	?	?	?

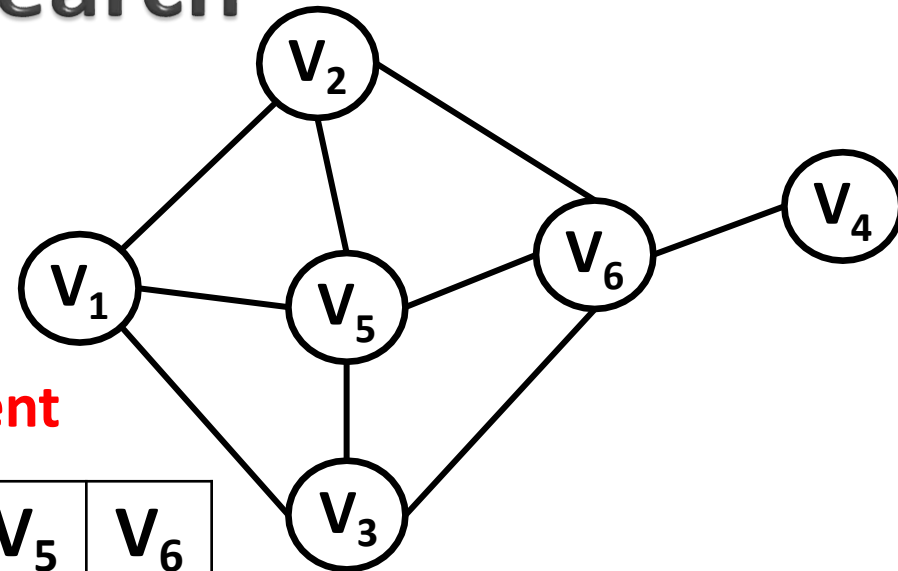
$V_1$	$V_2$	$V_3$	$V_4$	$V_5$	$V_6$
R	?	?	?	?	?

# CSP as a Standard Search

$V_1$	$V_2$	$V_3$	$V_4$	$V_5$	$V_6$
B	?	?	?	?	?

Dumb Assignment

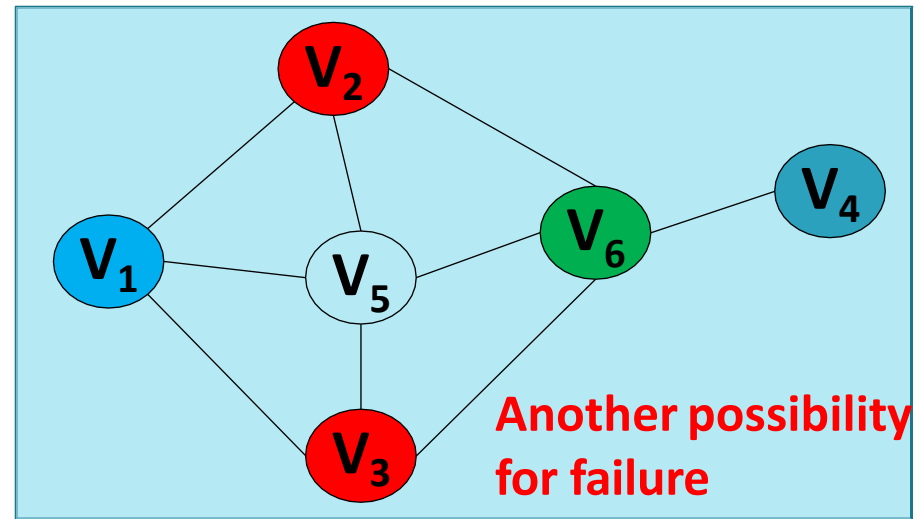
$V_1$	$V_2$	$V_3$	$V_4$	$V_5$	$V_6$
B	B	?	?	?	?



**Recursively:**

- For every possible value in  $D$ :
  - Set the next unassigned variable in the successor to that value
- Evaluate the successor of current state with this variable assignment
- Stop as soon as a solution is found

$V_1$	$V_2$	$V_3$	$V_4$	$V_5$	$V_6$
B	?	?	?	?	?

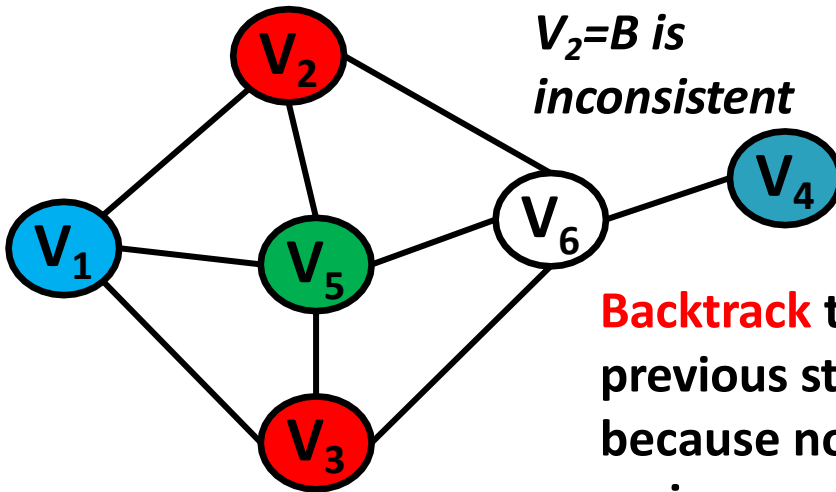


$V_1$	$V_2$	$V_3$	$V_4$	$V_5$	$V_6$
B	B	?	?	?	?

$V_1$	$V_2$	$V_3$	$V_4$	$V_5$	$V_6$
B	R	?	?	?	?

$V_1$	$V_2$	$V_3$	$V_4$	$V_5$	$V_6$
B	R	R	?	?	?

$V_1$	$V_2$	$V_3$	$V_4$	$V_5$	$V_6$
B	R	R	B	G	?



Backtrack to the previous state because no valid assignment is for  $V_6$

# CSP as a Standard Search

- For every possible value for  $x$  in  $D$ :
  - If assigning  $x$  to the next unassigned variable  $V_{k+1}$  **does not violate** any constraint with the  $k$  already assigned variables:
    - ▮ Set the value of the variable  $V_{k+1}$  to  $x$
    - ▮ Evaluate the successors of the current state with this variable assignment
- If **no valid assignment** is found:
  - Backtrack to previous state
- Stop as soon as a solution is found

# CSP as a Standard Search

- Additional computation: At each step, we *need to evaluate the constraints associated* with the current candidate assignment (**variable, value**).
- Uninformed search, we can improve by predicting:
  - What is the **effect of assigning a variable** on all of the other variables?
  - Which **variable should be assigned next** and in **which order** should the values be evaluated?
  - When a branch fails, how can we **avoid repeating the same mistake**?

# Consistency

# Node Consistency

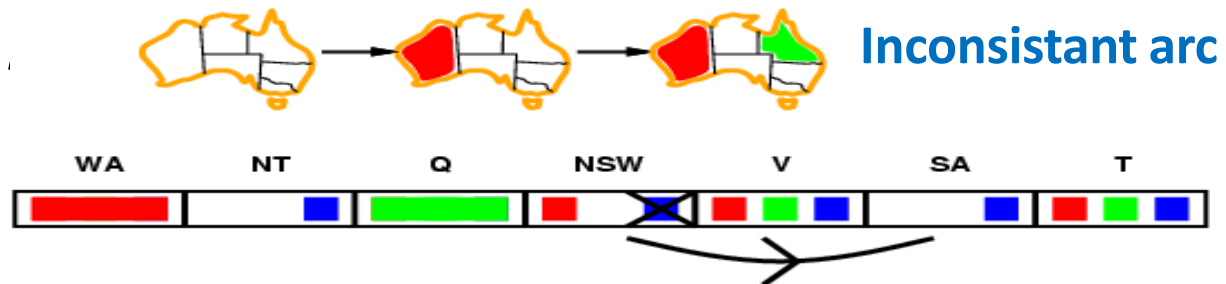
- A single variable (corresponding to a node in the CSP network) is **node-consistent** if all the values in the variable's domain satisfy the variable's *unary constraints*.

## Example:

- In a map-coloring problem, where
  - SA dislikes green,
  - the variable SA starts with domain  $\{red, green, blue\}$ ,
  - we can make it node consistent by eliminating green,
  - SA with the reduced domain  $\{red, blue\}$

# Arc Consistency

- A variable in a CSP is **arc-consistent** if every value in its domain satisfies the variable's *binary constraints*.
- *Arc consistency* eliminates values from the domain of variables that can never be part of a consistent solution.
- Directed arc  $(V_i, V_j)$  is arc consistent if
 
$$\forall x \in D_i \quad \exists y \in D_j$$
 such that  $(x, y)$  is allowed by constraint
- For every value





# Arc Consistency

## Example:

- Consider the **constraint**  $Y = X^2$
- The **domain** of both  $X$  and  $Y$  is the set of digits. We can write this constraint explicitly as
$$(X, Y), \{(0, 0), (1, 1), (2, 4), (3, 9)\}.$$
- To **make  $X$  arc-consistent** with respect to  $Y$ , we reduce  $X$ 's domain to  $\{0, 1, 2, 3\}$ .
- If we also **make  $Y$  arc-consistent** with respect to  $X$ , then  $Y$ 's domain becomes  $\{0, 1, 4, 9\}$
- The whole CSP is arc-consistent.

# Path Consistency

- **Path consistency** tightens the binary constraints by using implicit constraints that are inferred by looking at triples of variables.
- A two-variable set  $\{X_i, X_j\}$  is path-consistent with respect to a third variable  $X_m$  if,
  - for every assignment  $\{X_i = a, X_j = b\}$  consistent with the constraints on  $\{X_i, X_j\}$ ,
  - there is an assignment to  $X_m$  that satisfies the constraints on  $\{X_i, X_m\}$  and  $\{X_m, X_j\}$ .

# $k$ -consistency

- Stronger form of propagation
- A **CSP is  $k$ -consistent** if,
  - for any set of  $k - 1$  variables and
  - for any consistent assignment to those variables,
    - a consistent value can always be assigned to any  $k^{\text{th}}$  variable

## 1-consistency:

- given the empty set, we can make any set of one variable consistent: *node consistency*.

# $k$ -consistency

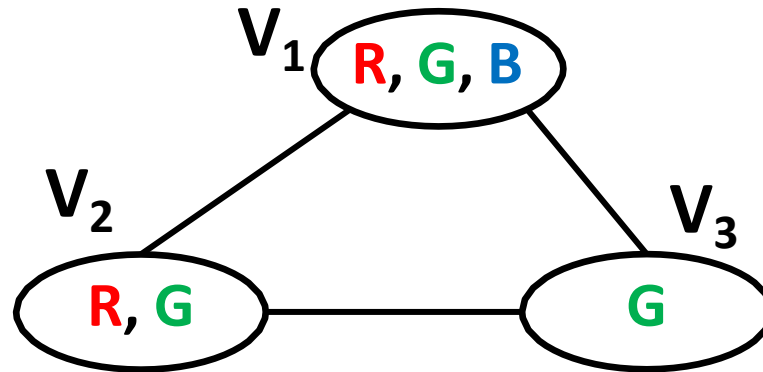
## 2-consistency:

- is the same as *arc consistency*.
- Suppose a CSP with  $n$  nodes and make it strongly  $n$ -consistent (i.e., *strongly  $k$ -consistent* for  $k = n$ ).

## $k$ -consistency:

- A CSP is **strongly  $k$ -consistent** if it is  $k$ -consistent and is also  $(k - 1)$ -consistent,  $(k - 2)$ -consistent, . . . all the way down to 1-consistent

# Arc Consistency



## Different Colour Constraints

- Each undirected constraint arc is really **two directed constraint arcs**, the effects must be then from examining BOTH arcs.