BITS Pilani
Pilani Campus

Artificial & Computational Intelligence

DSE CLZG557

M3 : Game Playing & Constraint Satisfaction

Raja vadhana P
Assistant Professor,
BITS - CSIS

Course Plan

M1 Introduction to AI

M2 Problem Solving Agent using Search

M3 Game Playing, Constraint Satisfaction Problem

M4 Knowledge Representation using Logics

M5 Probabilistic Representation and Reasoning

M6 Reasoning over time, Reinforcement Learning

Module 3 : Part -2

Constraint Satisfaction Problem

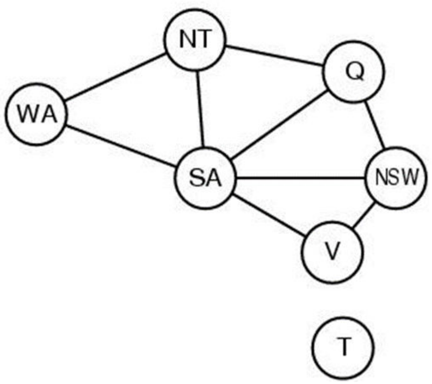
A. Formulating a CSP problem

B. Constraint propagation

C. Local search for CSP

Problem Formulation

Map Coloring Problem



Variables

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

Domain

D_i = {red , green, blue}

D_i = {R, G, B}

Constraints

WA ≠ NT

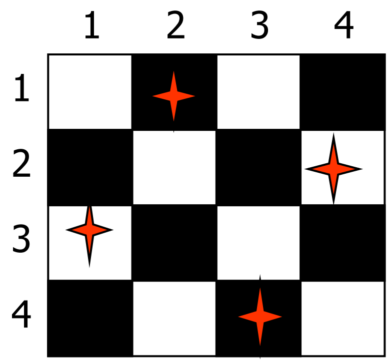
(WA,NT) =
{(R,G), (R,B), (G,R), …}

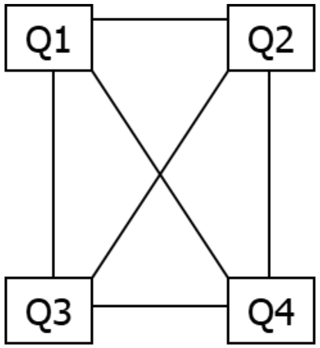
Solution

{WA = R, NT = G,Q = R,NSW = G, V = R, SA =B ,T = G }

Problem Formulation

N-Queen

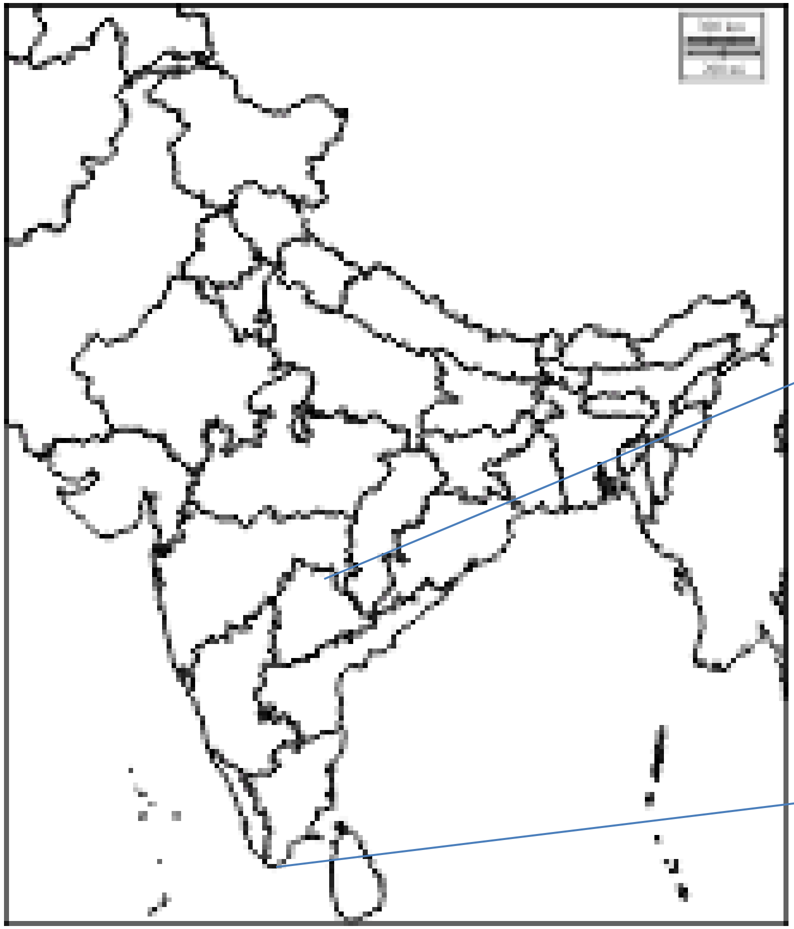


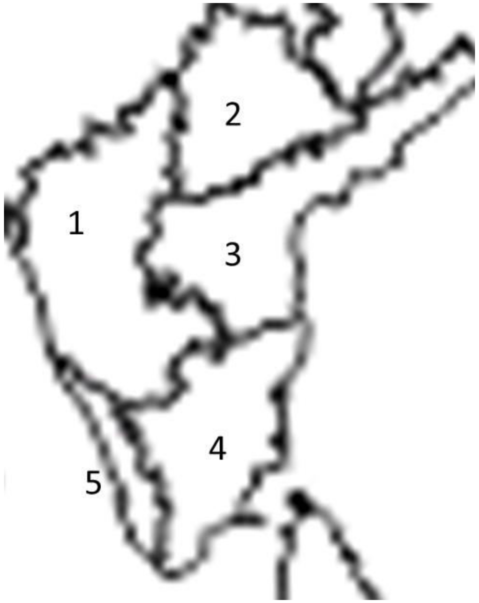


Variables
Q1, Q2, Q3, Q4.. QN
Domain
{1,2,…..N}
Constraints
(Q1,Q2)={(1,3), (1,4), (2,4)…}
Solution
Q1=3, Q2=1, Q3=4, Q4=2

Introduction to Constraint Propagation

Constraint Satisfaction Problem





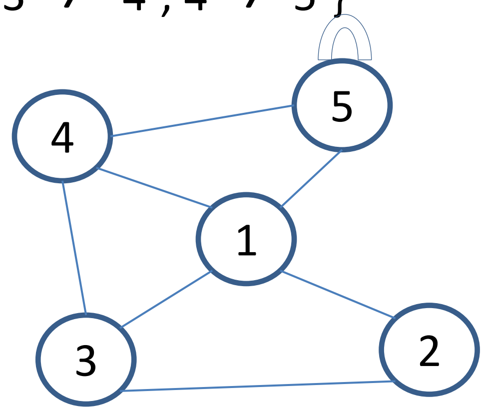
Constraint Satisfaction Problem

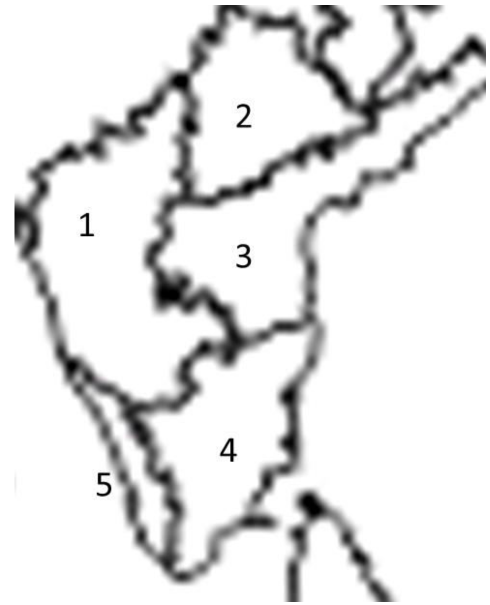
Variables = {1, 2, 3, 4, 5}
Domain = { R , G, B, Y }
Constraints =

{ 5 ≠ R ,

1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}





Objective: Color the marked states with available colors (from the domain set) such that no neighboring states share the same color.
Another restriction(constraint) is that state coded as 5 should not have Red(R) color

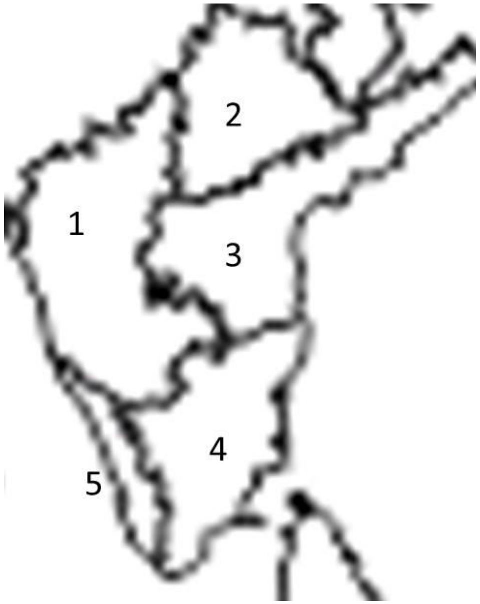
Constraint Satisfaction Problem

Variables = {1, 2, 3, 4, 5}
Domain = { R , G, B, Y }
Constraints =

{ 5 ≠ R ,

1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}



Problem Formulation
Initial State : Empty Assignment .
Successor Function : Consistent current assignment
Goal Test : Complete Consistent Assignment as of this state
Path Cost : Every backtracking = 10 penalty or Every Dead End = 5

Avenues to Improve - Heuristics

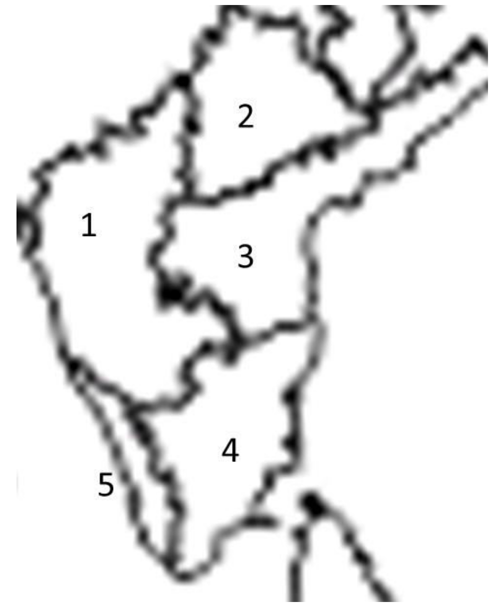
Depth-first search for CSPs with single-variable assignments is called backtracking search

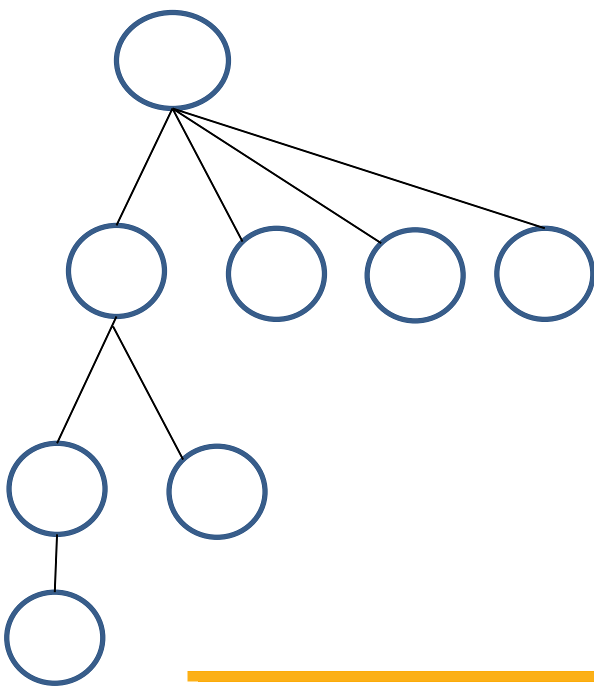
```
function BACKTRACKING-SEARCH(csp) return a solution or failure
    return RECURSIVE-BACKTRACKING({} , csp)

function RECURSIVE-BACKTRACKING(assignment, csp) return a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to CONSTRAINTS[csp] then
            add {var=value} to assignment
            result ← RECURSIVE-BACTRACKING(assignment, csp)
            if result ≠ failure  then return result
            remove {var=value} from assignment
    return failure
```

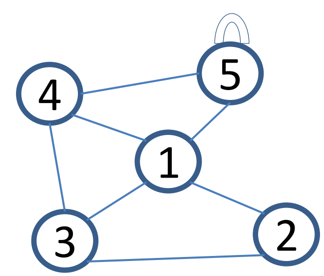
Constraint Satisfaction Problem

- MRV / Most constrained variable
  - choose the variable with the fewest legal values
- MCV / Most constraining variable
  - choose the variable with the most constraints on remaining variables
- LCV/ Least constraining value
  - Choose value that rules out the fewest values in the remaining variables
- Forward checking
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

	1	**2**	**3**	**4**	**5**
	R, G, B, Y	R, G, B, Y	R, G, B, Y	R, G, B, Y	R, G, B, Y
MCV					
MRV					
LCV					







Constraints =

{ 5 ≠ R ,

1\` ≠ 2\`, 1\` ≠ 3\`, 1\` ≠ 4\` , 1\` ≠ 5\`,

2\` ≠ 3\` , 3\` ≠ 4\`, 4\` ≠ 5\`}

Heuristics

- Frequent Techniques
  - Most constrained variable
  - Most constraining variable
  - Least constraining value
  - Forward checking

- MRV / Most constrained variable
  - choose the variable with the fewest legal values

- MCV / Most constraining variable
  - choose the variable with the most constraints on remaining variables

- LCV/ Least constraining value
  - Choose value that rules out the fewest values in the remaining variables

- Forward checking
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

Local Search for CSP

Sudoku Problem

Constraint Satisfaction Graph - Subgraph

Sudoku

	1	2	3	4
A		2	1	
B	4			
C				2
D		3		

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	2	3

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	1	3

2+2

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	2	3

1+2

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	3	3

3+2

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	4	3

2

Local optima achieved

Constraint Satisfaction Graph - Subgraph

Sudoku

	1	2	3	4
A		2	1	
B	4			
C				2
D		3		

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	2	3

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	1	3

2+2

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	2	3

1+2

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	3	3

3+2

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	4	3

2

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	1	3	4	3

0

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	2	3	4	3

2

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	3	3	4	3

3+2

	1	2	3	4
A	3	2	1	3
B	4	2	3	4
C	2	4	4	2
D	4	3	4	3

3+2

Local optima achieved

Constraint Propagation

Arc Consistency Algorithm

AC-3 (Arc Consistency Algorithm)

function AC-3(csp) return the CSP, possibly with reduced domains

inputs: csp, a binary csp with variables $\{X_1, X_2, \ldots, X_n\}$

local variables: queue, a queue of arcs initially the arcs in csp

while queue is not empty do

$(X_i, X_j) \leftarrow$ *REMOVE-FIRST(queue)*

if REMOVE-INCONSISTENT-VALUES(X_i, X_j) *then*

for each X_k *in NEIGHBORS*$[X_i]$ *do*

add (X_k, X_i) *to queue*

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) *return true iff we remove a value*

removed $\leftarrow$ *false*

for each x in DOMAIN$[X_i]$ *do*

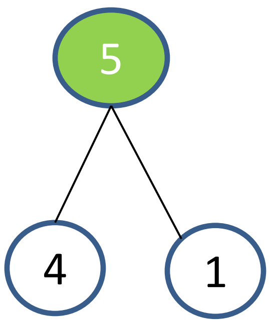
if no value y in DOMAIN$[X_j]$ *allows (x,y) to satisfy the constraints between* X_i *and* X_j

then delete x from DOMAIN$[X_i]$*; removed* $\leftarrow$ *true*

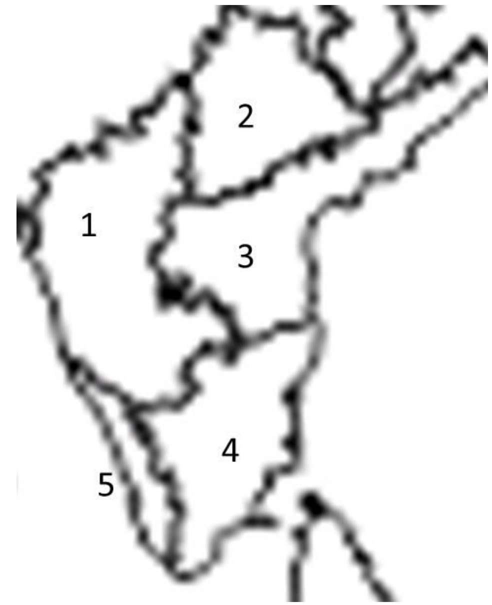
return removed

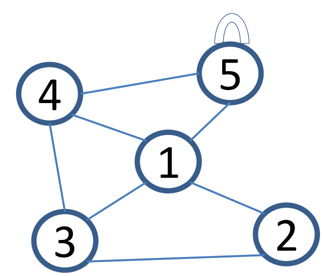
AC-3 (Arc Consistency Algorithm)

1	2	3	4	5
R, ~~G~~, B, Y	R, G, B, Y	R, G, B, Y	R, ~~G~~, B, Y	R, G, B, Y
				G



Xj	Xi	Domain(Xi)	Domain(Xi)
5	1	{R,G,B,Y}	{R,B,Y}
	4	{R,G,B,Y}	{R,B,Y}
1	2	{R,G,B,Y}	
	3	{R,G,B,Y}	
	4	{R,G,B,Y}	
	5	{G}	
4	1	{R,B,Y}	
	3	{R,G,B,Y}	
	5	{G}	





Constraints =

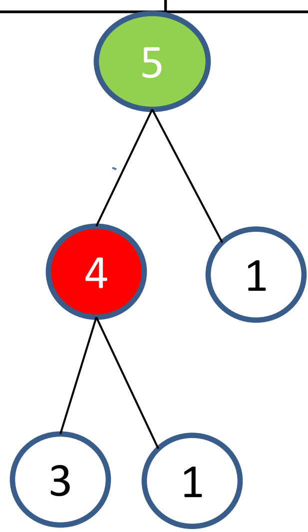
{ 5 ≠ R ,

1\` ≠ 2\`, 1\` ≠ 3\`, 1\` ≠ 4\` , 1\` ≠ 5\`,

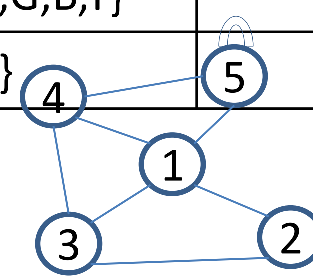
2\` ≠ 3\` , 3\` ≠ 4\`, 4\` ≠ 5\`}

AC-3 (Arc Consistency Algorithm)

1	2	3	4	5
~~R, G~~, B, Y	R, G, B, Y	~~R,~~ G, B, Y	R, ~~G,~~ B, Y	~~R,~~ G, B, Y
				G
			R	G



Xj	Xi	Domain(Xi)	Domain(Xi)
4	1	{R,B,Y}	{B,Y}
	3	{R,G,B,Y}	{G,B,Y}
	5	{G}	
1	2	{R,G,B,Y}	
	3	{R,G,B,Y}	
	4	{R}	
	5	{G}	
3	1	{B,Y}	
	2	{R,G,B,Y}	
	4	{R}	



Constraints =

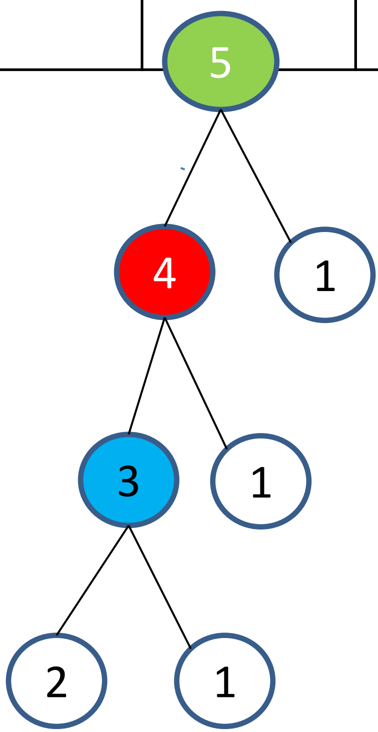
{ 5 ≠ R ,

1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

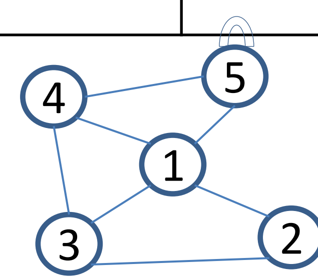
2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}

AC-3 (Arc Consistency Algorithm)

1	2	3	4	5
~~R, G, B~~, Y	R, G~~, B,~~ Y	~~R,~~ G, B, Y	R, ~~G, B,~~ Y	~~R,~~ G, B, Y
				G
			R	G
		B	R	G



Xj	Xi	Domain(Xi)	Domain(Xi)
3	1	{B,Y}	{Y}
	2	{R,G,B,Y}	
	4	{R}	
1	2	{R,G,B,Y}	{R,G,B}
	3	{B}	
	4	{R}	
	5	{G}	
2	1	{Y}	
	3	{B}	



Constraints =

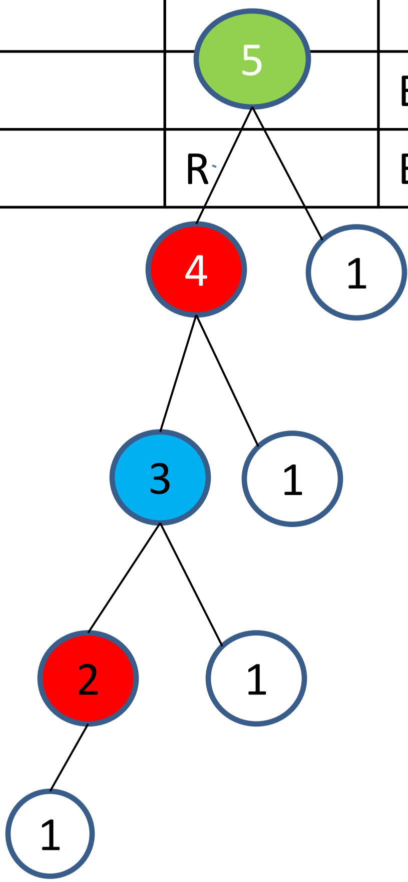
{ 5 ≠ R ,

1` ≠ 2`, 1` ≠ 3`, 1` ≠ 4` , 1` ≠ 5`,

2` ≠ 3` , 3` ≠ 4`, 4` ≠ 5`}

Constraint Satisfaction Problem-Earlier attempt

1	2	3	4	5
~~R, G, B, Y~~	R, G~~, B,~~ Y	~~R,~~ G, B, ~~Y~~	R, ~~G, B,~~ Y	~~R~~, G, B, Y
				G
			R	G
		B	R	G
	R	B	R	G



Xj	Xi	Domain(Xi)	Domain(Xi)
3	1	{B,Y}	{Y}
	2	{R,G,B}	{R,G}
	4	{R}	
1	2	{R,G,B,Y}	{R,G,B}
	3	{B}	
	4	{R}	
	5	{G}	
2	1	{Y}	
	3	{B}	

Constraints =

{ 5 ≠ R ,

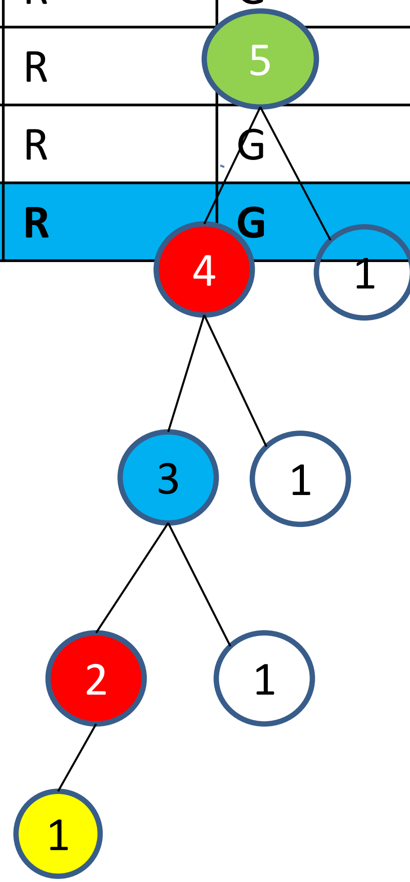
1\` ≠ 2\`, 1\` ≠ 3\`, 1\` ≠ 4\` , 1\` ≠ 5\`,

2\` ≠ 3\` , 3\` ≠ 4\`, 4\` ≠ 5\`}

AC-3 (Arc Consistency Algorithm)

1	**2**	**3**	**4**	**5**
~~R,~~ ~~G,~~ ~~B,~~ Y	R, G~~, B,~~ Y	~~R,~~ G, B~~, Y~~	R~~, G, B,~~ Y	~~R~~, G, B, Y
				G
			R	G
		B	R	
	R	B	R	G
Y	**R**	**B**	**R**	**G**

Xj	Xi	Domain(Xi)	Domain(Xi)
2	1	{Y}	
	3	{B}	



Constraints =

{ 5 ≠ R ,

1\` ≠ 2\`, 1\` ≠ 3\`, 1\` ≠ 4\` , 1\` ≠ 5\`,

2\` ≠ 3\` , 3\` ≠ 4\`, 4\` ≠ 5\`}

PC-2 (Path Consistency Algorithm)

- Path consistency extends it to look at triples of variables.
- A two-variable set X, Y is path-consistent with a third variable Z iff for every value a of X and b of Y which satisfies the constraints on {X, Y}, there is a value c of Z which satisfies the constraints on {X,Z} and {Z,Y}.
- Path consistency algorithm PC-2 is extension of AC-3.

K-Consistency:

- A CSP is k-consistent if for any set of k - 1 variables and a consistent assignment to those variables, a consistent value can be assigned to the kth variable.
- In practice, don't usually go beyond AC-3 (2-consistency) and PC-2 (3-consistency)

Constraint Satisfaction Problem

Application of Learning in the Heuristic Ordering

Heuristics for variable priority:

1. Smallest ratio of current domain size to dynamic degree
2. Cutoff limit : No.of.allowable backtracking – Restart : Parallel strategy

K-Armed Bandit Framework

Reward function : Pruned Tree Size
Optimization Objective : Mminimize the expected regret over T trials, which is defined as the expectation of the difference between the total reward obtained by the best arm and the total reward obtained by the bandit algorithm.

Source Credit:24th European Conference on Artificial Intelligence - ECAI 2020

Constraint Satisfaction Problem

Application of Learning in the Heuristic Ordering

Algorithm 1: MAB Framework

Input: constraint network P, heuristics $H_1, \cdots, H_K$, bandit policy B

Initialize the bandit policy

    $\text{INITARMS}_B(K)$

Trials

for *each run* $t = 1, \cdots, T$ **do**

    Select an arm according to the bandit policy

    $i_t \leftarrow \text{SELECTARM}_B(K)$

    Run the solver and observe a reward

    $r_t(i_t) \leftarrow \text{MAC}(H_{i_t})$

    Update the bandit policy

    $\text{UPDATEARMS}_B(r_t)$

Source Credit:[24th European Conference on Artificial Intelligence - ECAI 2020]

Required Reading: AIMA - Chapter #5.1, #5.2, #5.3, #5.4

Thank You for all your Attention

Note : Some of the slides are adopted from AIMA TB materials

BITS Pilani
Pilani Campus

Artificial & Computational Intelligence

DSE CLZG557

M4 : Knowledge Representation using Logics

Raja vadhana P

Assistant Professor,

BITS - CSIS

Module 4 :

Knowledge Representation using Logics

A. Logical Representation

B. Propositional Theorem Proving

C. DPLL Algorithm

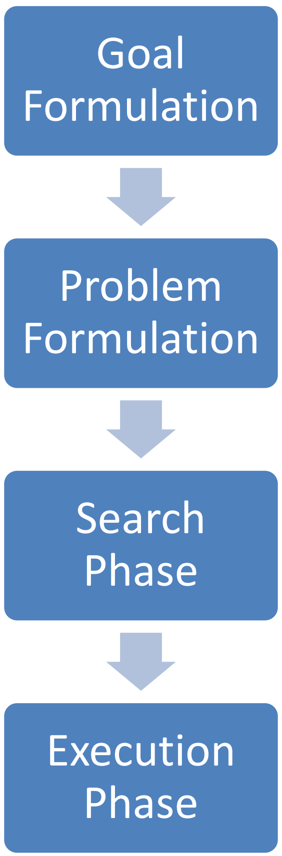
D. First Order Logic

E. FOL Inference

Problem Solving Agents

Goal based decision making agents which finds sequence of actions that leads to the desirable states.

Phases of Solution Search by PSA



Problem Solving Agents

Phases of Solution Search by PSA

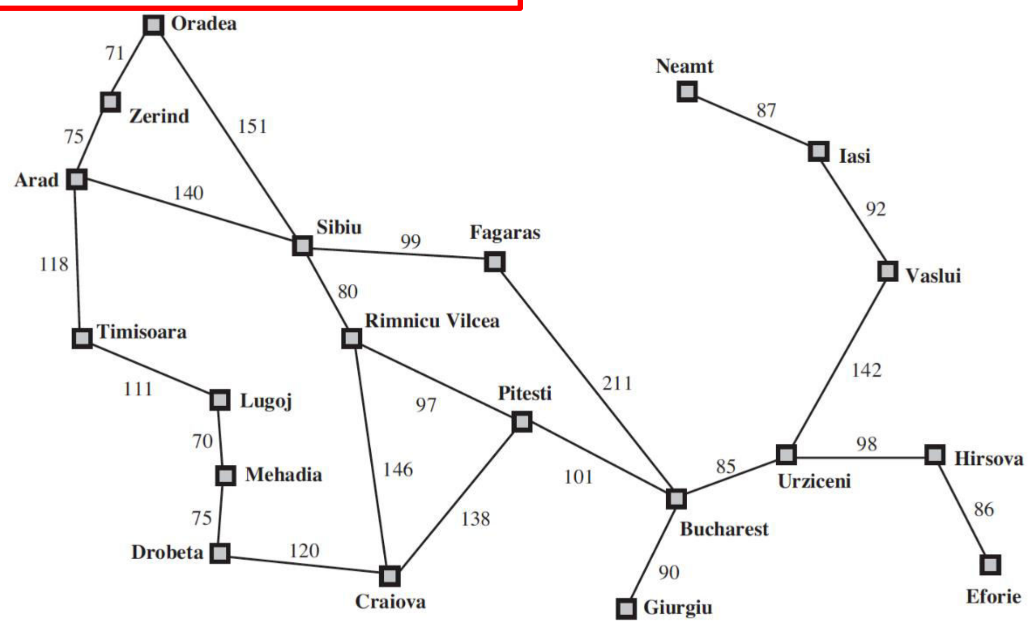
Goal Formulation

Problem Formulation

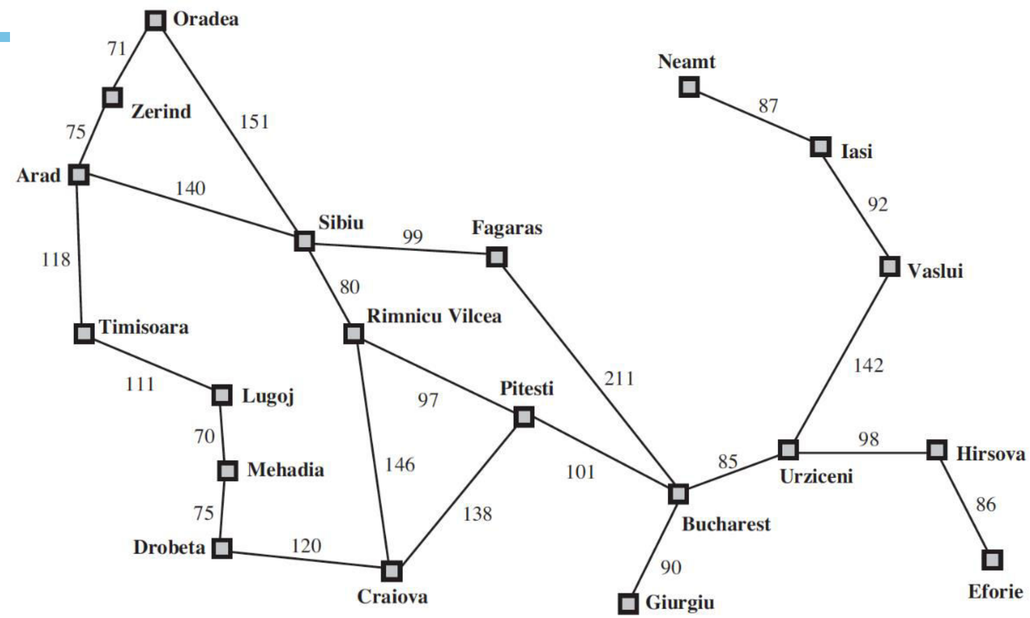
State Space Creations [in the path of Goal]
Lists the Actions

Search Phase

Execution Phase



Problem Solving Agents – Problem Formulation



Initial State –E.g., *In(Arad)*

Possible Actions – *ACTIONS(s)* ➔ *{Go(Sibiu), Go(Timisoara), Go(Zerind)}*

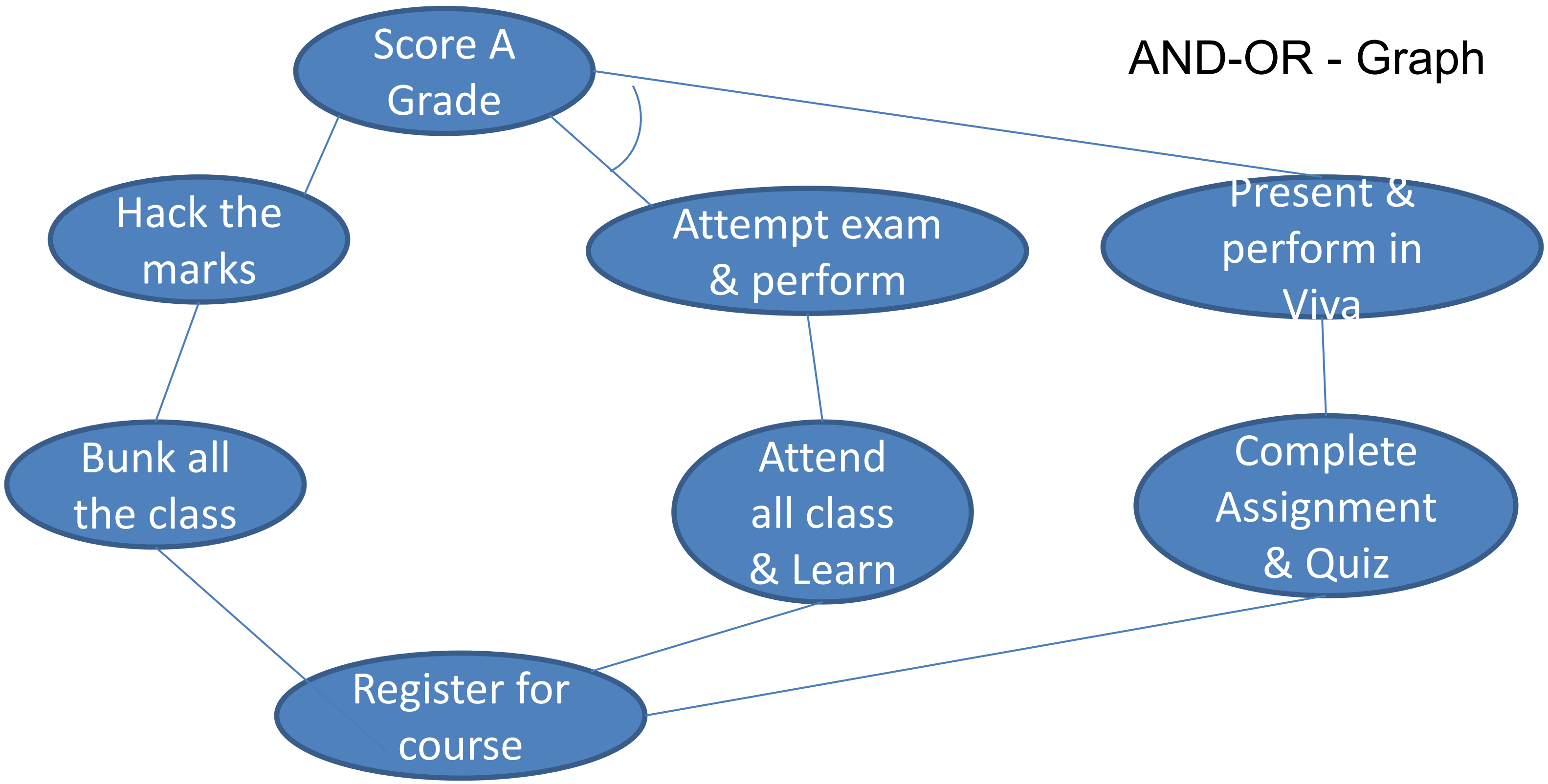
Transition Model – *RESULT(In(Arad), Go(Sibiu)) = In(Sibiu)*

Goal Test – *IsGoal(In(Bucharest)) = Yes*

Path Cost – *cost(In(Arad), go(Sibiu)) = 140 kms*

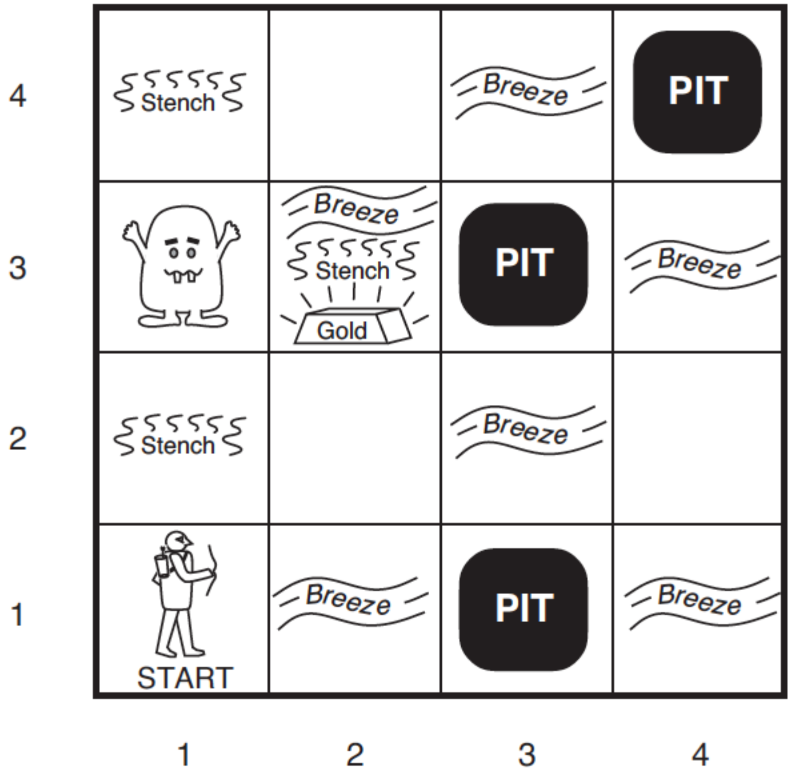
Search Graph Types

AND-OR - Graph



Knowledge based Agent : Model & Represent

Concepts, logic Representation of a sample agent



Wumpus World Problem:

PEAS:

Performance Measure:

+1000 for climbing out with gold,

-1000 for falling into a pit or being eaten by Wumpus,

-1 for each action taken and

-10 for using an arrow

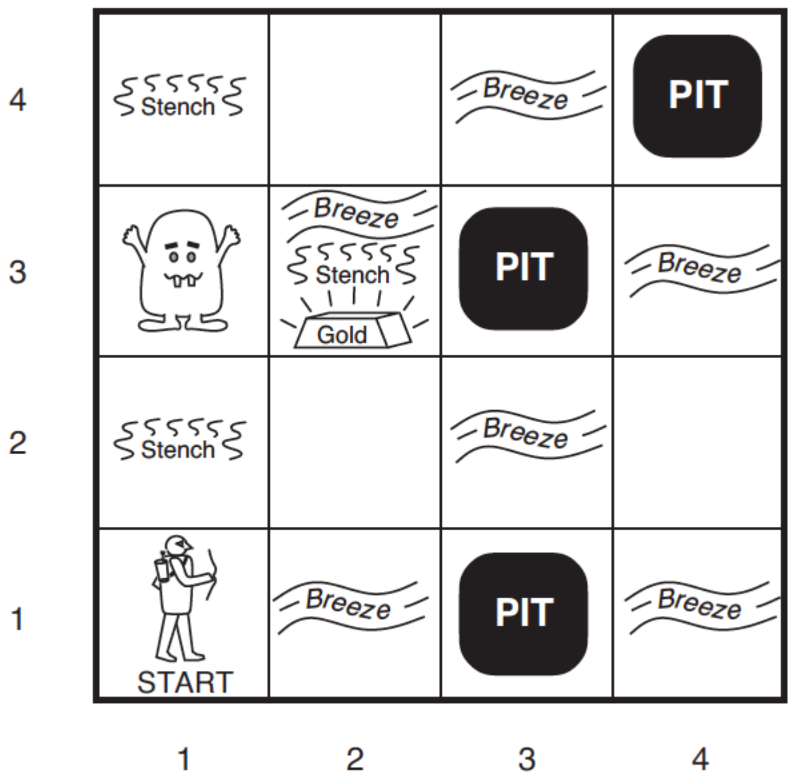
Environment: 4x4 grid of rooms. Always starts at [1, 1] facing right.

The location of Wumpus and Gold are random.

Agent dies if entered a pit or live Wumpus.

Knowledge based Agent : Model & Represent

Concepts, logic Representation of a sample agent



Wumpus World Problem:

PEAS:

Actuators –

Forward,

TurnLeft by 90,

TurnRight by 90,

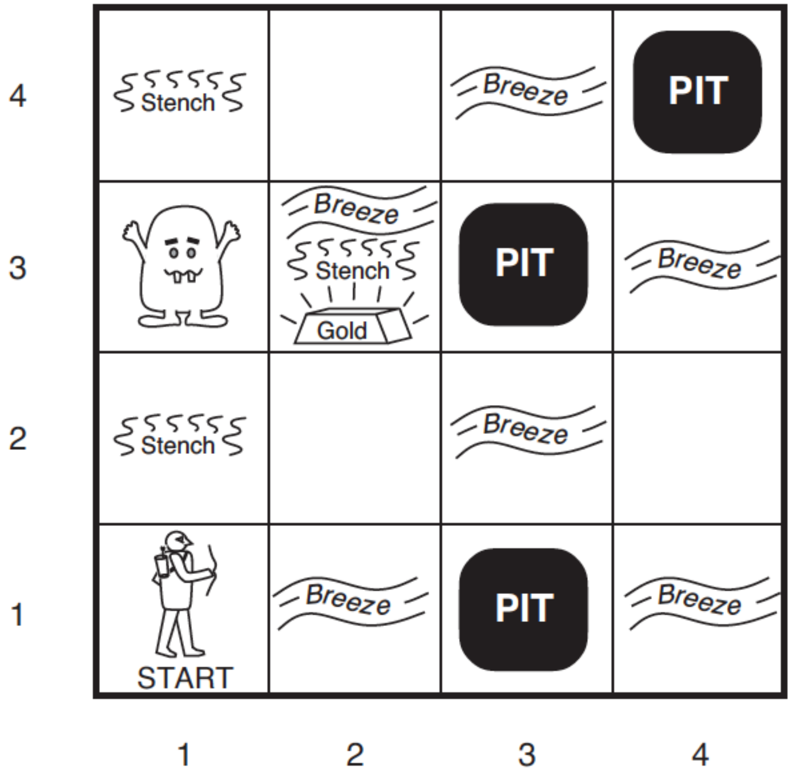
Grab – pick gold if present,

Shoot – fire an arrow, it either hits a wall or kills wumpus. Agent has only one arrow.

Climb – Used to climb out of cave, only from [1, 1]

Knowledge based Agent : Model & Represent

Concepts, logic Representation of a sample agent



Wumpus World Problem:

PEAS:

Sensors. The agent has five sensors

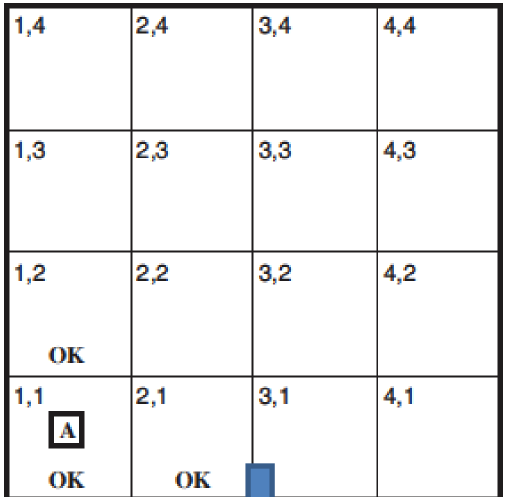
- **S**tench: In all adjacent (but not diagonal) squares of Wumpus
- **B**reeze: In all adjacent (but not diagonal) squares of a pit
- **G**litter: In the square where gold is
- **B**ump: If agent walks into a wall
- **S**cream: When Wumpus is killed, it can be perceived everywhere

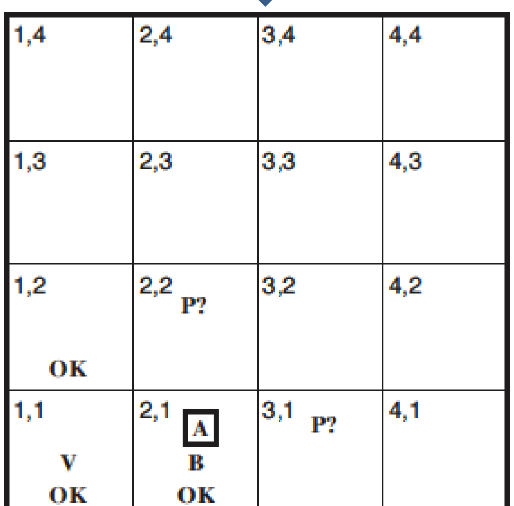
Percept Format:

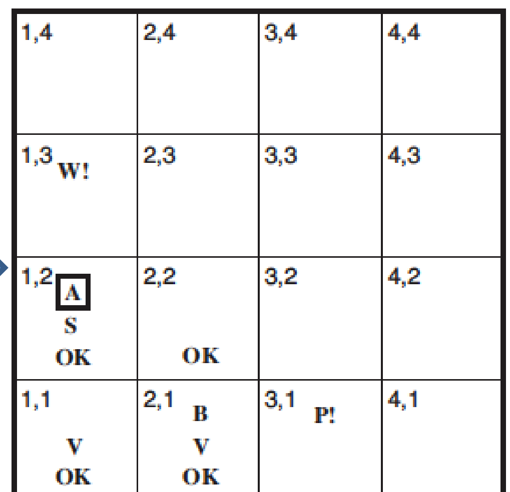
[Stench?, Breeze?, Glitter?, Bump?, Scream?]

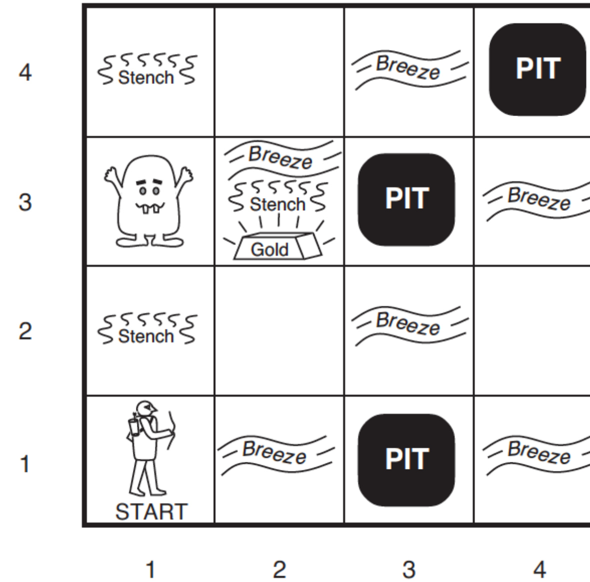
E.g., [Stench, Breeze, None, None, None]

Percept 3: [Stench, None, None, None, None]
Action: Move to [2, 2]
Remembers (2,2) as possible PIT and no Stench.



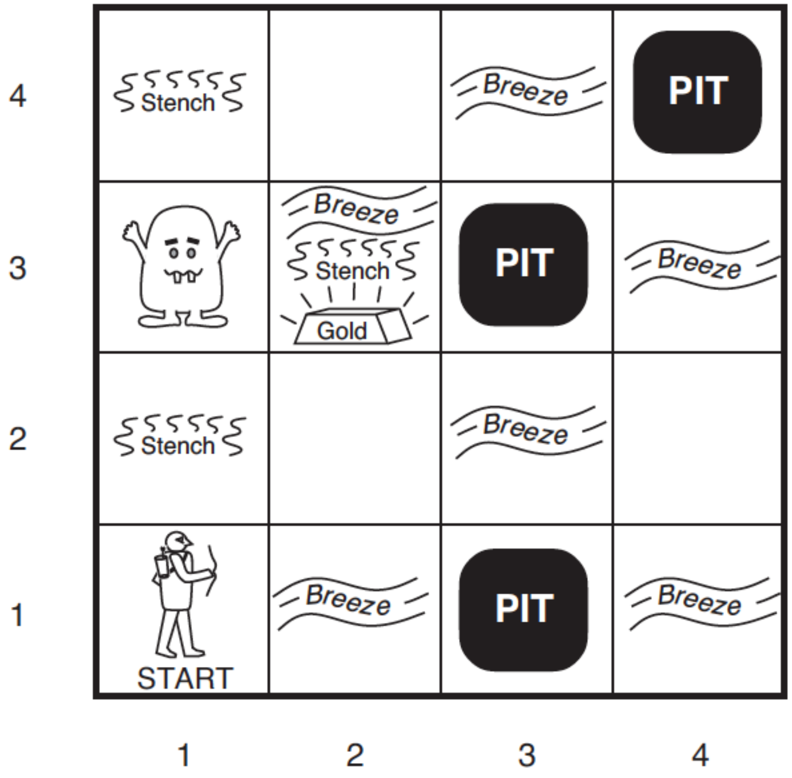






Knowledge based Agent : Model & Represent

Concepts, logic Representation of a sample agent



Why do we need Factored representation

- To reason about steps
- To learn new knowledge about the environment
- To adapt to changes to the existing knowledge
- Accept new tasks in the form of explicit goals
- To overcome partial observability of environment

Representation

Agents based on Propositional logic ,TT-Entail for inference from truth table

Syntax

Semantics

Model

Entailment

$X \models Y$

Logic

Propositional Logic

Predicate Logic

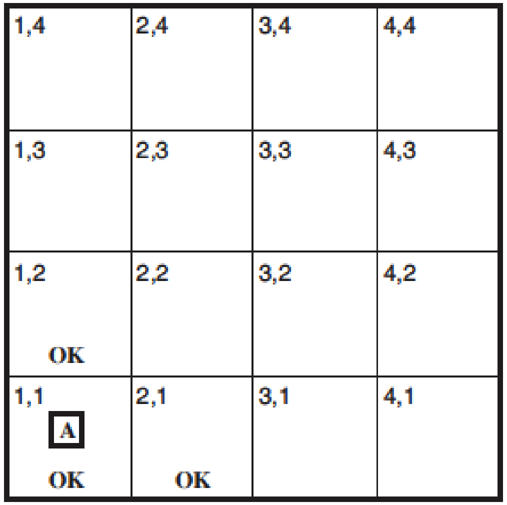
Propositional Logic

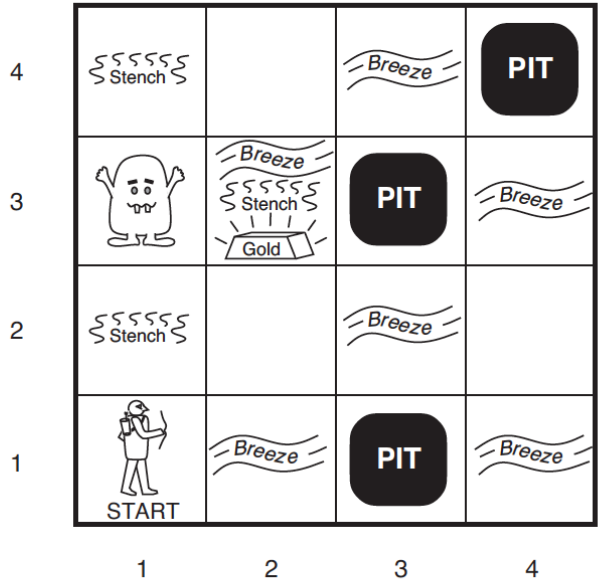
Agents based on Propositional logic ,TT-Entail for inference from truth table

A simple representation language for building knowledge-based agents

Proposition Symbol – A symbol that stands for a proposition.
E.g., W1,3 – "Wumpus in [1,3]" is a proposition and $W_{1,3}$ is the symbol
Proposition can be true or false

Atomic : $W_{1,3}$
Conjuncts : $W_{1,3} \wedge P_{3,1}$
Disjuncts : $W_{1,3} \vee P_{3,1}$
Implications :
$(W_{1,3} \wedge P_{3,1}) \Rightarrow \neg W_{2,2}$
Biconditional : $W_{1,3} \Leftrightarrow \neg W_{2,2}$





Propositional Logic

Agents based on Propositional logic ,TT-Entail for inference from truth table

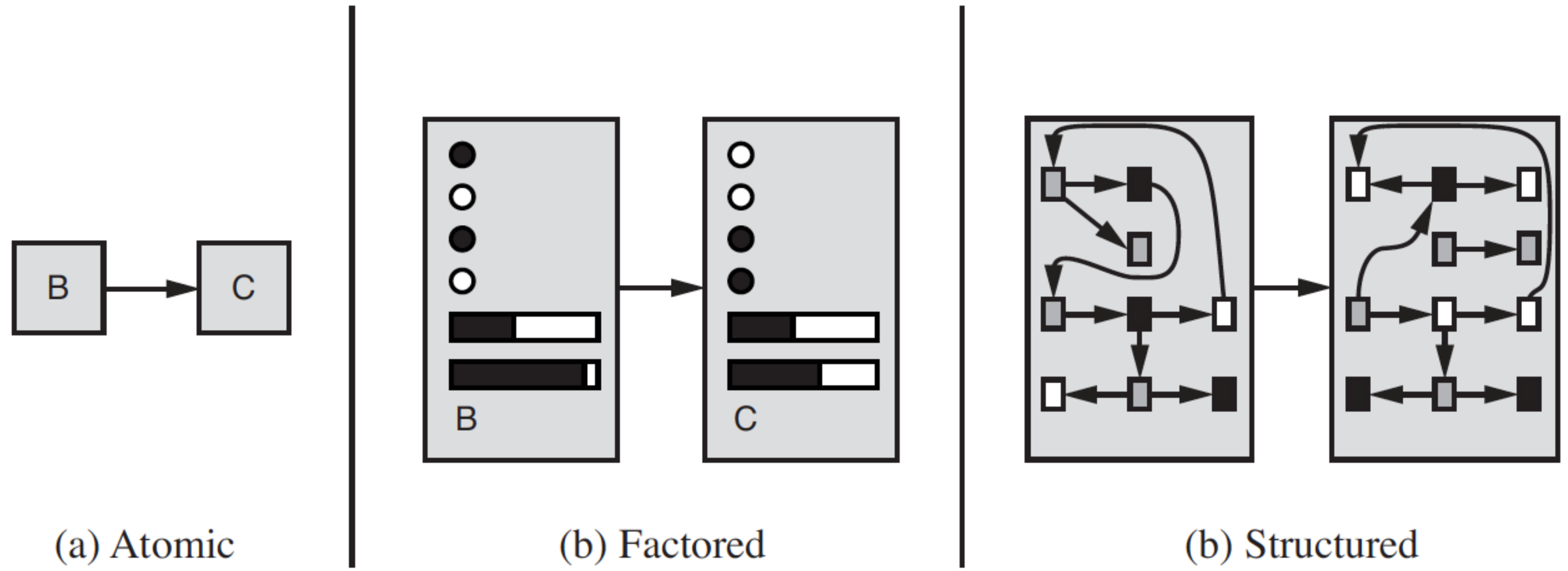
Tie break in search:

¬ , ∧ , ∨, ⇒ , ⇔

$(\neg A) \wedge B$ has precedence over $\neg (A \wedge B)$

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Predicate Logic



Search Strategies | Propositional Logic | First Order Logic

Predicate Logic

Squares neighboring the wumpus are smelly

Objects: squares, wumpus

Unary Relation (properties of an object): smelly

N-ary Relation (between objects): neighboring

Function: -

Primary difference between propositional and first-order logic lies in “ontological commitment” – the assumption about the nature of reality.

Predicate Logic – Sample Modelling

1. "Squares neighboring the wumpus are smelly"

$\forall x,y\ Neighbour(x,y) \wedge Wumpus(y) \Longrightarrow Smelly(x)$

2. "Everybody loves somebody"

$\forall x\ \exists y\ Loves(x, y)$

3. "There is someone who is loved by everyone"

$\exists y\ \forall x\ Loves(x, y)$

Order of quantifiers is important

Example: Propositional Logic

Student likes maths course and likes interesting courses

Student like datamining course

If student likes math course then they like statistics course

If student likes statistics and datamining then student is good in data analysis

Next Session Plan

- Theorem proving using Propositional Logic (Refresh the pre-reading material that was shared last week)
- DPLL algorithm
- Chaining (Forward & backward in predicate logic)
- Introduction to the Bayesian Network (Refresh the basics of probability theory)