CSC242: Introduction to Artificial Intelligence

Lecture 2.1

Please put away all electronic devices

Announcements

- Exam 1 has been graded
    - Feedback in "My Grades"
    - Complaints: email me today
    - Help: Study session
- Project 1 is being graded
    - Watch BlackBoard for announcements

Announcements

- Project 2 available after class
  - Due Sunday October 10
  - Before Fall Break
  - Exam 2 Thursday October 14

Course Calendar

State-Space Search

Problem (Domain): $\langle S,\ A,\ \textsc{Actions},\ \textsc{Result},\ \textsc{Cost}\rangle$

$\textsc{Actions} : s \in S \rightarrow$

$\{\ a \in A : a$ can be executed (is applicable) in $s\ \}$

$\textsc{Result} : s \in S\ ,\ a \in A \rightarrow$

$s' \in S$ s.t. s' is the result of performing a in s

$\textsc{Cost}$: Assigns a cost to each path/step $c(s, a, s')$

Problem (Instance): $\langle I \in S,\ G \subseteq S\rangle$

Solution: $\langle a_1,\ a_2,\ \ldots,\ a_n\rangle \in A^n$ s.t.

$\textsc{Result}(\cdots\ \textsc{Result}(\textsc{Result}(I,\ a_1),\ a_2)\ \cdots,\ a_n) \in G$

Universal Problem-Solving Procedure

Initialize the frontier to just I

While the frontier is not empty:

    Remove a state s from the frontier

    If $s \in G$:

        Return solution to s

    else:

        Add $successors(s)$ to the frontier

Fail!

State

Problem (Domain): $\langle S,\ A,\ \textsc{Actions},\ \textsc{Result},\ \textsc{Cost}\rangle$

$\textsc{Actions} : s \in S \rightarrow$

$\{\ a \in A : a \text{ can be executed (is applicable) in } s\ \}$

$\textsc{Result} : s \in S\ ,\ a \in A \rightarrow$

$s' \in S \text{ s.t. } s' \text{ is the result of performing } a \text{ in } s$

$\textsc{Cost} : \text{Assigns a cost to each path/step } c(s, a, s')$

Problem (Instance): $\langle I \in S,\ G \subseteq S\rangle$

Solution: $\langle a_1,\ a_2,\ \ldots,\ a_n\rangle \in A^n$ s.t.

$\textsc{Result}(\cdots \textsc{Result}(\textsc{Result}(I,\ a_1),\ a_2)\ \cdots,\ a_n) \in G$

Problem (Domain): $\langle S,\ A,\ \textsc{Actions},\ \textsc{Result},\ \textsc{Cost}\rangle$

$\textsc{Actions} : s \in S \to$

$\{\ a \in A : a \text{ can be executed (is applicable) in } s\ \}$

$\textsc{Result} : s \in S\ ,\ a \in A \to$

$s' \in S \text{ s.t. } s' \text{ is the result of performing } a \text{ in } s$

$\textsc{Cost} : \text{Assigns a cost to each path/step } c(s, a, s')$

Problem (Instance): $\langle I \in S,\ G \subseteq S\rangle$

Solution: $\langle a_1,\ a_2,\ \ldots,\ a_n\rangle \in A^n$ s.t.

$\textsc{Result}(\cdots\ \textsc{Result}(\textsc{Result}(I,\ a_1),\ a_2)\ \cdots,\ a_n) \in G$

Problem (Domain): $\langle S,\ A,\ \textsc{Actions},\ \textsc{Result},\ \textsc{Cost}\rangle$

$\textsc{Actions} : s \in S \rightarrow$

$\{\ a \in A : a$ can be executed (is applicable) in $s\ \}$

$\textsc{Result} : s \in S\ ,\ a \in A \rightarrow$

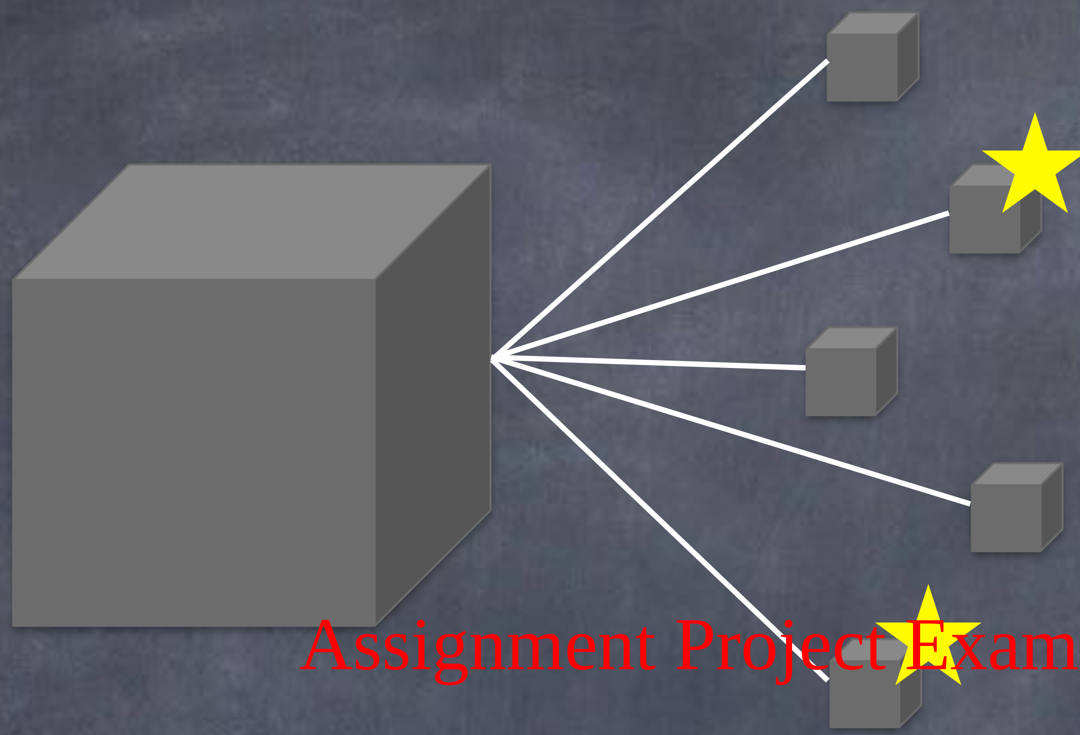
$s' \in S$ s.t. s' is the result of performing a in s

$\textsc{Cost} :$ Assigns a cost to each path/step $c(s, a, s')$

Problem (Instance): $\langle I \in S,\ \boxed{G \subseteq S}\rangle$

Solution: $\langle a_1,\ a_2,\ \ldots,\ a_n\rangle \in A^n$ s.t.

$\textsc{Result}(\cdots\ \textsc{Result}(\textsc{Result}(I,\ a_1),\ a_2)\ \cdots,\ a_n) \in G$



State

```
public class State {
    protected Board board;
    protected Player nextToMove;
    ...
}

public class Board {
    int nrows;
    int ncols;
    Piece[][] grid;

    public Board(int nrows, int ncols) {
        this.nrows = nrows;
        this.ncols = ncols;
        this.grid = new Piece[nrows][ncols];
    }
  ...
}
```

```
public Player getWinner() {
    int nwhite = 0;
    int nblack = 0;
    for (int c=0; c < this.numColumns; c++) {
        for (int r=0; r < this.numRows; r++) {
            Piece piece = this.grid[c][r];
            if (piece != null) {
                if (piece.player == Player.BLACK)
                    nblack += 1;
                } else if (piece.player == Player
                    nwhite += 1;
                }
            }
        }
    }
    if (nblack == 0) {
        return Player.WHITE;
    } else if (nwhite == 0) {
        return Player.BLACK;
    } else {
        return null;
    }
}
```

Representation

Representation

- Impose a structure on the representation of states

Representation

- Impose a structure on the representation of states

- Using that representation, successor generation and goal tests are domain-independent

Representation

- Impose a structure on the representation of states

- Using that representation, successor generation and goal tests are domain-independent

- Can also develop effective problem- and domain-independent heuristics

Bottom Line

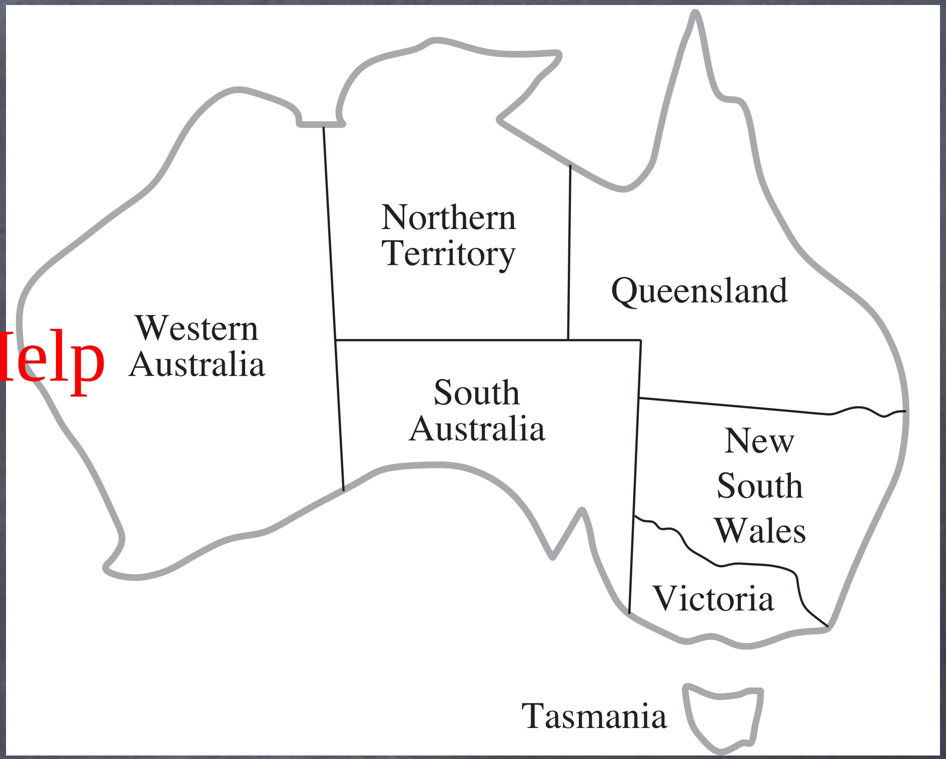
Represent State This Way → Write Code Once → Solve Any Problem

Example

Assign a color to each region such that no two neighboring regions have the same color

Variables

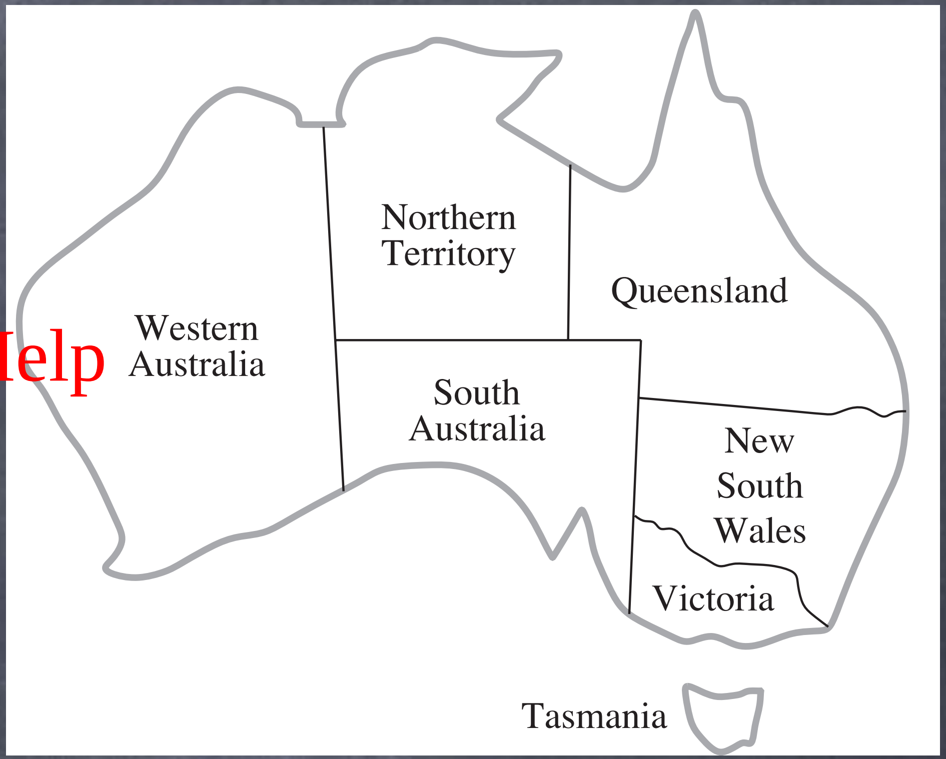
Color WA, NT, Q, NSW, V, SA, T



Values

`Color` WA, NT, Q, NSW, V, SA, T

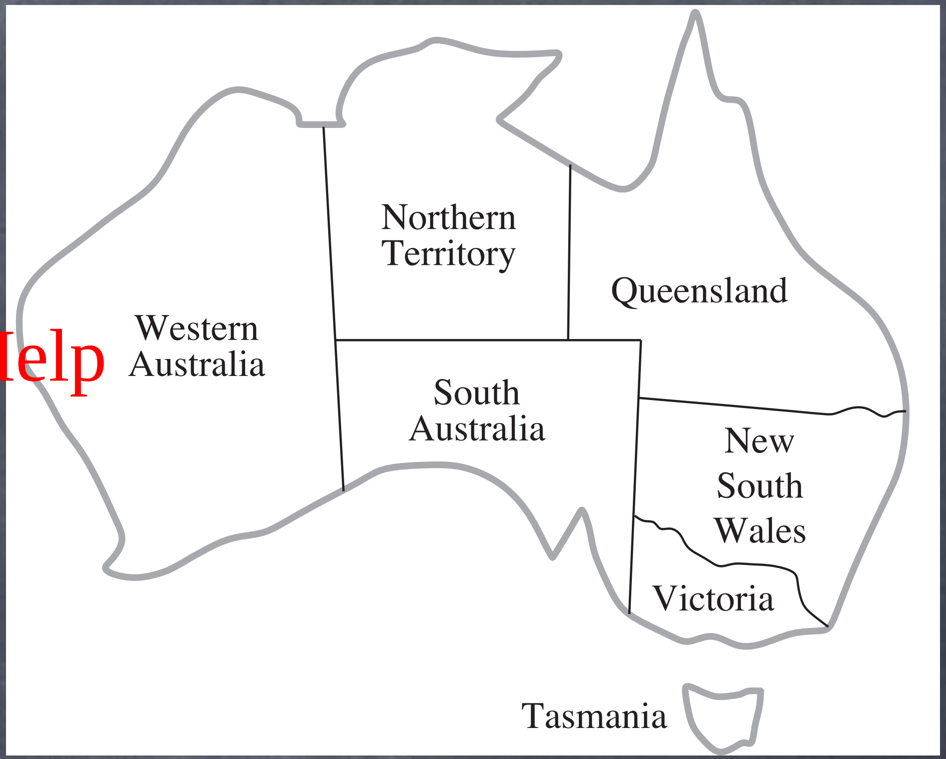
`enum Color` $\{red, green, blue\}$



Assignment

`Color` WA, NT, Q, NSW, V, SA, T

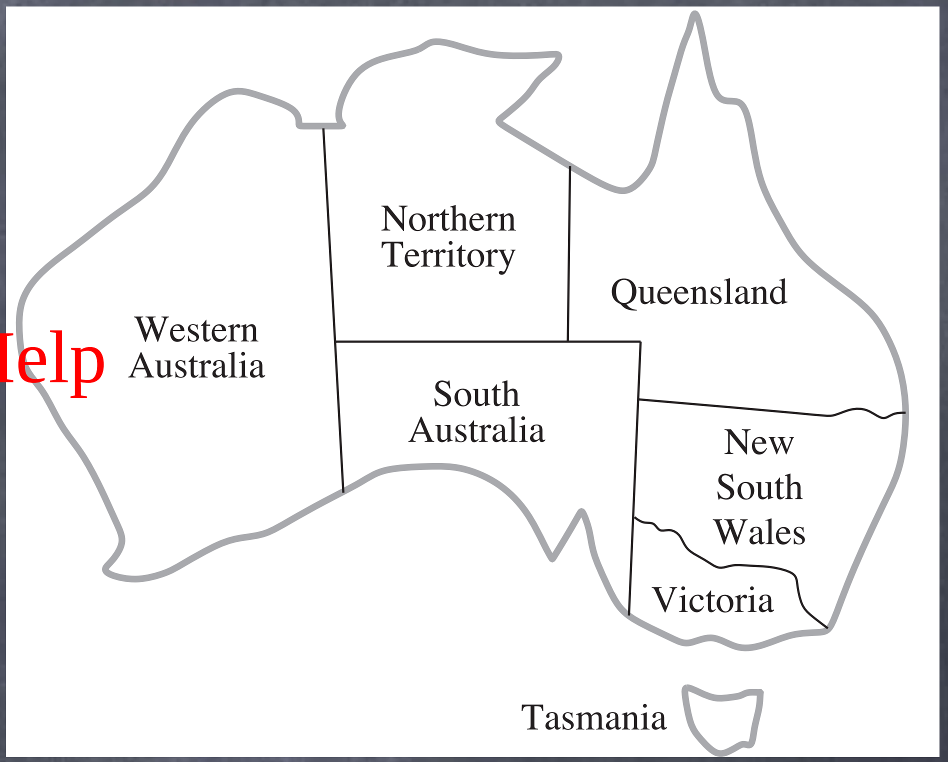
`enum Color` $\{red, green, blue\}$



Assignment

`Color` WA, NT, Q, NSW, V, SA, T

`enum Color` $\{red, green, blue\}$

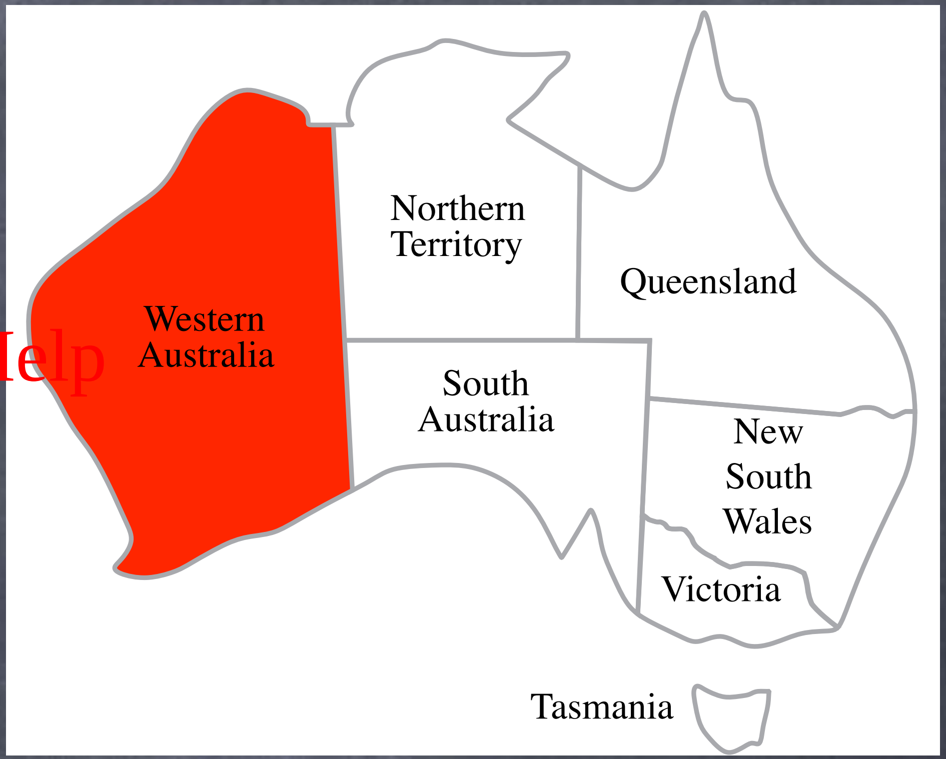


Empty assignment: $\{\ \}$

Assignment

`Color` WA, NT, Q, NSW, V, SA, T

`enum Color` $\{red, green, blue\}$

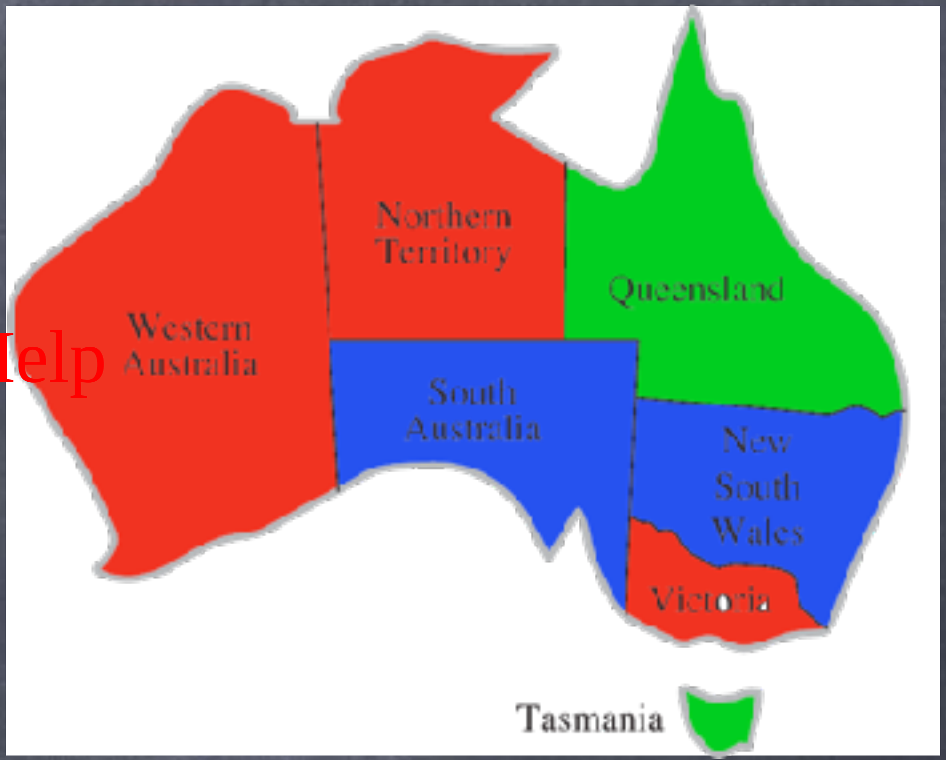


Partial assignment: $\{\ WA{=}red\ \}$

Assignment

`Color` WA, NT, Q, NSW, V, SA, T

`enum Color {`$red, green, blue$`}`



Complete assignment:

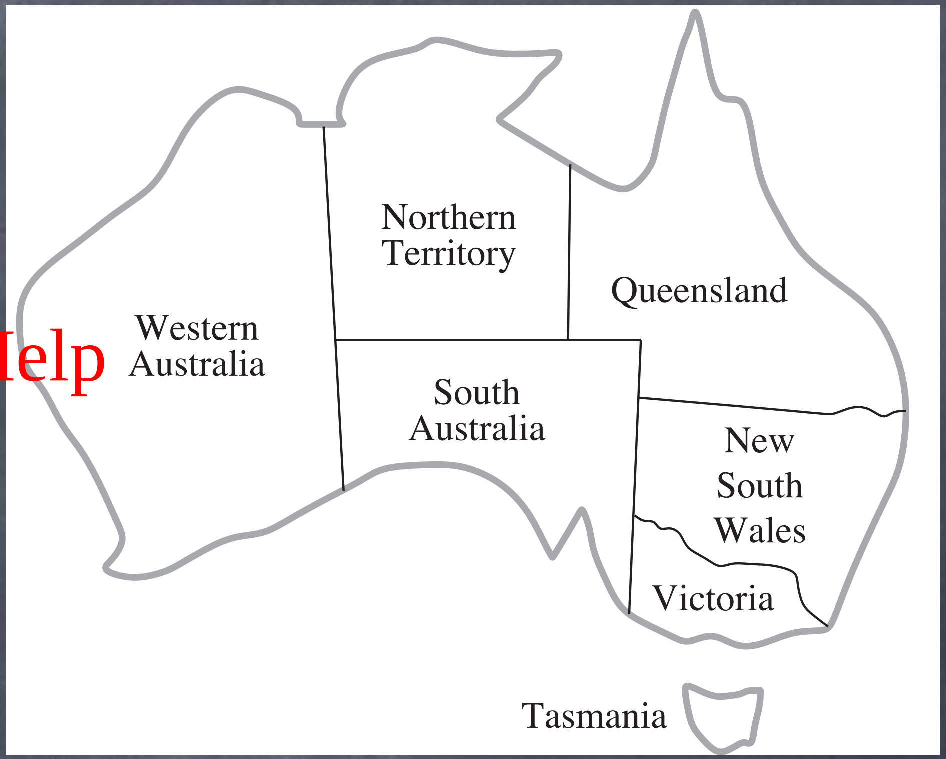
$$\{\ WA{=}red,\ NT{=}red,\ Q{=}green,\ NSW{=}blue,\ V{=}red,\ SA{=}blue,\ T{=}green\ \}$$

Assign a color to each region such that no two neighboring regions have the same color

Constraints

`Color` WA, NT, Q, NSW, V, SA, T

`enum Color` $\{red, green, blue\}$

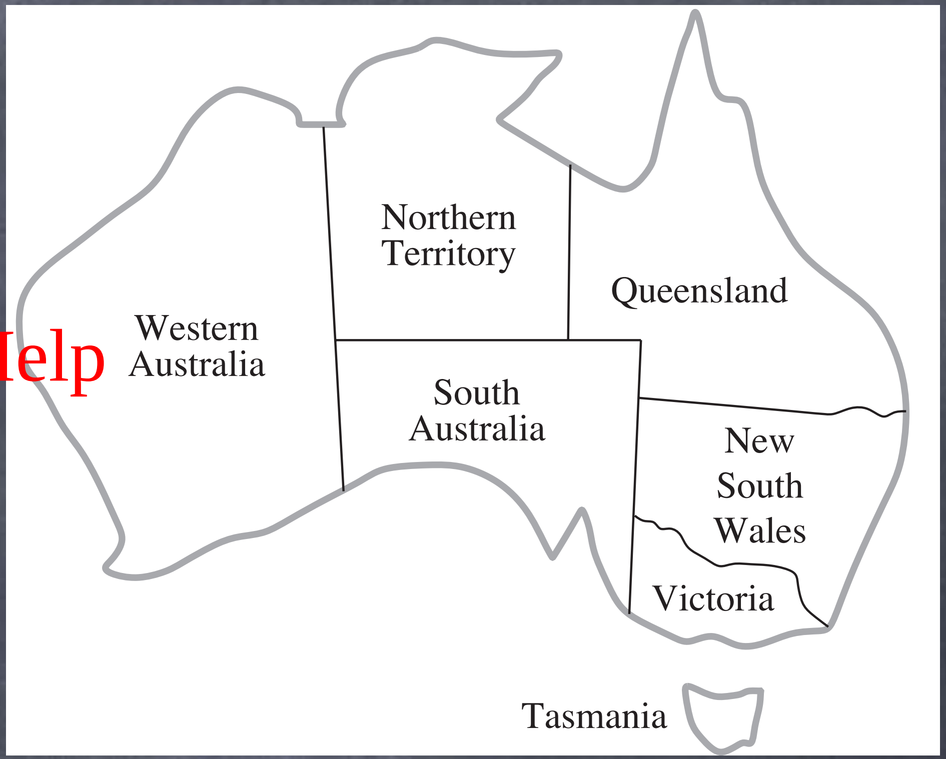


Constraints

`Color` WA, NT, Q, NSW, V, SA, T

`enum Color` $\{red, green, blue\}$

$WA \neq NT, WA \neq SA, NT \neq Q,$
$NT \neq SA, Q \neq NSW, Q \neq SA,$
$NSW \neq V, NSW \neq SA, V \neq SA$

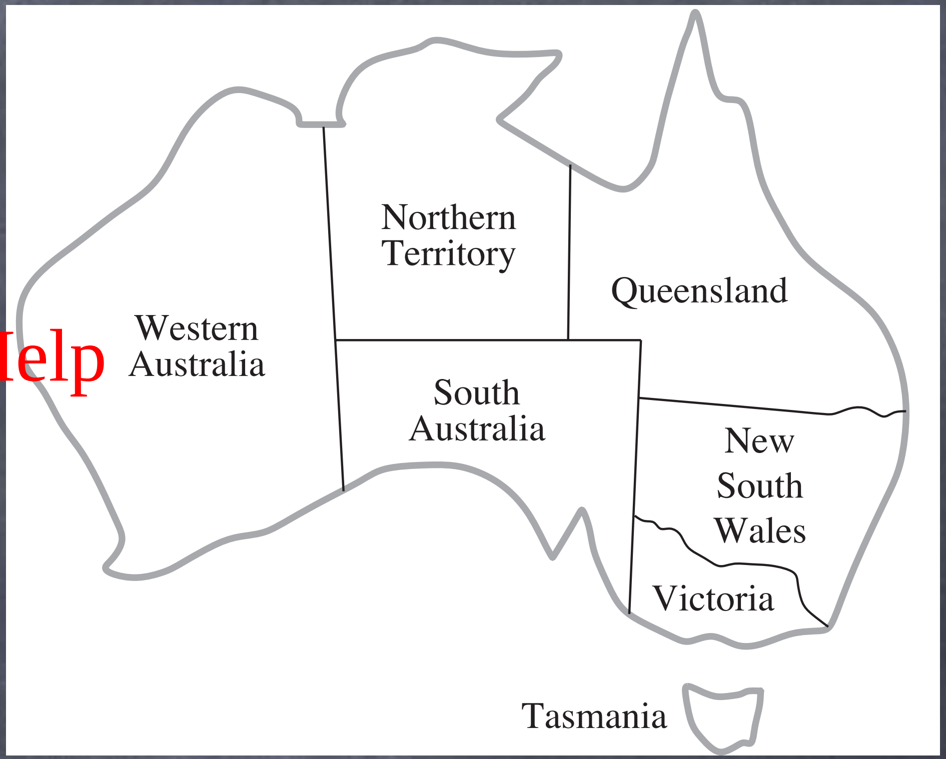


Constraints

`Color` WA, NT, Q, NSW, V, SA, T

`enum Color` $\{red, green, blue\}$

$WA \neq NT, WA \neq SA, NT \neq Q,$
$NT \neq SA, Q \neq NSW, Q \neq SA,$
$NSW \neq V, NSW \neq SA, V \neq SA$



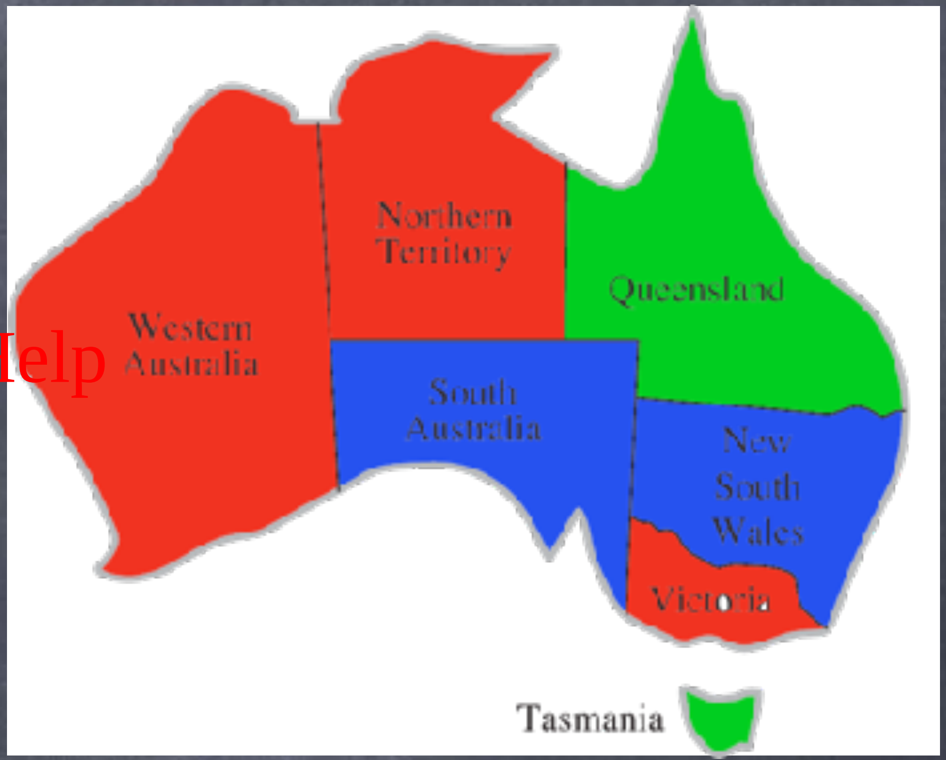
Rule out impossible assignments

Inconsistency

`Color` WA, NT, Q, NSW, V, SA, T

`enum Color` $\{red, green, blue\}$

$$WA \neq NT, WA \neq SA, NT \neq Q,$$
$$NT \neq SA, Q \neq NSW, Q \neq SA,$$
$$NSW \neq V, NSW \neq SA, V \neq SA$$



$$\{\ WA{=}red,\ NT{=}red,\ Q{=}green,\ NSW{=}blue,\ V{=}red,\ SA{=}blue,\ T{=}green\ \}$$

✘

Consistency (Satisfaction)

`Color` WA, NT, Q, NSW, V, SA, T

`enum Color` $\{red, green, blue\}$

$$WA \neq NT, WA \neq SA, NT \neq Q,$$

$$NT \neq SA, Q \neq NSW, Q \neq SA,$$

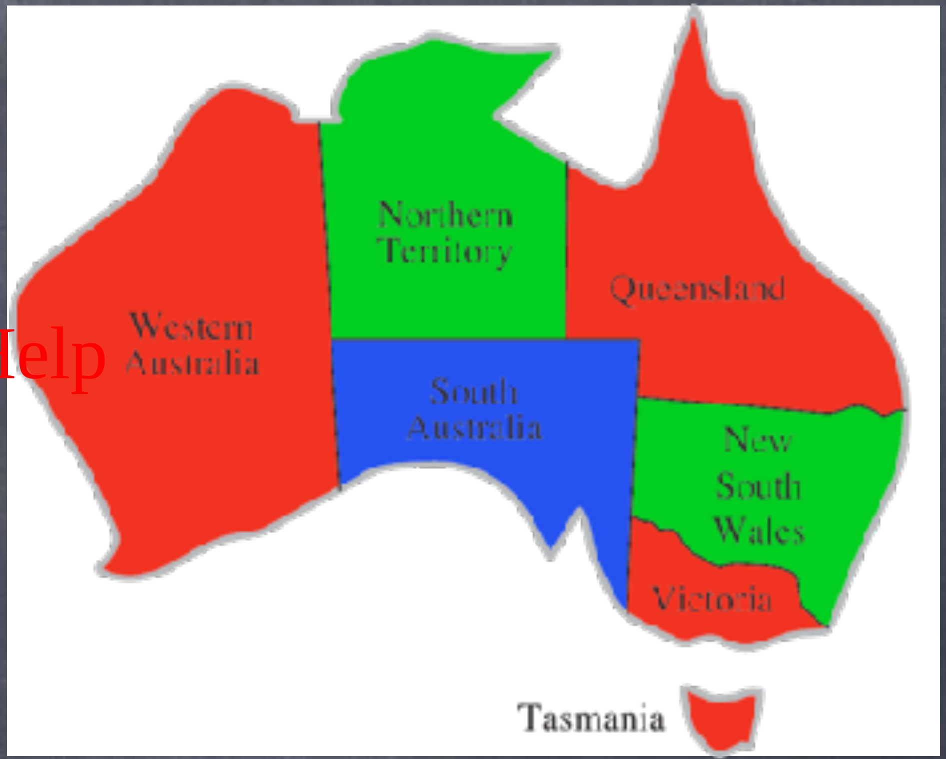
$$NSW \neq V, NSW \neq SA, V \neq SA$$

$$\{\ WA{=}red,\ NT{=}green,\ Q{=}red,\ NSW{=}green,\ V{=}red,\ SA{=}blue,\ T{=}red\ \}$$ ✔

Solution

`Color` $WA,\ NT,\ Q,\ NSW,\ V,\ SA,\ T$

`enum Color` $\{red,\ green,\ blue\}$



$WA \neq NT, WA \neq SA, NT \neq Q,$
$NT \neq SA, Q \neq NSW, Q \neq SA,$
$NSW \neq V, NSW \neq SA, V \neq SA$

$\{\ WA{=}red,\ NT{=}green,\ Q{=}red,\ NSW{=}green,$
$V{=}red,\ SA{=}blue,\ T{=}red\ \}$ ✔

Complete & Consistent

Constraint Satisfaction Problem (CSP)

- X: Set of variables $\{ X_1, \ldots, X_n \}$
- D: Set of domains $\{ D_1, \ldots, D_n \}$
  - Each D_i : set of values $\{ v_1, \ldots, v_{ki} \}$
- C: Set of constraints $\{ C_1, \ldots, C_m \}$
- Solution: Assign to each X_i a value from D_i such that all the C_j are satisfied

Constraint Satisfaction Problem (CSP)

- X: Set of variables $\{ X_1, \ldots, X_n \}$
- D: Set of domains $\{ D_1, \ldots, D_n \}$
  - Each D_i : set of values $\{ v_1, \ldots, v_{ki} \}$
- C: Set of constraints $\{ C_1, \ldots, C_m \}$
- Solution: Assign to each X_i a value from D_i such that all the C_j are satisfied

Constraint Satisfaction Problem (CSP)

- X: Set of variables $\{ X_1, \ldots, X_n \}$
- D: Set of domains $\{ D_1, \ldots, D_n \}$
  - Each D_i : set of values $\{ v_1, \ldots, v_{ki} \}$
- C: Set of constraints $\{ C_1, \ldots, C_m \}$
- Solution: Assign to each X_i a value from D_i such that all the C_j are satisfied

Constraint Satisfaction Problem (CSP)

- X: Set of variables $\{ X_1, ..., X_n \}$
- D: Set of domains $\{ D_1, ..., D_n \}$
  - Each D_i : set of values $\{ v_1, ..., v_{ki} \}$
- C: Set of constraints $\{ C_1, ..., C_m \}$
- Solution: Assign to each X_i a value from D_i such that all the C_j are satisfied

Constraint Satisfaction Problem (CSP)

- X: Set of variables $\{ X_1, \ldots, X_n \}$
- D: Set of domains $\{ D_1, \ldots, D_n \}$
  - Each D_i : set of values $\{ v_1, \ldots, v_{ki} \}$
- C: Set of constraints $\{ C_1, \ldots, C_m \}$
- Solution: Assign to each X_i a value from D_i such that all the C_j are satisfied

Factored Representation

- Splits a state into factors (attributes, features, variables) that can have values

Factored Representation

- Splits a state into factors (attributes, features, variables) that can have values
- Factored states can be more or less similar (unlike atomic states)

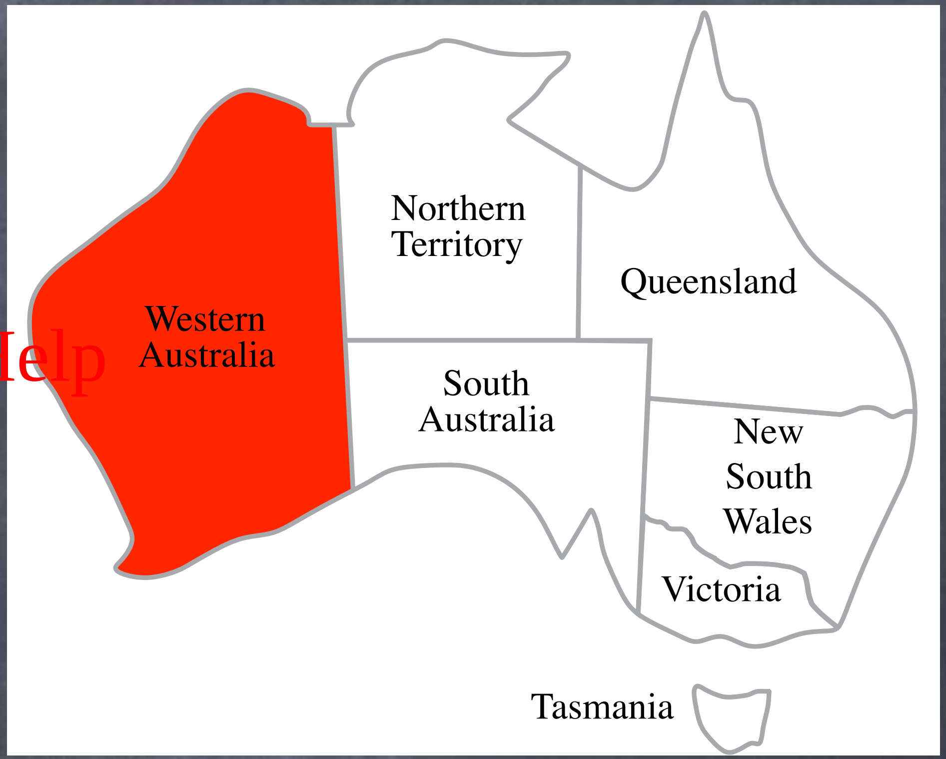
Factored Representation

- Splits a state into factors (attributes, features, variables) that can have values
- Factored states can be more or less similar (unlike atomic states)
- Can also represent uncertainty (don't know the value of some attribute)

Assignment

`Color` WA, NT, Q, NSW, V, SA, T

`enum Color` $\{red, green, blue\}$



Partial assignment: $\{\ WA{=}red\ \}$

Factored Representation

- Splits a state into factors (attributes, features, variables) that can have values
- Factored states can be more or less similar (unlike atomic states)
- Can also represent uncertainty (don't know the value of some attribute)

Constraint Satisfaction Problem (CSP)

- X: Set of variables $\{ X_1, \ldots, X_n \}$
- D: Set of domains $\{ D_1, \ldots, D_n \}$

  - Each D_i : set of values $\{ v_1, \ldots, v_{ki} \}$

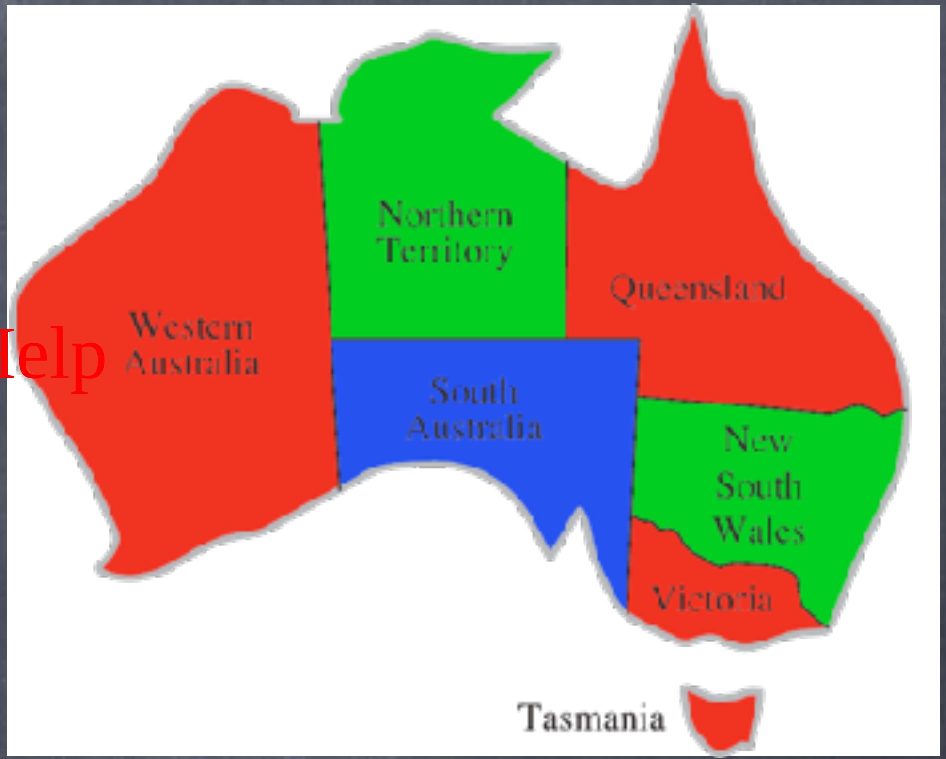
- C: Set of constraints $\{ C_1, \ldots, C_m \}$
- Solution: Assign to each X_i a value from D_i such that all the C_j are satisfied

Solution

`Color` $WA,\ NT,\ Q,\ NSW,\ V,\ SA,\ T$

`enum Color` $\{red,\ green,\ blue\}$

$WA \neq NT, WA \neq SA, NT \neq Q,$
$NT \neq SA, Q \neq NSW, Q \neq SA,$
$NSW \neq V, NSW \neq SA, V \neq SA$



$\{\ WA{=}red,\ NT{=}green,\ Q{=}red,\ NSW{=}green,$
$V{=}red,\ SA{=}blue,\ T{=}red\ \}$ ✔

Complete & Consistent

State-Space Search for CSPs

State-Space Search for CSPs

- States: assignments (possibly partial)

State-Space Search for CSPs

- States: assignments (possibly partial)
- Actions: pick an unassigned variable and assign it a value from its domain

State-Space Search for CSPs

- States: assignments (possibly partial)
- Actions: pick an unassigned variable and assign it a value from its domain
- Result: extends assignment

State-Space Search for CSPs

- States: assignments (possibly partial)
- Actions: pick an unassigned variable and assign it a value from its domain
- Result: extends assignment
- Cost: ?

State-Space Search for CSPs

- States: assignments (possibly partial)
- Actions: pick an unassigned variable and assign it a value from its domain
- Result: extends assignment
- Cost: constant

State-Space Search for CSPs

- States: assignments (possibly partial)
- Actions: pick an unassigned variable and assign it a value from its domain
- Result: extends assignment
- Cost: constant
- Initial state: empty assignment

State-Space Search for CSPs

- States: assignments (possibly partial)
- Actions: pick an unassigned variable and assign it a value from its domain
- Result: extends assignment
- Cost: constant
- Initial state: empty assignment
- Goal states: complete, consistent assignments

State-Space Search for CSPs

- Search strategy: ?

State-Space Search for CSPs

- Search strategy: ?
  - n variables: depth of solution = n

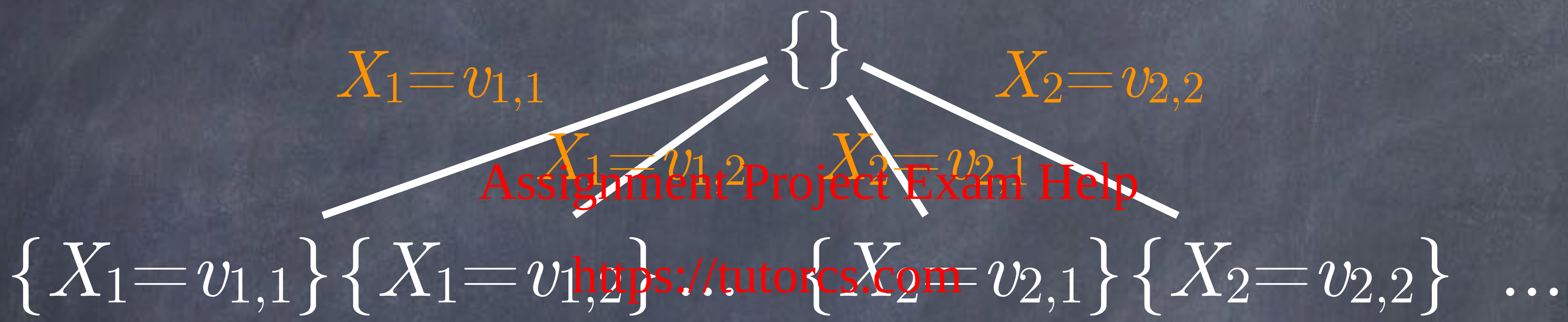
State-Space Search for CSPs

- Search strategy: ?
  - n variables: depth of solution = n
  - Depth-limited search to depth n

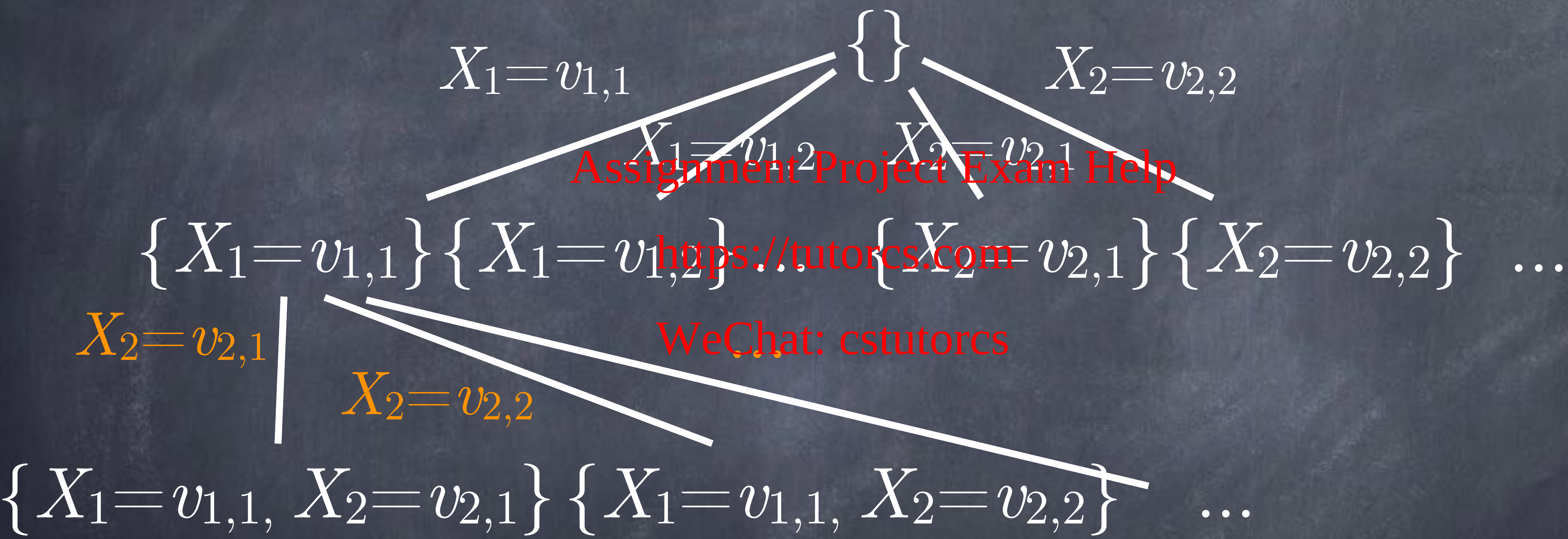
State-Space Search for CSPs

{}

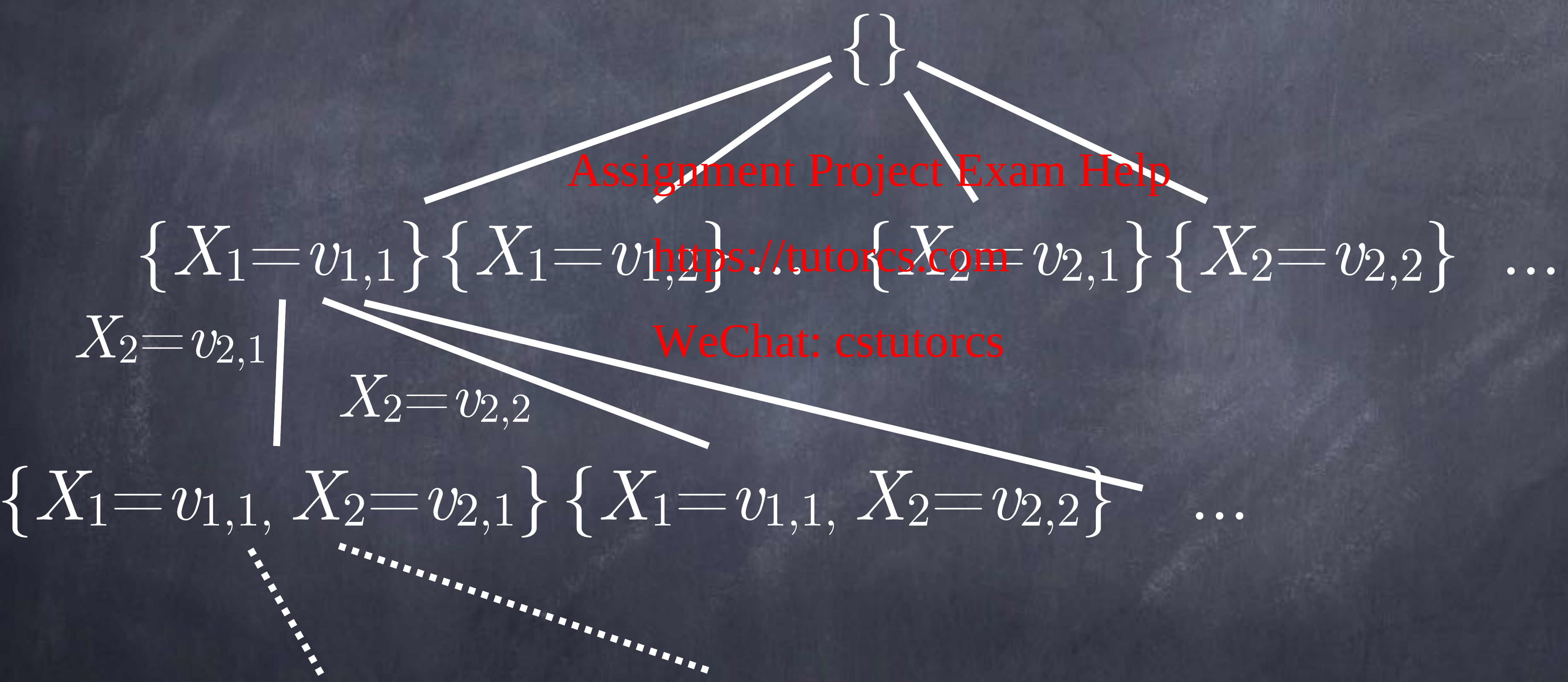
State-Space Search for CSPs



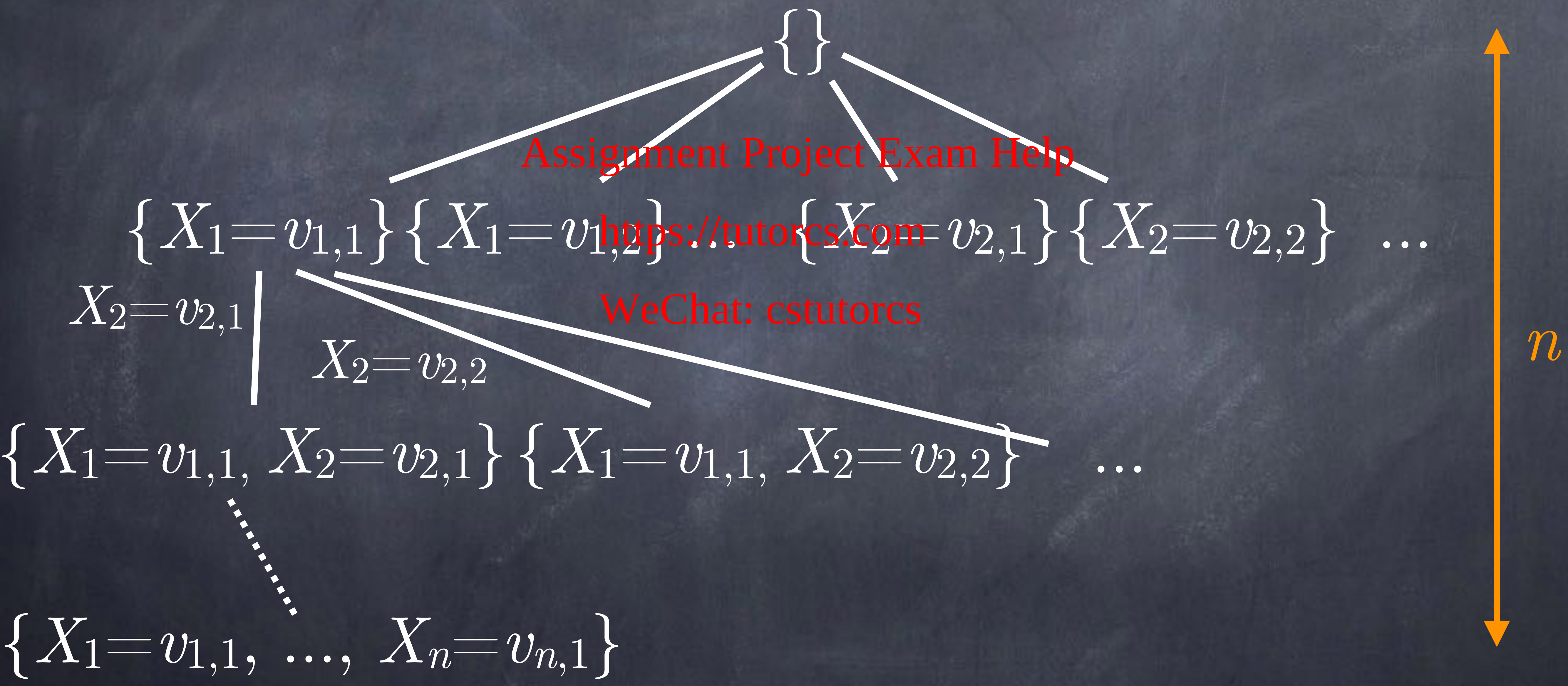
State-Space Search for CSPs



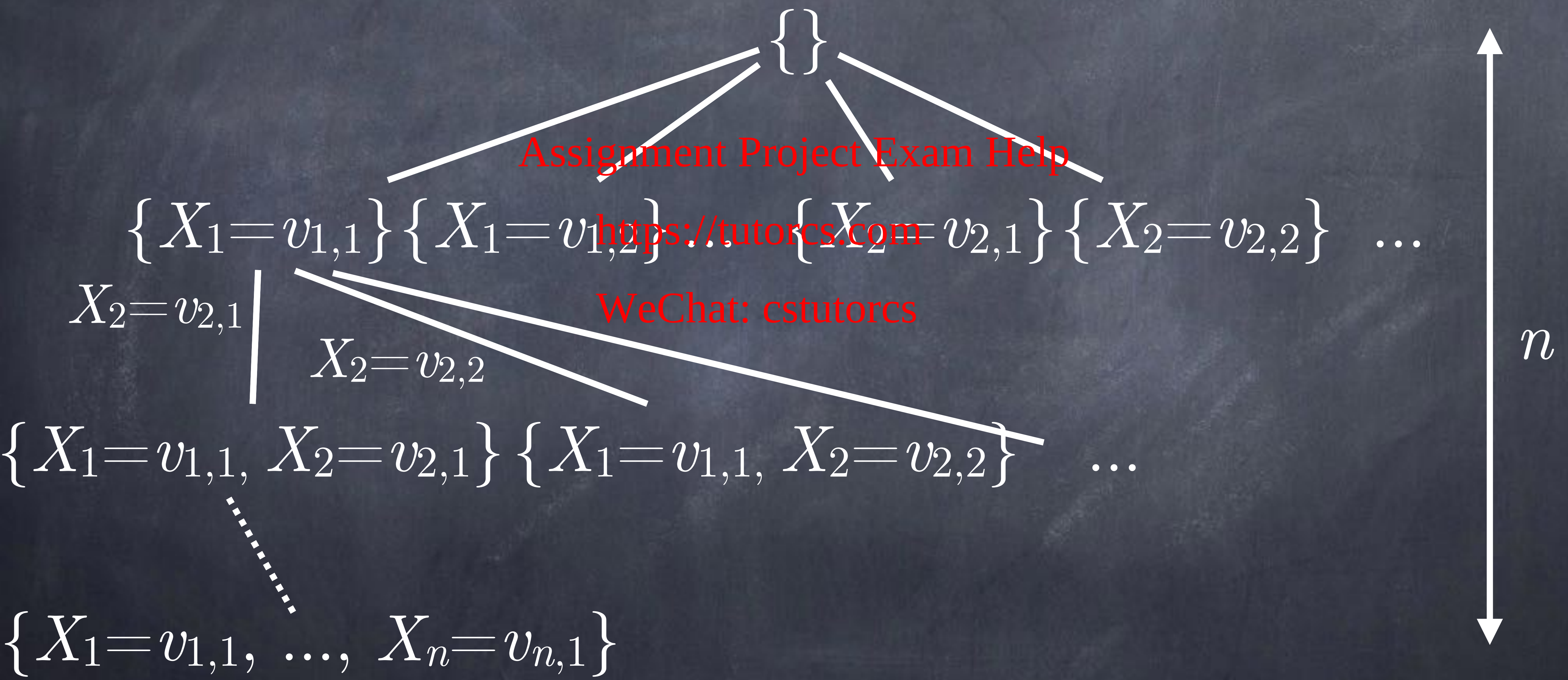
State-Space Search for CSPs



State-Space Search for CSPs

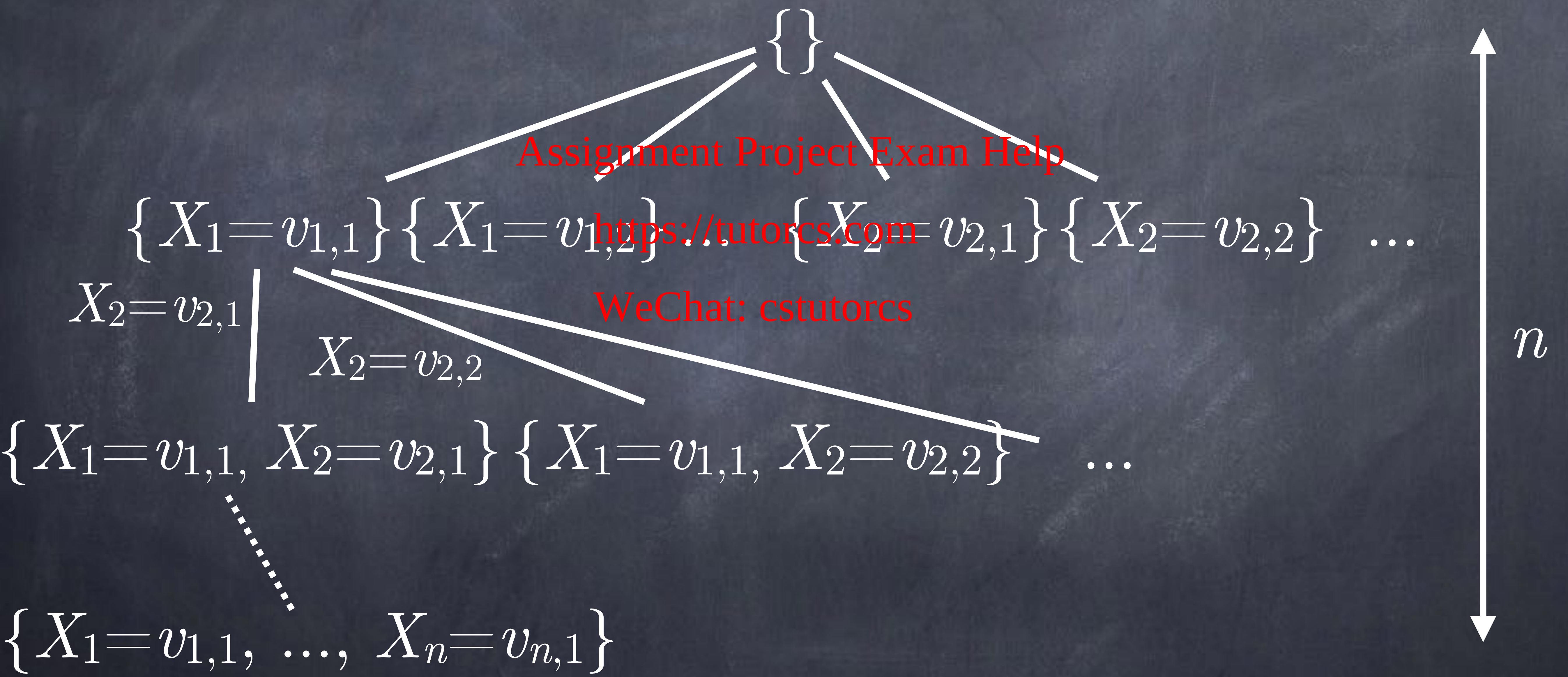


State-Space Search for CSPs



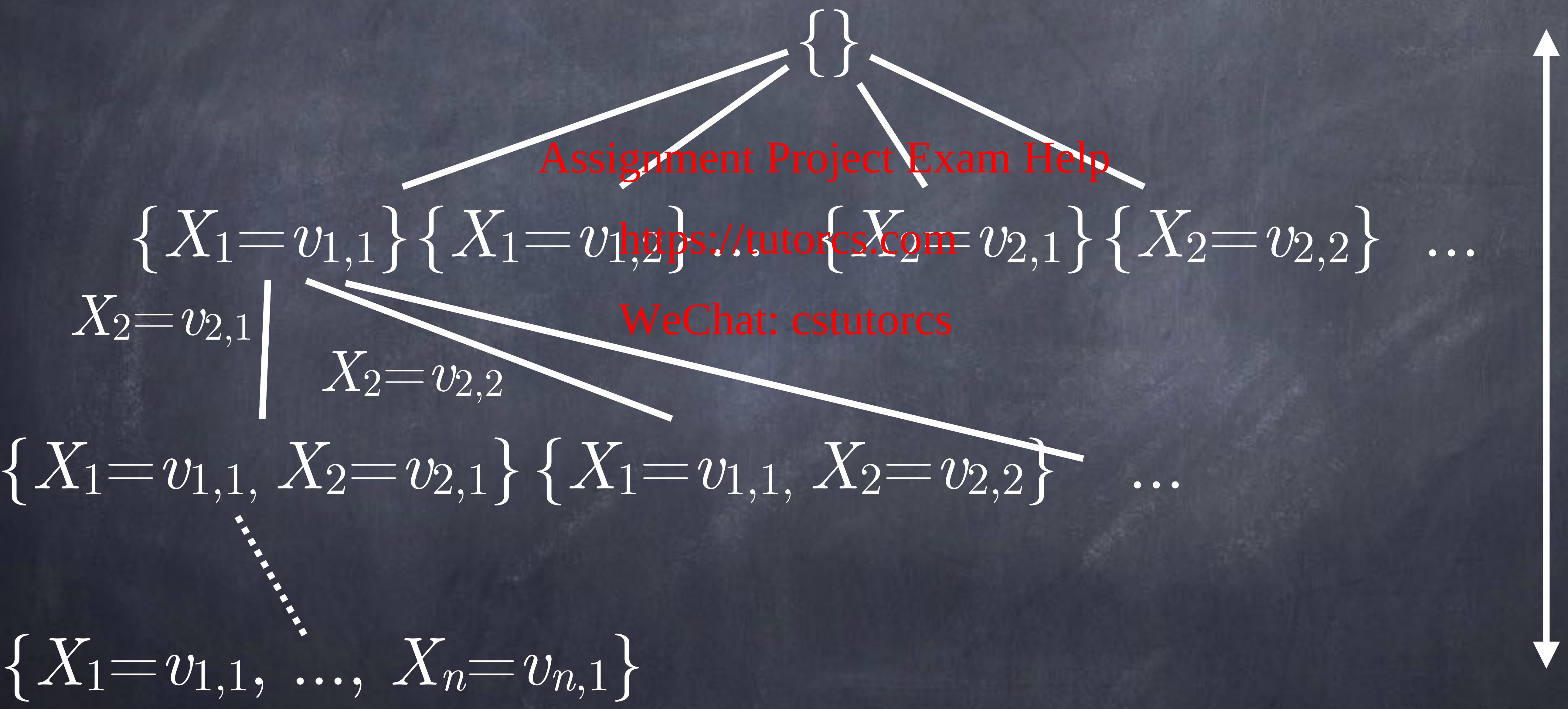
Consistent?

State-Space Search for CSPs



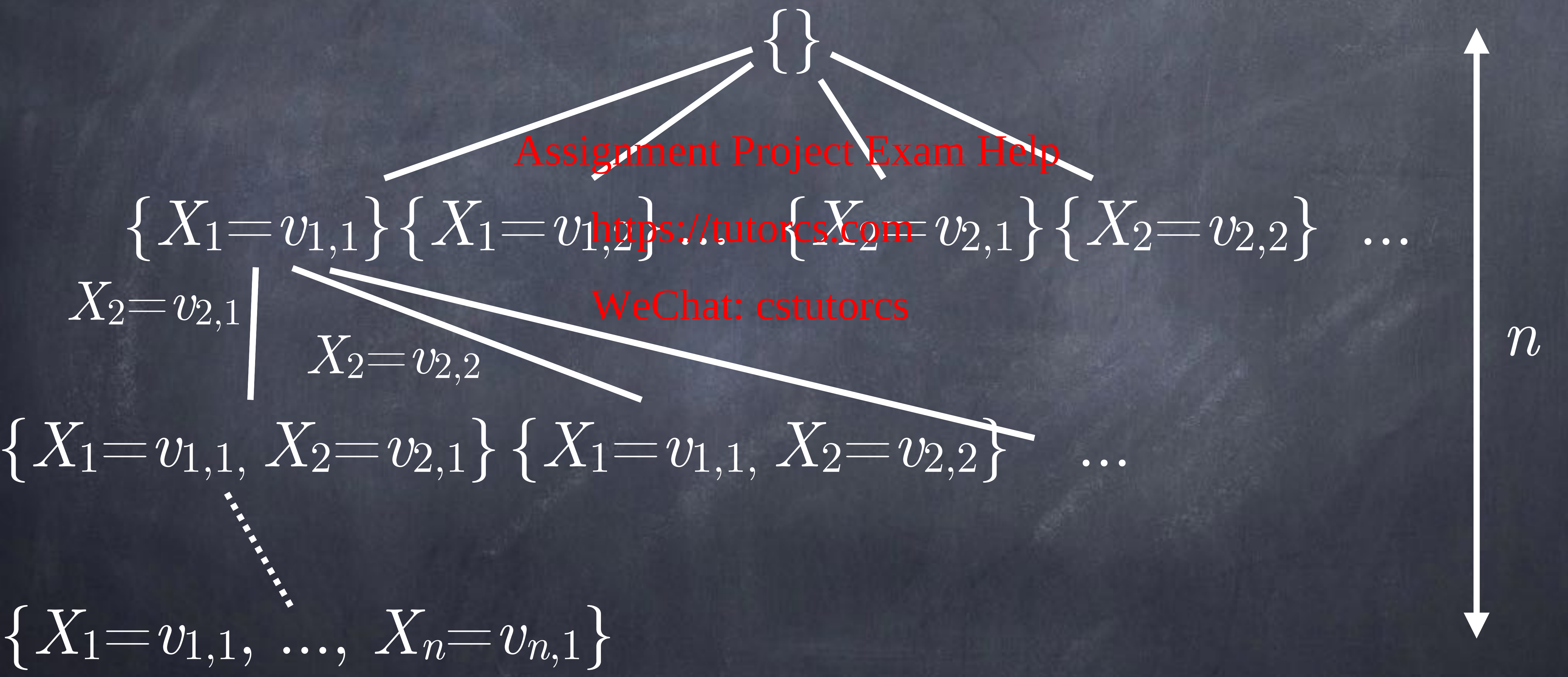
Consistent? Yes: solution!

State-Space Search for CSPs



Consistent? No: Backtrack (keep searching)

State-Space Search for CSPs



State-Space Search for CSPs

$\{\}$

$X_1 = v_{1,1}$ $X_1 = v_{1,2}$ $X_2 = v_{2,1}$ $X_2 = v_{2,2}$

$\{X_1 = v_{1,1}\} \{X_1 = v_{1,2}\} \ldots \{X_2 = v_{2,1}\} \{X_2 = v_{2,2}\} \ \ldots \ n \times d$

State-Space Search for CSPs

$\{\}$

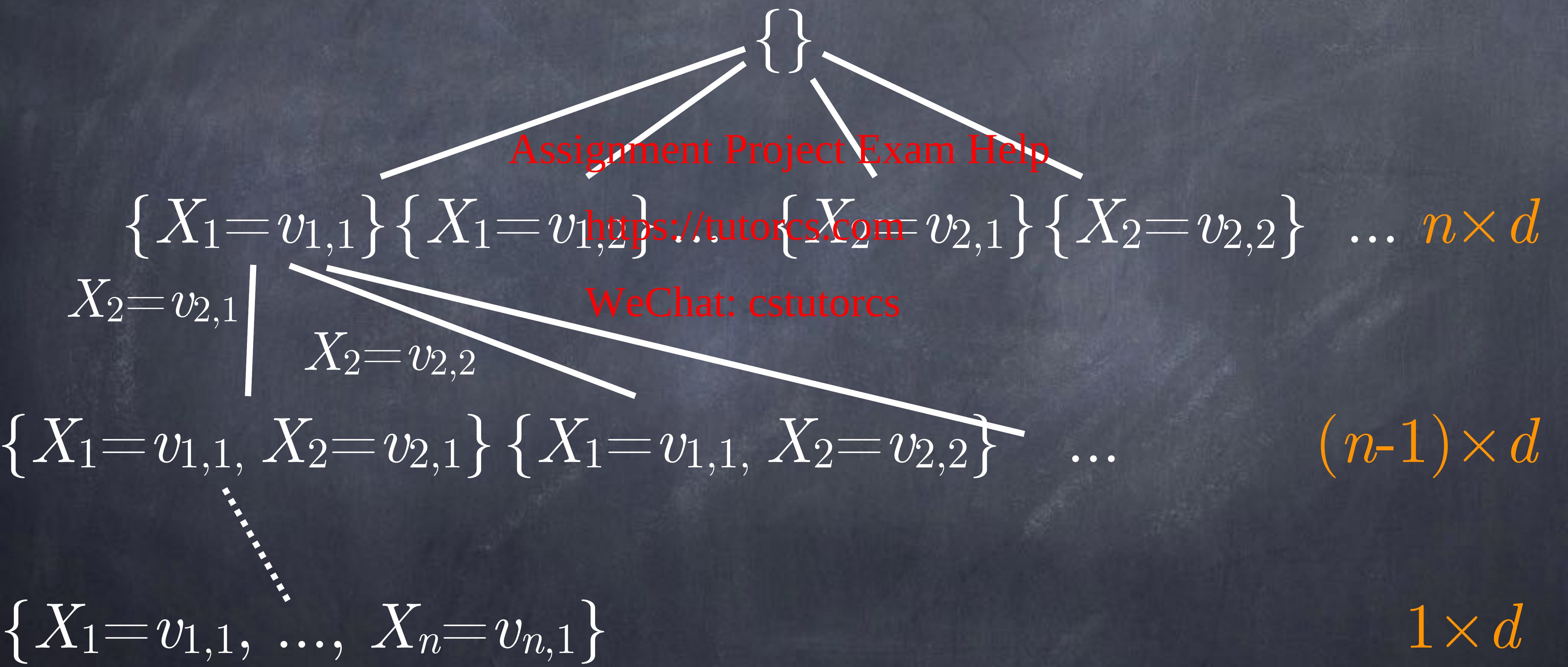
$\{X_1 = v_{1,1}\}\ \{X_1 = v_{1,2}\}\ \ldots\ \{X_2 = v_{2,1}\}\ \{X_2 = v_{2,2}\}\ \ldots\ n \times d$

$X_2 = v_{2,1}$

$X_2 = v_{2,2}$

$\{X_1 = v_{1,1},\ X_2 = v_{2,1}\}\ \{X_1 = v_{1,1},\ X_2 = v_{2,2}\}\ \ldots\ (n\text{-}1) \times d$

State-Space Search for CSPs



State-Space Search for CSPs

$$\{\}$$

$\{X_1{=}v_{1,1}\}$ $\{X_1{=}v_{1,2}\}$ ... $\{X_2{=}v_{2,1}\}$ $\{X_2{=}v_{2,2}\}$ ... $n \times d$

$X_2{=}v_{2,1}$

$X_2{=}v_{2,2}$

$\{X_1{=}v_{1,1},\ X_2{=}v_{2,1}\}$ $\{X_1{=}v_{1,1},\ X_2{=}v_{2,2}\}$ ... $(n\text{-}1) \times d$

$\{X_1{=}v_{1,1},\ \ldots,\ X_n{=}v_{n,1}\}$ $1 \times d$

$$(n \times d) \times ((n\text{-}1) \times d) \times ((n\text{-}2) \times d) \times \ldots \times (2 \times d) \times (1 \times d)$$

State-Space Search for CSPs

$\{\}$

$\{X_1 = v_{1,1}\}$ $\{X_1 = v_{1,2}\}$ … $\{X_2 = v_{2,1}\}$ $\{X_2 = v_{2,2}\}$ … $n \times d$

$X_2 = v_{2,1}$

$X_2 = v_{2,2}$

n

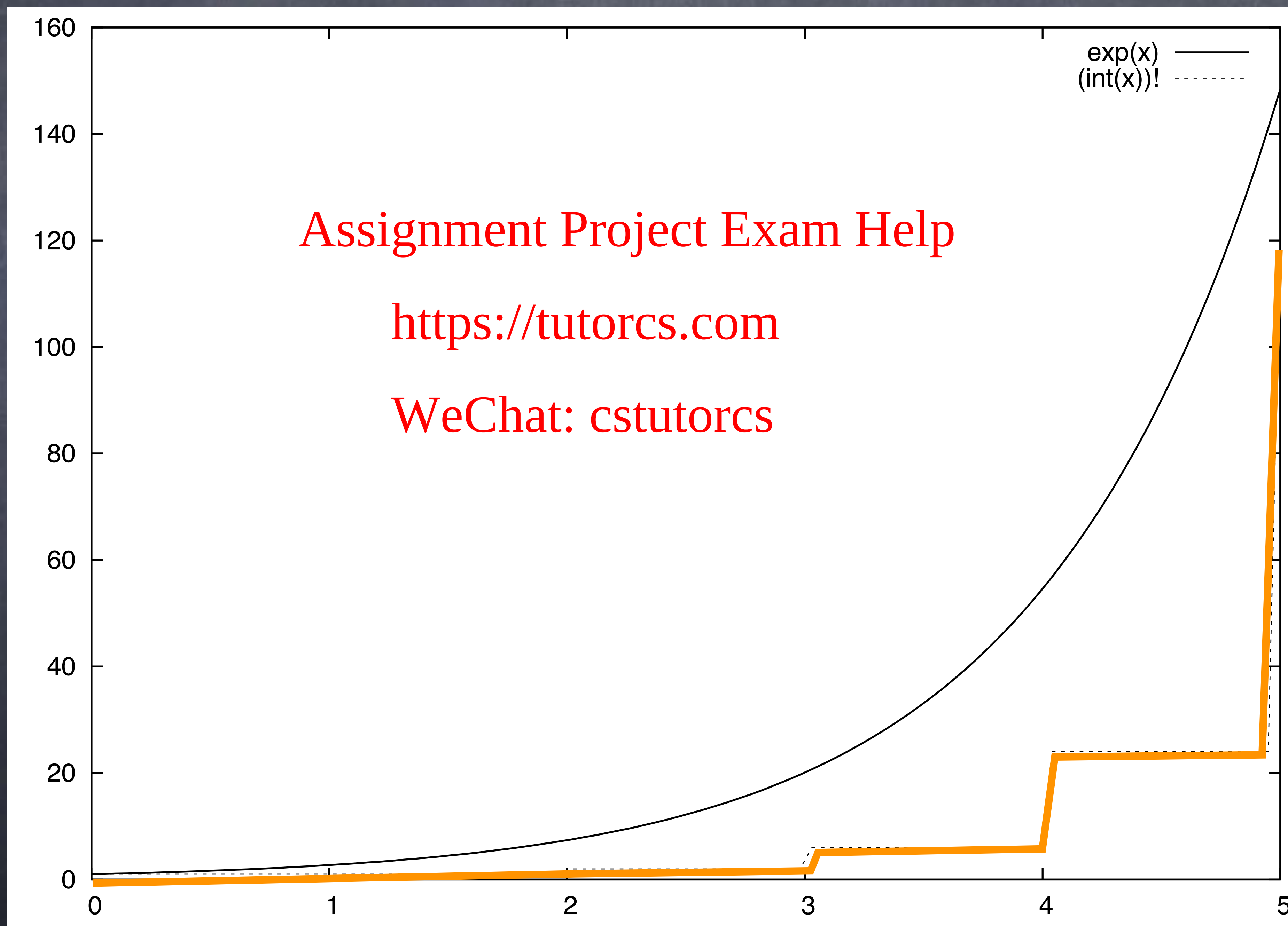
$\{X_1 = v_{1,1},\ X_2 = v_{2,1}\}$ $\{X_1 = v_{1,1},\ X_2 = v_{2,2}\}$ … $(n\text{-}1) \times d$

$\{X_1 = v_{1,1},\ \ldots,\ X_n = v_{n,1}\}$ $1 \times d$

$O(n!\,d^n)$

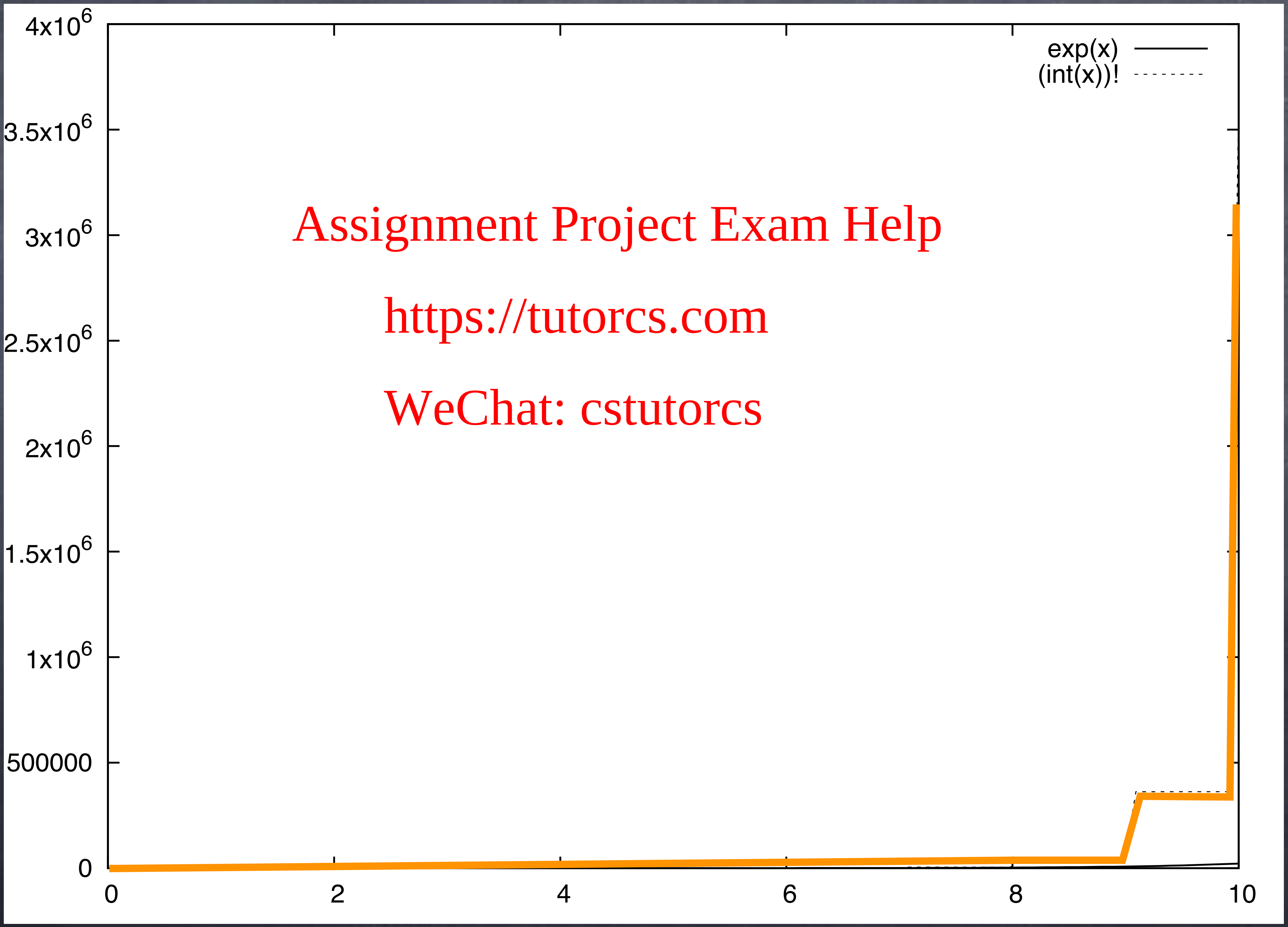
$$e^5 = 148$$

$$5! = 120$$



$$e^{10} \approx 22{,}026$$

$$10! = 3{,}628{,}800$$



State-Space Search for CSPs

$\{\}$

$\{X_1 = v_{1,1}\}$ $\{X_1 = v_{1,2}\}$ … $\{X_2 = v_{2,1}\}$ $\{X_2 = v_{2,2}\}$ … $n \times d$

$X_2 = v_{2,1}$

$X_2 = v_{2,2}$

$\{X_1 = v_{1,1},\ X_2 = v_{2,1}\}$ $\{X_1 = v_{1,1},\ X_2 = v_{2,2}\}$ … $(n\text{-}1) \times d$

$\{X_1 = v_{1,1},\ \ldots,\ X_n = v_{n,1}\}$ $1 \times d$

$O(n!\,d^n)$

State-Space Search for CSPs

$\{\}$

$\{X_1{=}v_{1,1}\}\ \{X_1{=}v_{1,2}\}\ \ldots\ \{X_2{=}v_{2,1}\}\ \{X_2{=}v_{2,2}\}\ \ldots\ n \times d$

$X_2{=}v_{2,1}$

$X_2{=}v_{2,2}$

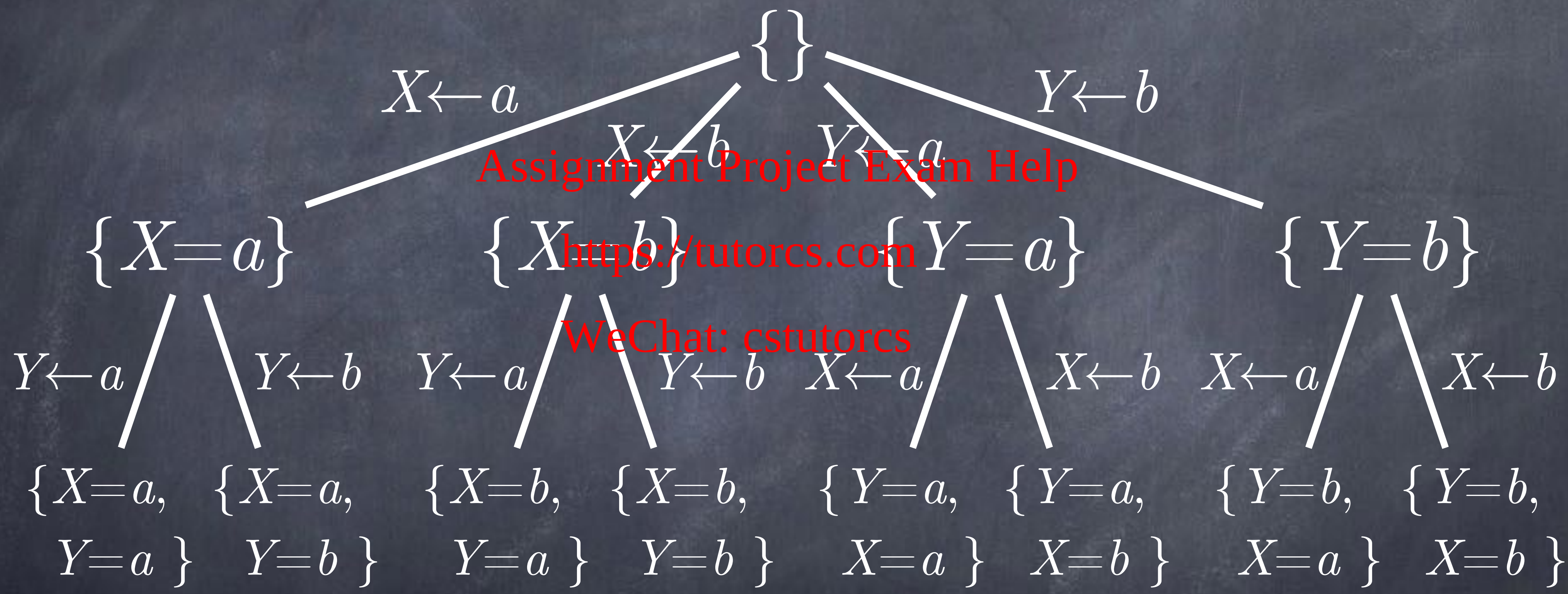
$\{X_1{=}v_{1,1},\ X_2{=}v_{2,1}\}\ \{X_1{=}v_{1,1},\ X_2{=}v_{2,2}\}\ \ldots\ (n\text{-}1) \times d$

$\{X_1{=}v_{1,1},\ \ldots,\ X_n{=}v_{n,1}\}\ \ 1 \times d$

$O(n!\,d^n) \gg O(d^n)$

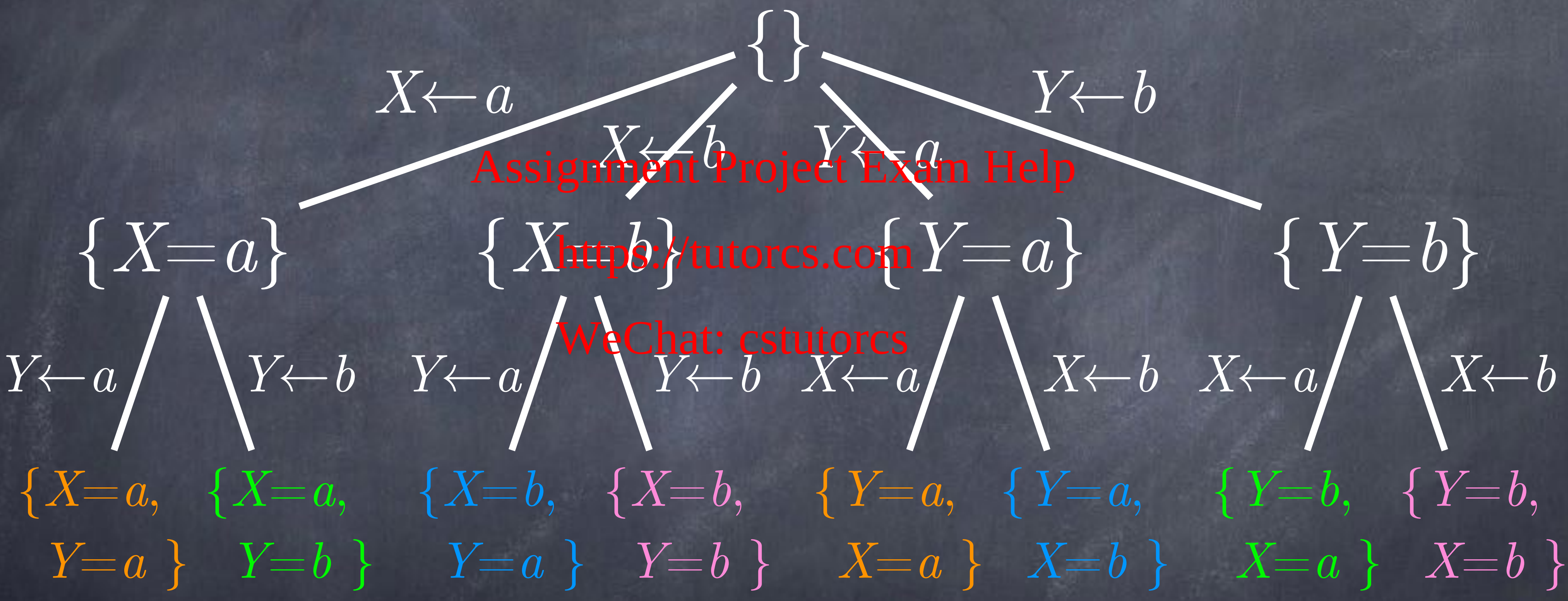
Variables: X, Y

Domains: $\{\ a,\ b\ \}$



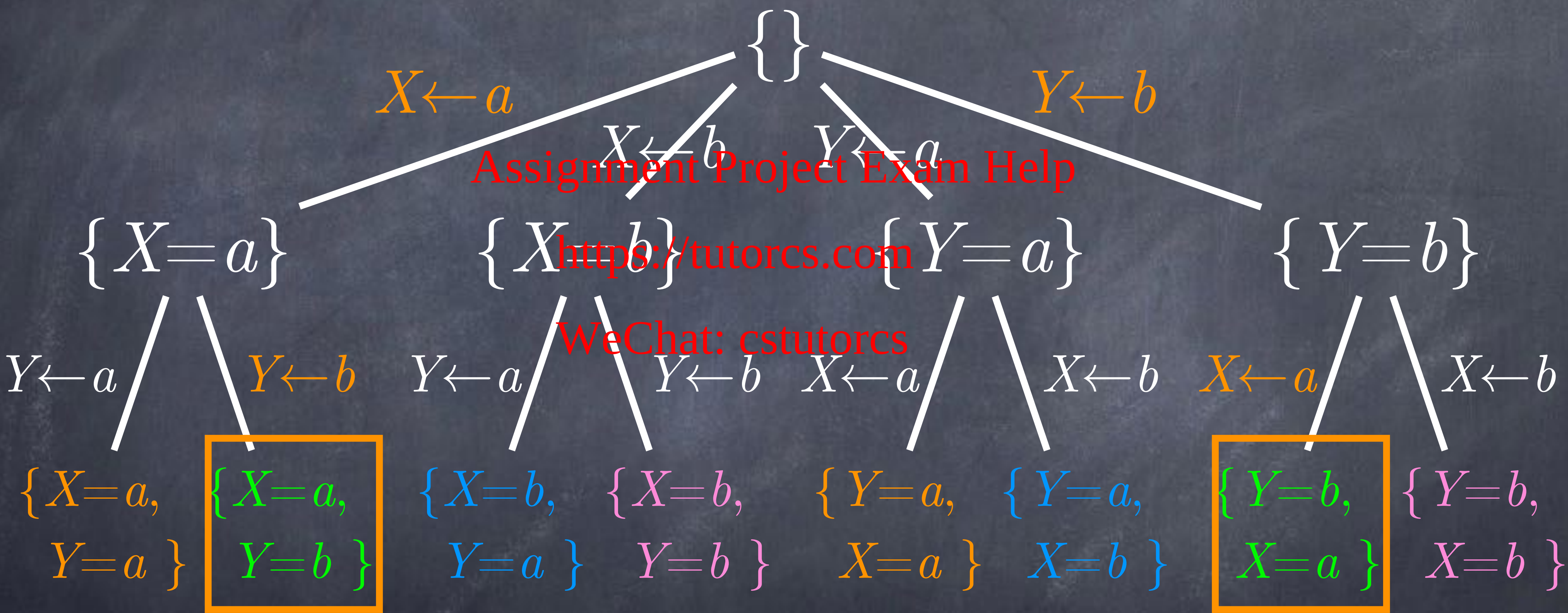
Variables: $X,\ Y$

Domains: $\{\ a,\ b\ \}$



Variables: X, Y

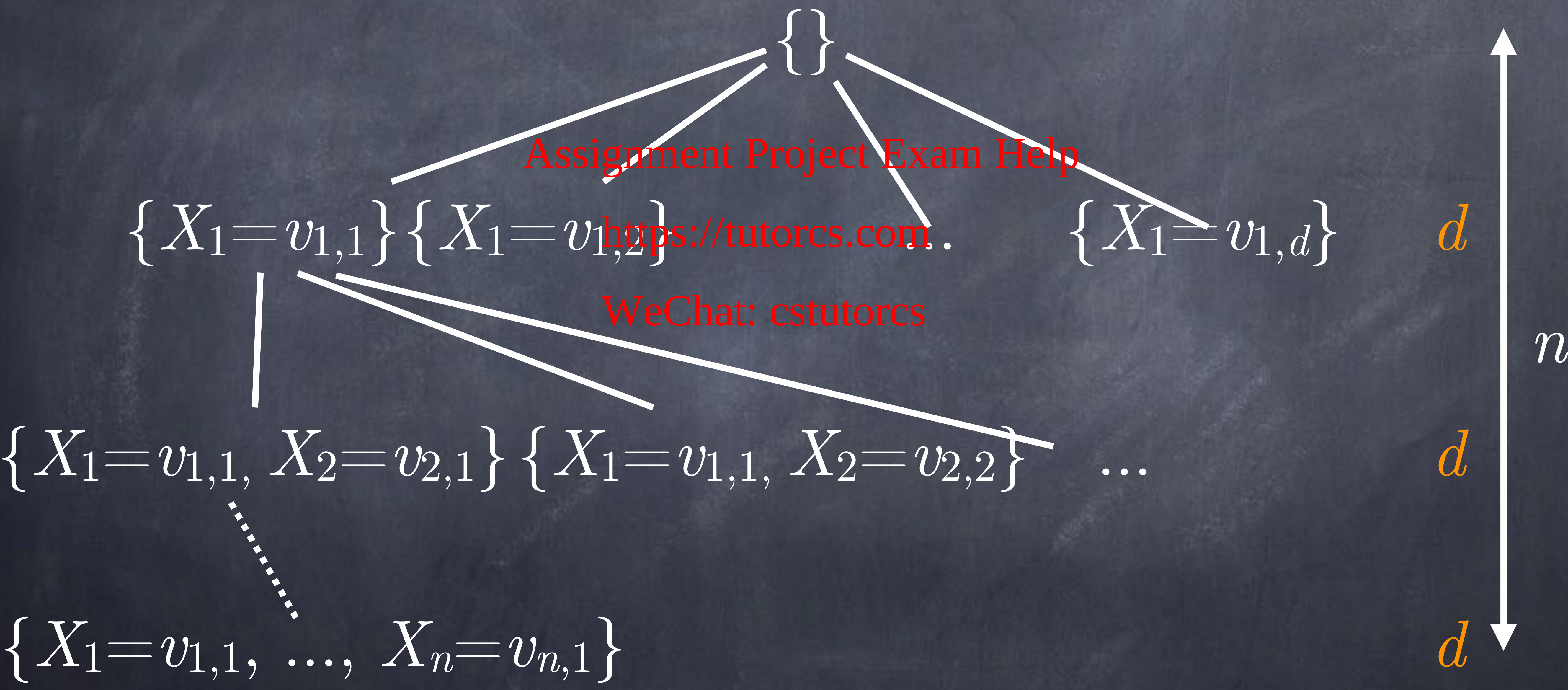
Domains: $\{ a, b \}$



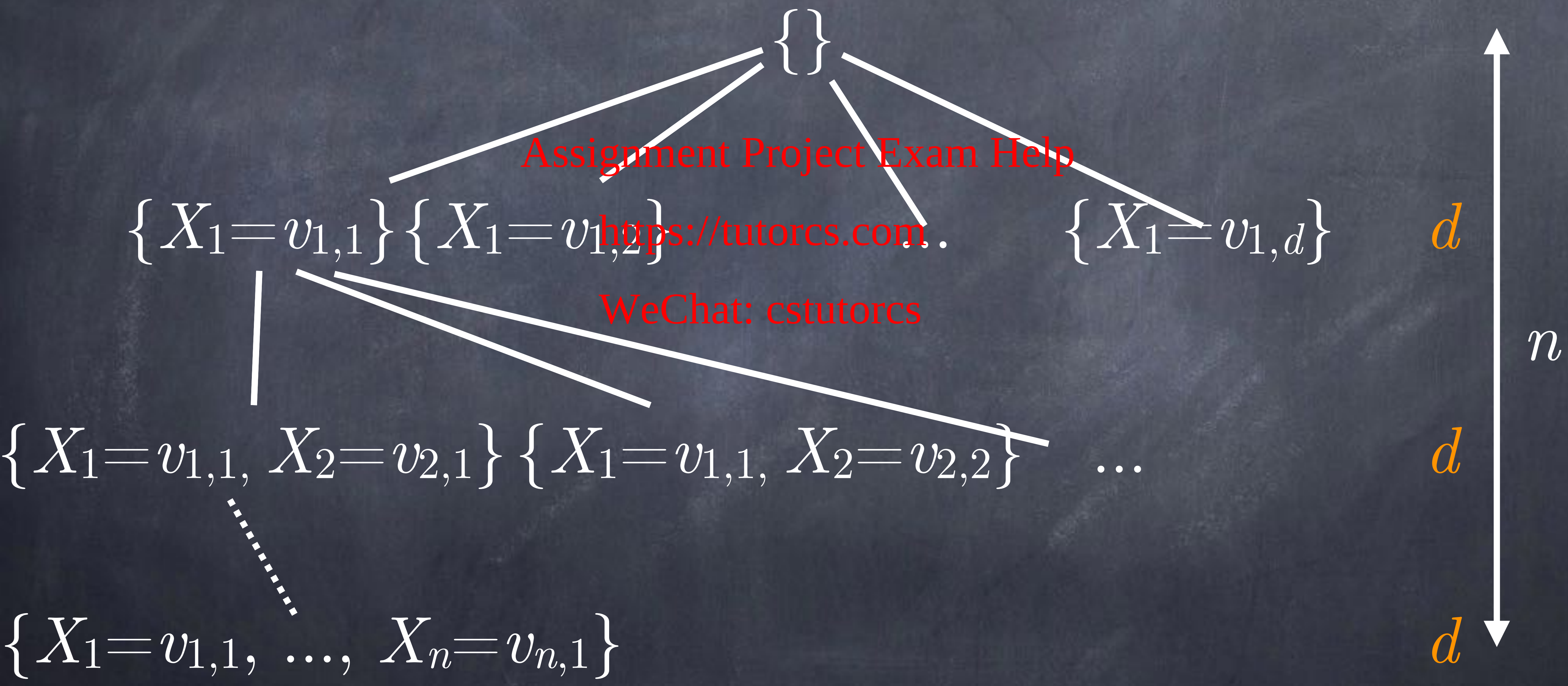
CSPs are Commutative

- CSPs are commutative because we reach the same partial assignment regardless of order
- Need only consider assignment to a single variable at each node in the search tree

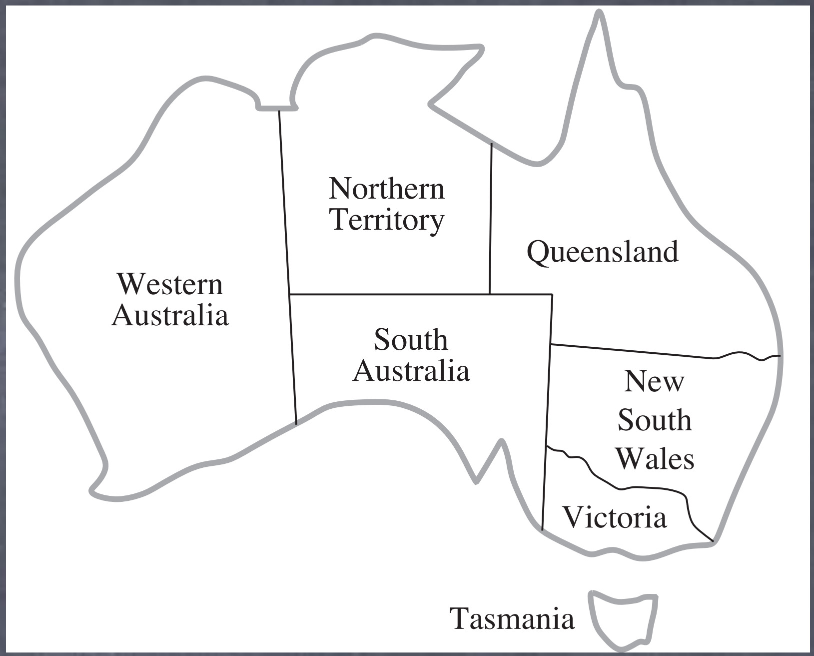
State-Space Search for CSPs



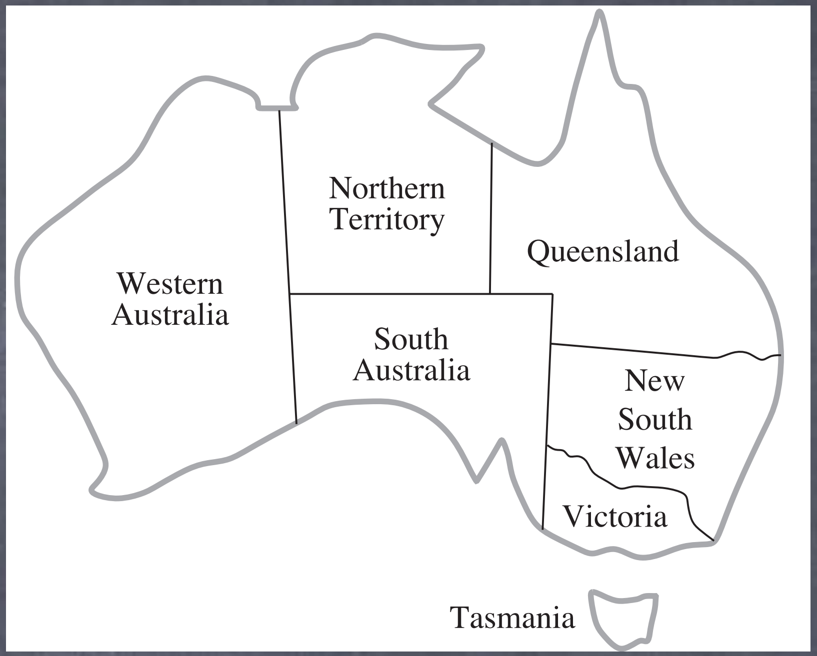
State-Space Search for CSPs

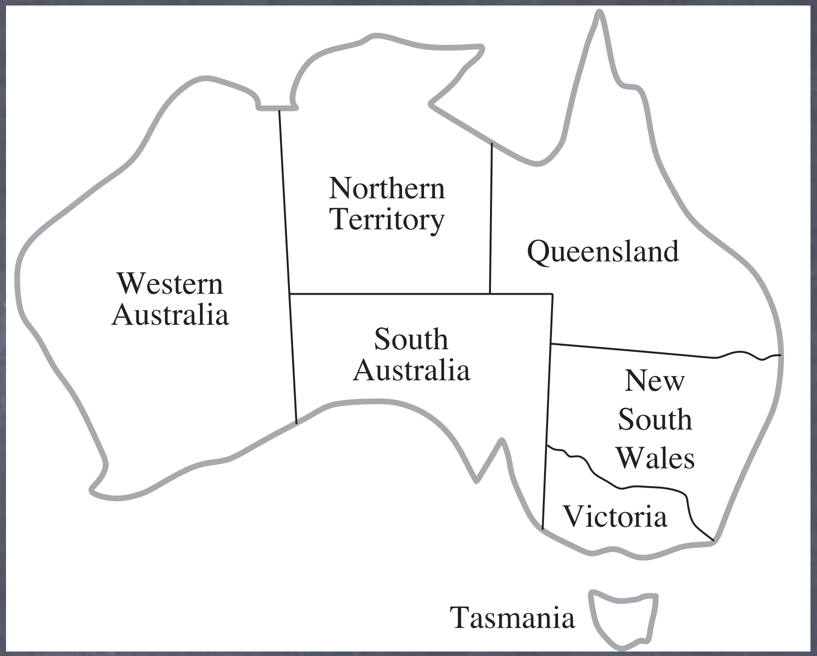


$O(d^n)$


Northern Territory
Queensland
Western Australia
South Australia
New South Wales
Victoria
Tasmania

$\varnothing$

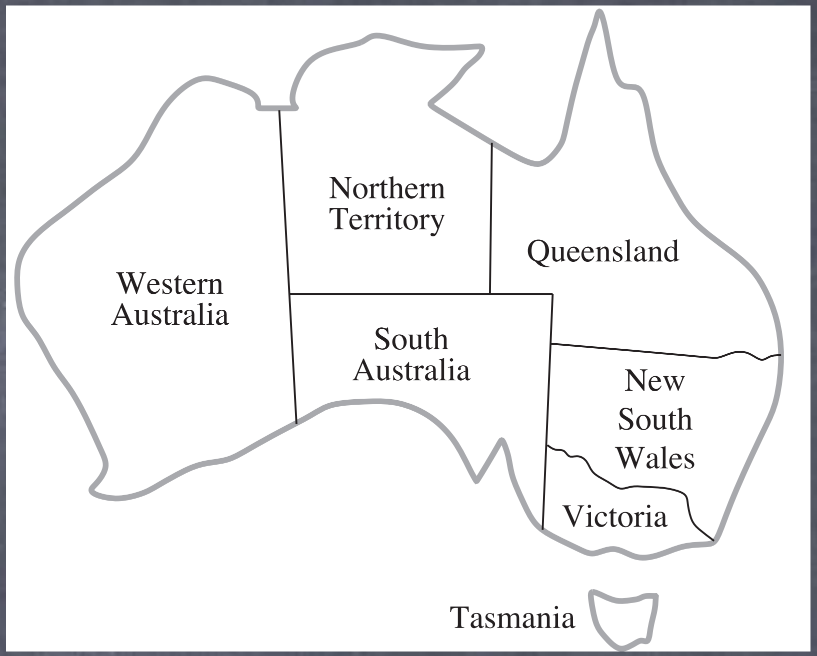

Northern Territory
Queensland
Western Australia
South Australia
New South Wales
Victoria
Tasmania



$\emptyset$

$WA = red$ | $WA = green$ | $WA = blue$

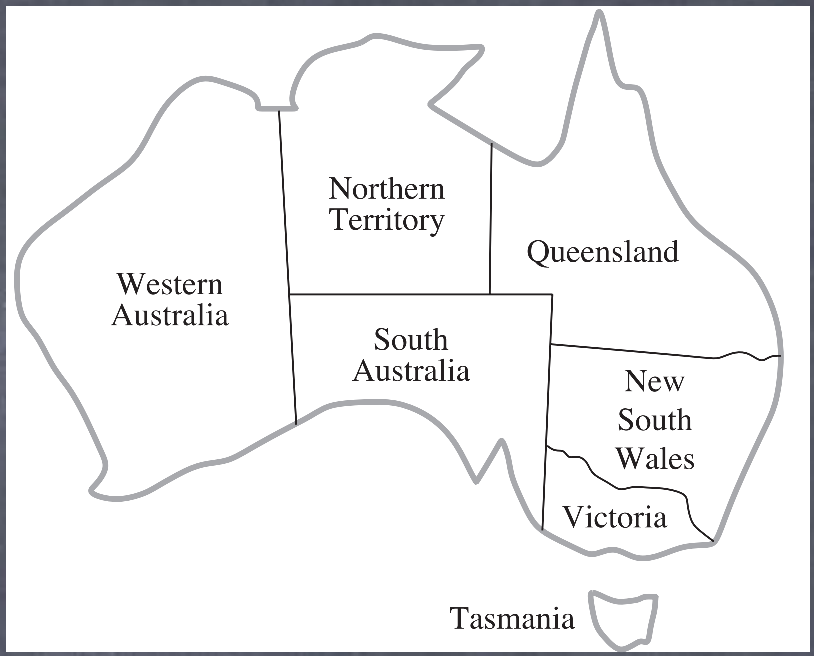
WA



$\varnothing$

WA

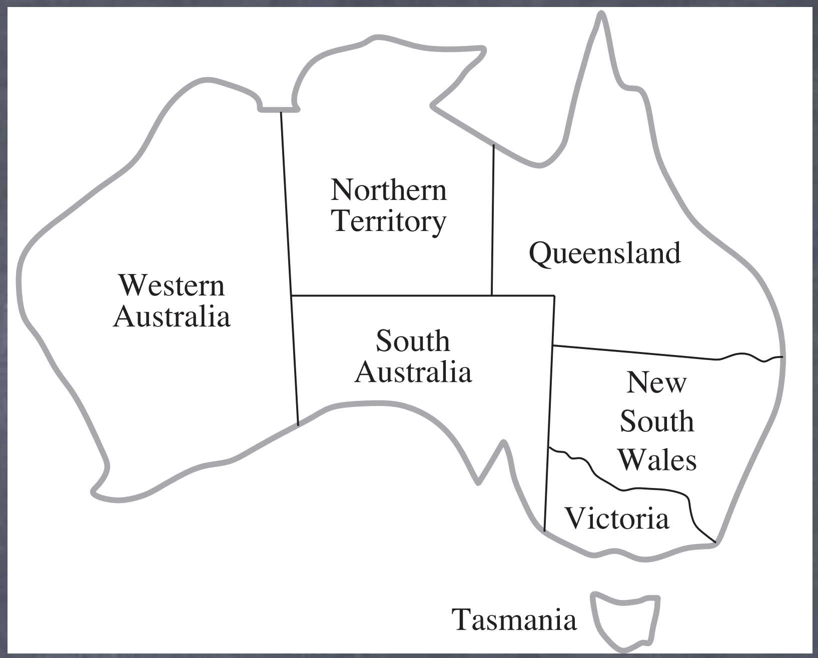
$WA = red$ | $WA = green$ | $WA = blue$



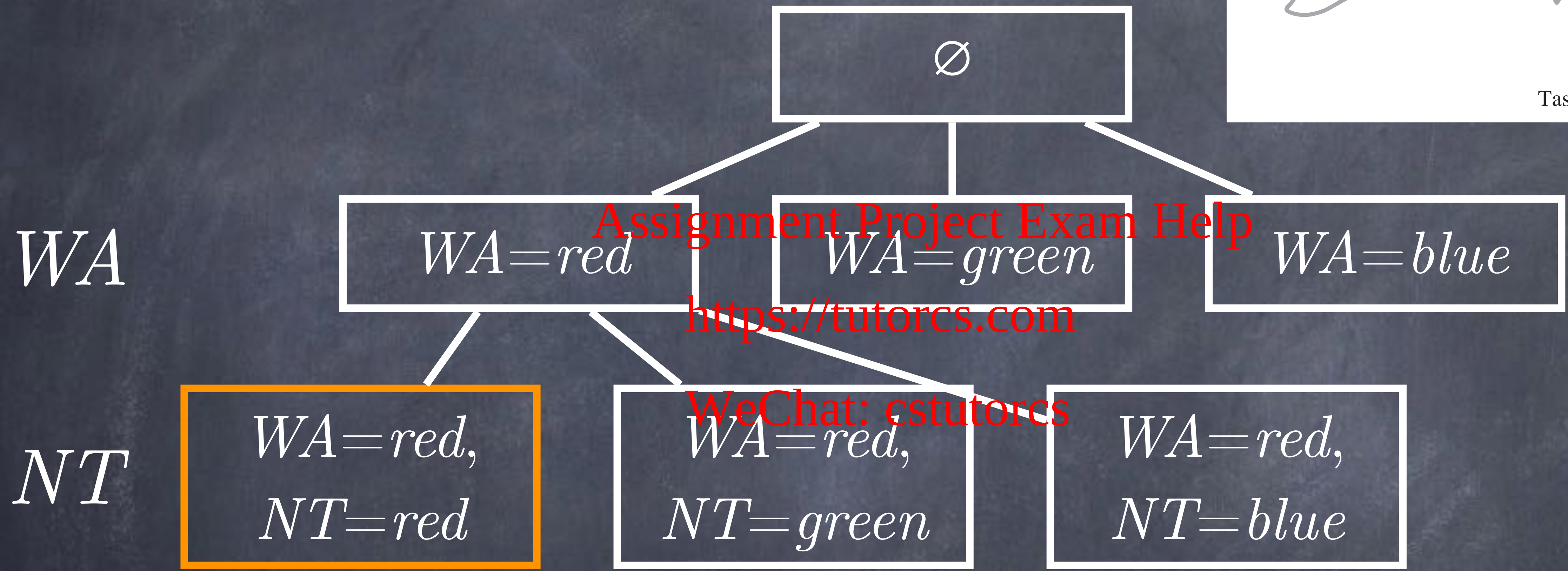
$\emptyset$

WA | $WA = red$ | $WA = green$ | $WA = blue$

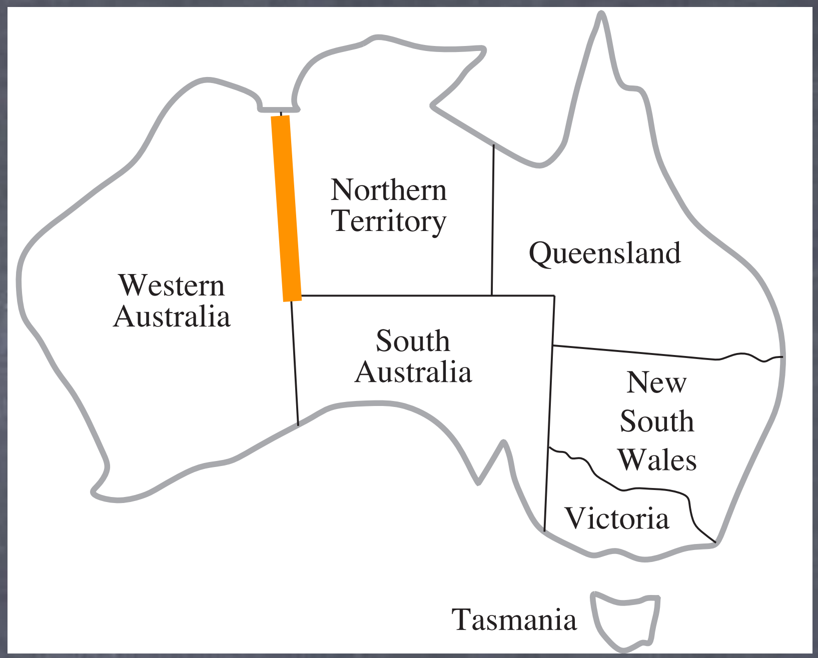
NT | $WA = red, NT = red$ | $WA = red, NT = green$ | $WA = red, NT = blue$

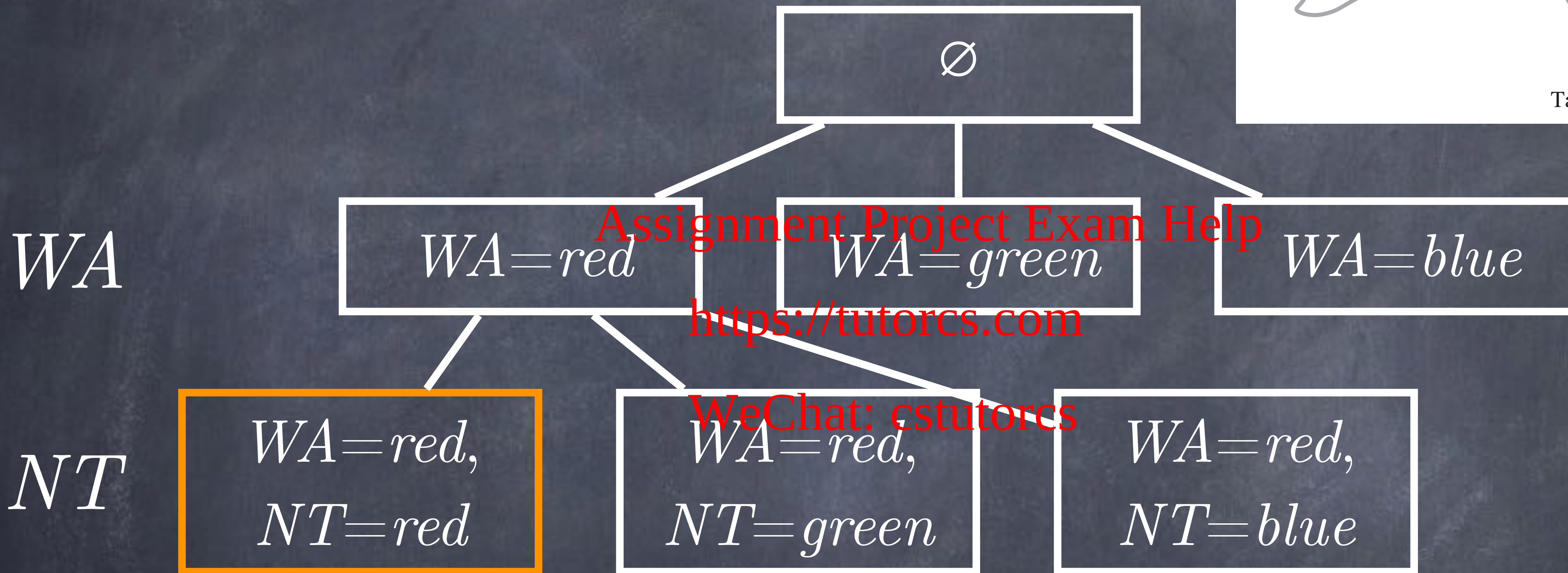


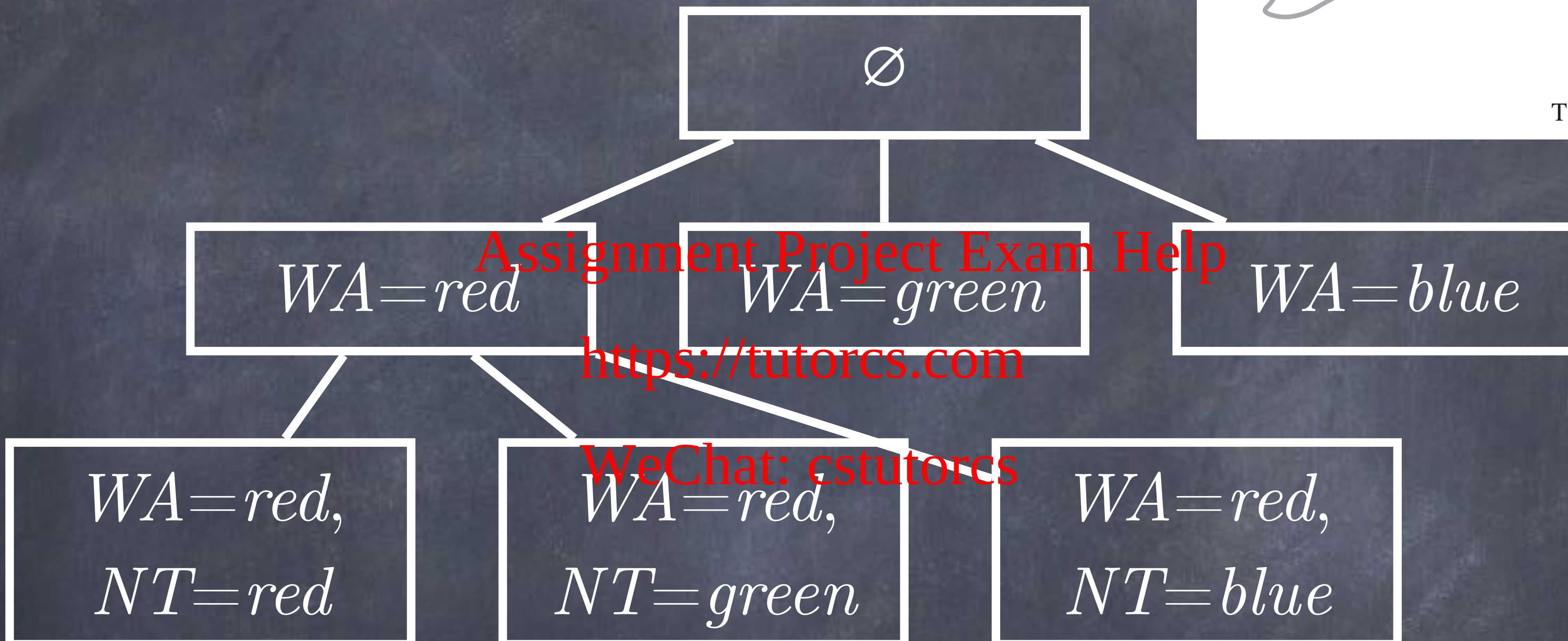
Western Australia
Northern Territory
Queensland
South Australia
New South Wales
Victoria
Tasmania



$\varnothing$
WA
$WA = red$
$WA = green$
$WA = blue$
NT
$WA = red, NT = red$
$WA = red, NT = green$
$WA = red, NT = blue$


Northern Territory
Western Australia
Queensland
South Australia
New South Wales
Victoria
Tasmania

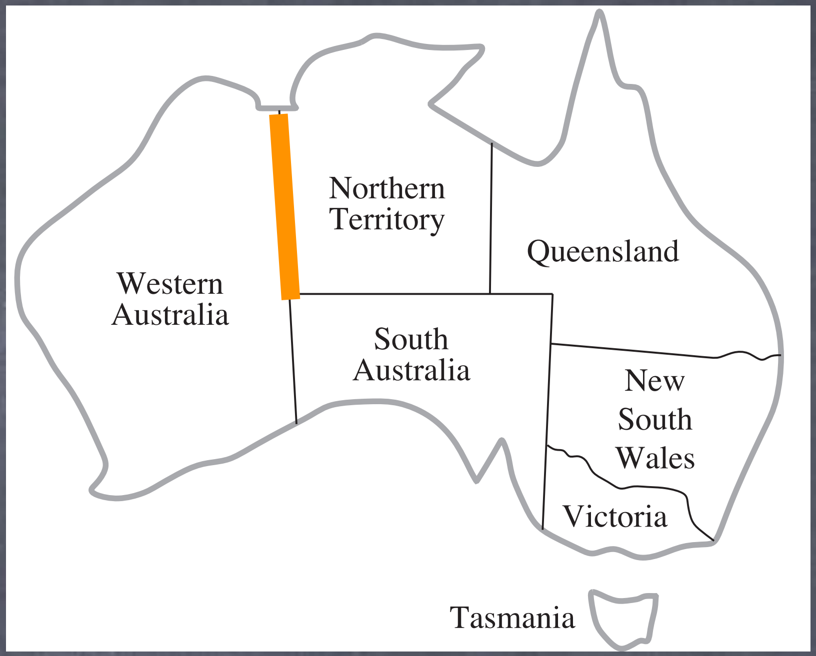

$\varnothing$
WA
$WA = red$
$WA = green$
$WA = blue$
NT
$WA = red$, $NT = red$
$WA = red$, $NT = green$
$WA = red$, $NT = blue$



WA

NT

$WA \neq NT$



$\varnothing$

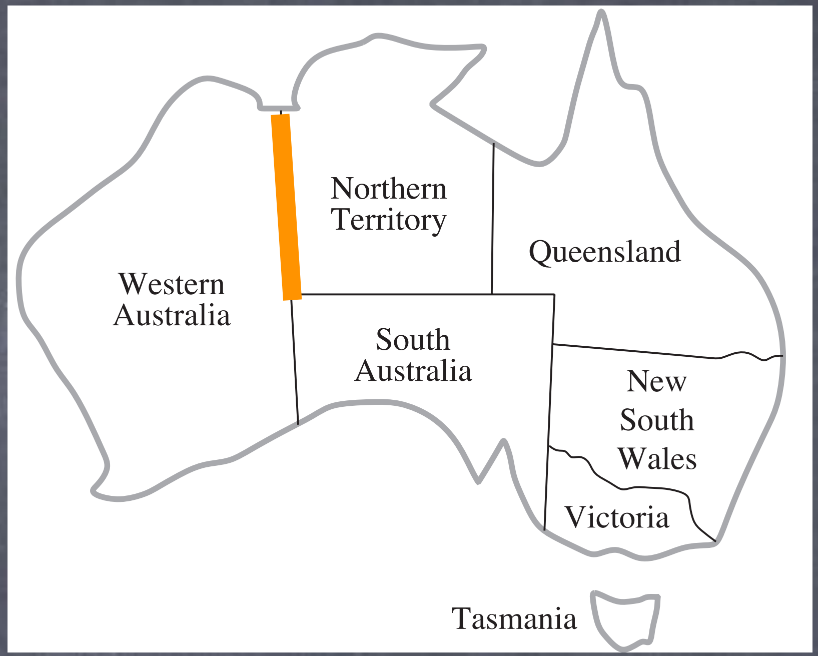
WA

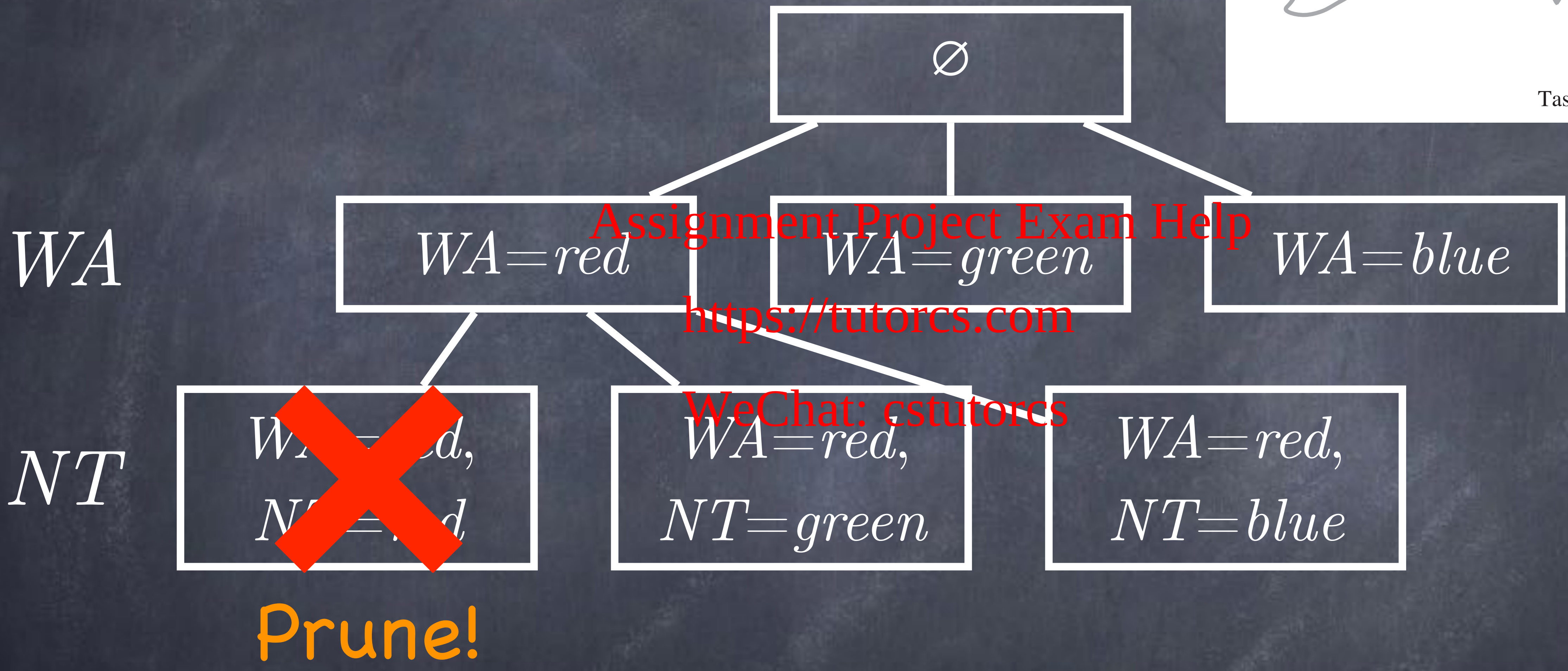
$WA=red$ | $WA=green$ | $WA=blue$

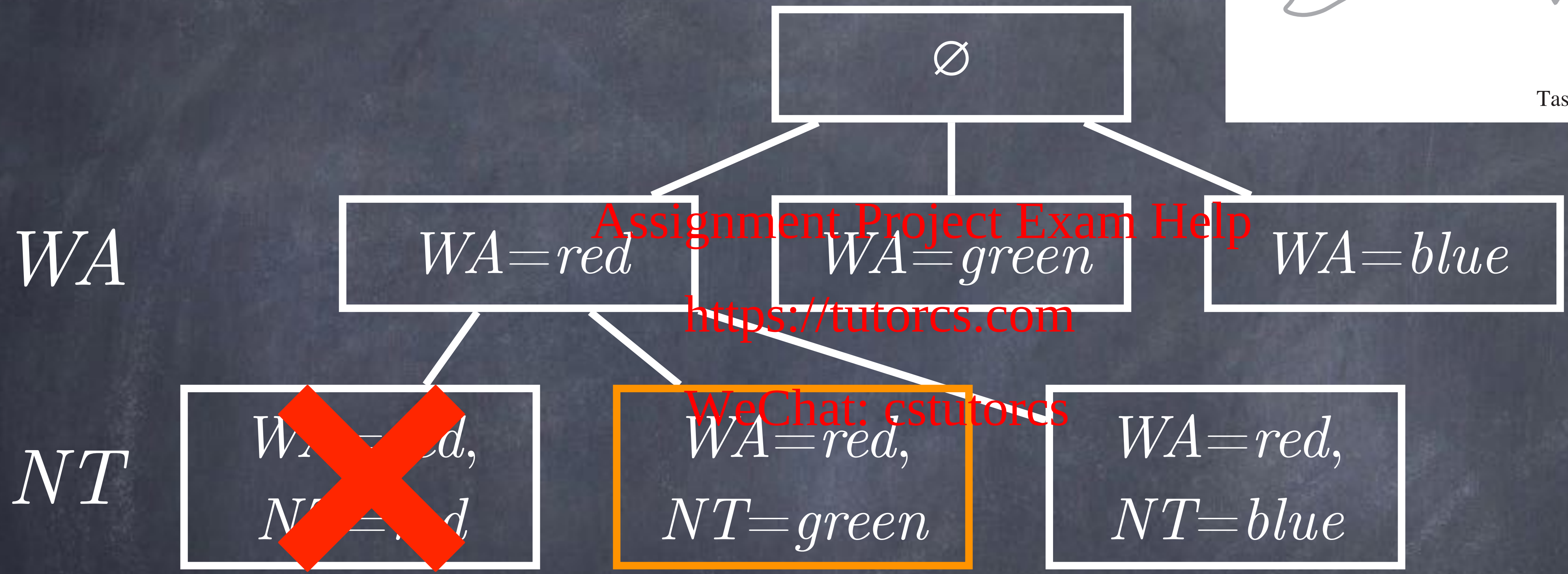
NT

$WA=red,$ $NT=red$ | $WA=red,$ $NT=green$ | $WA=red,$ $NT=blue$

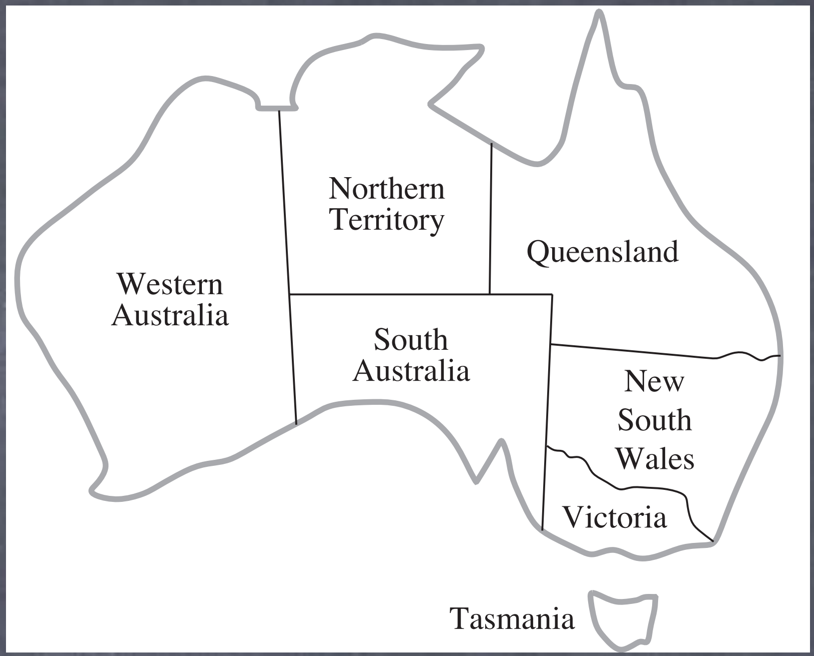
Inconsistent!


Northern
Territory
Queensland
Western
Australia
South
Australia
New
South
Wales
Victoria
Tasmania

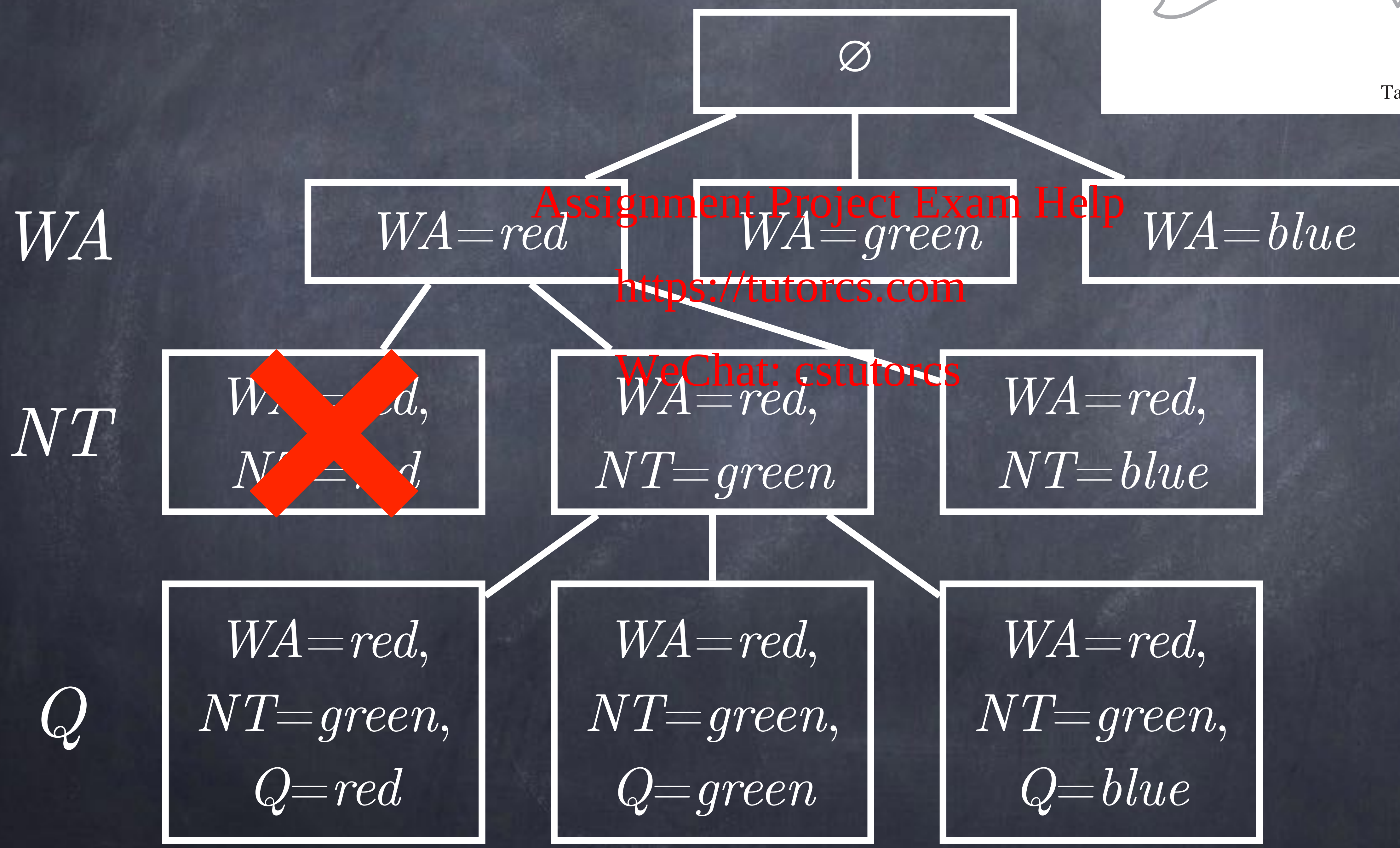

∅
WA
WA=red
WA=green
WA=blue
NT
WA=red,
NT=green
WA=red,
NT=blue
Prune!



$\varnothing$
WA
$WA\!=\!red$
$WA\!=\!green$
$WA\!=\!blue$
NT
$WA\!=\!red,$
$NT\!=\!green$
$WA\!=\!red,$
$NT\!=\!blue$

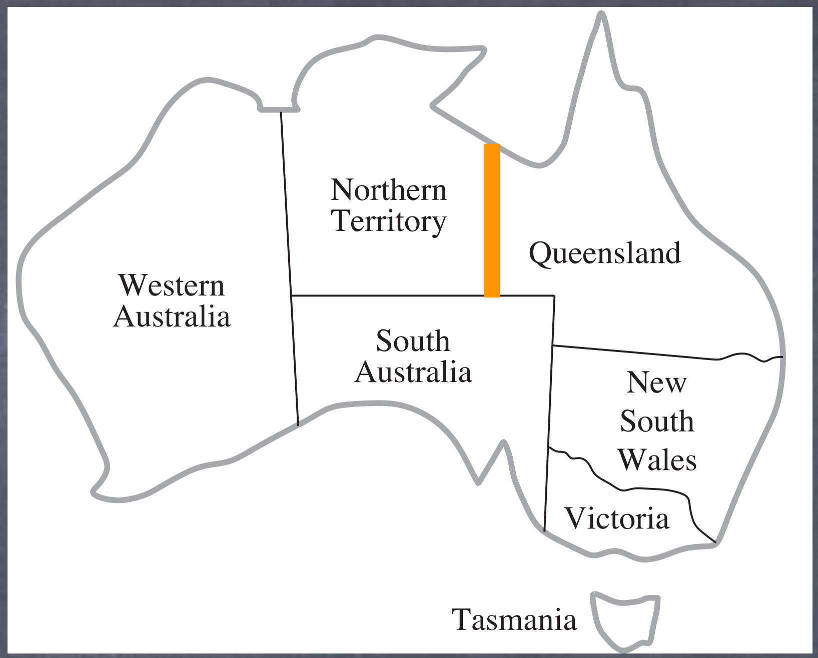


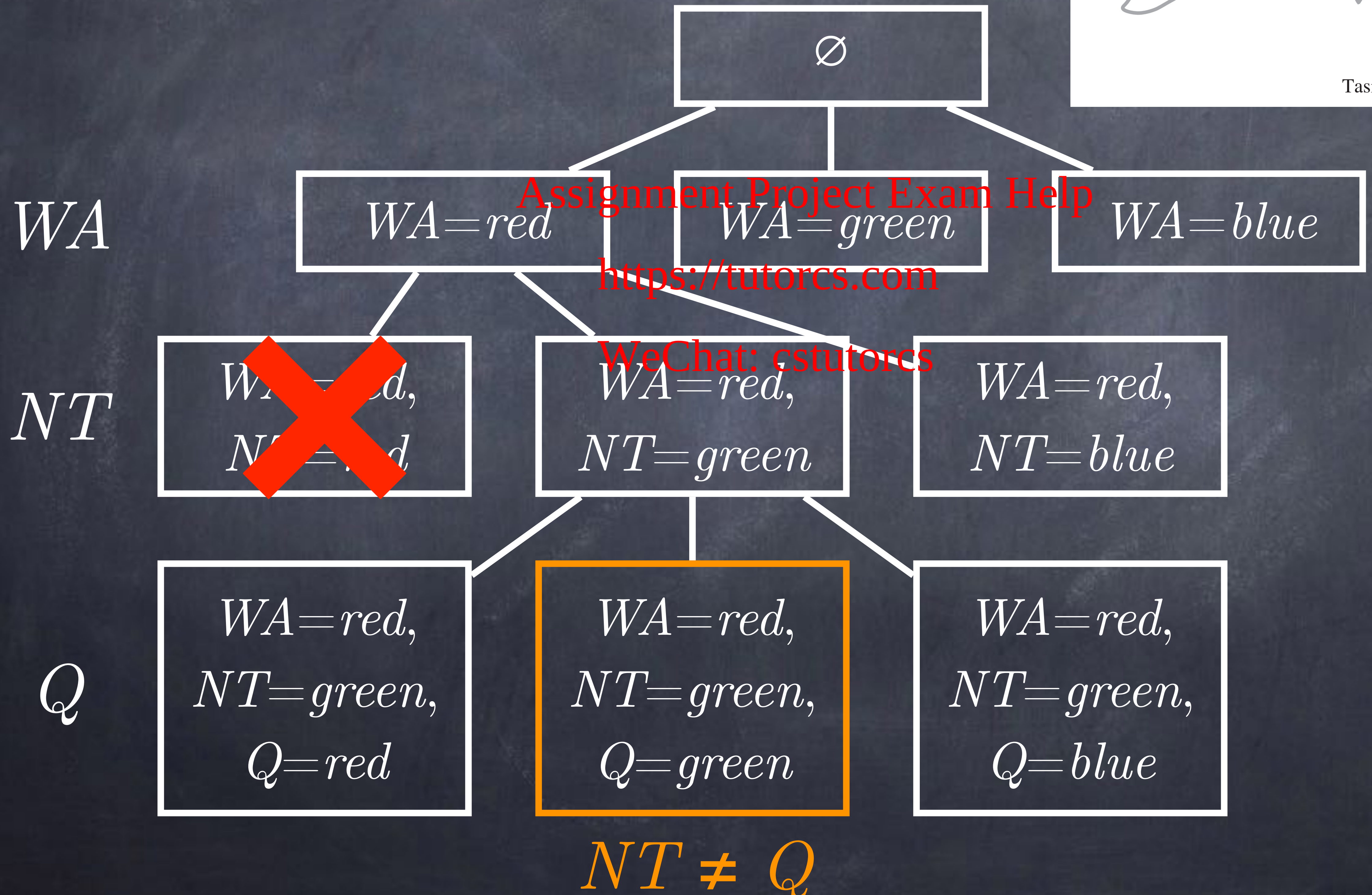
Northern Territory
Queensland
Western Australia
South Australia
New South Wales
Victoria
Tasmania



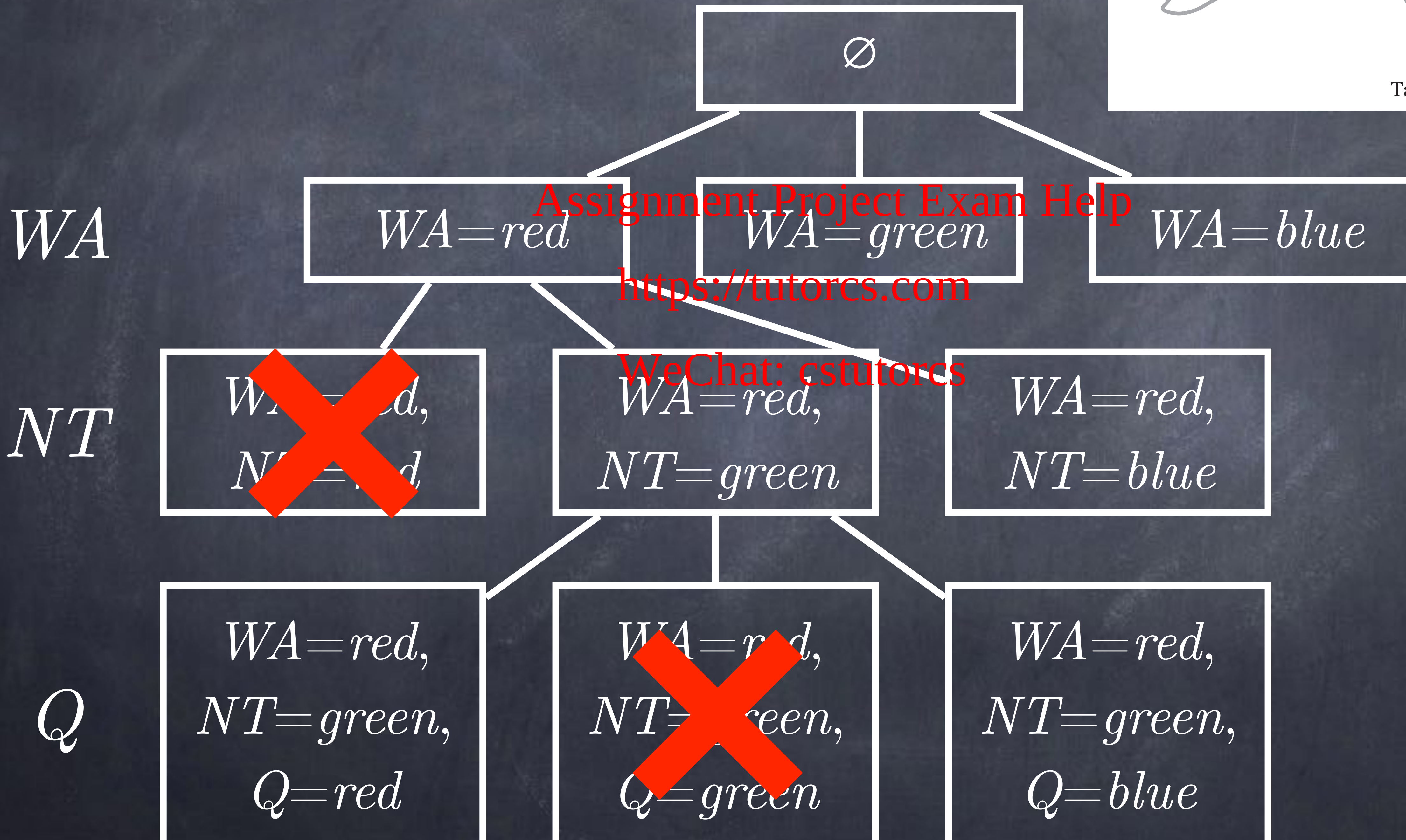
∅
WA
WA=red
WA=green
WA=blue
NT
WA=red, NT=green
WA=red, NT=blue
Q
WA=red, NT=green, Q=red
WA=red, NT=green, Q=green
WA=red, NT=green, Q=blue

Northern Territory
Queensland
Western Australia
South Australia
New South Wales
Victoria
Tasmania


Northern Territory
Queensland
Western Australia
South Australia
New South Wales
Victoria
Tasmania

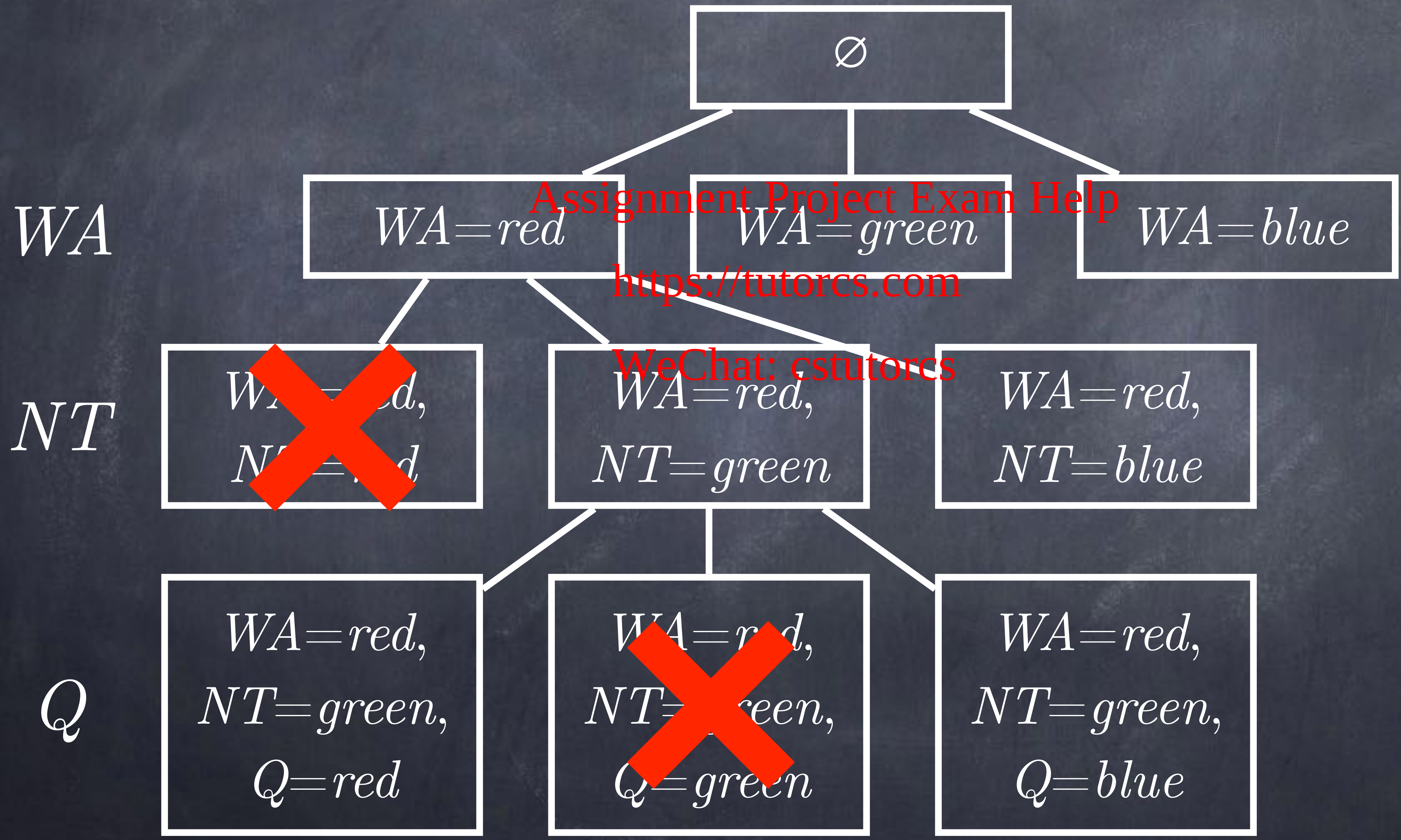

$\emptyset$
WA
$WA = red$
$WA = green$
$WA = blue$
NT
$WA = red, NT = green$
$WA = red, NT = blue$
Q
$WA = red, NT = green, Q = red$
$WA = red, NT = green, Q = green$
$WA = red, NT = green, Q = blue$
$NT \neq Q$


Northern Territory
Queensland
Western Australia
South Australia
New South Wales
Victoria
Tasmania


∅
WA
WA=red
WA=green
WA=blue
NT
WA=red, NT=green
WA=red, NT=blue
Q
WA=red, NT=green, Q=red
WA=red, NT=green, Q=blue

Prune!

Early Pruning of Inconsistent States



Backtracking Search

AIMA Fig 6.5

```
function BT(csp)
  return backtrack({}, csp)

function backtrack(assignment, csp)
  if (assignment is complete)
    return assignment
  var = SelectUnassignedVar(csp)
  foreach value in OrderDomainValues(var,assignment,csp)
    if (value is consistent with assignment)
      add <var,value> to assignment
      result = backtrack(assignment, csp)
      if (result != failure)
        return result
      else
        remove <var,value> from assignment
  return failure
```

Backtracking Search

- DFS search through the space of assignments
- Assign one variable at a time
- Early pruning of inconsistent states
- Solves ANY CSP

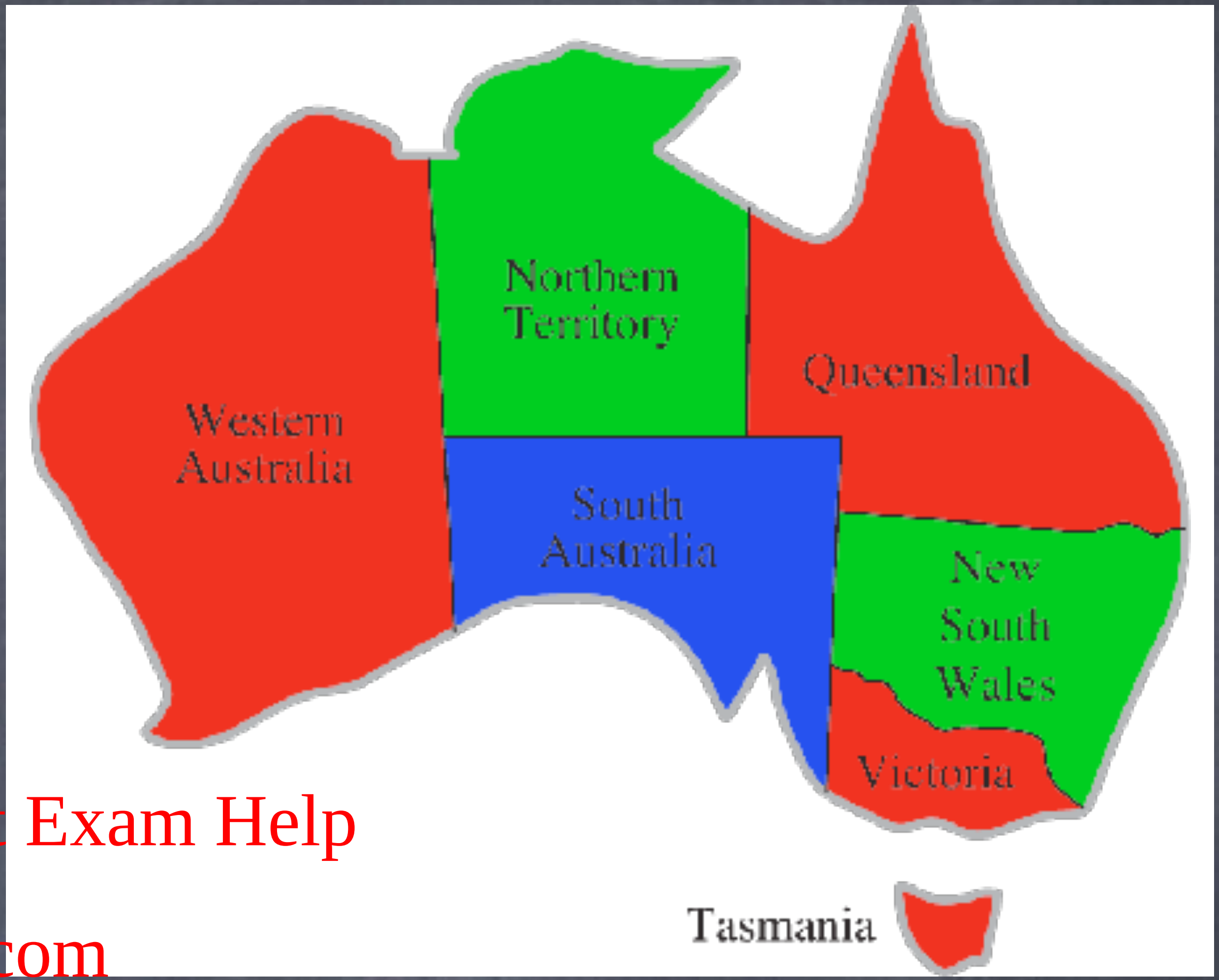
Heuristics for CSPs

- Minimum-remaining values (most constrained variable)
- Degree heuristic (variable involved in most constraints with unassigned variables)
- Least constraining value (if we only want to find one solution)

But wait, there's more!

WA	$r,\ g,\ b$
NT	$r,\ g,\ b$
SA	$r,\ g,\ b$
Q	$r,\ g,\ b$
NSW	$r,\ g,\ b$
V	$r,\ g,\ b$
T	$r,\ g,\ b$

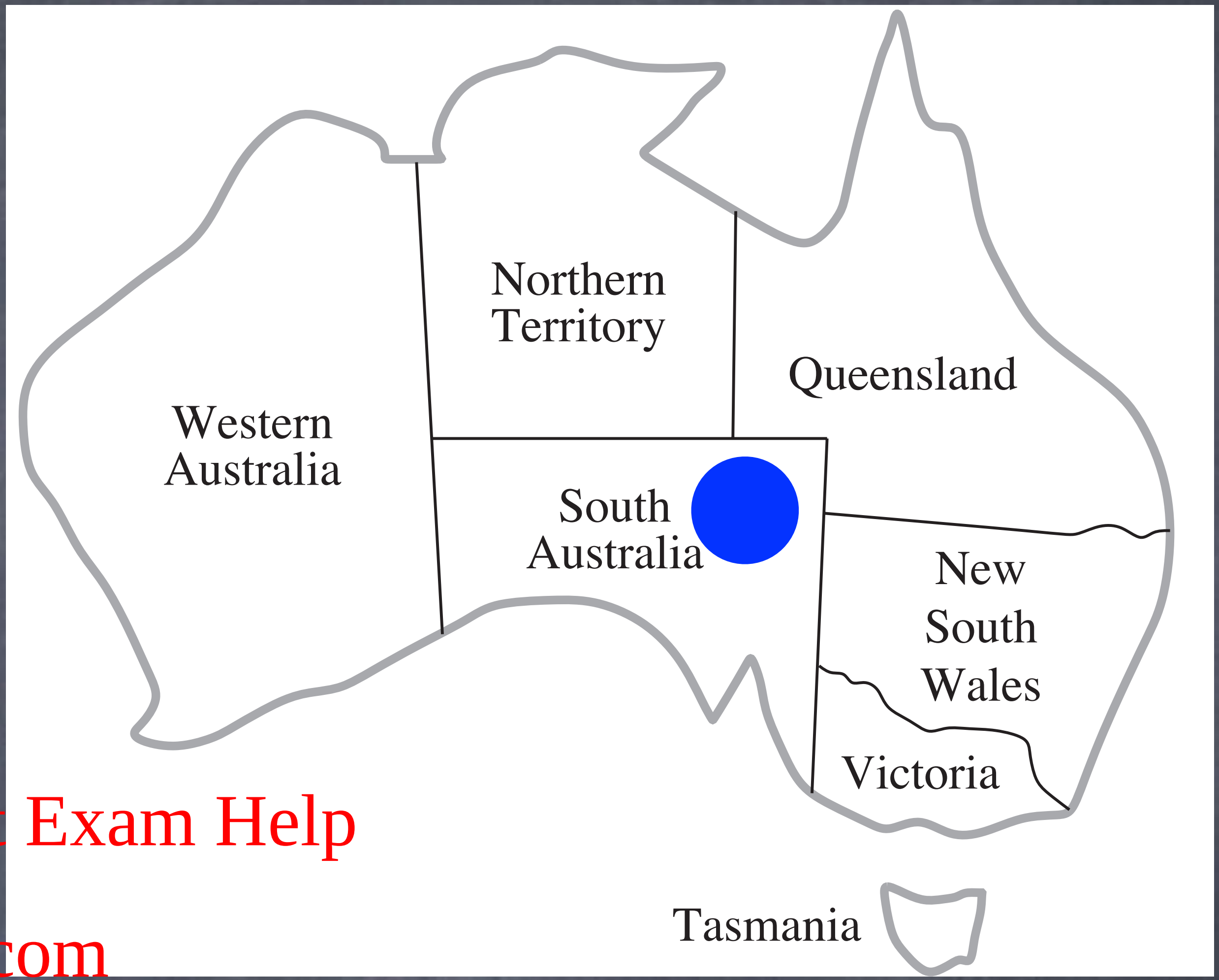
WA	r
NT	g
SA	b
Q	r
NSW	g
V	r
T	r



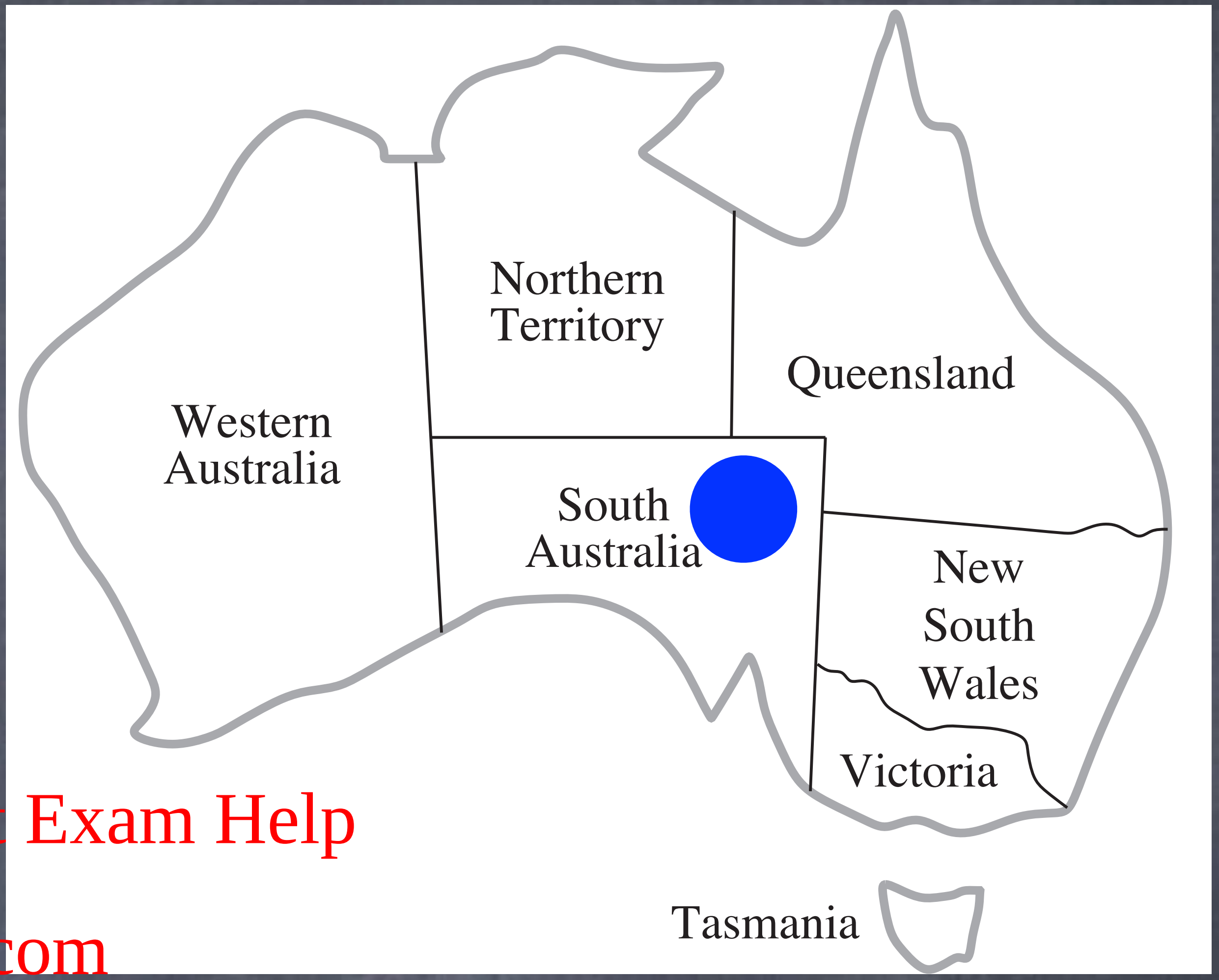
Solution

WA	$r,\ g,\ b$
NT	$r,\ g,\ b$
SA	$r,\ g,\ b$
Q	$r,\ g,\ b$
NSW	$r,\ g,\ b$
V	$r,\ g,\ b$
T	$r,\ g,\ b$

WA	r, g, b
NT	r, g, b
SA	b
Q	r, g, b
NSW	r, g, b
V	r, g, b
T	r, g, b

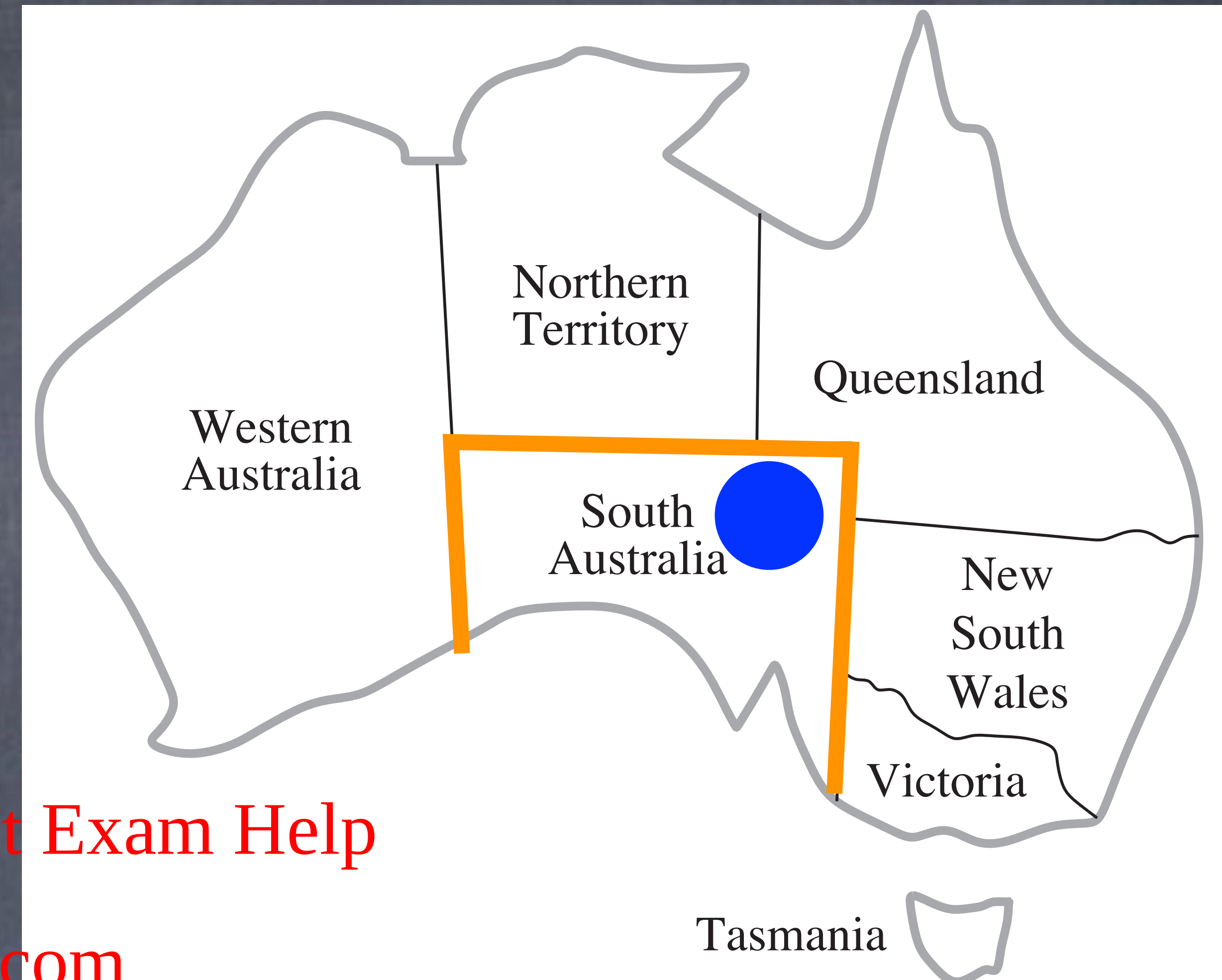


WA	$r,\ g,\ b$
NT	$r,\ g,\ b$
SA	b
Q	$r,\ g,\ b$
NSW	$r,\ g,\ b$
V	$r,\ g,\ b$
T	$r,\ g,\ b$



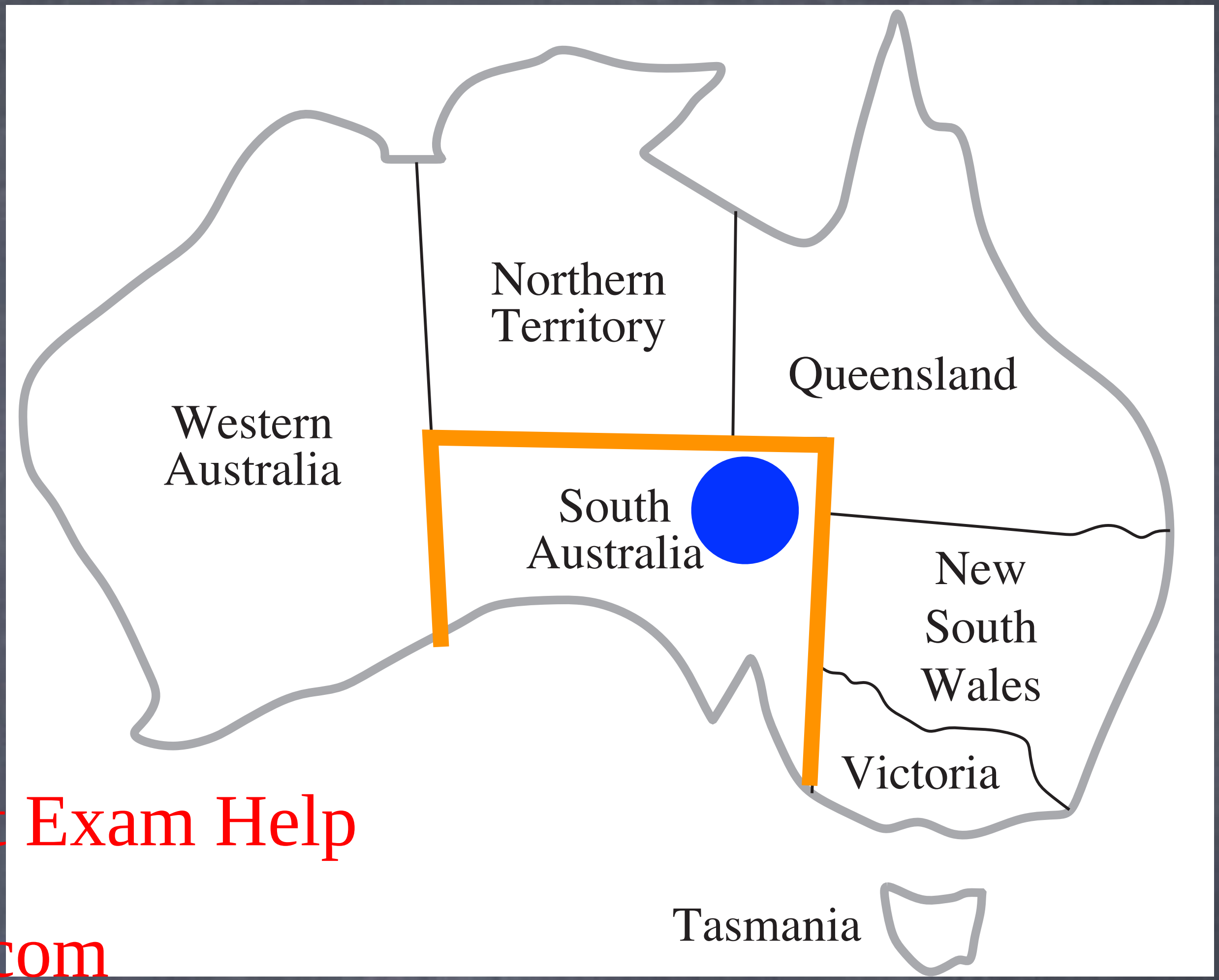
Remaining possibilities: $3^6 = 729$

WA	$r,\ g,\ b$
NT	$r,\ g,\ b$
SA	b
Q	$r,\ g,\ b$
NSW	$r,\ g,\ b$
V	$r,\ g,\ b$
T	$r,\ g,\ b$



$$WA \neq NT, WA \neq SA, NT \neq Q, NT \neq SA,$$
$$Q \neq NSW, Q \neq SA, NSW \neq V, NSW \neq SA,$$
$$V \neq SA$$

WA	$r,\ g$
NT	$r,\ g$
SA	b
Q	$r,\ g$
NSW	$r,\ g$
V	$r,\ g$
T	$r,\ g,\ b$



Remaining possibilities: $2^5 \times 3 = 96$

$$WA \neq NT, WA \neq SA, NT \neq Q, NT \neq SA,$$
$$Q \neq NSW, Q \neq SA, NSW \neq V, NSW \neq SA,$$
$$V \neq SA$$

Constraint Propagation

- Using the constraints to reduce the set of possible values of a variable, which can in turn reduce the possible values of another variable, and so on
- Not a search process!
- Part of state update during state-space search

Constraint Propagation

- Using the constraints to reduce the set of possible values of a variable, which can in turn reduce the possible values of another variable, and so on
- Not a search process!
- Part of state update in state-space search
- A form of inference

Constraint Propagation

- Good:
  - Can significantly reduce the space of assignments left to search
- Bad:
  - It takes time. How much?

Constraints

- Unary constraint: one variable
  - e.g., $NSW \neq red$, X_i is even, $X_i = 2$

- Binary constraint: two variables

  - e.g., $NSW \neq WA$, $X_i > X_j$, $X_i + X_j = 2$

- "Global" constraint: more than two vars
  - e.g., X_i is between X_j and X_k, $AllDiff(X_i, X_j, X_k)$

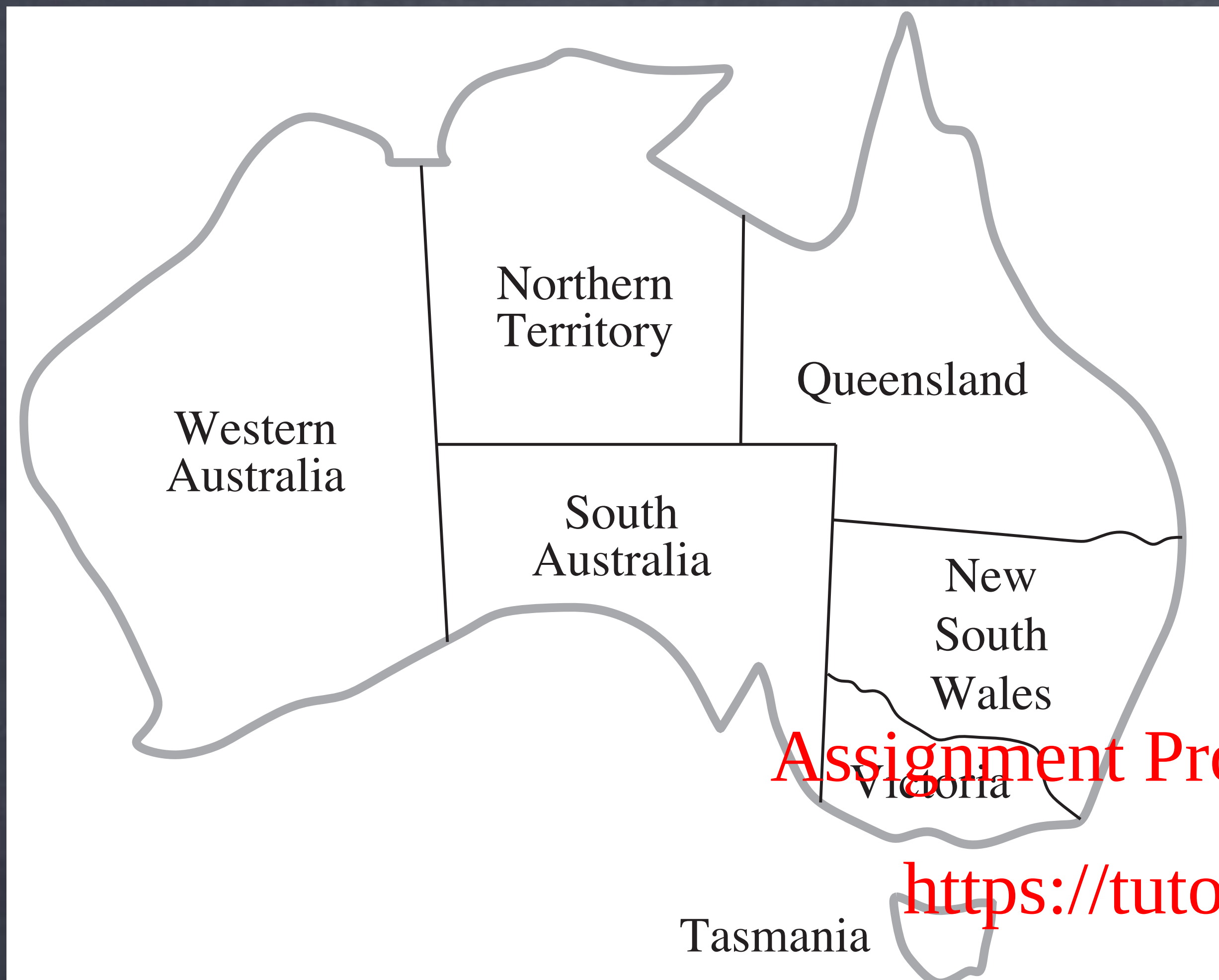
Constraints

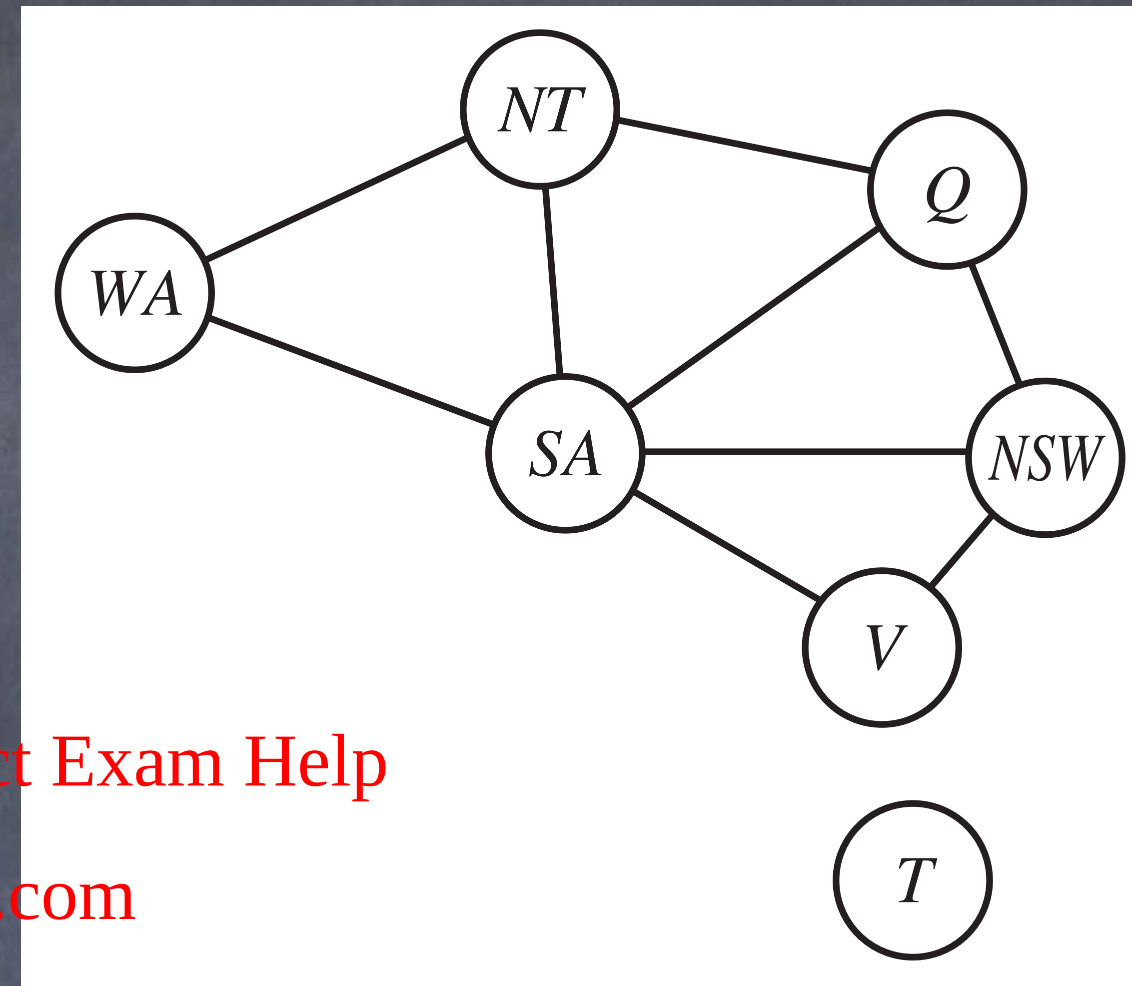
- Unary constraint: one variable
  - e.g., $NSW \neq red$, X_i is even, $X_i = 2$

- Binary constraint: two variables

  - e.g., $NSW \neq WA$, $X_i > X_j$, $X_i + X_j = 2$

- "Global" constraint: more than two vars
  - e.g., X_i is between X_j and X_k, $AllDiff(X_i, X_j, X_k)$
  - Can be reduced to set of binary constraints (possibly inefficiently)





$$WA \neq NT, WA \neq SA, NT \neq Q, NT \neq SA,$$
$$Q \neq NSW, Q \neq SA, NSW \neq V, NSW \neq SA,$$
$$V \neq SA$$

Constraints

- Unary constraint: one variable
  - e.g., $NSW \neq red$, X_i is even, $X_i = 2$

- Binary constraint: two variables
  - e.g., $NSW \neq WA$, $X_i > X_j$, $X_i + X_j = 2$
- "Global" constraint: more than two vars
  - e.g., X_i is between X_j and X_k, $AllDiff(X_i, X_j, X_k)$
  - Can be reduced to set of binary constraints (possibly inefficiently)

WA	$r,\ g,\ b$
NT	$r,\ g,\ b$
SA	$r,\ g,\ b$
Q	$r,\ g,\ b$
NSW	$r,\ g,\ b$
V	$r,\ g,\ b$
T	$r,\ g,\ b$

$SA \neq g$

Unary constraint

WA	$r,\ g,\ b$
NT	$r,\ g,\ b$
SA	$r,\ g,\ b$
Q	$r,\ g,\ b$
NSW	$r,\ g,\ b$
V	$r,\ g,\ b$
T	$r,\ g,\ b$

$SA \neq g$

Unary constraint

WA	$r,\ g,\ b$
NT	$r,\ g,\ b$
SA	$r,\ b$
Q	$r,\ g,\ b$
NSW	$r,\ g,\ b$
V	$r,\ g,\ b$
T	$r,\ g,\ b$

$SA \neq g$

Unary constraint

Node Consistency

- Every possible value of every variable is consistent with the unary constraints

Node Consistency

- Every possible value of every variable is consistent with the unary constraints
- Propagate unary constraints at the start
  - Inconsistent? Fail!
  - Otherwise never have to test them again

Node Consistency

- Every possible value of every variable is consistent with the unary constraints

- Propagate unary constraints at the start

  - Inconsistent? Fail!

  - Otherwise never have to test them again
- Complexity: Each variable, each value, each unary constraint

Arc Consistency

- Unary constraint: one variable
  - e.g., $NSW \neq red$, X_i is even, $X_i = 2$

- Binary constraint: two variables
  - e.g., $NSW \neq WA$, $X_i > X_j$, $X_i + X_j = 2$
- "Global" constraint: more than two vars
  - e.g., X_i is between X_j and X_k, $AllDiff(X_i, X_j, X_k)$
  - Can be reduced to set of binary constraints (possibly inefficiently)

Arc Consistency

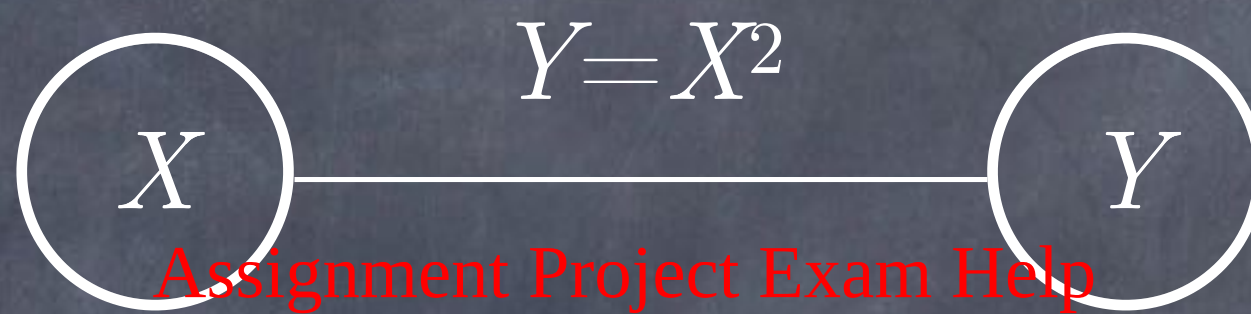
X_i is arc-consistent w.r.t. X_j if

for every value in the domain D_i,

there is some value in the domain D_j

that satisfies the binary constraint on the arc (X_i, X_j)

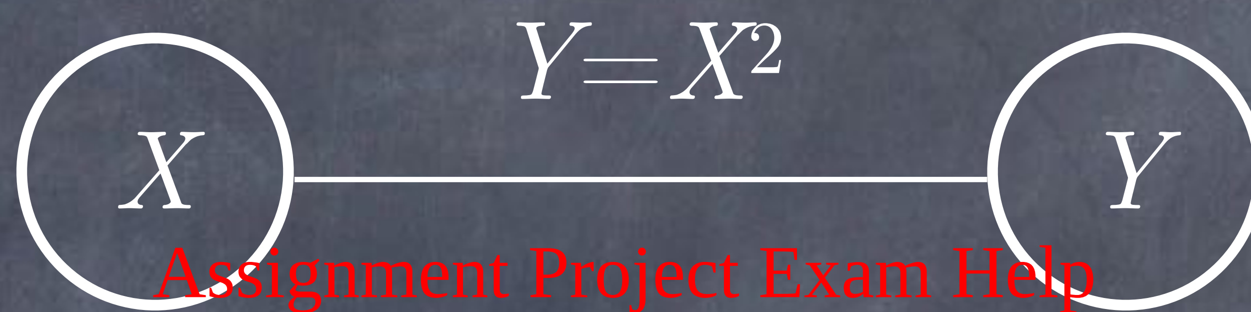
Arc Consistency



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

\# possible assignments: $10 \times 10 = 100$

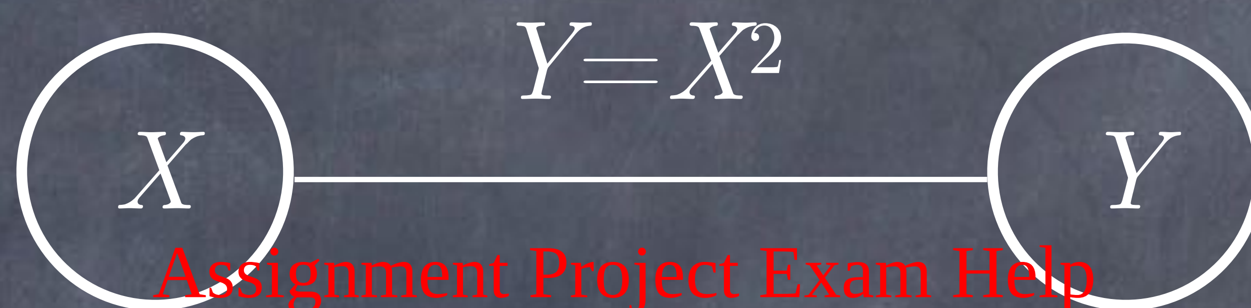
Arc Consistency



$\{\ 0,1,2,3,4,5,6,7,8,9\ \}$ $\qquad$ $\{\ 0,1,2,3,4,5,6,7,8,9\ \}$

Make X arc-consistent with respect to Y

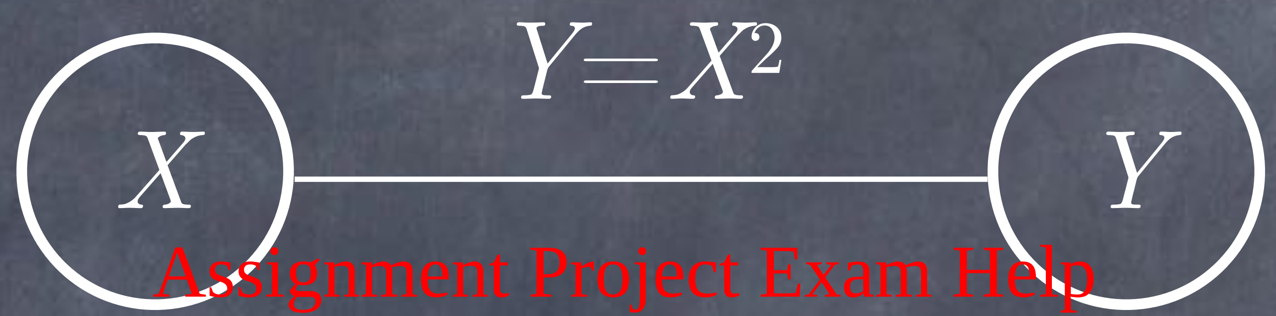
Arc Consistency



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

X arc-consistent with respect to Y

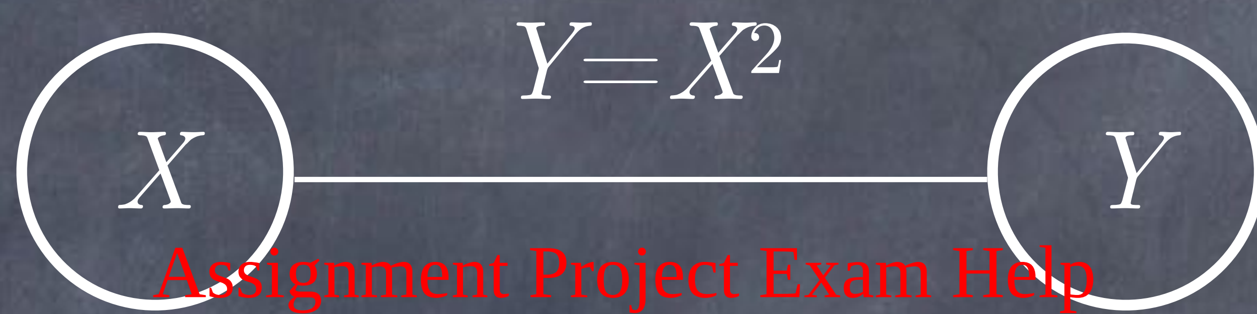
Arc Consistency



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

Make Y arc-consistent with respect to X

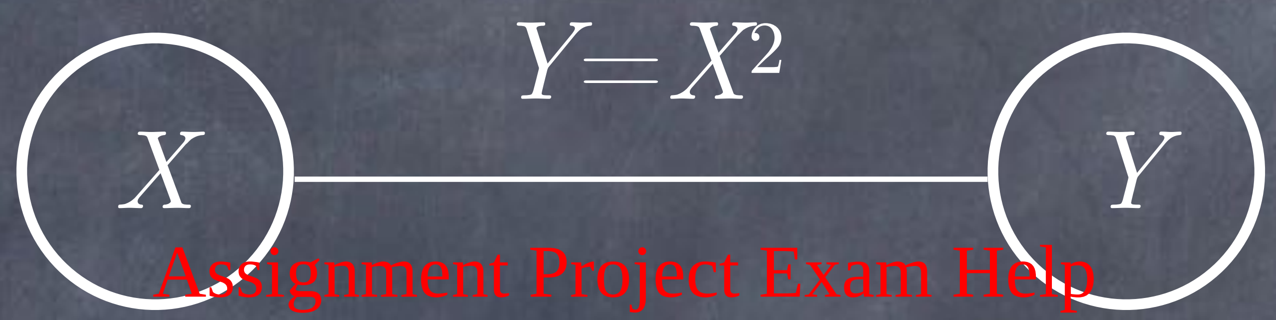
Arc Consistency



$\{ 0,1,2,3 \}$ $\{ 0,1,4,9 \}$

Y arc-consistent with respect to X

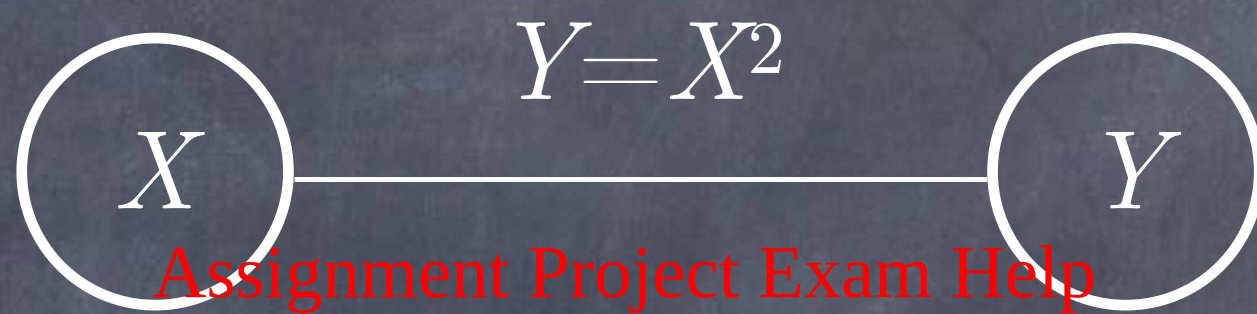
Arc Consistency



$\{ 0,1,2,3 \}$ $\{ 0,1,4,9 \}$

X and Y arc-consistent

Arc Consistency



$\{ 0,1,2,3 \}$ $\{ 0,1,4,9 \}$

X and Y arc-consistent

possible assignments: $4 \times 4 = 16$

AC-3

```
boolean AC3(csp) {
  Set queue = all arcs in csp
  while (queue is not empty) {
    <i,j> = queue.removeFirst()
    if (revise(csp, i, j)) {
      if Di is empty {
        return false
      }
      foreach k in neighbors(i) {
        add <k,i> to queue
      }
    }
  }
  return true
}
```

```
boolean revise(csp, i, j) {
  boolean changed = false
  foreach vi in Di {
    boolean ok = false
    foreach vj in Dj {
      if (<vi,vj> satisfies Cij )
        ok = true
      }
    }
    if (!ok) {
      delete vi from Di
      changed = true
    }
  }
  return changed
}
```

AC-3 Analysis

- CSP with n variables, domain size $O(d)$, c constraints (arcs)
- Each arc can be inserted in the queue at most d times
- Checking a single arc takes $O(d^2)$ time
- Total time: $O(cd^3)$
  - Independent of n

More Constraint Propagation

- Path consistency
- k-consistency
  - Generalization of node (1-), arc (2-), and path (3-) consistency
  - Establishing k-consistency is exponential in k
  - Typically use node- and arc-consistency and rarely path-consistency

Constraint Propagation

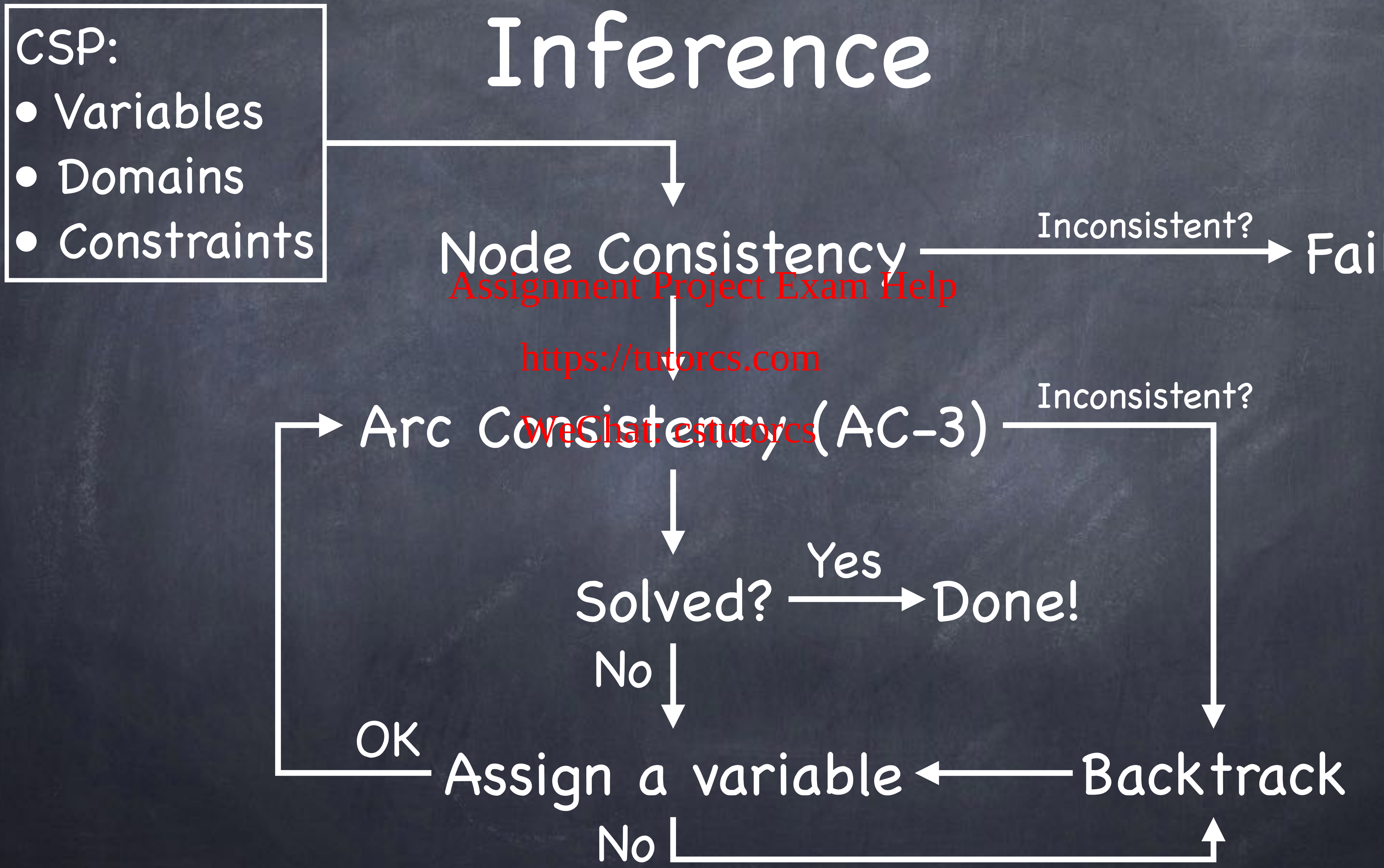
- "After constraint propagation, we are left with a CSP that is equivalent to the original CSP—they both have the same solutions—but the new CSP will in most cases be faster to search because its variables have smaller domains."

Constraint Propagation (inference) ⟷ State-Space Search

Interleaving Search and Inference

- After each choice during search, we can perform inference to reduce future search

Interleaving Search and Inference



Interleaving Search and Inference

- After each choice during search, we can perform inference to reduce future search
- Forward checking
- MAC: Maintaining Arc Consistency
- Bottom line: Cost of inference is subsumed by cost of search, so do it

Solving CSPs

- Search through space of assignments
  - Commutativity => Only have to consider assignment to one variable at a time
- Interleave search and inference
  - Constraint propagation to reduce domains of variables for subsequent search

Constraint Satisfaction

- Impose a structure on the representation of states: Variables, Domains, Constraints
- Backtracking (DFS) search for complete, consistent assignment of values to variables
- Inference (constraint propagation) can reduce the domains of variables
  - Preprocessing and/or interleaved with search
- Useful problem-independent heuristics

CSP Secret Sauce

- Factored representation of state:
  - Variables, Domains, Constraints
- Allows:
  - Early pruning of inconsistent states
  - Inference during search to reduce alternatives

For next time:

AIMA 7.0 - 7.4