# IS-ZC444: Artificial Intelligence

Lecture-11: Constraint Satisfaction, Logical Agents

**Dr. Kamlesh Tiwari**
Assistant Professor
Department of Computer Science and Information Systems,
BITS Pilani, Pilani, Jhunjhunu-333031, Rajasthan, INDIA

Nov 07, 2020 FLIPPED (WILP @ BITS-Pilani Jul-Nov 2020)

# Preferential Constraints

- Many real world CSPs include **preference constraints**
- Indicating some solution are preferred over other
- Consider university class scheduling problem
  - Apart from absolute constraints such as no professor could simultaneously teach two classes
  - There are some preferential constraints such as Prof. A prefer teaching in morning whereas Prof. B prefer teaching on evening.
  - A solution that schedules Prof. A in evening and Prof. B in morning is still ok
  - But, we do not prefer it
- Such problems are sometimes called **constraint optimization problem** (COP)

# Inference in CSP

- In CSP, an algorithm can either
  1. Search or
  2. Do inference: constraint propagation (that reduces number of legal values for another variable)

Enforcing **local consistency** in each part of the graph can cause inconsistency elimination throughout the graph.

- **Node consistency:** if all the values in variable's domain satisfy the variable's unary constraints [1]. It is always possible to eliminate all unary constraints by applying *Node consistency*
- **Arc consistency:** $X_i$ is arc consistent with $X_j$ if for every value in current domain $D_i$ there is some value in $D_j$ satisfying binary constraint on $(X_i, X_j)$. Note [2]

[1] If SA do not like green color then use {red,blue} instead of {red,green,blue}

[2] Generalization is possible: using more than two variables in a constraint

# AC-3 Algorithm

**function** AC-3( *csp*) **returns** false if an inconsistency is found and true otherwise
  **inputs**: *csp*, a binary CSP with components $(X, D, C)$
  **local variables**: *queue*, a queue of arcs, initially all the arcs in *csp*

```
while queue is not empty do
   (X_i, X_j) ← REMOVE-FIRST(queue)
   if REVISE(csp, X_i, X_j) then
      if size of D_i = 0 then return false
      for each X_k in X_i.NEIGHBORS - {X_j} do
         add (X_k, X_i) to queue
return true
```

---

**function** REVISE( *csp*, $X_i$, $X_j$) **returns** true iff we revise the domain of $X_i$

```
revised ← false
for each x in D_i do
   if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j then
      delete x from D_i
      revised ← true
return revised
```

Takes $O(cd^3)$ time in worst case

# Inference in CSP: Path consistency

Arc consistency can help if some domain becomes empty, or size of every domain reduces to 1

- **Path consistency:** two variable set $\{X_i, X_j\}$ is path consistent wrt $X_m$ if for every $\{X_i = a, X_j = b\}$ consistent with constraints on $\{X_i, X_j\}$ there is an assignment to $X_m$ that satisfies constraints on $\{X_i, X_m\}$ and $\{X_m, X_j\}$
- **K-consistency:** for any set of $k-1$ variables and for any constraint assignment to those variables, a consistent value can always be assigned to any $k^{th}$ variable.
  A CSP is strongly $k$-consistent is it is $k$-consistent, $k$-1-consistent, $k$-2-consistent, ..., 1-consistent [3]
- **Global Constraints:** like *alldiff*. If all $m$ variable involved in alldiff have only $n$ possible values where $m > n$ then there is no solution.

[3] Finding such considition is hard

# Example: 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 | | |

(a)

# Example: 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 | | |

(a)

| | 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 |

(b)

# Example: 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 | | |

(a)

| | 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 |

(b)

$$alldiff\{A1, A2, A3, A4, A5, A6, A7, A8, A9\}$$
$$alldiff\{B1, B2, B3, B4, B5, B6, B7, B8, B9\}$$
$$\vdots$$
$$alldiff\{A1, A2, A3, B1, B2, B3, C1, C2, C3\}$$
$$\vdots$$

# Backtracking Search for CSP

- When inference only do not work, use search
- CSP with $n$ variable and $d$ domain size can have branching factor $nd$ at top level. Then $(n-1)d$ in next level and so on.
- Tree with $!n.d^n$ leaves get generated (however valid assignments are only $d^n$)
- It is why we have ignored **commutativity** [4]
- So consider single variable at a node.
- Now we need to **backtrack**, if no legal value is left for assignment

[4] No effect of assignment order

# Backtracking Search

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

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
     if value is consistent with assignment then
        add {var = value} to assignment
        inferences ← INFERENCE(csp, var, value)
        if inferences ≠ failure then
           add inferences to assignment
           result ← BACKTRACK(assignment, csp)
           if result ≠ failure then
              return result
     remove {var = value} and inferences from assignment
  return failure
```

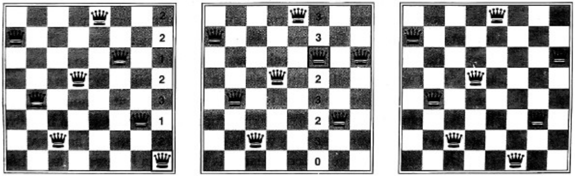
# Backtracking Search

- Which variable to choose next? **minimum remaining value** [5] or "fail first"
- **Degree heuristic**, choose one which is involved in many constraints
- Which value to choose? **least constrained value**
- **Forward checking**: interleaving search and inference would help. Whenever a variable $X$ is assigned, forward checking establishes arc consistency for it.
- **Intelligent Backtracking:** Some time it is needed to backtrack upward more than a single step to resolve the inconsistency. Conflicting set is used to find most suitable node.

[5] chose whose domain have fewer entries

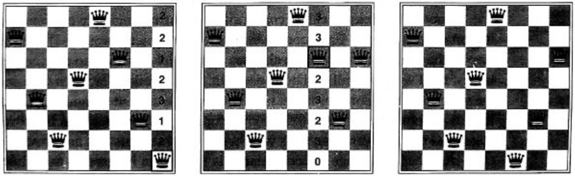
# Local Search for CSP

Complete state space formulation can also be used for search



# Local Search for CSP

Complete state space formulation can also be used for search



```
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
  inputs: csp, a constraint satisfaction problem
          max_steps, the number of steps allowed before giving up

  current ← an initial complete assignment for csp
  for i = 1 to max_steps do
     if current is a solution for csp then return current
     var ← a randomly chosen conflicted variable from csp.VARIABLES
     value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
     set var = value in current
  return failure
```

# Local Search for CSP

- **Min-conflict:** can solve Million Queen problem in 50 steps on an average
- **Tabu-search:** keeps a small list of recently visited states and forbidding algorithm to return to these states.
- **Constraint weighting:** starting from weight 1 for each variable; in each step, algorithm choses variable/value such that sum of weights is minimized. Weights of violated variables are incremented.
- Big advantage that the local search can be implemented in online environment.[6]

[6] Airline wants to repair its schedule with minimum changes.

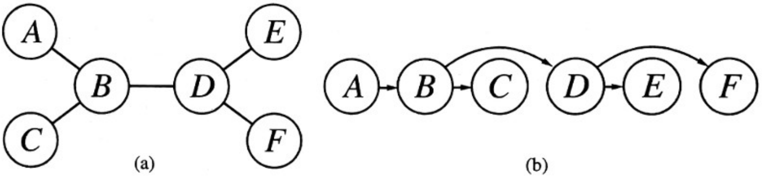
# Exploit Problem Structure

Sometime structure of the problem could help to find solution

- **Independent Subproblems:** see that Tasmania is not connected to mainland in Australia map.

  $$\text{Compare } O(d^c n/c) \text{ with } O(d^n) \text{ it is linear}$$

  where each sub problem has $c$ variables
- **Tree structured CSP** is solvable in linear time $O(nd^2)$ with topological sort

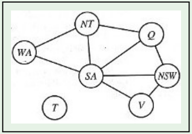


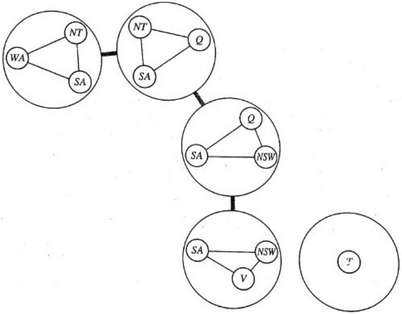

  - **Cut set conditioning:** Assign few, to get tree from remaining vars $O(d^c(n-c)d^2)$
  - Another approach is **tree decomposition**

# Tree Decomposition for CSP

## Divide the problem in sub-problems

- Every variable in original problem appears at least one of the subproblems
- If two variables are connected by a constraint in original problem, then they must appear together in at least one of the subproblem
- If a variable appears in two subproblems in the tree, it must appear in every subproblem along the path connecting those subproblems



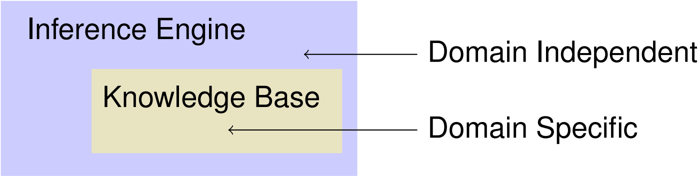
# Logical Agents

- Humans, it seems, know things
- And what they know, helps them to think/reason
- Knowledge base is set of **sentences**[7]

[7] Sentences are derived from knowledge representation language. An underived sentence is **axiom**

# Logical Agents

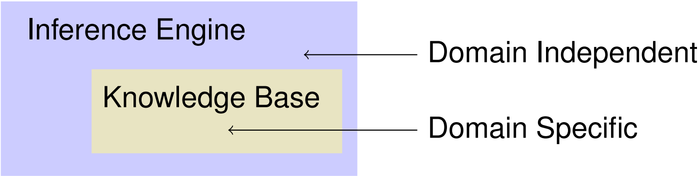
- Humans, it seems, know things
- And what they know, helps them to think/reason
- Knowledge base is set of **sentences**[7]
- Knowledge-based agents




[7] Sentences are derived from knowledge representation language. An underived sentence is **axiom**

# Logical Agents

- Humans, it seems, know things
- And what they know, helps them to think/reason
- Knowledge base is set of **sentences**[7]
- Knowledge-based agents




- Agent program does three operation
  1. **TELL** the knowledge base what it have perceived
  2. **ASK** what to do
  3. **TELL** which action is chosen and the agent execute that action

[7] Sentences are derived from knowledge representation language. An underived sentence is **axiom**

# A Generic Knowledge Based Agent

**Algorithm 1:** KB-Agent(percept)

**Input**: *KB* is knowledge base *t* a counter indicating time

TELL(*KB*, Make-Percept-Sentence(percept,*t*))
*action* $\leftarrow$ ASK(*KB*, Make-Action-Query(*t*))
TELL(*KB*, Make-Action-Sentence(*action*,*t*))
$t \leftarrow t + 1$
**return** *action*

# A Generic Knowledge Based Agent

**Algorithm 2:** KB-Agent(percept)

**Input**: *KB* is knowledge base *t* a counter indicating time

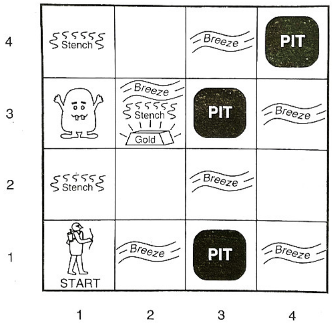
TELL(*KB*, Make-Percept-Sentence(percept,*t*))
*action* $\leftarrow$ ASK(*KB*, Make-Action-Query(*t*))
TELL(*KB*, Make-Action-Sentence(*action*,*t*))
*t* $\leftarrow$ *t* +1
**return** *action*

## Knowledge base could be built through

- **Declarative** procedure that tell everything
- **Procedural** approach that writes a program code

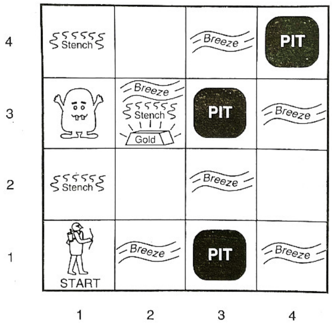
# Example: Wumpus World

- **Performance** gold +100, death -100, step -1, arrow -10
- **Environment** smell around wumpus, breeze around pit
- **Actuator** turn left/right, forward, grab, release, shoot
- **Sensor** breeze, glitter, smell, bump, scream



# Example: Wumpus World

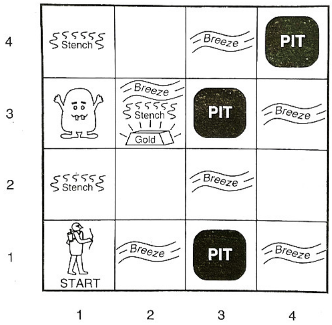
- **Performance** gold +100, death -100, step -1, arrow -10
- **Environment** smell around wumpus, breeze around pit
- **Actuator** turn left/right, forward, grab, release, shoot
- **Sensor** breeze, glitter, smell, bump, scream




| Observable? | No |
|---|---|

# Example: Wumpus World

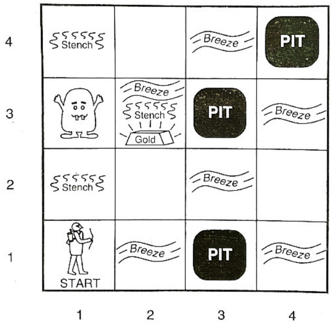
- **Performance** gold +100, death -100, step -1, arrow -10
- **Environment** smell around wumpus, breeze around pit
- **Actuator** turn left/right, forward, grab, release, shoot
- **Sensor** breeze, glitter, smell, bump, scream




| | |
|---|---|
| Observable? | No |
| Deterministic? | Yes |

# Example: Wumpus World

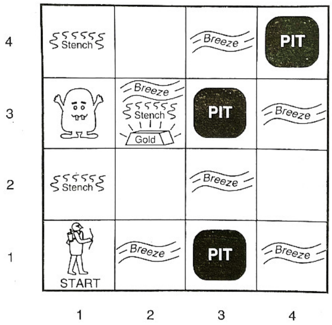
- **Performance** gold +100, death -100, step -1, arrow -10
- **Environment** smell around wumpus, breeze around pit
- **Actuator** turn left/right, forward, grab, release, shoot
- **Sensor** breeze, glitter, smell, bump, scream




| | |
|---|---|
| Observable? | No |
| Deterministic? | Yes |
| Episodic? | No |

# Example: Wumpus World

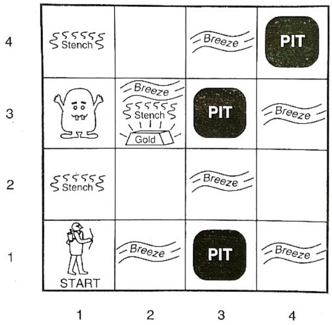
- **Performance** gold +100, death -100, step -1, arrow -10
- **Environment** smell around wumpus, breeze around pit
- **Actuator** turn left/right, forward, grab, release, shoot
- **Sensor** breeze, glitter, smell, bump, scream




| | |
|---|---|
| Observable? | No |
| Deterministic? | Yes |
| Episodic? | No |
| Static? | Yes |

# Example: Wumpus World

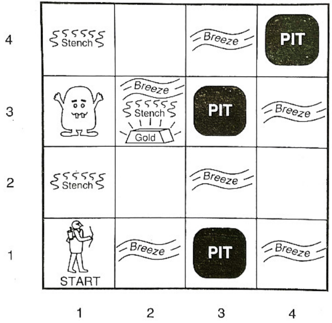
- **Performance** gold +100, death -100, step -1, arrow -10
- **Environment** smell around wumpus, breeze around pit
- **Actuator** turn left/right, forward, grab, release, shoot
- **Sensor** breeze, glitter, smell, bump, scream




| | |
|---|---|
| Observable? | No |
| Deterministic? | Yes |
| Episodic? | No |
| Static? | Yes |
| Discrete? | Yes |

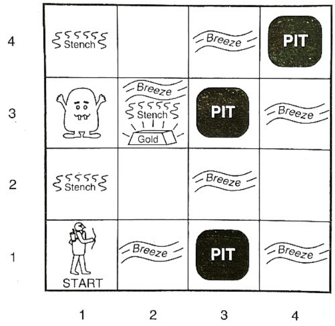
# Example: Wumpus World

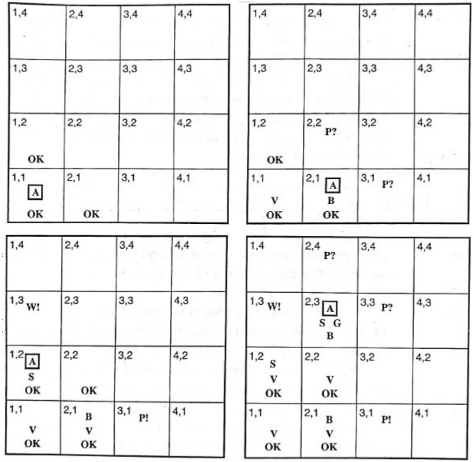
- **Performance** gold +100, death -100, step -1, arrow -10
- **Environment** smell around wumpus, breeze around pit
- **Actuator** turn left/right, forward, grab, release, shoot
- **Sensor** breeze, glitter, smell, bump, scream




| | |
|---|---|
| Observable? | No |
| Deterministic? | Yes |
| Episodic? | No |
| Static? | Yes |
| Discrete? | Yes |
| Single Agent? | Yes |

# Wumpus World



# Logic

- Knowledge base contains sentences that are written according to some syntax that specified whether they are well formed or not.

# Logic

- Knowledge base contains sentences that are written according to some syntax that specified whether they are well formed or not.
- Sementics defines truth of each sentence. $x + y = 4$ is
  - **true** if $x = 1$ and $y = 3$ but
  - **not true** when $x = 2$ and $y = 3$

# Logic

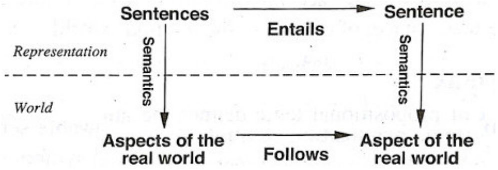
- Knowledge base contains sentences that are written according to some syntax that specified whether they are well formed or not.
- Sementics defines truth of each sentence. $x + y = 4$ is
  - **true** if $x = 1$ and $y = 3$ but
  - **not true** when $x = 2$ and $y = 3$
- Here $x + y = 4$ is a model that is satisfied by some sentence $\alpha$
- We use $M(\alpha)$ to represent a set of all models of $\alpha$

# Logic

- Knowledge base contains sentences that are written according to some syntax that specified whether they are well formed or not.
- Sementics defines truth of each sentence. $x + y = 4$ is
  - **true** if $x = 1$ and $y = 3$ but
  - **not true** when $x = 2$ and $y = 3$
- Here $x + y = 4$ is a model that is satisfied by some sentence $\alpha$
- We use $M(\alpha)$ to represent a set of all models of $\alpha$
- Logical entailment between sentences means that a sentence logically follows from another one $\alpha \models \beta$ it means $M(\alpha) \subseteq M(\beta)$

# Logic

- Knowledge base contains sentences that are written according to some syntax that specified whether they are well formed or not.
- Sementics defines truth of each sentence. $x + y = 4$ is
  - **true** if $x = 1$ and $y = 3$ but
  - **not true** when $x = 2$ and $y = 3$
- Here $x + y = 4$ is a model that is satisfied by some sentence $\alpha$
- We use $M(\alpha)$ to represent a set of all models of $\alpha$
- Logical entailment between sentences means that a sentence logically follows from another one $\alpha \models \beta$ it means $M(\alpha) \subseteq M(\beta)$
- Relationship between representation and real world



# Propositional Logic

- **Propositions** or **declarative sentences** can be true or false
  - Sum of 5 and 4 is 9
  - Could you give me your pen
  - Every even natural number grater then two can be written as sum of two primes
  - Best of luck

# Propositional Logic

- **Propositions** or **declarative sentences** can be true or false
  - Sum of 5 and 4 is 9
  - Could you give me your pen
  - Every even natural number grater then two can be written as sum of two primes
  - Best of luck
- Sometime it is better to assign symbols to a atomic sentences
  $p$: "I won a lottery last week"
  $q$: "I have purchased a lottery ticket last week"

# Propositional Logic

- **Propositions** or **declarative sentences** can be true or false
  - Sum of 5 and 4 is 9
  - Could you give me your pen
  - Every even natural number grater then two can be written as sum of two primes
  - Best of luck
- Sometime it is better to assign symbols to a atomic sentences
  $p$: "I won a lottery last week"
  $q$: "I have purchased a lottery ticket last week"
- Complex sentences could be formed by using

| | | |
|---|---|---|
| $\neg$ | : | negative |
| $\vee$ | : | disjunction, at least one is true |
| $\wedge$ | : | conjunction, both should be true |
| $\rightarrow$ | : | implication |

# Truth Table and Natural Deduction

| $P$ | $Q$ | $\neg P$ | $P \wedge Q$ | $P \vee Q$ | $P \rightarrow Q$ | $P \Leftrightarrow Q$ |
|---|---|---|---|---|---|---|
| T | T | F | T | T | T | T |
| T | F | F | F | T | F | F |
| F | T | T | F | T | T | F |
| F | F | T | F | F | T | T |

# Truth Table and Natural Deduction

| $P$ | $Q$ | $\neg P$ | $P \wedge Q$ | $P \vee Q$ | $P \to Q$ | $P \Leftrightarrow Q$ |
|---|---|---|---|---|---|---|
| T | T | F | T | T | T | T |
| T | F | F | F | T | F | F |
| F | T | T | F | T | T | F |
| F | F | T | F | F | T | T |

## Natural Deduction

Suppose we have formulas $\phi_1$, $\phi_2$, ..., $\phi_n$ and we have applied some proof rules to get another formula $\psi$ then we denote

$$\phi_1, \phi_2, ..., \phi_n \vdash \psi$$

This equation is called sequent; and is valid if a proof can be found

# Rules for Conjunction

$$\frac{\phi \qquad \psi}{\phi \wedge \psi} \wedge \mathrm{i} \qquad\qquad \frac{\phi \wedge \psi}{\phi} \wedge e_1 \qquad\qquad \frac{\phi \wedge \psi}{\psi} \wedge e_2$$

# Rules for Conjunction

$$\frac{\phi \qquad \psi}{\phi \wedge \psi} \wedge \mathrm{i} \qquad\qquad \frac{\phi \wedge \psi}{\phi} \wedge e_1 \qquad\qquad \frac{\phi \wedge \psi}{\psi} \wedge e_2$$

**Show $p \wedge q, r \vdash q \wedge r$**

# Rules for Conjunction

$$\frac{\phi \qquad \psi}{\phi \wedge \psi} \wedge\text{i} \qquad \frac{\phi \wedge \psi}{\phi} \wedge e_1 \qquad \frac{\phi \wedge \psi}{\psi} \wedge e_2$$

**Show $p \wedge q, r \vdash q \wedge r$**

| | | |
|---|---|---|
| 1 | $p \wedge q$ | premise |
| 2 | $r$ | premise |
| 3 | $q$ | $\wedge e_2$ 1 |
| 4 | $q \wedge r$ | $\wedge i$ 3,2 |

# Rules for Conjunction

$$\frac{\phi \quad \psi}{\phi \wedge \psi} \wedge \mathrm{i} \qquad \frac{\phi \wedge \psi}{\phi} \wedge e_1 \qquad \frac{\phi \wedge \psi}{\psi} \wedge e_2$$

**Show $p \wedge q, r \vdash q \wedge r$**

| | | |
|---|---|---|
| 1 | $p \wedge q$ | premise |
| 2 | $r$ | premise |
| 3 | $q$ | $\wedge e_2$ 1 |
| 4 | $q \wedge r$ | $\wedge i$ 3,2 |

- Show $(p \wedge q) \wedge r, s \wedge t \vdash q \wedge s$

# Rules for Double Negation

Introduction $$\frac{\phi}{\neg\neg\phi}\ \neg\neg i$$

Elimination $$\frac{\neg\neg\phi}{\phi}\ \neg\neg e$$

# Rules for Double Negation

Introduction $\dfrac{\phi}{\neg\neg\phi}\ \neg\neg i$

Elimination $\dfrac{\neg\neg\phi}{\phi}\ \neg\neg e$

$p, \neg\neg(q \wedge r) \vdash \neg\neg p \wedge r$

# Rules for Double Negation

Introduction $\dfrac{\phi}{\neg\neg\phi}\ \neg\neg i$ Elimination $\dfrac{\neg\neg\phi}{\phi}\ \neg\neg e$

**$p, \neg\neg(q \wedge r) \vdash \neg\neg p \wedge r$**

| | | |
|---|---|---|
| 1 | $p$ | premise |
| 2 | $\neg\neg(q \wedge r)$ | premise |
| 3 | $\neg\neg p$ | $\neg\neg i$ 1 |
| 4 | $q \wedge r$ | $\neg\neg e$ 2 |
| 5 | $r$ | $\wedge e_2$ 4 |
| 6 | $\neg\neg p \wedge r$ | $\wedge i$ 3,5 |

# Rules for Implication Elimination

Elimination $$\frac{\phi \qquad \phi \to \psi}{\psi} \to e$$

# Rules for Implication Elimination

Elimination $\dfrac{\phi \quad \phi \rightarrow \psi}{\psi} \rightarrow e$

**Can $p, p \rightarrow q, p \rightarrow (q \rightarrow r)$ infer $r$**

# Rules for Implication Elimination

$$\text{Elimination} \quad \frac{\phi \qquad \phi \to \psi}{\psi} \to e$$

## Can $p, p \to q, p \to (q \to r)$ infer $r$

| | | |
|---|---|---|
| 1 | $p \to (q \to r)$ | premise |
| 2 | $p \to q$ | premise |
| 3 | $p$ | premise |
| 4 | $q \to r$ | $\to e$ 1,3 |
| 5 | $q$ | $\to e$ 2,3 |
| 6 | $r$ | $\to e$ 4,5 |

# Rules for Implication Elimination

Elimination $\dfrac{\phi \quad \phi \to \psi}{\psi} \to e$

### Can $p, p \to q, p \to (q \to r)$ infer $r$

| | | |
|---|---|---|
| 1 | $p \to (q \to r)$ | premise |
| 2 | $p \to q$ | premise |
| 3 | $p$ | premise |
| 4 | $q \to r$ | $\to e$ 1,3 |
| 5 | $q$ | $\to e$ 2,3 |
| 6 | $r$ | $\to e$ 4,5 |

### Modus Tollens

$$\frac{\phi \to \psi \qquad \neg\psi}{\neg\phi} \; MT$$

# Rules for Implication Elimination

Elimination $\dfrac{\phi \qquad \phi \rightarrow \psi}{\psi} \rightarrow e$

## Can $p, p \rightarrow q, p \rightarrow (q \rightarrow r)$ infer $r$

| | | |
|---|---|---|
| 1 | $p \rightarrow (q \rightarrow r)$ | premise |
| 2 | $p \rightarrow q$ | premise |
| 3 | $p$ | premise |
| 4 | $q \rightarrow r$ | $\rightarrow e$ 1,3 |
| 5 | $q$ | $\rightarrow e$ 2,3 |
| 6 | $r$ | $\rightarrow e$ 4,5 |

## Modus Tollens

$$\frac{\phi \rightarrow \psi \qquad \neg\psi}{\neg\phi} \; MT$$

Show: $p \rightarrow (q \rightarrow r), p, \neg r \vdash \neg q$

# Rules for Implication Introduction

$$\frac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \psi\end{array}}}{\phi \to \psi} \to \mathrm{i}$$

# Rules for Implication Introduction

$$\frac{\boxed{\begin{matrix}\phi \\ \vdots \\ \psi\end{matrix}}}{\phi \to \psi} \to \mathrm{i}$$

**Show** $(q \to r) \to ((\neg q \to \neg p) \to (p \to r))$

# Rules for Implication Introduction

$$\frac{\boxed{\begin{array}{c}\phi \\ \vdots \\ \psi\end{array}}}{\phi \to \psi} \to \text{i}$$

**Show $(q \to r) \to ((\neg q \to \neg p) \to (p \to r))$**

| | | |
|---|---|---|
| 1. | $q \to r$ | Assumption |
| 9. | $(\neg q \to \neg r) \to (p \to r)$ | |
| 10. | $(q \to r) \to ((\neg q \to \neg p) \to (p \to r))$ | $\to$ i 1-9 |

# Rules for Implication Introduction

$$\frac{\boxed{\begin{array}{c}\phi \\ \vdots \\ \psi\end{array}}}{\phi \to \psi} \to \mathrm{i}$$

## Show $(q \to r) \to ((\neg q \to \neg p) \to (p \to r))$

| | | |
|---|---|---|
| 1. | $q \to r$ | Assumption |
| 2. | $\neg q \to \neg p$ | Assumption |
| 8. | $p \to r$ | |
| 9. | $(\neg q \to \neg r) \to (p \to r)$ | $\to$ i 2-8 |
| 10. | $(q \to r) \to ((\neg q \to \neg p) \to (p \to r))$ | $\to$ i 1-9 |

# Rules for Implication Introduction

$$\frac{\boxed{\begin{matrix}\phi\\ \vdots\\ \psi\end{matrix}}}{\phi \to \psi} \to \mathrm{i}$$

**Show $(q \to r) \to ((\neg q \to \neg p) \to (p \to r))$**

| | | |
|---|---|---|
| 1. | $q \to r$ | Assumption |
| 2. | $\neg q \to \neg p$ | Assumption |
| 3. | $p$ | |
| | $r$ | |
| 8. | $p \to r$ | |
| 9. | $(\neg q \to \neg r) \to (p \to r)$ | $\to$ i 2-8 |
| 10. | $(q \to r) \to ((\neg q \to \neg p) \to (p \to r))$ | $\to$ i 1-9 |

# Rules for Implication Introduction

$$\frac{\boxed{\begin{array}{c}\phi \\ \vdots \\ \psi\end{array}}}{\phi \to \psi} \to \text{i}$$

## Show $(q \to r) \to ((\neg q \to \neg p) \to (p \to r))$

| | | | |
|---|---|---|---|
| 1. | $q \to r$ | | Assumption |
| 2. | $\neg q \to \neg p$ | | Assumption |
| 3. | $p$ | | Assumption |
| 4. 5. | $\neg\neg p,\ \neg\neg q$ | $\neg\neg i$ 3, $MT$ 2,4 | |
| 6. | $q$ | $\neg\neg e$ 5 | |
| 7. | $r$ | | $\to$ e 1-6 |
| 8. | $p \to r$ | | $\to$ i 3-7 |
| 9. | $(\neg q \to \neg r) \to (p \to r)$ | | $\to$ i 2-8 |
| 10. | $(q \to r) \to ((\neg q \to \neg p) \to (p \to r))$ | | $\to$ i 1-9 |

# Examples

Show following

1. $p \wedge q \rightarrow r \vdash p \rightarrow (q \rightarrow r)$

# Examples

Show following

1. $p \wedge q \rightarrow r \vdash p \rightarrow (q \rightarrow r)$
2. $(p \rightarrow (q \rightarrow r)) \rightarrow ((p \wedge q) \rightarrow r)$

# Examples

Show following

1. $p \wedge q \to r \vdash p \to (q \to r)$
2. $(p \to (q \to r)) \to ((p \wedge q) \to r)$
3. $p \to q \vdash (p \wedge r) \to (q \wedge r)$

# Recap: Propositional Logic

**Propositions** or **declarative sentences** can be true or false. Complex sentences could be formed by using $\neg$, $\wedge$, $\vee$ $\rightarrow$

# Recap: Propositional Logic

**Propositions** or **declarative sentences** can be true or false. Complex sentences could be formed by using $\neg$, $\wedge$, $\vee$ $\rightarrow$

**Natural Deduction** applies proof rules on sentences $\phi_1$, $\phi_2$, ..., $\phi_n$ to get new $\psi$; we denote it as $\phi_1, \phi_2, ..., \phi_n \vdash \psi$

# Recap: Propositional Logic

**Propositions** or **declarative sentences** can be true or false. Complex sentences could be formed by using $\neg$, $\wedge$, $\vee$ $\rightarrow$

**Natural Deduction** applies proof rules on sentences $\phi_1$, $\phi_2$, ..., $\phi_n$ to get new $\psi$; we denote it as $\phi_1, \phi_2, ..., \phi_n \vdash \psi$

$$\frac{\phi \qquad \psi}{\phi \wedge \psi} \wedge \mathrm{i} \qquad \frac{\phi \wedge \psi}{\phi} \wedge e_1 \qquad \frac{\phi \wedge \psi}{\psi} \wedge e_2$$

$$\frac{\phi}{\neg\neg\phi} \neg\neg\,\mathrm{i} \qquad \frac{\neg\neg\phi}{\phi} \neg\neg e \qquad \frac{\phi \qquad \phi \rightarrow \psi}{\psi} \rightarrow e$$

$$\frac{\phi \rightarrow \psi \qquad \neg\psi}{\neg\phi} \; MT \qquad \frac{\boxed{\begin{array}{c}\phi \\ \vdots \\ \psi\end{array}}}{\phi \rightarrow \psi} \rightarrow \mathrm{i}$$

Box nesting is important

# Backus-Naur Form (BNF)

There is a rule (syntax) to form sentences

$$
\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \rightarrow & \mathit{True} \mid \mathit{False} \mid P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} & \rightarrow & (\ \mathit{Sentence}\ ) \mid [\ \mathit{Sentence}\ ] \\
& \mid & \neg\ \mathit{Sentence} \\
& \mid & \mathit{Sentence} \land \mathit{Sentence} \\
& \mid & \mathit{Sentence} \lor \mathit{Sentence} \\
& \mid & \mathit{Sentence} \Rightarrow \mathit{Sentence} \\
& \mid & \mathit{Sentence} \Leftrightarrow \mathit{Sentence} \\
\\
\text{OPERATOR PRECEDENCE} & : & \neg, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

# Rules for Disjunction

$$\frac{\phi}{\phi \vee \psi} \vee i_1 \qquad \frac{\psi}{\phi \vee \psi} \vee i_2 \qquad \frac{\phi \vee \psi \quad \boxed{\begin{matrix}\phi \\ \vdots \\ \chi\end{matrix}} \quad \boxed{\begin{matrix}\psi \\ \vdots \\ \chi\end{matrix}}}{\chi} \vee e$$

# Rules for Disjunction

$$\frac{\phi}{\phi \vee \psi} \vee i_1 \qquad \frac{\psi}{\phi \vee \psi} \vee i_2 \qquad \frac{\phi \vee \psi \quad \boxed{\begin{array}{c}\phi \\ \vdots \\ \chi\end{array}} \quad \boxed{\begin{array}{c}\psi \\ \vdots \\ \chi\end{array}}}{\chi} \vee e$$

**Show $p \vee q \vdash q \vee p$**

# Rules for Disjunction

$$\frac{\phi}{\phi \vee \psi} \vee i_1 \qquad \frac{\psi}{\phi \vee \psi} \vee i_2 \qquad \frac{\phi \vee \psi \quad \boxed{\begin{matrix}\phi \\ \vdots \\ \chi\end{matrix}} \quad \boxed{\begin{matrix}\psi \\ \vdots \\ \chi\end{matrix}}}{\chi} \vee e$$

**Show $p \vee q \vdash q \vee p$**

| | | |
|---|---|---|
| 1. | $p \vee q$ | Premise |

- Show $q \rightarrow r \vdash p \vee q \rightarrow p \vee r$

# Rules for Disjunction

$$\frac{\phi}{\phi \vee \psi} \vee i_1 \qquad \frac{\psi}{\phi \vee \psi} \vee i_2 \qquad \frac{\phi \vee \psi \quad \boxed{\begin{matrix}\phi \\ \vdots \\ x\end{matrix}} \quad \boxed{\begin{matrix}\psi \\ \vdots \\ x\end{matrix}}}{x} \vee e$$

## Show $p \vee q \vdash q \vee p$

| | | |
|---|---|---|
| 1. | $p \vee q$ | Premise |
| 2. | $p$ | Assumption |
| 3. | $q \vee p$ | $\vee i_2$ |

- Show $q \to r \vdash p \vee q \to p \vee r$

# Rules for Disjunction

$$\frac{\phi}{\phi \vee \psi}\ \vee i_1 \qquad\qquad \frac{\psi}{\phi \vee \psi}\ \vee i_2 \qquad\qquad \frac{\phi \vee \psi \quad \boxed{\begin{array}{c}\phi\\ \vdots\\ x\end{array}} \quad \boxed{\begin{array}{c}\psi\\ \vdots\\ x\end{array}}}{x}\ \vee e$$

## Show $p \vee q \vdash q \vee p$

| | | |
|---|---|---|
| 1. | $p \vee q$ | Premise |
| 2. | $p$ | Assumption |
| 3. | $q \vee p$ | $\vee i_2$ |
| 4. | $q$ | Assumption |
| 5. | $q \vee p$ | $\vee i_1$ |

- Show $q \rightarrow r \vdash p \vee q \rightarrow p \vee r$

# Rules for Disjunction

$$\frac{\phi}{\phi \vee \psi} \vee i_1 \qquad \frac{\psi}{\phi \vee \psi} \vee i_2 \qquad \frac{\phi \vee \psi \quad \boxed{\begin{matrix}\phi \\ \vdots \\ x\end{matrix}} \quad \boxed{\begin{matrix}\psi \\ \vdots \\ x\end{matrix}}}{x} \vee e$$

## Show $p \vee q \vdash q \vee p$

| | | |
|---|---|---|
| 1. | $p \vee q$ | Premise |
| 2. | $p$ | Assumption |
| 3. | $q \vee p$ | $\vee i_2$ |
| 4. | $q$ | Assumption |
| 5. | $q \vee p$ | $\vee i_1$ |
| 6. | $q \vee p$ | $\vee e$ 1, 3, 5 |

# Rules for Disjunction

$$\frac{\phi}{\phi \vee \psi} \vee i_1 \qquad \frac{\psi}{\phi \vee \psi} \vee i_2 \qquad \frac{\phi \vee \psi \quad \boxed{\begin{matrix}\phi \\ \vdots \\ x\end{matrix}} \quad \boxed{\begin{matrix}\psi \\ \vdots \\ x\end{matrix}}}{x} \vee e$$

**Show $p \vee q \vdash q \vee p$**

| | | |
|---|---|---|
| 1. | $p \vee q$ | Premise |
| 2. | $p$ | Assumption |
| 3. | $q \vee p$ | $\vee i_2$ |
| 4. | $q$ | Assumption |
| 5. | $q \vee p$ | $\vee i_1$ |
| 6. | $q \vee p$ | $\vee e$ 1, 3, 5 |

- Show $q \to r \vdash p \vee q \to p \vee r$

# Rules for Negation

$$\frac{\bot}{\phi}\ \bot e \qquad \frac{\phi \qquad \neg\phi}{\bot}\ \neg e \qquad \frac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \bot\end{array}}}{\neg\phi}\ \neg\, \mathrm{i}$$

# Rules for Negation

$$\frac{\perp}{\phi}\ \perp e \qquad \frac{\phi \quad \neg\phi}{\perp}\ \neg e \qquad \frac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \perp\end{array}}}{\neg\phi}\ \neg\, i$$

**Show $\neg p \vee q \vdash p \rightarrow q$**

# Rules for Negation

$$\frac{\perp}{\phi}\ \perp e \qquad \frac{\phi \quad \neg\phi}{\perp}\ \neg e \qquad \frac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \perp\end{array}}}{\neg\phi}\ \neg\, \mathrm{i}$$

## Show $\neg p \vee q \vdash p \to q$

| | | |
|---|---|---|
| 1. | $\neg p \vee q$ | |
| 2. | $\neg p$ | Premise |
| 3. | $p$ | Assumption |
| 4. | $\perp$ | $\neg e$ 3,2 |
| 5. | $q$ | $\perp e$ 4 |
| 6. | $p \to q$ | $\to i$ 3,5 |

- Show $p \to q, p \to \neg q \vdash \neg p$

# Rules for Negation

$$\frac{\bot}{\phi}\ \bot e \qquad\qquad \frac{\phi \qquad \neg\phi}{\bot}\ \neg e \qquad\qquad \frac{\boxed{\begin{matrix}\phi\\ \vdots \\ \bot\end{matrix}}}{\neg\phi}\ \neg\, \mathrm{i}$$

## Show $\neg p \vee q \vdash p \rightarrow q$

| | | | | |
|---|---|---|---|---|
| 1. | $\neg p \vee q$ | | | |
| 2. | $\neg p$ | Premise | $q$ | Premise |
| 3. | $p$ | Assumption | $p$ | Assumption |
| 4. | $\bot$ | $\neg e$ 3,2 | $q$ | Copy 2 |
| 5. | $q$ | $\bot e$ 4 | $p \rightarrow q$ | $\rightarrow i$ 3,4 |
| 6. | $p \rightarrow q$ | $\rightarrow i$ 3,5 | | |

- Show $p \rightarrow q, p \rightarrow \neg q \vdash \neg p$

# Rules for Negation

$$\frac{\bot}{\phi}\ \bot e \qquad\qquad \frac{\phi \qquad \neg\phi}{\bot}\ \neg e \qquad\qquad \frac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \bot\end{array}}}{\neg\phi}\ \neg\, \mathrm{i}$$

## Show $\neg p \lor q \vdash p \to q$

| | | | | |
|---|---|---|---|---|
| 1. | $\neg p \lor q$ | | | |
| 2. | $\neg p$ | Premise | $q$ | Premise |
| 3. | $p$ | Assumption | $p$ | Assumption |
| 4. | $\bot$ | $\neg e$ 3,2 | $q$ | Copy 2 |
| 5. | $q$ | $\bot e$ 4 | $p \to q$ | $\to i$ 3,4 |
| 6. | $p \to q$ | $\to i$ 3,5 | | |
| 7. | $p \to q$ | | | $\lor e$ 1,2-6 |

# Rules for Negation

$$\frac{\bot}{\phi}\ \bot e \qquad \frac{\phi \quad \neg\phi}{\bot}\ \neg e \qquad \frac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \bot\end{array}}}{\neg\phi}\ \neg\, \mathrm{i}$$

## Show $\neg p \vee q \vdash p \rightarrow q$

| | | | | |
|---|---|---|---|---|
| 1. | $\neg p \vee q$ | | | |
| 2. | $\neg p$ | Premise | $q$ | Premise |
| 3. | $p$ | Assumption | $p$ | Assumption |
| 4. | $\bot$ | $\neg e$ 3,2 | $q$ | Copy 2 |
| 5. | $q$ | $\bot e$ 4 | $p \rightarrow q$ | $\rightarrow i$ 3,4 |
| 6. | $p \rightarrow q$ | $\rightarrow i$ 3,5 | | |
| 7. | $p \rightarrow q$ | | | $\vee e$ 1,2-6 |

- Show $p \rightarrow q, p \rightarrow \neg q \vdash \neg p$

# Logical Equivalences

$(\alpha \land \beta) \quad \equiv \quad (\beta \land \alpha)$ Commutativity of $\land$

# Logical Equivalences

| | | | |
|---|---|---|---|
| $(\alpha \wedge \beta)$ | = | $(\beta \wedge \alpha)$ | Commutativity of $\wedge$ |
| $(\alpha \vee \beta)$ | = | $(\beta \vee \alpha)$ | Commutativity of $\vee$ |

# Logical Equivalences

| | | | |
|---|---|---|---|
| $(\alpha \wedge \beta)$ | = | $(\beta \wedge \alpha)$ | Commutativity of $\wedge$ |
| $(\alpha \vee \beta)$ | = | $(\beta \vee \alpha)$ | Commutativity of $\vee$ |
| $(\alpha \wedge \beta) \wedge \gamma$ | = | $\alpha \wedge (\beta \wedge \gamma)$ | Associativity of $\wedge$ |

# Logical Equivalences

$$
\begin{array}{lcl}
(\alpha \wedge \beta) & = & (\beta \wedge \alpha) \\
(\alpha \vee \beta) & = & (\beta \vee \alpha) \\
(\alpha \wedge \beta) \wedge \gamma & = & \alpha \wedge (\beta \wedge \gamma) \\
(\alpha \vee \beta) \vee \gamma & = & \alpha \vee (\beta \vee \gamma)
\end{array}
$$

Commutativity of $\wedge$
Commutativity of $\vee$
Associativity of $\wedge$
Associativity of $\vee$

# Logical Equivalences

| | | | |
|---|---|---|---|
| $(\alpha \wedge \beta)$ | $=$ | $(\beta \wedge \alpha)$ | Commutativity of $\wedge$ |
| $(\alpha \vee \beta)$ | $=$ | $(\beta \vee \alpha)$ | Commutativity of $\vee$ |
| $(\alpha \wedge \beta) \wedge \gamma$ | $=$ | $\alpha \wedge (\beta \wedge \gamma)$ | Associativity of $\wedge$ |
| $(\alpha \vee \beta) \vee \gamma$ | $=$ | $\alpha \vee (\beta \vee \gamma)$ | Associativity of $\vee$ |
| $\neg\neg\alpha$ | $=$ | $\alpha$ | Double negation elimination |

# Logical Equivalences

| | | | |
|---|---|---|---|
| $(\alpha \wedge \beta)$ | $=$ | $(\beta \wedge \alpha)$ | Commutativity of $\wedge$ |
| $(\alpha \vee \beta)$ | $=$ | $(\beta \vee \alpha)$ | Commutativity of $\vee$ |
| $(\alpha \wedge \beta) \wedge \gamma$ | $=$ | $\alpha \wedge (\beta \wedge \gamma)$ | Associativity of $\wedge$ |
| $(\alpha \vee \beta) \vee \gamma$ | $=$ | $\alpha \vee (\beta \vee \gamma)$ | Associativity of $\vee$ |
| $\neg\neg\alpha$ | $=$ | $\alpha$ | Double negation elimination |
| $\alpha \rightarrow \beta$ | $=$ | $\neg\beta \rightarrow \neg\alpha$ | Contraposition |

# Logical Equivalences

| | | | |
|---|---|---|---|
| $(\alpha \wedge \beta)$ | $=$ | $(\beta \wedge \alpha)$ | Commutativity of $\wedge$ |
| $(\alpha \vee \beta)$ | $=$ | $(\beta \vee \alpha)$ | Commutativity of $\vee$ |
| $(\alpha \wedge \beta) \wedge \gamma$ | $=$ | $\alpha \wedge (\beta \wedge \gamma)$ | Associativity of $\wedge$ |
| $(\alpha \vee \beta) \vee \gamma$ | $=$ | $\alpha \vee (\beta \vee \gamma)$ | Associativity of $\vee$ |
| $\neg\neg\alpha$ | $=$ | $\alpha$ | Double negation elimination |
| $\alpha \rightarrow \beta$ | $=$ | $\neg\beta \rightarrow \neg\alpha$ | Contraposition |
| $\alpha \rightarrow \beta$ | $=$ | $\neg\alpha \vee \beta$ | Implication Elimination |

# Logical Equivalences

| | | | |
|---|---|---|---|
| $(\alpha \wedge \beta)$ | = | $(\beta \wedge \alpha)$ | Commutativity of $\wedge$ |
| $(\alpha \vee \beta)$ | = | $(\beta \vee \alpha)$ | Commutativity of $\vee$ |
| $(\alpha \wedge \beta) \wedge \gamma$ | = | $\alpha \wedge (\beta \wedge \gamma)$ | Associativity of $\wedge$ |
| $(\alpha \vee \beta) \vee \gamma$ | = | $\alpha \vee (\beta \vee \gamma)$ | Associativity of $\vee$ |
| $\neg\neg\alpha$ | = | $\alpha$ | Double negation elimination |
| $\alpha \rightarrow \beta$ | = | $\neg\beta \rightarrow \neg\alpha$ | Contraposition |
| $\alpha \rightarrow \beta$ | = | $\neg\alpha \vee \beta$ | Implication Elimination |
| $\alpha \leftrightarrow \beta$ | = | $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$ | Biconditional Elimination |

# Logical Equivalences

| | | | |
|---|---|---|---|
| $(\alpha \wedge \beta)$ | $=$ | $(\beta \wedge \alpha)$ | Commutativity of $\wedge$ |
| $(\alpha \vee \beta)$ | $=$ | $(\beta \vee \alpha)$ | Commutativity of $\vee$ |
| $(\alpha \wedge \beta) \wedge \gamma$ | $=$ | $\alpha \wedge (\beta \wedge \gamma)$ | Associativity of $\wedge$ |
| $(\alpha \vee \beta) \vee \gamma$ | $=$ | $\alpha \vee (\beta \vee \gamma)$ | Associativity of $\vee$ |
| $\neg\neg\alpha$ | $=$ | $\alpha$ | Double negation elimination |
| $\alpha \rightarrow \beta$ | $=$ | $\neg\beta \rightarrow \neg\alpha$ | Contraposition |
| $\alpha \rightarrow \beta$ | $=$ | $\neg\alpha \vee \beta$ | Implication Elimination |
| $\alpha \leftrightarrow \beta$ | $=$ | $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$ | Biconditional Elimination |
| $\neg(\alpha \wedge \beta)$ | $=$ | $\neg\alpha \vee \neg\beta$ | De Morgen |

# Logical Equivalences

| | | | |
|---|---|---|---|
| $(\alpha \wedge \beta)$ | $=$ | $(\beta \wedge \alpha)$ | Commutativity of $\wedge$ |
| $(\alpha \vee \beta)$ | $=$ | $(\beta \vee \alpha)$ | Commutativity of $\vee$ |
| $(\alpha \wedge \beta) \wedge \gamma$ | $=$ | $\alpha \wedge (\beta \wedge \gamma)$ | Associativity of $\wedge$ |
| $(\alpha \vee \beta) \vee \gamma$ | $=$ | $\alpha \vee (\beta \vee \gamma)$ | Associativity of $\vee$ |
| $\neg\neg\alpha$ | $=$ | $\alpha$ | Double negation elimination |
| $\alpha \rightarrow \beta$ | $=$ | $\neg\beta \rightarrow \neg\alpha$ | Contraposition |
| $\alpha \rightarrow \beta$ | $=$ | $\neg\alpha \vee \beta$ | Implication Elimination |
| $\alpha \leftrightarrow \beta$ | $=$ | $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$ | Biconditional Elimination |
| $\neg(\alpha \wedge \beta)$ | $=$ | $\neg\alpha \vee \neg\beta$ | De Morgen |
| $\neg(\alpha \vee \beta)$ | $=$ | $\neg\alpha \wedge \neg\beta$ | De Morgen |

# Logical Equivalences

| | | | |
|---|---|---|---|
| $(\alpha \wedge \beta)$ | $=$ | $(\beta \wedge \alpha)$ | Commutativity of $\wedge$ |
| $(\alpha \vee \beta)$ | $=$ | $(\beta \vee \alpha)$ | Commutativity of $\vee$ |
| $(\alpha \wedge \beta) \wedge \gamma$ | $=$ | $\alpha \wedge (\beta \wedge \gamma)$ | Associativity of $\wedge$ |
| $(\alpha \vee \beta) \vee \gamma$ | $=$ | $\alpha \vee (\beta \vee \gamma)$ | Associativity of $\vee$ |
| $\neg\neg\alpha$ | $=$ | $\alpha$ | Double negation elimination |
| $\alpha \rightarrow \beta$ | $=$ | $\neg\beta \rightarrow \neg\alpha$ | Contraposition |
| $\alpha \rightarrow \beta$ | $=$ | $\neg\alpha \vee \beta$ | Implication Elimination |
| $\alpha \leftrightarrow \beta$ | $=$ | $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$ | Biconditional Elimination |
| $\neg(\alpha \wedge \beta)$ | $=$ | $\neg\alpha \vee \neg\beta$ | De Morgen |
| $\neg(\alpha \vee \beta)$ | $=$ | $\neg\alpha \wedge \neg\beta$ | De Morgen |
| $\alpha \wedge (\beta \vee \gamma)$ | $=$ | $(\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$ | Distribution of $\wedge$ on $\vee$ |

# Logical Equivalences

| | | | |
|---|---|---|---|
| $(\alpha \wedge \beta)$ | $=$ | $(\beta \wedge \alpha)$ | Commutativity of $\wedge$ |
| $(\alpha \vee \beta)$ | $=$ | $(\beta \vee \alpha)$ | Commutativity of $\vee$ |
| $(\alpha \wedge \beta) \wedge \gamma$ | $=$ | $\alpha \wedge (\beta \wedge \gamma)$ | Associativity of $\wedge$ |
| $(\alpha \vee \beta) \vee \gamma$ | $=$ | $\alpha \vee (\beta \vee \gamma)$ | Associativity of $\vee$ |
| $\neg\neg\alpha$ | $=$ | $\alpha$ | Double negation elimination |
| $\alpha \rightarrow \beta$ | $=$ | $\neg\beta \rightarrow \neg\alpha$ | Contraposition |
| $\alpha \rightarrow \beta$ | $=$ | $\neg\alpha \vee \beta$ | Implication Elimination |
| $\alpha \leftrightarrow \beta$ | $=$ | $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$ | Biconditional Elimination |
| $\neg(\alpha \wedge \beta)$ | $=$ | $\neg\alpha \vee \neg\beta$ | De Morgen |
| $\neg(\alpha \vee \beta)$ | $=$ | $\neg\alpha \wedge \neg\beta$ | De Morgen |
| $\alpha \wedge (\beta \vee \gamma)$ | $=$ | $(\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$ | Distribution of $\wedge$ on $\vee$ |
| $\alpha \vee (\beta \wedge \gamma)$ | $=$ | $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$ | Distribution of $\vee$ on $\wedge$ |

# Soundness and Completeness

- Soundness: doing right
- Completeness: full coverage

# Soundness and Completeness

- Soundness: doing right
- Completeness: full coverage

There are 10 defective bulbs in a box of 25.

- Mr. A gives me 10 bulbs none of them is defective
- Mr. B gives me 20 bulbs; 5 of them is defective

# Soundness and Completeness

- Soundness: doing right
- Completeness: full coverage

There are 10 defective bulbs in a box of 25.

- Mr. A gives me 10 bulbs none of them is defective
- Mr. B gives me 20 bulbs; 5 of them is defective

A is sound
B is complete

# Soundness and Completeness

- Soundness: doing right
- Completeness: full coverage

There are 10 defective bulbs in a box of 25.

- Mr. A gives me 10 bulbs none of them is defective
- Mr. B gives me 20 bulbs; 5 of them is defective

A is sound
B is complete

Evaluate a legal system "guilty until proven innocent" and "innocent until proven guilty"

# Soundness and Completeness

- Soundness: doing right
- Completeness: full coverage

There are 10 defective bulbs in a box of 25.

- Mr. A gives me 10 bulbs none of them is defective
- Mr. B gives me 20 bulbs; 5 of them is defective

A is sound

B is complete

Evaluate a legal system “guilty until proven innocent” and “innocent until proven guilty”

What we want? both.

# CNF, IMPL_FREE and NNF

Conjunctive normal form[8], implication free[9] and negative normal form[10]

**Find CNF(NNF(IMPL_FREE($A$)))**

Where $A = \neg p \wedge q \rightarrow p \wedge (r \rightarrow q)$

[8] everything is conjunctions of disjunction

[9] no $\rightarrow$

[10] no double negation

# CNF, IMPL_FREE and NNF

Conjunctive normal form[8], implication free[9] and negative normal form[10]

**Find CNF(NNF(IMPL_FREE($A$)))**

Where $A = \neg p \wedge q \rightarrow p \wedge (r \rightarrow q)$

$$\neg(\neg p \wedge q) \vee (p \wedge (\neg r \vee q))$$
$$(p \vee \neg q) \vee (p \wedge (\neg r \vee q))$$
$$(p \vee \neg q \vee p) \vee (p \wedge \neg q \wedge \neg r \vee q)$$

[8] everything is conjunctions of disjunction

[9] no $\rightarrow$

[10] no double negation

# Horn Clause

Formula that can be generated by $H$

$$\begin{array}{lcl} P & ::= & \bot \mid \top \mid p \mid q \mid r \mid \ldots \\ A & ::= & P \mid P \wedge A \\ C & ::= & A \to P \\ H & ::= & C \mid C \wedge H \end{array} \tag{1}$$

# Horn Clause

Formula that can be generated by $H$

$$\begin{aligned} P &::= \bot \mid \top \mid p \mid q \mid r \mid \ldots \\ A &::= P \mid P \wedge A \\ C &::= A \rightarrow P \\ H &::= C \mid C \wedge H \end{aligned} \tag{1}$$

## Satisfiability

1. It marks $\top$ if it occurs in that list.
2. If there is a conjunct $P_1 \wedge P_2 \wedge \cdots \wedge P_{k_i} \rightarrow P'$ of $\phi$ such that all $P_j$ with $1 \leq j \leq k_i$ are marked, mark $P'$ as well and go to 2. Otherwise (= there is no conjunct $P_1 \wedge P_2 \wedge \cdots \wedge P_{k_i} \rightarrow P'$ such that all $P_j$ are marked) go to 3.
3. If $\bot$ is marked, print out 'The Horn formula $\phi$ is unsatisfiable.' and stop. Otherwise, go to 4.
4. Print out 'The Horn formula $\phi$ is satisfiable.' and stop.

# Horn Clause

Formula that can be generated by $H$

$$
\begin{aligned}
P &::= \bot \mid \top \mid p \mid q \mid r \mid \ldots \\
A &::= P \mid P \wedge A \\
C &::= A \to P \\
H &::= C \mid C \wedge H
\end{aligned}
\tag{1}
$$

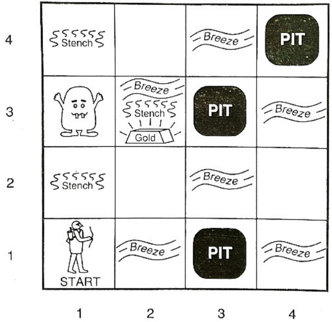
## Satisfiability

1. It marks $\top$ if it occurs in that list.
2. If there is a conjunct $P_1 \wedge P_2 \wedge \cdots \wedge P_{k_i} \to P'$ of $\phi$ such that all $P_j$ with $1 \le j \le k_i$ are marked, mark $P'$ as well and go to 2. Otherwise (= there is no conjunct $P_1 \wedge P_2 \wedge \cdots \wedge P_{k_i} \to P'$ such that all $P_j$ are marked) go to 3.
3. If $\bot$ is marked, print out 'The Horn formula $\phi$ is unsatisfiable.' and stop. Otherwise, go to 4.
4. Print out 'The Horn formula $\phi$ is satisfiable.' and stop.

(a) $(p \wedge q \wedge w \to \bot) \wedge (t \to \bot) \wedge (r \to p) \wedge (\top \to r) \wedge (\top \to q) \wedge (u \to s) \wedge (\top \to u)$
(b) $(p \wedge q \wedge w \to \bot) \wedge (t \to \bot) \wedge (r \to p) \wedge (\top \to r) \wedge (\top \to q) \wedge (r \wedge u \to w) \wedge (u \to s) \wedge (\top \to u)$
(c) $(p \wedge q \wedge s \to p) \wedge (q \wedge r \to p) \wedge (p \wedge s \to s)$
(d) $(p \wedge q \wedge s \to \bot) \wedge (q \wedge r \to p) \wedge (\top \to s)$
(e) $(p_5 \to p_{11}) \wedge (p_2 \wedge p_3 \wedge p_5 \to p_{13}) \wedge (\top \to p_5) \wedge (p_5 \wedge p_{11} \to \bot)$
(f) $(\top \to q) \wedge (\top \to s) \wedge (w \to \bot) \wedge (p \wedge q \wedge s \to \bot) \wedge (v \to s) \wedge (\top \to r) \wedge (r \to p)$

# Recall Wumpus World

- **Performance** gold +100, death -100, step -1, arrow -10
- **Environment** smell around wumpus, breeze around pit
- **Actuator** turn left/right, forward, grab, release, shoot
- **Sensor** breeze, glitter, smell, bump, scream




Single Agent, Deterministic, Static, Discrete, !Observable & !Episodic

- $P_{x,y}$ if there is a pit in $[x, y]$
- $W_{x,y}$ if wumpus is in $[x, y]$
- $B_{x,y}$ if breeze is in $[x, y]$
- $S_{x,y}$ if stench is in $[x, y]$

We know $R_1$: $\neg P_{1,1}$, $R_2$: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$,
$R_3$: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$, $R_4$: $\neg B_{1,1}$, $R_5$: $B_{2,1}$

# Model Checking for Inference

- Seven symbols $P_{1,1}$, $B_{1,1}$, $P_{1,2}$, $P_{2,1}$, $B_{2,1}$, $P_{2,2}$, $P_{3,1}$ have $2^7 = 128$ models. In three of these knowledge base is true.

| $B_{1,1}$ | $B_{2,1}$ | $P_{1,1}$ | $P_{1,2}$ | $P_{2,1}$ | $P_{2,2}$ | $P_{3,1}$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $KB$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *false* | *false* | *false* | *false* | *false* | *false* | *false* | *true* | *true* | *true* | *true* | *false* | *false* |
| *false* | *false* | *false* | *false* | *false* | *false* | *true* | *true* | *true* | *false* | *true* | *false* | *false* |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| *false* | *true* | *false* | *false* | *false* | *false* | *false* | *true* | *true* | *false* | *true* | *true* | *false* |
| *false* | *true* | *false* | *false* | *false* | *false* | *true* | *true* | *true* | *true* | *true* | *true* | *true* |
| *false* | *true* | *false* | *false* | *false* | *true* | *false* | *true* | *true* | *true* | *true* | *true* | *true* |
| *false* | *true* | *false* | *false* | *false* | *true* | *true* | *true* | *true* | *true* | *true* | *true* | *true* |
| *false* | *true* | *false* | *false* | *true* | *false* | *false* | *true* | *false* | *false* | *true* | *true* | *false* |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| *true* | *true* | *true* | *true* | *true* | *true* | *true* | *false* | *true* | *true* | *false* | *true* | *false* |

In all those three $\neg P_{1,2}$ is true, hence there is no pit in [1,2]. On the other hand $P_{2,2}$ is true on two and false in one so it is not confirmed whether there is pit in [2,2] or not.

# Validity and Satisfiability

- **Validity**: sentence is true in all models (tautologies)

$$A \vee \neg A$$
$$A \vee B \rightarrow A \vee B$$

- **Satisfiability**: sentence is true in some models

$$A \vee \neg B$$
$$A \rightarrow B$$

Determine whether following sentence is valid or satisfiable

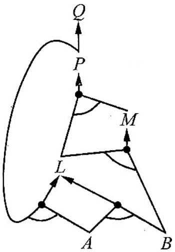
$$((A \wedge B) \rightarrow C) \leftrightarrow (A \rightarrow (B \rightarrow C))$$

# Forward Chaining

Determines if a single proposition symbol $q$ is entailed by the knowledge? (data driven reasoning)

- It begins from known facts and adds conclusions of the implication whose all the premises are known
- for $L_{1,1} \wedge breeze \rightarrow B_{1,1}$ if we know $L_{1,1}$ and *breeze* then $B_{1,1}$ is added in knowledge base [11]

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
$A$
$B$




- Applies Modus Ponens

$$\frac{\phi \quad \phi \rightarrow \psi}{\psi}$$

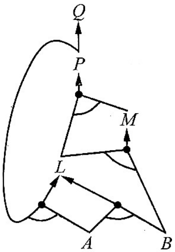
[11] $L_{1,1}$: location is [1,1]

# Forward Chaining

Determines if a single proposition symbol $q$ is entailed by the knowledge? (data driven reasoning)

- It begins from known facts and adds conclusions of the implication whose all the premises are known
- for $L_{1,1} \wedge breeze \rightarrow B_{1,1}$ if we know $L_{1,1}$ and *breeze* then $B_{1,1}$ is added in knowledge base [11]

$$
\begin{aligned}
&P \Rightarrow Q \\
&L \wedge M \Rightarrow P \\
&B \wedge L \Rightarrow M \\
&A \wedge P \Rightarrow L \\
&A \wedge B \Rightarrow L \\
&A \\
&B
\end{aligned}
$$




- Applies Modus Ponens

$$\frac{\phi \quad \phi \rightarrow \psi}{\psi}$$

- An and-or tree gets constructed

[11] $L_{1,1}$: location is [1,1]

# Backward Chaining

- Works backward from query
- If query $Q$ is known to be true, then no work is needed.
- Otherwise, find those implications whose conclusion is $Q$
- If all the premises of one of those implications can be proven true (by backward chaining) then $Q$ is true

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$\boxed{A \wedge P \Rightarrow L}$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

- test($Q$) is it true ?
- test($P$) is it true ?
- test($L \wedge M$) ?
- ((test($A \wedge B$) or test($A \wedge P$)) and test($B \wedge L$) ? we know $A$ and $B$ so we have $L$ this gives $M$
- Therefore $P$ and hence $Q$

# First Order Logic (Predicate Logic)

- We have **constants**, **variables**, **predicates** and **functions**
- Here $P(x)$ could means $\forall x$ we have $P(x)$ or $\exists x$ such that $P(x)$
- Variable $x$ has a domain from where it gets values
- $\forall x, \exists y P(x, y)$ is not always same as $\exists y, \forall x P(x, y)$
- When we say $\exists$ a predicate then it is higher order logic

## Examples

1. Not every customer have purchased milk and bread

# First Order Logic (Predicate Logic)

- We have **constants**, **variables**, **predicates** and **functions**
- Here $P(x)$ could means $\forall x$ we have $P(x)$ or $\exists x$ such that $P(x)$
- Variable $x$ has a domain from where it gets values
- $\forall x, \exists y P(x, y)$ is not always same as $\exists y, \forall x P(x, y)$
- When we say $\exists$ a predicate then it is higher order logic

## Examples

1. Not every customer have purchased milk and bread

$$\exists c \;\; Cust(c) \wedge [\neg shop(milk, c) \vee \neg shop(bread, c)]$$

2. Only one customer have purchased guitar

# First Order Logic (Predicate Logic)

- We have **constants**, **variables**, **predicates** and **functions**
- Here $P(x)$ could means $\forall x$ we have $P(x)$ or $\exists x$ such that $P(x)$
- Variable $x$ has a domain from where it gets values
- $\forall x, \exists y P(x, y)$ is not always same as $\exists y, \forall x P(x, y)$
- When we say $\exists$ a predicate then it is higher order logic

## Examples

1. Not every customer have purchased milk and bread

$$\exists c \;\; Cust(c) \wedge [\neg shop(milk, c) \vee \neg shop(bread, c)]$$

2. Only one customer have purchased guitar

$$\exists x \;\; [Cust(x) \wedge shop(G, x) \wedge \forall y[\neg(x = y) \wedge Cust(y) \Rightarrow \neg shop(G, y)]]$$

3. Only one customer have purchased guitar and pen
4. Highest purchase in forenoon is more than afternoon.

# Inference in First Order Logic

- **Universal Elimination** $\forall x \;\; Feels(x, king)$ could be $Feels(Raju, king)$ substitution $\{x/\text{Raju}\}$ is done using some ground term.
- **Existential Elimination** $\exists x \;\; Feels(x, king)$ could be $Feels(man, king)$ if *man* does not appear in knowledge base [12]
- **Existential Introduction** If $Feels(Raju, king)$ then we can say $\exists x \;\; Feels(x, king)$

1. It is crime for Magadh to sell formula to a hostile country
2. Country Bhind, an enemy of Magadh have purchased some formula from Dara
3. Dara is from Magadh
4. Question: Is Dara a criminal?

[12] *man* is a name of person who feels like king

# Prolog

- A logic programming language [13]
- Compile as ['a.pl'].
- If :- and , or ; not not
- write('hello'), nl

```
warm_blood(penguin).
warm_blood(human).
produce_milk(penguin).
produce_milk(human).
have_feather(penguin).
have_hair(human).
mammal(X) :-
warm_blood(X),
produce_milk(X),

have_hair(X).
```

?- mammal(penguin)
no
?- mammal(X).
X = human.

- is_even(X) :-
  Y is X//2, X =:= 2*Y.
- write('what is your name/ '), read(X), write('Hi '),write(X).

Many more things are possible

[13] http://www.swi-prolog.org/

# Thank You!

**Thank you very much for your attention!**

**Queries ?**

(Reference[14])

[14] 1) Book - *AIMA*, ch-07, Russell and Norvig. 2) Book - *Logic in CS*, ch-01, Mitchel Huth and Mark Ryan.