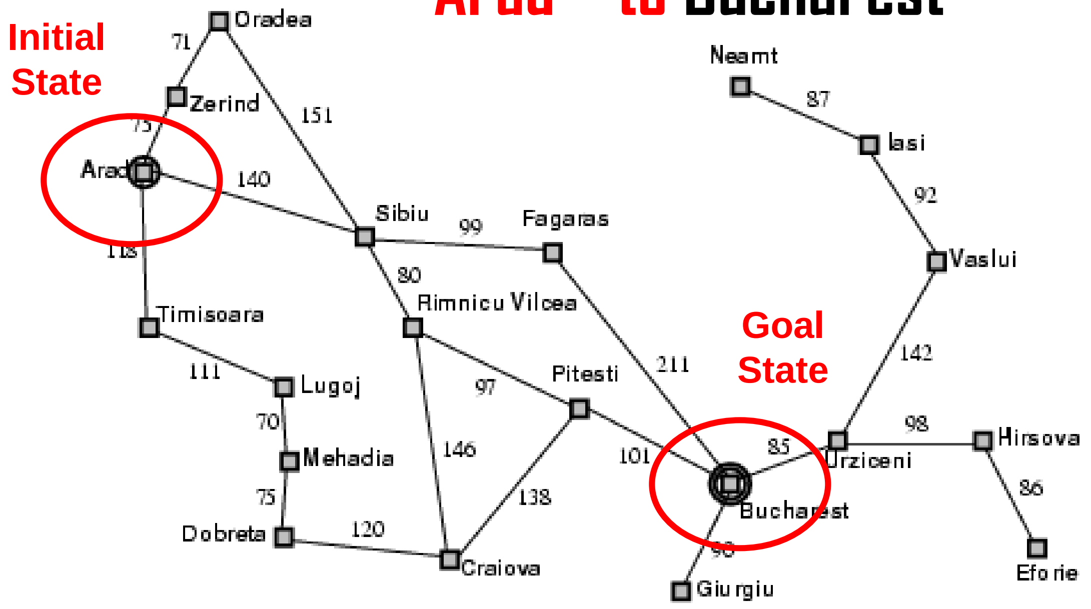
Steps in problem solving

- **Goal formulation**
  - is a step that ***specifies exactly what the agent is trying to achieve***
  - This step narrows down the **scope** that the agent has to look at
- **Problem formulation**
  - is a step that puts down the **actions** and **states** that the agent has to consider **given a goal** (***avoiding any redundant states***), like:
    - the initial state
    - the allowable actions etc…
- **Search**
  - is the process of looking for the **various sequence of actions** **that lead to a goal state,** **evaluating them** and choosing the **optimal** sequence.
- **Execute**
  - is ***the final step*** that the agent executes **the chosen sequence of actions** to get it to the solution/goal

Example: Path Finding problem

- Formulate goal:
  - be in Bucharest (Romania)
- Formulate problem:
  - action: drive between pair of connected cities (direct road)
  - state: be in a city
    (20 world states)
- Find solution:
  - sequence of cities leading from start to goal state, e.g., Arad, Sibiu, Fagaras, Bucharest
- Execution
  - drive from Arad to Bucharest according to the solution

Arad to Bucharest



Problem Solving Agent Algorithm

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
    static: seq, an action sequence, initially empty
            state, some description of the current world state
            goal, a goal initially null
            problem, a problem formulation

    state ← UPDATE-STATE(state, percept)
    if seq is empty then do
        goal ← FORMULATE-GOAL(state)
        problem ←FORMULATE-PROBLEM(state,goal)
        seq ← SEARCH(problem)
    action ← FIRST(seq)
    seq ← REST(seq)
    return  action
```

Searching

- **search** consists of **a set of methods** to **systematically determine the next best action** **in order to search for a solution** of a problem

- **parts of a problem:** initial state, operators, goal test function, path cost function.

Searching…

- Examine different possible sequences of actions & states, and come up with **specific optimal sequence** of actions that will take you from the initial state to the goal state
  - *Given a state space with **initial state** and **goal state,** find* optimal ***sequence of actions** leading through a **sequence of states** to the **final goal state***
- **Searching in State Space**
  - **select optimal** (min-cost/max-profit) **node/state** in the state space
  - ***test whether*** **the state selected** ***is the goal state or not***
  - **if not, expand the node** further to identify its successor.

Search Tree

- The searching process is like building the search tree that is super imposed over the state space
  - A search tree is a representation in which nodes denote paths and branches connect paths. The node with no parent is the **root node**. The nodes with no children are called **leaf nodes**.

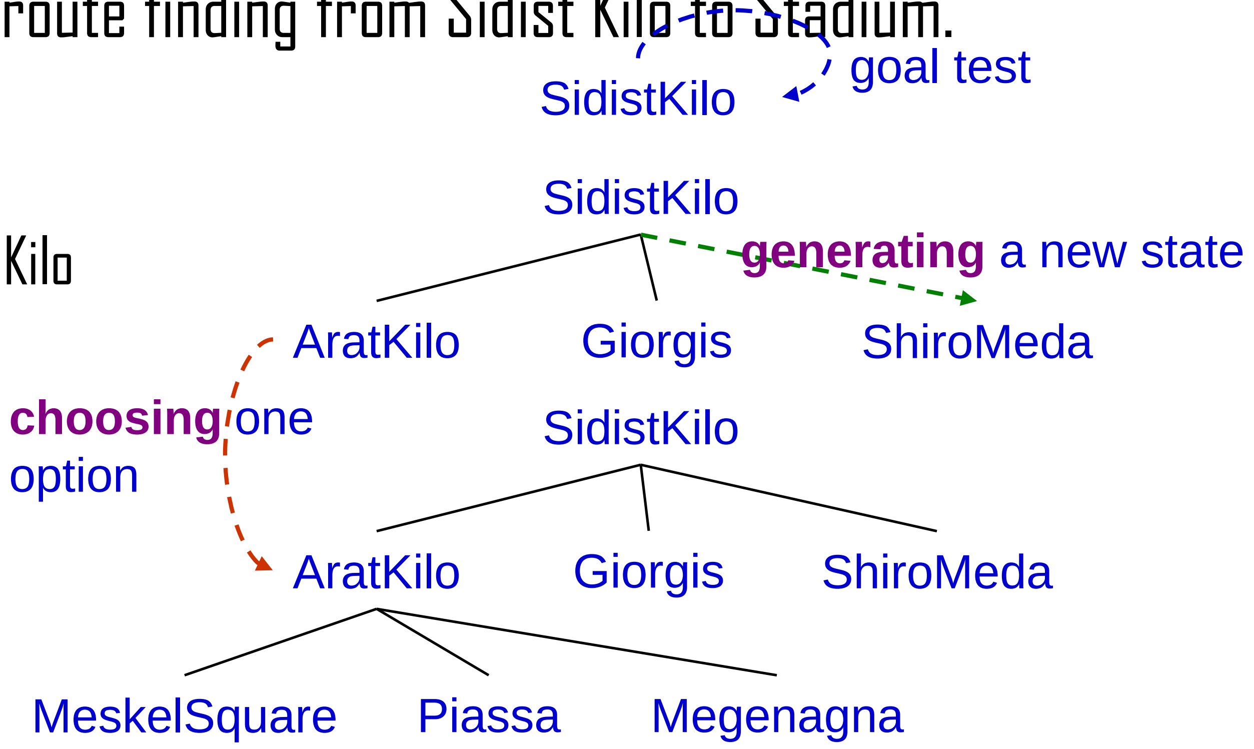
Example: Route finding Problem

- Partial **search tree** for route finding from Sidist Kilo to Stadium.

(a) The initial state

(b) After **expanding** Sidist Kilo

(c) After expanding Arat Kilo



Search algorithm

- **Two functions** needed for conducting search
  - **Generator (or successors) function**: Given a state and action, **produces its successor states** (in a state space)
  - **Tester (or IsGoal) function**: Tells whether given state S is a goal state IsGoal(S) → True/False

  ✓ IsGoal and Successors functions depend on problem domain.

- **Two lists maintained** during searching
  - **OPEN list**: stores the nodes expanded but not explored
  - **CLOSED list**: the nodes expanded and explored

  ✓ Generally search proceeds by examining each node on the OPEN list; performing some expansion operation that adds its children to the OPEN list, & moving the node to the CLOSED list

General Search Algorithm

I. identify initial state
II. expand (generate new states)
III. choose option(**search strategy**)
IV. test goal function
V. expand until goal not attained/no more states to expand

Search algorithm

- **Input**: a given problem (Initial State + Goal state + transit states and operators)
- **Output**: **returns optimal sequence of actions** to reach the goal.

```
function GeneralSearch (problem, strategy)
    open = (initialState); //put initial state in the List
    closed = {}; //maintain list of nodes examined earlier
    while (not (empty (open)))
        f = remove_first(open);
        if IsGoal (f) then return (f);
        closed = append (closed, f);
        s = Successors (f)
        left = not-in-closed (s, closed );
        open = merge (rest(open), left); //append or prepend  to open list
    end while
    return ('fail')
end  GeneralSearch
```

Evaluation Searching Algorithm

Searching algorithm can be evaluated by the following four criteria

- Completeness
- Optimality
- Time complexity
- Space complexity

Cont…

- **Completeness**
  - Guarantees finding a solution whenever one exists.
  - Think about the density of solutions in space and evaluate whether the searching technique guaranteed to find *all* solutions or not.
- **Optimality**
  - is the algorithm **finding an optimal solution;** i.e. the one with minimize cost or maximize profit
  - If a solution is found, is it guaranteed to be an optimal one? That is, is it the one with minimum cost?
    - how good is our solution?
- **Time complexity**: ***how long does it take to find a solution***
  - Usually measured in terms of the number of nodes expanded
- **Space complexity:** ***how much space is used by the algorithm***

Assignment IV (due: 5 days)

Consider one of the following problem:

- Towers of Hanoi [1]
- Travelling salesperson problem[3]
- 8-queens problem[4]
- Remove 5 sticks[5]
- Water Jug Problem [5]
- Monkey and Banana problem[7]

1. **Identify the set of states and operators;** based on which construct the state space of the problem
2. Write **goal test function** and also **determine path cost**

Individual Assignment I (For Next Class- Hard Copy)

1. Discus ***the different types of problem based on the level of knowledge*** that an agent can have concerning its action and the state of the world with example?

Dcs

Year IV
Semester I
2016

Knowledge Representation(NR)

gechb21@gmail.com

20 November 2016

Year IV
Semester I
2016

SESSION VII

Contents will be covered

- Knowledge
    - What????
    - Why?????
- KBA
    - KE
    - KR
- KR
    - KR using Logic
    - Why????

Knowledge: What and Why?

- **Knowledge** includes facts about the real world entities and the relationship between them
- Knowledge-based Systems (KBSs) are **useless** ***without the ability to represent knowledge.***

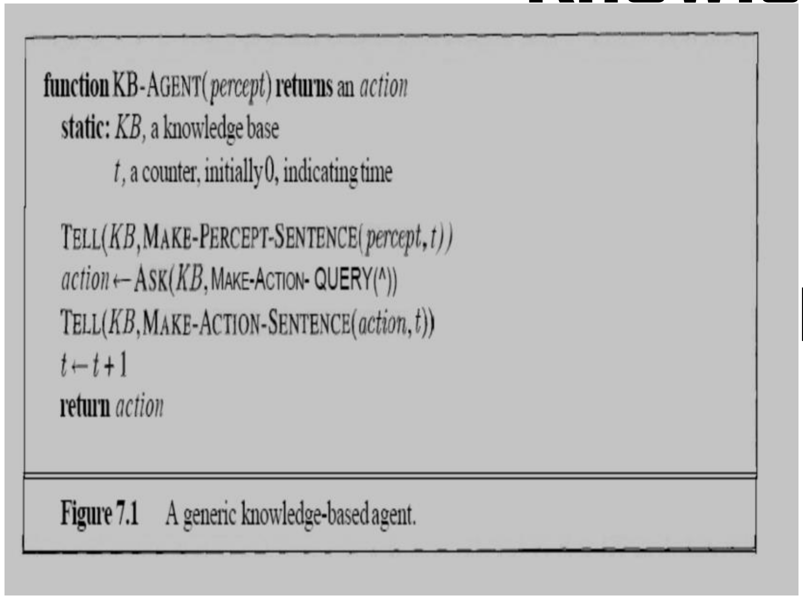
Why Knowledge is important ?

- We are living in complex environment.
- Hence, there is a need to represent knowledge to ease ***the development of an intelligent system.***
- It enables to:
  - ***Automate reasoning*** , ***Discover new facts, Deduce new facts that follow from the KB,*** *and* ***Answer users queries***
  - ***Make quality decisions*** - select courses of actions, etc.

Knowledge-based Agent (KBA)

- **Agents** can be seen ***as knowing about their world***, and ***reasoning about their possible courses of action.***
- **KBA** begins with some knowledge of the world and of its actions.
  - It **uses logical reasoning** to maintain a description of the world as new percepts arrive
  - **Learn new facts/knowledge** that are inferred and unseen by current percepts
  - **Deduce a course of actions** that will achieve its goals
- One can also design an **autonomous agent** that
  - learns from experience and construct knowledge with less human interventions

Knowledge-based Agent (KBA)…



```
function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(^))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Figure 7.1 A generic knowledge-based agent.

We can describe a knowledge-based agent at three levels:

I. Knowledge Level

- *The most abstract level:* describe agent by saying what it knows about the world and what its goals are.

II. Logical Level.

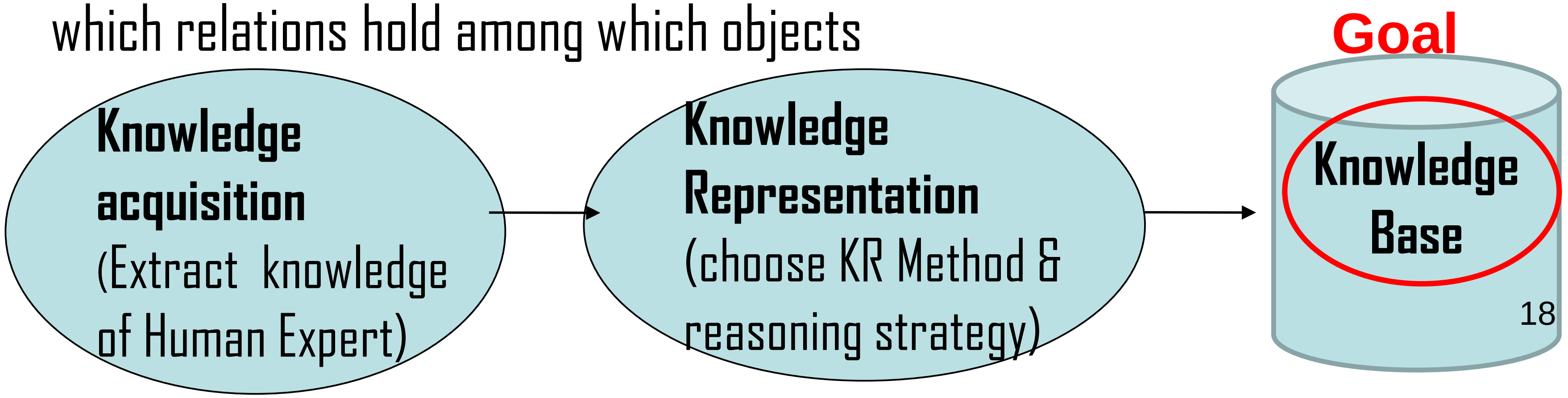
- The level at which *the knowledge is encoded into sentences* of some logical language.

III. Implementation Level.

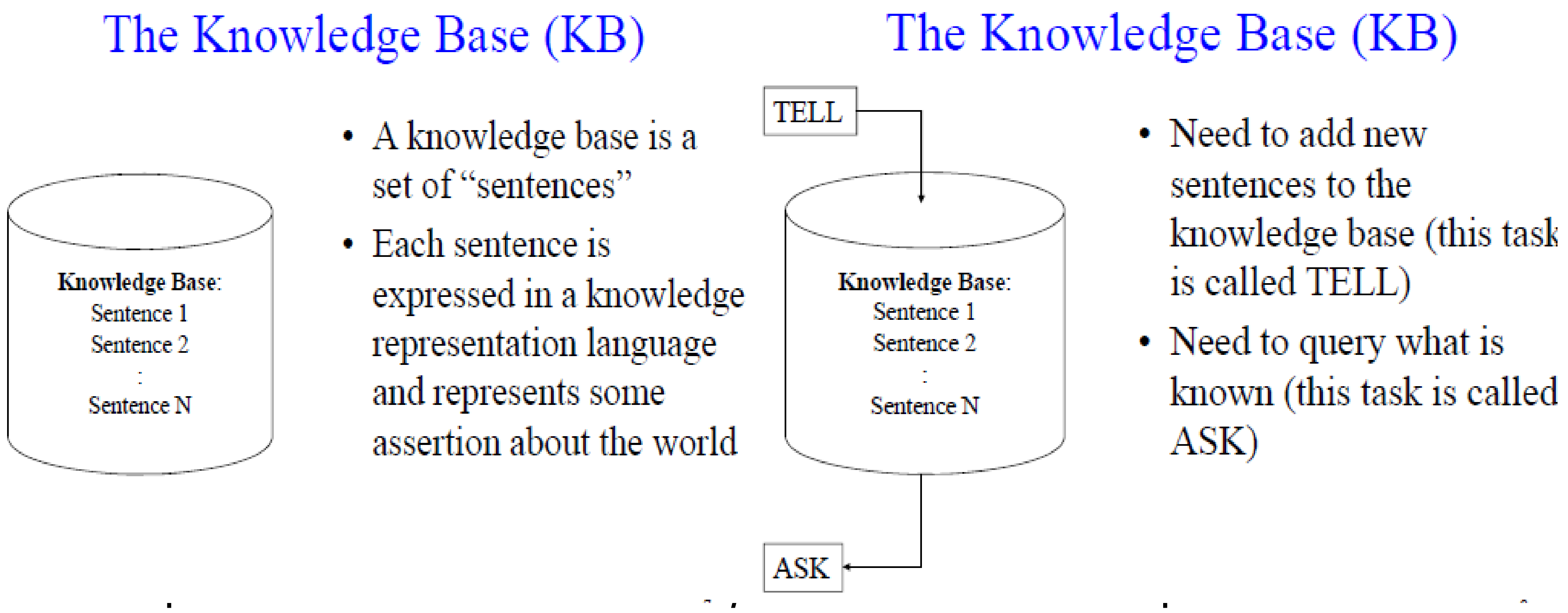
- This is the level where *sentences are implemented*. This level runs on the agent architecture.

Knowledge engineering (KE)

- KE is the **process of building a knowledge base** through **extracting knowledge from the human expert.**
- **Knowledge engineering** is the process of
  - *Extracting knowledge* from the human expert.
  - *Choose knowledge representation formalism*
  - *Choose reasoning* and *problem solving strategy.*
- A knowledge engineer is **someone who investigates a particular domain**, ***determines*** **what concepts are** ***important*** in that domain, and ***creates a*** **formal representation of the objects** and **relations** *in the domain.*
  - A KE has to decide what objects and relations are worth representing, and which relations hold among which objects


Goal
Knowledge acquisition (Extract knowledge of Human Expert)
Knowledge Representation (choose KR Method & reasoning strategy)
Knowledge Base

The two main tasks of KE



The Knowledge Base (KB)

- A knowledge base is a set of "sentences"
- Each sentence is expressed in a knowledge representation language and represents some assertion about the world

The Knowledge Base (KB)

- Need to add new sentences to the knowledge base (this task is called TELL)
- Need to query what is known (this task is called ASK)

- **Knowledge base** is used to store **a set of facts and rules** about the domain expressed ***in a suitable representation language.***
  - *Each individual representation* are called **sentences**
  - Sentences are expressed in a (formal) **knowledge representation (KR) language**

Knowledge Representation & Reasoning

- **Knowledge Representation (KR)**: express knowledge explicitly *in a computer-tractable* way such that the agent can reason out.
- **Parts of KR language:**
  - **Syntax** of a language: ***describes the possible configuration*** to form sentences. **E.g.**: if x & y denote numbers, then x > y is a sentence about numbers
  - **Semantics**: ***determines the facts in the world*** to which the sentences refer. **E.g.**: x > y is false when y is greater than x and true otherwise

- **Reasoning**: is ***the process of constructing new sentences*** from existing facts in the KB.
  - Proper reasoning ensures that the new configuration represent facts that actually follow from the facts in the KB.

Logic

- A Logic is a **formal language** in which knowledge can be represented such that conclusions can easily be drawn.
  - It is a ***declarative language*** to assert sentences and deduce from sentences.
- **Components of a logic include syntax, semantics, reasoning** and **inference mechanism.**
  - **Syntax:** ***what expressions/structures are allowed in the language***.
    - *Describes how to make sentences.*

    E.g. my car (red) is ok, but my car(grey or green) is not.
  - **Semantics:** ***express what sentences mean***, in terms of a mapping to real world.
    - The meaning of a sentence is not *intrinsic* to that sentence. Semantics relate sentences to reality.
    - E.g. my car (red) means that my car is red.
  - **Proof Theory:** It is a means of ***carrying our reasoning using a set of rules***.
    - It helps to draw new conclusions from existing statements in the logic.

Why formal languages (Logic) ?

- An ***obvious way of expressing or representing facts*** and thoughts is by writing them in a natural language such as English, Amharic, etc. However,
  - The meaning of a sentence ***depends on the sentence itself*** and on ***the context on which the sentence was spoken***
  - Natural languages exhibit **ambiguity.**

    E.g. small dogs and cats.

- **Ambiguity** ***makes reasoning difficult and incomplete.***
  - Hence we need formal languages to **express facts and concepts in an unambiguous and well-defined way.**

Revision

1. Steps for problem solving?
2. Searching?
   - Searching Algorithm Should consider
     - % Two functions
     - % Two lists
3. What are the Criteria's for evaluating Searching algorithm performance
4. What is k/ge? Why k/ge is needed for developing KBS?
5. What is the main purpose of KE? – The two main tasks of KE?
6. Logic? Components of Logic? Why we use logic to represent language in the KB?

Propositional logic

- A simple language **useful for showing key ideas and definitions**
- **Syntax:** PL allows facts about the world to be represented as sentences formed from:
  - ✓ **Logical constants**: True, False
  - ✓ **Proposition symbols (*P, Q, R, …*)** are ***used to represent facts about the world***: e.g.: P = "It is hot", Q = "It is humid", R = "It is raining"
  - ✓ **Logical connectives**: not **(¬)**, and **(∧)**, or **(∨)**, implies **(→)**, is equivalent, if and only if (↔).
    - ➢ *Precedence order from* **highest to lowest** is: **¬, ∧, ∨, →, ↔**

      e.g. The sentence ¬**P v Q** ∧ ***R***→ ***S*** *is equivalent to* *[(*¬**P) v (Q** ∧ ***R)]***→ ***S***
    - ➢ **Parenthesis** ( ): ***Used for grouping sentences*** and to ***specify order of precedence***

Propositional logic (PL) sentences

- A sentence is made ***by linking prepositional symbols together*** using **logical connectives.**
  - There are ***atomic and complex*** sentences.
  - **Atomic sentences** consist of ***propositional symbol*** (e.g. P, Q, TRUE, FALSE)
  - **Complex sentences** are combined by using **connectives** or **parenthesis:**
  - while S and T are **atomic sentences**, $S \lor T$, $(S \lor T)$, $(S \land T)$, $(S \rightarrow T)$, and $(S \leftrightarrow T)$ are **complex sentences.**

Examples: Given the following sentences about the "**weather problem**" convert them into **PL sentences:**

- "It is humid.": **Q**
- "If it is humid, then it is hot" : $\mathbf{Q \rightarrow P}$
- "If it is hot and humid, then it is raining": $\mathbf{(P \land Q) \rightarrow R}$

Syntax of Propositional logic (PL)

Prop Logic: Syntax

$$
\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} &\rightarrow \textit{True} \mid \textit{False} \mid \textit{Symbol} \\
\text{Symbol} &\longrightarrow P \mid Q \mid R \mid \ldots \\
\textit{ComplexSentence} &\rightarrow (\ \textit{Sentence}\ ) \\
&\quad \mid \textit{Sentence Connective Sentence} \\
&\quad \mid \neg \textit{Sentence} \\
\textit{Connective} &\rightarrow \wedge \mid \vee \mid \Leftrightarrow \mid \Rightarrow
\end{aligned}
$$

Precedence (high to low): $\neg \wedge \vee \Rightarrow \Leftrightarrow$

E.g. $\neg P \vee Q \wedge R \Rightarrow S$
is equivalent to
$((\neg P) \vee (Q \wedge R)) \Rightarrow S$

use α to denote a sentence

Exercise

Examples: Convert the following **English sentences to Propositional logic**

Let A = Lectures are active and R = Text is readable, P = Aleazar will pass the exam, then represent the following:

- the lectures are *not* active: $\neg A$
- the lectures are active and the text is readable: $A \wedge R$
- either the lectures are active or the text is readable: $A \vee R$
- if the lectures are active, then the text is not readable: $A \rightarrow \neg R$
- the lectures are active if and only if the text is readable: $A \leftrightarrow R$
- if the lectures are active, then if the text is not readable, Aleazar will not pass the exam: $A \rightarrow (\neg R \rightarrow \neg P)$

Terminology

- **Valid sentence:** A sentence is **valid sentence** or **tautology** ***if and only if it is True under all possible interpretations*** in all possible worlds.

  Example: "It's raining **or** it's not raining." ($R \vee \neg R$).

- **Unsatisfiable:** A sentence is unsatisfiable (*inconsistent sentence* or *self- contradiction*) if and only if it is not satisfiable, i.e. ***a sentence that is False under all interpretations.*** The world is never like what it describes.

  Example: "It's raining **and** it's not raining." $R \wedge \neg R$

- **Satisfiable**: A sentence is satisfiable if and only if there is some interpretations in some world for which the sentence is True.

  Example: "It is raining or it is humid". R v Q, R

Proof: Inference Rules

- **Inference** ***is used to create new sentences that logically follow from a given set of sentences in the KB.***
  - It captures patterns of inferences that occur over & over again.
  - Once a rule is established, it can be used to make inferences without going through the tedious process of building truth tables

- Given **set of inference rules** (I) and **set of sentences** (KB); Inference is the process of applying successive inference rules from the KB, each rule inferring new facts and adding its conclusion to KB

- There are **different inference rules** (including logical equivalence) that can be **used for proofing** and **reasoning purpose.**

Logical equivalence

- Two sentences are logically equivalent iff they are true in same models

$p \vee q \equiv q \vee p$	*Commutativity of disjunction*
$p \wedge q \equiv q \wedge p$	*Commutativity of conjunction*
$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	*Associativity of conjunction*
$(p \vee q) \vee r \equiv p \vee (q \vee r)$	*Associativity of disjunction*
$(\neg (\neg p) \equiv p$	*Double Negation elimination*
$p \Rightarrow q \equiv \neg q \Rightarrow \neg p$	*contraposition*
$p \Rightarrow q \equiv \neg p \vee q$	*implication elimination*
$\neg (p \vee q) \equiv (\neg p \wedge \neg q)$	*De-Morgan*
$\neg (p \wedge q) \equiv (\neg p \vee \neg q)$	*De-Morgan*
$(p \Leftrightarrow q) \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$	*Biconditional elimination*
$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	*Distributive of $\wedge$ over $\vee$*
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	*Distributive of $\vee$ over $\wedge$*

Inference Rules

RULE	PREMISE	CONCLUSION
Modus Ponens	$A, A \rightarrow B$	B
And Elimination	$A \wedge B$	A B
And Introduction	A, B	$A \wedge B$
Or Introduction	A	$A_1 \vee A_2 \vee \ldots \vee A_n$
Double Negation Elimination	$\neg \neg A$	A
Hypothetical Syllogism	$P \rightarrow Q, Q \rightarrow R$	$P \rightarrow R$

In the case of modus ponens, if A is true and $A \rightarrow B$ is true, then we know that B is true.

Example

- *E.g. Given the following facts and relationship between facts; What can we say about the weather condition?*
  - *It is humid:*
  - *If it is humid, then it is hot :*
  - *If it is hot and humid, then it is raining:*

1. Q
2. $Q \rightarrow P$
3. $(P \wedge Q) \rightarrow R$
4. *(using Modes Ponens on 1 & 2)* P
5. *(using AND introduction on 1 & 4)* $P \wedge Q$

Propositional logic is a weak language

- PL cannot handle even a domain with small worlds. The problem is that there are just too many propositions to handle since it only has one representational device: the **proposition**
- In PL world consists ***of just facts***. **It is hard to** :
  - **Identify individuals:** E.g., Mary, 3
  - **Describe properties of** (or relations between) **individuals.** E.g. Belete is taller than Gelaw
  - **Generalize for a given universe**. E.g., all triangles have 3 sides

- Example: Prove that "my dog Fido is Nice, given that "all dogs are Nice."
  - This requires to get at the structure and meanings of statements (where **FOL is useful).**

First Order Logic

- **First-Order Logic (FOL)** is expressive enough to concisely represent any kind of situation that are expressed in natural language.
  - FOL represents objects and relations between objects, variables, and quantifiers **in addition to propositions**
  - **Also k.as predicate logic.**

Components of FOL

- **Constants symbol**
  - names (like Yonas, Kebede, …), numbers (like 1, 2, … n), …
- **Predicates**:
  - Predicates ***used to relate one object with another***. E.g. brother, >, ,…

  Eg greater(5,3)
- **Functions**: ***Returns value*** (Sqrt, mother-of, color-of,…) eg father-of(Mary) = John
- **Variables**: X, Y, A, B,…
  - Important ***to increase generalization capability of KB***
- **Connectives**:
  - retains connectives used in PL ($\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$)
- **Quantifiers**:
  - Quantifiers ***specify whether all or some objects satisfy properties*** or relations between objects
  - Two standard quantifiers**: Universal (**" for all, for every) and **Existential** ("there exists, some")

Components of FOL…

- **Term**
  - Constant, e.g. Red
  - Function of constant, e.g. Color(Block1)

  Term= Constant | Variable | Function(Term,….)

- **Atomic Sentence**
  - ***Predicate relating objects*** (no variable)

    Atomic Sentence= Predicate(term,…….)|Term=Term

    - Brother (John, Richard)
    - Married (Mother(John), Father(John))
- **Complex Sentences**
  - ***Atomic sentences + logical connectives***
    - Brother (John, Richard) $\wedge \neg$Brother (John, Father(John))

Components of FOL…

- **Quantifiers**
  - Each quantifier defines a variable for the duration of the following expression, and **indicates the truth of the expression…**

❖ **Universal quantifier** “for all” $\forall$

- The expression is **true** ***for every possible value*** of the variable
- $\Rightarrow$ is the main connective with $\forall$

Every elephant is gray:

$\forall$ x (elephant(x) $\Rightarrow$ gray(x))

❖ **Existential quantifier** “there exists” $\exists$

- The expression is **true** ***for at least one value*** of the variable
- $\wedge$ is the main connective with $\exists$

There is a white alligator:

$\exists$ x (alligator(X) ^ white(X))

Universal quantification

- **Universal Quantifiers**: makes statements **about every object**

  $\forall$ ***<variables> <sentence>***

  - All cats are mammals:

    $\forall_x \text{cat}(x) \Rightarrow \text{mammal}(x)$

  - Everyone at ASU is smart:

    $\forall_x \text{At}(x,\text{ASU}) \Rightarrow \text{Smart}(x)$

- $\forall_x$ sentence *P* is true iff *P* is true with *x* being each possible object in the given universe
  - The above statement is equivalent to the **conjunction**

    At(Alеazar, ASU) $\Rightarrow$ Smart(Aleazar) $\wedge$

    At(Rawuda, ASU) $\Rightarrow$ Smart(Rawuda) $\wedge$

    ..

    Conjunction

- ***A* common mistake to avoid**
  - Typically, $\Rightarrow$ is the main connective with $\forall$
  - **Common mistake**: the use of $\wedge$ as the main connective with $\forall$:

    $\forall_x$ At(x, ASU) $\wedge$ Smart(x) is true if "Everyone is at ASU & everyone is smart"

Existential quantification

- Makes statements about some objects in the universe

  $\exists$<***variables***> <***sentence*** >

  - There is a student who is smart

    $(\exists x)\ student(x) \wedge smart(x)$

  - Someone at ASU is smart:

    $\exists_x\ At(x,ASU) \wedge Smart(x)$

- $\exists_x$ sentence *P* is true iff *P* is true with *x* being some possible objects in the universe.
  - The above statement is equivalent to the **disjunction**

    $At(Jonas, ASU) \wedge Smart(Jonas) \vee$

    $At(Alemu, ASU) \wedge Smart(Alemu) \vee$ **disjunction**

    .....

- **Common mistake to avoid**
  - Typically, $\wedge$ is the main connective with $\exists$
  - Common mistake: using $\Rightarrow$ as the main connective with $\exists$:

    $\exists_x$ At(x,ASU) $\Rightarrow$ Smart(x) is true if there is anyone who is not at ASU

Nested quantifiers

- $\forall_{x,y}$ **parent(x,y) $\Rightarrow$ child(y,x)**
  - for all x and y, if x is the parent of y then y is the child of x.
- $\exists_x \forall_y$ **Loves(x,y)**
  - There is a person who loves everyone in the given world
- $\forall_y \exists_x$ **Loves(x,y)**
  - Everyone in the given universe is loved by at least one person

Properties of quantifiers

  - **$\forall_x \forall_y$ is the same as $\forall_y \forall_x$**
  - **$\exists_x \exists_y$ is the same as $\exists_y \exists_x$**
  - **$\exists_x \forall_y$ is not the same as $\forall_y \exists_x$**

- **Quantifier duality**: each can be expressed using the other, using negation ($\neg$)

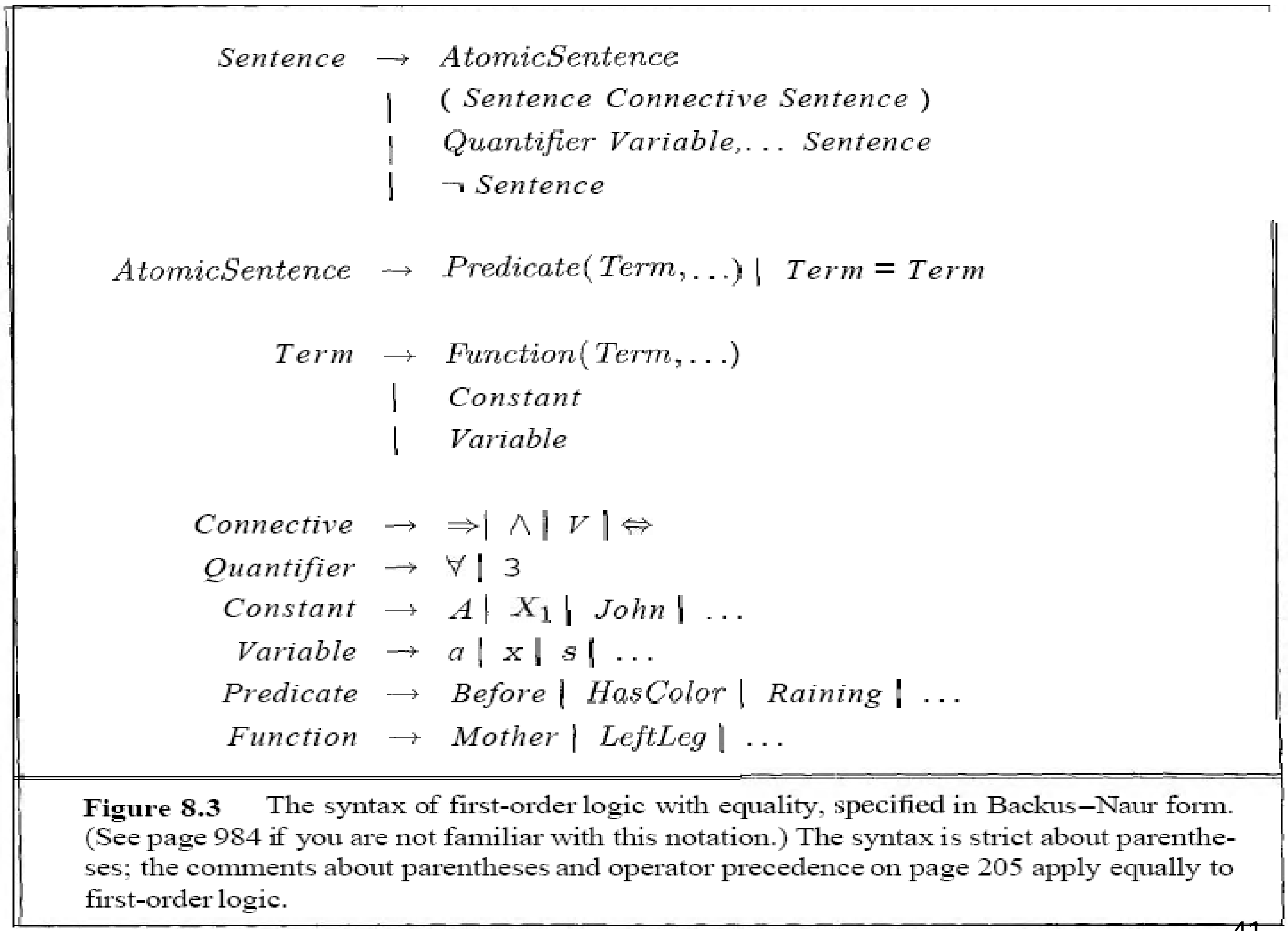
  $\forall_x$ Likes(x,icecream) $\neg\exists_x \neg$Likes(x,icecream)

  - **Everyone likes ice cream** means that there is **nobody who dislikes ice cream**

  $\exists_x$ Likes(x,cake) $\neg\forall_x \neg$Likes(x,cake)

  - **There is someone who likes cake** means that there **is no one who dislikes cake**

Syntax of FOL



$$
\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \\
 & | & (\ \mathit{Sentence}\ \mathit{Connective}\ \mathit{Sentence}\ ) \\
 & | & \mathit{Quantifier}\ \mathit{Variable}, \ldots\ \mathit{Sentence} \\
 & | & \neg\ \mathit{Sentence} \\
 & & \\
\mathit{AtomicSentence} & \rightarrow & \mathit{Predicate}(\mathit{Term}, \ldots) \mid \mathit{Term} = \mathit{Term} \\
 & & \\
\mathit{Term} & \rightarrow & \mathit{Function}(\mathit{Term}, \ldots) \\
 & | & \mathit{Constant} \\
 & | & \mathit{Variable} \\
 & & \\
\mathit{Connective} & \rightarrow & \Rightarrow \mid \wedge \mid \vee \mid \Leftrightarrow \\
\mathit{Quantifier} & \rightarrow & \forall \mid \exists \\
\mathit{Constant} & \rightarrow & A \mid X_1 \mid \mathit{John} \mid \ldots \\
\mathit{Variable} & \rightarrow & a \mid x \mid s \mid \ldots \\
\mathit{Predicate} & \rightarrow & \mathit{Before} \mid \mathit{HasColor} \mid \mathit{Raining} \mid \ldots \\
\mathit{Function} & \rightarrow & \mathit{Mother} \mid \mathit{LeftLeg} \mid \ldots
\end{array}
$$

Figure 8.3 The syntax of first-order logic with equality, specified in Backus–Naur form. (See page 984 if you are not familiar with this notation.) The syntax is strict about parentheses; the comments about parentheses and operator precedence on page 205 apply equally to first-order logic.

Sentence structure

In FOL the basic unit is a **predicate (argument/terms)** structure called sentence to represent facts.

- Terms refer to **objects**

  Example: likes(muhe, chocolate); tall(Silesh)

  – Terms or arguments can be any of: **Constant symbol**, such as 'muhe', **variable symbol**, such as X , **functions**, such as motherof(Silesh), father-of(father-of(john))

- **A predicate** is the one **that says something about the subject**. E.g., There is a red book

  - Subject: color of the book, represented as: $\exists_x$ book(x)$\rightarrow$red(x)

  – **Predicate** also refers to **a particular relation between objects**

    - Example: likes(X, richard)

      friends(motherof(yonas) , motherof(sami))

  – **A predicate statement** takes the value **true** or **false**

Sentences

Atomic sentences: formed from **a predicate symbol** followed by **a parenthesized list of terms**

Atomic sentence = *predicate* ($term_1,\dots,term_n$)

Example: *Brother(John, Richard)*

- *Atomic sentences* ***can have arguments that are complex terms*** *(e.g.* term = *function* ($term_1,\dots,term_n$) *)*

  Example: *married(fatherof(Richard),motherof(John))*

Complex sentences: Complex sentences are made by **combining atomic sentences using connectives:**

$\neg S_1 \quad S_1 \wedge S_2 \quad S_1 \vee S_2 \quad S_1 \Rightarrow S_2 \quad S_1 \Leftrightarrow S_2$

Ex. likes(john, mary) $\wedge$ tall(mary)

tall(john) $\Rightarrow$ handsome(john)

Sibling(John, Richard) $\Rightarrow$ *Sibling(Richard, John)*

- Sentences can also be formed using quantifiers to indicate how to treat variables:
  - Universal quantifier: $\forall$x lovely(x) - Everything is lovely.
  - Existential quantifier: $\exists$x lovely(x) - Something is lovely.

Sentences…

- *Can have several quantifiers, e.g.,*

  $\forall_x \; \exists_y \; loves(x, y)$

  $\forall_x \; handsome(x) \Rightarrow \; \exists_y \; loves(y, x)$

Represent the following in FOL:

- *Everything in the garden is lovely* → $\forall x \; in(x, garden) \Rightarrow lovely(x)$
- *Everyone* ***likes*** *ice cream* → $\forall x \; likes(x, icecream)$
- *Peter has some* ***friends*** → $\exists x \; friends(x, Peter)$
- *Bereket* ***plays*** *the piano or the violin*

  plays(bereket, piano) v plays(bereket, violin)
- *Some* ***people like*** *snakes* - $\exists x(person(x) \land likes(x, snakes))$
- *Fikadu did not* ***write*** *city* – $\neg write(fikadu, city)$
- *Nobody* ***wrote*** *City* – $\neg \exists x \; write(x, city)$
- *Nobody did not* ***write*** *City* $\neg \exists x \; \neg write(x, city)$

Semantics

- There is a precise meaning to expressions in predicate logic.
- Like in propositional logic, it is all about determining whether something is true or false.
- **∀x P(x)** means that P(x) must be true for **every object x** in the *domain of interest.*
- **∃x P(x)** means that P(x) must be true for **at least one object x** in the domain of interest.

- ***Atomic sentence*** is true if **the relation referred by the predicate holds b/n the objects** referred by **the arguments**

  brother(John, Richard)
- The truth value of **complex sentences** depends on **logical connectives used**

  older(John,30) ⇒ ¬younger(John,30)

Proof and inference

- We can define inference rules allowing us to say that if certain things are true, certain other things are sure to be true,

  E.g. All men are mortal

  Aristotle is a man

  using logical inferences we can deduce that: Aristotle is mortal

  $\forall x\ man(x) \Rightarrow mortal(x)$

  man(Aristotle)

  so we can conclude that: mortal(Aristotle)

  - This involves matching man(x) against man(Aristotle) and **binding the variable x to Aristotle**.

Unification

- Unification is an algorithm for determining the substitutions needed to make two expressions match
  - Unification is a "**pattern matching**" procedure that **takes two atomic sentences** as input, and returns the **most general unifier**, i.e., a shortest length substitution list that makes the two literals match.
  - E.g: To make, say p(X, X) and p(Y, Z) match, subst(X/ Y) or subst(X/ Z)

- Note: It is possible to substitute **two variables with the same value**, **but not the same variables with different values**.

- **Example**

Sentence 1	**Sentence 2**	**Unifier**
group(x, cat(x), dog(Bill))	group(Bill, cat(Bill), y)	{x/Bill, y/dog(Bill)}
group(x, cat(x), dog(Bill))	group(Bill, cat(y), z)	{x/Bill, y/Bill, z/dog(Bill)}
group(x, cat(x), dog(Jane))	group(Bill, cat(y), dog(y))	Failure

Unification…

- A variable can never be replaced by a term containing that variable. For example, x/f(x) is illegal.
- Unification and inference rules allows us to make inferences on a set of logical assertions. To do this, the logical database must be expressed in an appropriate form
- A substitution α unfies atomic sentences p and q if pα = qα

p	q	α
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y, Abe)	{y/John, x/Abe}
Knows(John,x)	Knows(y,Mother(y))	{y/John, x/Mother(John)}
Knows(John,x)	Knows(x,Abe)	Fail

Unify rule: Premises with known facts apply unifier to conclusion

Example. if we know q and Knows(John,x) → Likes(John,x)
then we conclude Likes(John,Jane)
Likes(John,Abe)
Likes(John,Mother(John))

Inference rules

- Rules for PL apply to FOL as well. For example,
  - Modus Ponens
  - Resolution
  - And-Introduction
  - And-Elimination, etc.

- Sound inference rules for use with quantifiers:
  - **Universal Elimination**
  - **Existential Introduction**
  - **Existential Elimination**

Sound inference rule- if a set of premises are all true, any conclusion drawn from those premises must also be true

Sound Inference Rules

- **Universal Elimination:** If $\forall x\, P(x)$ is true, then $P(c)$ is true, where c is a constant in the domain of x.

  Example: $\forall x$ eats(x, IceCream).
  Using the substitution (x/Aleazar) we can infer **eats(Aleazar, Icecream).**
  - The variable symbol can be replaced **by any constant symbol** or **function symbol**.

- **Existential Introduction:** If P(c) is true, then $\exists$**x P(x)** is inferred.

  Example: eats(John, IceCream) we can infer $\exists$**x eats(x, icecream).**
  - *All instances of the given constant symbol* are replaced by **the new variable symbol**.

- **Existential Elimination:** From $\exists x\, P(x)$ **infer P(c).**

  Example: $\exists x$ eats(Sol, x) infer **eats(Sol, Cheese)**
  - Note that the variable x is replaced by a *brand new* constant (like Cheese) that does not occur in this or *any other* sentence in the KB

Inference Mechanisms

- **Inference** is a means of *interpretation of knowledge in the KB* ***to reason*** and ***give advise*** to users query.
- There are **two inference strategies** **to control** and **organize** the steps taken to solve problems:
  - **Forward chaining**: also called **data-driven chaining**
    - It starts with ***facts and rules in the KB*** and try to draw conclusions from the data
  - **Backward chaining**: also called **goal-driven chaining**
    - It starts with ***possible solutions/goals*** and tries to gather information that verifies the solution

Cont….

The choice of strategy depends **on the nature of the problem.**

- **Forward chaining-** starts w' a set of rules and a set of facts and tries to find away of using those rules and facts ***to deduce a conclusion*** or come up with a suitable course of action.
- Forward chaining is the best choice if:
  - ***All the facts are provided with the problem statement;***

or:

  - **There is no any sensible way to guess what the goal** is at the beginning of the consultation.

or:

  - There are **many possible goals.**

Cont….

- **Backward chaining** is the best choice if:
  - The **goal is given** in the **problem statement**,

Or:

  - **can sensibly be guessed what the goal is?** at the beginning of the consultation;

or:

  - The system has been built so that it sometimes asks for pieces of data (e.g. "please now do the gram test on the patient's blood, and tell me the result"), rather than expecting all the facts to be presented to it.

- This is because (especially in the medical domain) the test may be
  - expensive,
  - or unpleasant,
  - or dangerous for the human participant

w/c inference mechanism is better? Why?

so one would want to avoid doing such a test unless there was a good reason for it.

Quiz

1. What are the two main tasks of KE? Discus it.
2. List part of KR language?
3.

What is Prolog?

What Is Prolog?

- Developed in 1972 by **Alain Colmerauer** and **Philippe Roussel**
- PROLOG stands for - "**PROgramming in LOGic**"
- **Prolog is a logic-based language**
  - It is a declarative language
  - Prolog reasons by backward chaining

  It Used in AI applications such as
    - Natural language processing (NLP),
    - automated reasoning systems,
    - Intelligent systems, …
- **Prolog is based on facts, rules, queries, constants and variables.**
  - Facts and rules make up the knowledge base.
  - Constants and variables are used to construct facts, rules and queries.
    - A variable represents some unspecified element of the system
    - A constant represents known, member of the system
  - **Queries drive the search processes**

Prolog program

Prolog files are with extension **.pl,** which contains list of :

- ❖ **Clauses** written in the form of:
  - predicate (argument(s)). Eg. Brother (Endris, Kedir), Male(Endris). .....

– Predicate represents **relations or properties among objects in the system.**

- **There are two types of clauses - facts and rules**
  - Rules are clauses which **contain the ":-" symbol**
  - Facts are clauses which don't.

  – Each fact consists of **just one predicate**

  - Each rule consists of **a predicate, followed by a ":-" symbol,**

  – followed by a **list of predicates separated by "," or ";"**

- In a rule, the predicate before the ":-" is called the **goal/ head** of the rule
  – The predicates coming after the ``:-'' are called **the body of the rule**

❑ **Comments in Prolog**

- Single line comments use **the "%" character**
- Multi-line comments use **/* and */**

Cont'd…

- **FACTS:**
  - male(sam).
  - male(alex).
  - child(sam,alex).

- tasty (cheese).
- made_from (cheese, milk).
- contains (milk, calcium).

Expressed

- Cheese is tasty.
- Cheese is made from milk.
- Milk contains calcium.

❖ **RULES:** combine facts to increase knowledge of the KBS

- son(X,Y) :- male(X),child(X,Y).

The body of the Rule

  - X is a son of Y if X is male and X is a child of Y

Goal/Head of the Rule

- son(X,Y) :- father(Y,X), male(X).
- son(X,Y) :- mother(Y,X), male(X).

- NOTE:
  - Variables always start with an **upper-case letter** or an **under score** (X and Y **are variables**)
  - Predicates and constants always start with **a lower-case letter** or **digit (**sam and alex are **constants)**
  - male, child and son **are predicates**
- – Every clause is terminated by a "." **(full-stop)**.

Cont'd…

- We "define a predicate" by writing down a number of clauses which have that predicate at their head
  - The order in which we write these down is important
  - Any predicates mentioned in the body must either:
    - – be defined somewhere else in the program, or
    - – be one of Prolog's "built-in" predicates.
- Defining a predicate in Prolog corresponds roughly to defining a procedure
- Predicates occurring in the body of a clause correspond roughly to procedure calls

Note also that:

- Constants and variables will never appear "on their own" in
  - a clause. They can only appear as the "arguments" to some predicate.
  - Predicates will (almost) **never appear as arguments** to another predicate

Cont'd…

❖Queries

The user requests more answers, typing semicolon

- A query is our "**goal**", and Prolog has **to infer the solution** to the query **from the knowledge base**
- Ask Prolog questions composed at the **prompt: ?-**

```
?- son(sam, alex).
yes
?- child(alex, sam).
No
```

If the fact is in the DB(if it can prove it)

If the fact is absent in the DB

The variable

```
?- Born(alex, X).
```

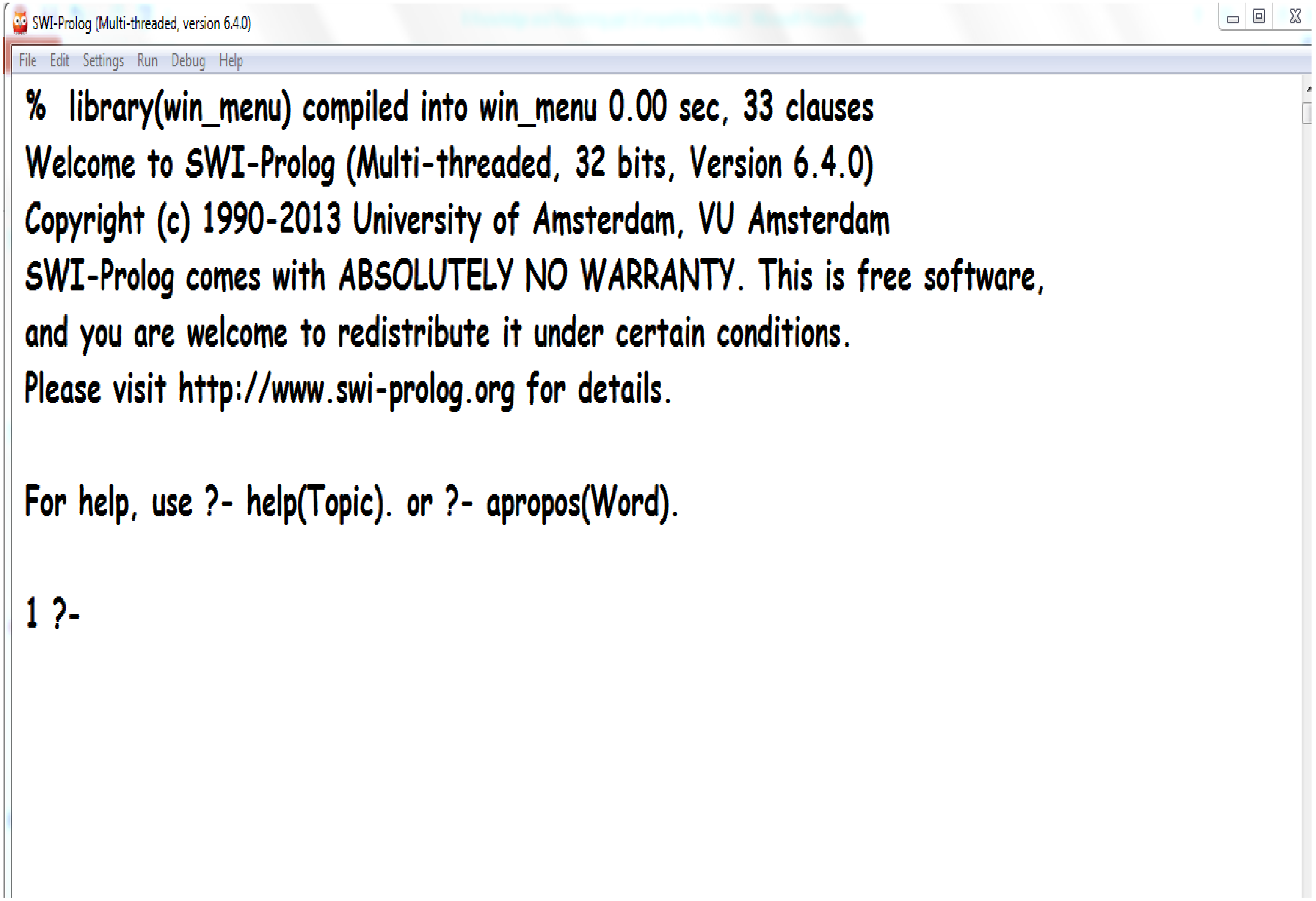
Query typed by the user

```
X=1989;
```

Prolog answers, unifying X with a value

Note that solving a query results in either: **failure**, in which case "no" is printed out, or **success**, in which case "yes" is displayed, or all sets of values satisfying the variables in the goal are printed out

Prolog Environment


SWI-Prolog (Multi-threaded, version 6.4.0)
File Edit Settings Run Debug Help

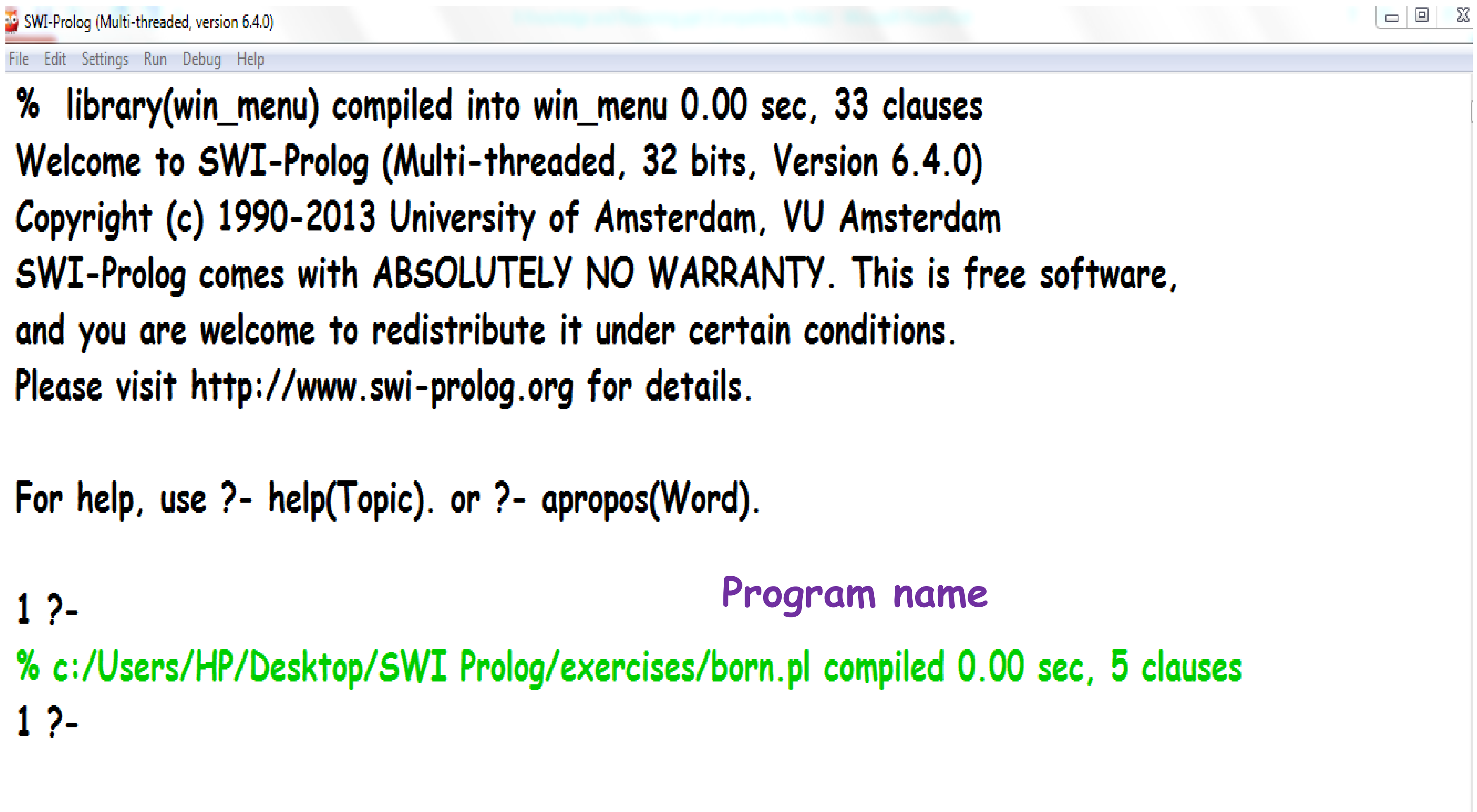
```
% library(win_menu) compiled into win_menu 0.00 sec, 33 clauses
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 6.4.0)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?-
```

Consult

Consult- load a program into prolog through the interpreter


SWI-Prolog (Multi-threaded, version 6.4.0)
File Edit Settings Run Debug Help
Program name

```
% library(win_menu) compiled into win_menu 0.00 sec, 33 clauses
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 6.4.0)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?-
% c:/Users/HP/Desktop/SWI Prolog/exercises/born.pl compiled 0.00 sec, 5 clauses
1 ?-
```

Logical connectives

Syntax

PL/FOL	Prolog
^	,
V	;
→	: -
¬	not

Example:

PL/FOL	Prolog
Hot ^ wet → humid	humid : - hot, wet.
(¬ humid v cool) → pleasant	pleasant : - humid; cool.
¬ likes(ted,chips) → Likes(yared, chips)	likes(yared, chips) : - not (likes (ted,chips)).
¬ likes(ted,x) → likes(sam, x)	likes(sam, X) : - not (likes (ted,X)).
male(X) ^ child(X,Y) → son(X,Y)	son(X,Y):- male(X), child(X,Y).

Example

Consider the following facts and rules about food composition Molalign likes.

- Molalign likes chips.
- Macaroni contains cheese.
- Lasagna contains cheese.
- Lasagna contains meat.
- There is Chinese noodles.
- Molalign likes Chinese foods.
- Molalign likes all foods that contain cheese and meat.

Exercise

Syllogism

Socrates is a man.

All men are mortal.

Is Socrates mortal?

Prolog

```
man(socrates).

mortal(X) :- man(X).

?- mortal(socrates).
```

Cont...

- To express this example in Prolog we must:
- ❖ Identify **the entities** (or **actual objects**), mentioned in the description
- ❖ Identify **the types of properties that things can have**, as well as the **relations** ***that can hold between these things***
- ❖ **Figure out which properties/relations** hold for which entities
- ▪ There is really no unique way of doing this; we must decide the best way to structure our data (based on what we want to do with it).

- ▪ We will choose the following:
  - **Objects/arguments**: molalign, cheese, Food, lasagna
  - **Relations/predicates**: "... likes ...", "... contains..."
- ▪ **Constructing our knowledge base** then consists of writing down which properties and relationships hold for which objects in the form of facts and rules as follows:

Prolog

Molalign likes chips.
Macaroni contains cheese.
Lasagna contains cheese.
Lasagna contains meat.
There is Chinese noodles.
Molalign **likes** Chinese **foods**.
Molalign **likes** all **foods** that contain cheese and meat.

```
likes (molalign, chips).
contains (macaroni, cheese).
contains (lasagna, cheese).
contains (lasagna, meat).
chinese(noodles).

likes (molalign, Food) :-   chinese(Food).
likes (molalign, Food) :-  contains (Food,
    cheese),
                           contains (Food, meat).
```

Query:

```
?- likes(molalign ,chips).
?- likes(molalign ,X).
```

Prolog…

- To run prolog programs we need to give it a query to prove
  - A query has exactly the same format as a **clause body**: **one or more predicates, separated by "," or ";"**, terminated by **a full-stop.**

Example…

- Let us consider the following description of a “system”;
    - Negasi likes every toy he plays with. A doll is a toy. A train is a toy. Negasi plays with trains. Aleazar likes everything Negasi likes.

    - Build A KB for the following description?
- **Knowledge base**

```
likes(negasi,X) :- toy(X), plays(negsai,X).
toy(doll).
toy(train).
plays(negasi,train).
likes(aleazar,Y) :- likes(negasi,Y).
```

Individual Assignment(Next Class)

- Translate the following sentences into Prolog:
  - Gebeyehu eats all kinds of food. Apples are food. Hens are food. Anything anyone eats is food. Balemlay eats goats. Mekides eats everything that Balemlay eats.

- Save the program in a file called **food.pl**. Now read them into Prolog, and formulate queries to find out:

A. What Gebeyehu eats
B. What Mekides eats
C. Is there anything which both Gebeyehu and Mekides eat.
D. Who eats goats

I/O in Prolog

- Use the write statement:
  - Prolog syntax: write('hello').
    - Example: write('Hello'), write('World').
- Use a Newline
  - write('hello'), nl, write('World').
- Reading a value from stdin
  - Prolog Syntax: read(X).
    - Example: read(X), write(X).

Arithmetic

- **Predefined** basic arithmetic operators.
  - Operators
    - + - * / // mod ** **(Basic Arithmetic Operators)**
    - < =< ==(=:=) !=(=\=) => > **(Comparison Operators)**
    - Arithmetic is done via evaluation then unification
- Arithmetic Example: X is Y
  - compute Y then unify X and Y
    - X is Y * 2
    - N is N - 1

Operator '**is**' is a built-in procedure.

- Prolog implementations usually also provide standard functions such as **sin(X)**, **cos(X)**, **atan(X)**, **log(X)**, **exp(X)**, etc.

Arithmetic Basics

- **X == Y**

  –This is the identity relation. In order for this to be true, X & Y must both be identical variables (i.e. have the same name), or both be identical constants, or both be identical operations applied to identical terms.

- **X = Y**

  – This is unification. It is true if X is unifiable with Y

- **X is Y**

  – This means compute Y and then unify X and Y

  – Y must be an arithmetic expression; X can either be an arithmetic expression (of the same form), or a variable

Example

- X = 2, Y is X+1
- X = 2, Y = X+1
- X = 2, Y == X+1
- X = 2, Y is X*1
- X = 2, Y is X-1

Project

- Select an area (may be agriculture, health, law, education, etc.) which ever is suitable for you.
  - Extract the explicit knowledge from documents and also tacit knowledge from human experts and represent it using FOL knowledge representation technique
  - Use prolog to develop a knowledge base that can provide suggestion as per users query