KNOWLEDGE REPRESENTATION

1st CLASS / AI BRANCH

م. سُــرى محمــود عبـدالله

Email: - 110050@uotechnology.edu.iq

Google Site: **Google scholar**: https://scholar.google.com/citations?user=ep015F0AAAAJ&hl=en

Research gate: https://www.researchgate.net/profile/Sura_Abdullah

Knowledge Representation

Knowledge representation (KR) is a part of the Artificial Intelligence which responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real-world problems such as communicating with humans in natural language.

It is also a way which describes how we can represent knowledge in Artificial Intelligence. Knowledge representation is not just storing data into database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human. There are many methods can be used for knowledge representation and some of them can be described as follow:-

1. Propositional Logic
2. Predicate Logic
3. Semantic Network
4. Conceptual Graph
5. Frame Representation
6. Script Representation

1- Prepositional Calculus (Logic)

Propositional Calculus Symbols

The symbols of propositional calculus are {P, Q, R, S, …}
Truth symbols: **{True, False}**
Connective tools: **{**∧, ∨, ┐, →, ≡**}**

Propositional symbols denote ***propositions*** or **statements** about the world that may be either true or false, Propositions are denoted by uppercase letters near the end of the English alphabet letters.

For example:
P: It is sunny today.
Q: The sun shines on the window.
R: The blinds are down.
(P→Q): If it is sunny today, then the sun shines on the window
(Q→R): If the sun shines on the window, the blinds are brought down.
(┐R): The blinds are not yet down.

Propositional Calculus Sentence

- Every propositional symbol and truth symbol is a sentence.

For example: **True, P, Q,** and **R** are sentences**.**

- The ***negation*** of a sentence is a sentence.

For example: ¬ **P** and ¬ **False** are sentences.

- The ***conjunction***, **AND**, of two sentences is a sentence.

For example: **P** ∧ ¬ **P** is a sentence.

- The ***disjunction***, **OR** of two sentences is a sentence.

For example: **P** ∨ ¬ **P** is a sentence.

- The ***implication*** of one sentence from another is a sentence.

For example: **P** → **Q** is a sentence.

- The ***equivalence*** of two sentences is a sentence.

For example: **P** ∨ **Q** ≡ **R** is a sentence.

- Legal sentences are also called ***well-formed formulas*** or ***WFF****s.*

In expressions of the form **P ∧ Q, P** and **Q** are called the ***conjuncts***. In **P ∨ Q**, **P** and **Q** are referred to as ***disjuncts***. In an implication, **P → Q, P** is the ***premise*** and ***Q***, the ***conclusion*** or ***consequent.***

In propositional calculus sentences, the symbols **( )** and **[ ]** are used to group symbols into sub-expressions and so to control their order of evaluation and meaning.

For Example:

(P ∨ Q) ≡ R is quite different from **P ∨ (Q ≡ R)** as can be demonstrated using truth tables. An expression is a sentence, or well- formed formula, of the propositional calculus if and only if it can be formed of legal symbols through some sequence of these rules.

For Example: **((P ∧ Q) → R) ≡ ¬ P ∨ ¬ Q ∨ R** is a well-formed sentence in the propositional calculus because:

P, Q, and **R** are propositions and thus sentences.

P ∧ Q, the conjunction of two sentences, is a sentence.

(P ∧ Q) → R, the implication of a sentence for another, is a sentence.

¬ P and **¬ Q**, the negations of sentences, are sentences.

¬ P ∨ ¬ Q the disjunction of two sentences is a sentence.

¬ P ∨ ¬ Q ∨ R, the disjunction of two sentences, is a sentence.

((P ∧ Q) → R) ≡ ¬ P ∨ ¬ Q ∨ R, the equivalence of two sentences, is a sentence.

This is our original sentence, which has been constructed through a series of applications legal rules and is therefore "***well formed***".

Propositional Calculus Semantics

An ***interpretation*** of a set of propositions is the assignment of truth value, either **T** or **F**, to each propositional symbol. The symbol **True** is always assigned **T**, and the symbol **False** is assigned **F**.

The interpretation or truth value for sentences is determined by:

- The truth assignment of ***negation***, ¬ **P**, where **P** is any propositional symbol, is **F** if the assignment to **P** is **T**, and **T** if the assignment to **P** is **F**.
- The truth assignment of ***conjunction***, ∧, is **T** only when both **conjuncts** have truth value **T**; otherwise, it is **F**.
- The truth assignment of ***disjunction***, ∨, is **F** only when both **disjuncts** have truth value **F**; otherwise, it is **T**.
- The truth assignment *of* ***implication***, →, is **F** only when the premise or symbol before the implication is **T** and the truth value of the consequent or symbol after the implication is **F**; otherwise, it is **T**.
- The truth assignment of ***equivalence***, ≡, is **T** only when both expressions have the same truth assignment for all possible interpretations; otherwise, it is **F**.

- The truth assignments of compound propositions are often described by ***truth tables***. A **truth table** contains all possible truth value assignments to the atomic propositions of expression and gives the truth value of the expression for each assignment. Thus, a truth table enumerates all possible worlds of interpretation that may be given to an expression.
- **For Example**, the truth table for **P ∧ Q**, **Fig. (A)** , lists truth values for each possible truth assignment of the operands.

P	Q	P ∧ Q
T	T	T
T	F	F
F	T	F
F	F	F

- **P ∧ Q** is true only when both **P** and **Q** are both **T**. **Or (∨), not(¬), implies (→)**, and **equivalence (≡)** is defined in a similar fashion. The construction of these truth tables is left as an exercise. Two expressions in the propositional calculus are equivalent if they have the same value under all truth value assignments.
- This equivalence may be demonstrated using truth tables. **For example**, a proof of the equivalence of **P → Q** and **¬ P ∨ Q** is given by the truth table **Fig. (B).**

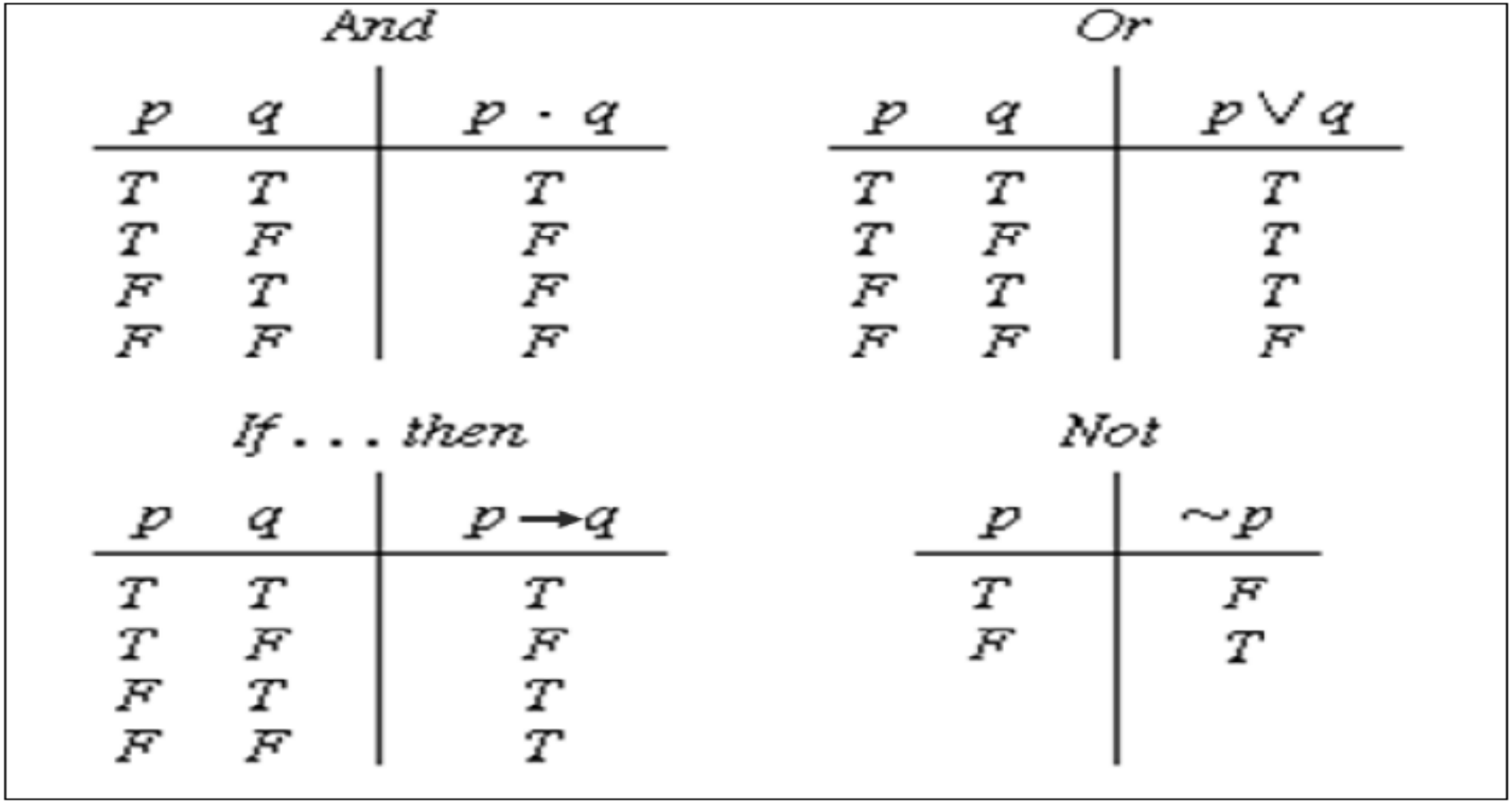
P	Q	¬P	P→Q	¬P∨Q	(¬P∨Q) ≡(P→Q)
T	T	F	T	T	T
T	F	F	F	F	T
F	T	T	T	T	T
F	F	T	T	T	T

Fig. (B) : Truth table demonstrating the equivalence of (¬ P∨Q) ≡ (P→Q)

By demonstrating that two different sentences in the propositional calculus have identical truth tables, we can prove the following equivalences. For propositional expressions **P, Q,** and **R**:

- The double negation law : $\neg\ (\neg P) \equiv P$
- The implication in terms of $\vee$ law: $P \rightarrow Q \equiv \neg P \vee Q$
- The contrapositive law: $(P \rightarrow Q) \equiv (\neg Q \rightarrow \neg P)$
- De Morgan's law: $\neg (P \vee Q) \equiv (\neg P \wedge \neg Q)$ and $\neg (P \wedge Q) \equiv (\neg P \vee \neg Q)$
- The commutative laws: $(P \wedge Q) \equiv (Q \wedge P)$ and $(P \vee Q) \equiv (Q \vee P)$
- The associative law: $((P \wedge Q) \wedge R) \equiv (P \wedge (Q \wedge R))$
- The associative law: $((P \vee Q) \vee R) \equiv (P \vee (Q \vee R))$
- The distributive law: $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$
- The distributive law: $P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$

Truth table then will list all possible truth value assignments to the propositions of expression, the standard truth tables are shown in the figure below:



And

p	q	$p \cdot q$
T	T	T
T	F	F
F	T	F
F	F	F

Or

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

If . . . then

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Not

p	$\sim p$
T	F
F	T

Example: Use a truth table to list all possible truth value assignments to the propositions of the expression **$(P \wedge Q) \vee (\neg Q \vee P)$.**

- **Answer:**

P	Q	$P \wedge Q$	$\neg Q$	$\neg Q \vee P$	$(P \wedge Q) \vee (\neg Q \vee P)$
T	T	T	F	T	T
T	F	F	T	T	T
F	T	F	F	F	F
F	F	F	T	T	T

Example: Prove that **(P ∧ Q)** is not equivalent to **(P → Q)**; in other word prove **(P∧ Q) ≢ (P → Q)**

Answer:

P	Q	(P∧ Q)	(P → Q)	(P∧ Q) ≢ (P → Q)
T	T	T	T	T
T	F	F	F	T
F	T	F	T	F
F	F	F	T	F

Example: Suggest a propositional logic expression that is equivalent to **X** in the following truth table.

P	*Q*	X
T	T	T
T	F	T
F	T	T
F	F	T

Sol:

$(P \wedge Q) \rightarrow Q$

P	*Q*	$(P \wedge Q) \rightarrow Q$
T	T	T
T	F	T
F	T	T
F	F	T

Example: *Represent the following knowledge using the propositional logic method.*

- It is hot *(fact)*

  P

- It is not hot *(fact)*

  ¬ P

- If it is raining, then will not go to the mountain *(rule)*

  P → ¬Q

- The food is good and the service is good *(fact)*

  X ∧ Y

- If the food is good and the service is good then the restaurant is good *(rule)*

  X ∧ Y → Z

Example: ***Represent the following knowledge using the propositional logic method.***

“If it is sunny today, then the sun shines on the screen. If the sun shines on the screen, the blinds are brought down. The blinds are not down. Is it sunny today?”

Sol:

P= it is sunny today.

Q= the sun shines on the screen.

R= blinds are brought down.

$P \rightarrow Q$.

$Q \rightarrow R$.

$\neg R$.

P?

- The propositional calculus has a **limitation** that it cannot deal properly with general statements because it represents each statement by using some symbols jointed with connectivity tools.

2- Predicate Logic (Calculus) (Also known as First-Order Logic)

To solve the limitation in the propositional calculus, you need to analyze propositions into predicates and arguments and deal explicitly with quantification. *Predicate Logic* provides formalism for performing this analysis of propositions and additional methods for reasoning with quantified expressions.

For example, instead of letting a single prepositional symbol, **P**, denotes the entire sentence "**it rained on Tuesday**“, we can create predicate **weather** that describes a relationship between a **date** and the **weather**:

rain (weather, tuesday)

through inference rules, we can manipulate predicate calculus expression accessing their individual components and inferring new sentences. Predicate calculus also allows expressions to contain variables. Variables let us create general assertions about classes of entities. For example, we could state that for all values, of X, where X is a day of the week, the statement:

rain (weather, X) is true ;

I.e., "**it rains every day**". As with propositional calculus, we will first define the syntax of the language and then discuss its semantics.

Example: *Represent the following knowledge using the predicate logic method:*

1-Facts

1- Maha is a girl
girl(maha)

2- I have a book
have (i, book).

3- Ali is a brave man
brave (ali,man).
brave (ali) ^ man (ali)

4- Ali has red car
has (ali,car) ^ color(car,red)

5- This is sunny day

sunny(day).

6- Maha has 4 books

has(maha, book) ^ number (book,4)

7- Ali going to school now

go(ali, school) ^ time(now)

8- I have one or two books

have (i, book) ^ (number(book,1) ∨ number(book,2))

have (i, book) ^ (number(book,1) ∨ (book,2)) Wrong

9- Nobody likes taxies

¬ ∃X like(X, taxi)

¬ ∀X like(X, taxi) Wrong

10- There is a person who writes computer class

∃X write(X, computer class)

11- John did not study but he is lucky

¬ study(john) ∧ lucky(john)

12- Either Jack or Sarah wrote homework.

write (jack, homework) ∨ write (sarah, homework)

13- Animals either male or female.

∀X male (X) ∨ female (X)

Example: ***Represent the following knowledge using the predicate logic method:***

2- Rules

1- If its winter then it is cold

winter(weather) → cold(weather)

2- When I'm sick, I will go to the doctor

sick(i) → go (i, doctor)

3- If student will read well, he will pass

read(X, well) → pass(X).

4- Ahmed goes to school when he is 6 years old

age (ahmed,6) → go(ahmed, school).

5- If it is raining, tom will not go to the mountain

$rain(weather) \rightarrow \neg go(tom, mountain)$

6- All basketball players are tall

$\forall X \quad play(X, basketball) \rightarrow tall(X)$

7- John likes anyone who likes books

$like(X, book) \rightarrow like(john, X)$

8- All dogs are animals

$\forall X \quad dog(X) \rightarrow animal(X)$

9- All cats and dogs are animals

$\forall X \forall Y \quad cat(X) \wedge dog(Y) \rightarrow animal(X) \wedge animal(Y)$

$\forall X \forall Y \quad cat(X) \wedge dog(Y) \rightarrow animal(X \wedge Y)$ Wrong

10- Ali eats anything john eats

$$eat(john, X) \rightarrow eat(ali, X)$$

11- John playing well therefore he will win the game

$$play(john, well) \rightarrow win(john, game)$$

12- There are no two adjacent countries have the same color.

$$\forall X \forall Y\ (county(X) \wedge county(Y) \wedge adjacent(X,Y)) \rightarrow \neg (color(X) \equiv color(Y)).$$

13- All blocks supported by blocks that have been moved have also been moved.

$$\forall X \forall Y\ block(X) \wedge block(Y) \wedge support(X,Y) \wedge move(X) \rightarrow move(Y)$$

Resolution Theorem Proving

Resolution is a technique for proving theorems in the predicate logic using the ***resolution by refutation algorithm***. The resolution refutation proof procedure answers a query or deduces a new result by reducing the set of clauses to a contradiction.

The **Resolution by Refutation Algorithm** includes the following steps:-

a) Convert the statements to **predicate calculus** (predicate logic).

b) Convert the statements from **predicate calculus** to **clause form**.

c) Add the negation of what is to be proved to the clause form.

d) Resolve the clauses to producing new clauses and producing a contradiction by generating the empty clause.

Clause Form

The statements that produced from **predicate calculus** method are nested and very complex to understand, so this will lead to more complexity in resolution stage, therefore the following steps are used to convert the **predicate calculus** to **clause form**:-

ملاحظة/ يتم تطبيق الخطوات الخاصة Clause form (9 خطوات) بالتسلسل لكون الخطوة اللاحقة تعتمد على نتيجة الخطوة السابقة.

1. **Eliminate (→)** by replacing each instance of the form **(P → Q)** by expression **(ך P ∨ Q)**

2. **Reduce the scope of negation.**
   - ך(ךa) ≡ a
   - ך(∀X) b(X) ≡ ∃X ך b(X)
   - ך(∃X) b(X) ≡ ∀X ך b(X)
   - ך(a∧b) ≡ ךa ∨ ךb
   - ך(a∨b) ≡ ךa ∧ ךb

3. **Standardize variables**: rename all variables so that each quantifier has its own unique variable name. ***For example***,
   - ∀X a(X) ∨ ∀X b(X) ≡ ∀X a(X) ∨ ∀Y b(Y)

4. **Move all quantifiers to the left** without changing their order. ***For example*****,**
   - ∀X a(X) ∨ ∀Y b(Y) ≡ ∀X ∀Y a(X) ∨ b(Y)

5. **Eliminate existential quantification** by using the equivalent function. ***For example***,
   - ∀X ∃Y mother(X,Y) ≡ ∀X mother(X,m(X))
   - ∀X ∀Y ∃Z p(X,Y,Z) ≡ ∀X ∀Y p(X,Y, f(X,Y))

6. **Remove universal quantification** symbols. ***For example***,
   - ∀X ∀Y p(X,Y, f(X,Y)) ≡ p(X,Y, f(X,Y))

7. **Use the associative and distributive properties** to get a conjunction of disjunctions called **conjunctive Normal Form (CNF)** . ***For example***,
   - $a \vee (b \vee c) \equiv (a \vee b) \vee c$
   - $a \wedge (b \wedge c) \equiv (a \wedge b) \wedge c$
   - $a \vee (b \wedge c) \equiv (a \vee b) \wedge (a \vee c)$
   - $a \wedge (b \vee c) \equiv (a \wedge b) \vee (a \wedge c)$

8. **Split each conjunct** into a separate clause. ***For example***,
   - $(\neg a(X) \vee \neg\ b(X) \vee e(W)) \wedge (\neg b(X) \vee \neg d(X,f(X)) \vee e(W))$
     - ✓ $\neg a(X) \vee \neg\ b(X) \vee e(W)$
     - ✓ $\neg b(X) \vee \neg d(X,f(X)) \vee e(W)$

9. **Standardize variables** again so that each clause contains variable names that do not occur in any other clause. ***For example***,
   - $(\neg a(X) \vee \neg\ b(X) \vee e(W)) \wedge (\neg b(X) \vee \neg d(X,f(X)) \vee e(W))$
     - ✓ $\neg a(X) \vee \neg\ b(X) \vee e(W)$
     - ✓ $\neg b(Y) \vee \neg d(Z,f(Z)) \vee e(V)$

Example1: ***Use the Resolution Algorithm for proving that John is happy with regard the following story:***

Everyone passing his AI exam and winning the lottery is happy. But everyone who studies or lucky can pass all his exams. John did not study but he is lucky. Everyone who is lucky wins the lottery.

Solution:

A. Convert all statements to predicate calculus.

- ∀X pass(X,ai_exam)∧win(X,lottery)→happy(X)
- ∀X study(X) ∨ lucky(X)→ ∀E pass(X,E)
- ¬study(john) ∧ lucky(john)
- ∀X lucky(X) →win(X,lottery)
- happy (john) **Required to prove it**

B. Convert the statements from predicate calculus to clause form.

$\forall X$ pass (X,ai_exam) $\wedge$ win(X,lottery) $\rightarrow$ happy(X)

$\forall X$ study(X) $\vee$ lucky(X) $\rightarrow$ $\forall E$ pass (X,E)

$\neg$ study(john) $\wedge$ lucky(john)

$\forall X$ lucky(X) $\rightarrow$ win(X,lottery)

1. Remove ($\rightarrow$)

- $\forall X$ $\neg$(pass(X,ai_exam) $\wedge$ win(X,lottery)) $\vee$ happy(X)
- $\forall X$ $\neg$(study(X) $\vee$ lucky(X)) $\vee$ $\forall E$ pass(X,E)
- $\neg$study(john) $\wedge$ lucky(john)
- $\forall X$ $\neg$(lucky(X)) $\vee$ win(X,lottery)

2. Reduce $\neg$

- $\forall X$ ($\neg$pass(X,ai_exam) $\vee$ $\neg$ win(X,lottery)) $\vee$ happy(X)
- $\forall X$ ($\neg$study(X) $\wedge$ $\neg$lucky(X)) $\vee$ $\forall E$ pass(X,E)
- $\neg$study(john) $\wedge$ lucky(john)
- $\forall X$ $\neg$lucky(X) $\vee$ win(X,lottery)

3. Standardize Variables

- ∀X (¬pass(X,ai_exam)∨¬win(X,lottery))∨ happy(X)
- ∀Y (¬study(Y)∧¬lucky(Y))∨ ∀E pass(Y,E)
- ¬study(john)∧lucky(john)
- ∀Z ¬lucky(Z)∨win(Z,lottery)

4. Move all quantifiers to the left

- ∀X (¬pass(X,ai_exam)∨¬win(X,lottery))∨ happy(X)
- ∀Y ∀E (¬study(Y)∧¬lucky(Y))∨ pass(Y,E)
- ¬study(john)∧lucky(john)
- ∀Z ¬lucky(Z)∨win(Z,lottery)

5. Remove ∃

- Nothing to do here.

6. Remove ∀

- (¬pass(X,ai_exam) ∨ ¬win(X,lottery))∨ happy(X)
- (¬study(Y)∧¬lucky(Y)) ∨ pass(Y,E)
- ¬study(john) ∧ lucky(john)
- ¬lucky(Z) ∨ win(Z,lottery)

7. CNF

- ¬pass(X,ai_exam)∨¬win(X,lottery)∨ happy(X)
- (¬study(Y)∧¬lucky(Y))∨ pass(Y,E) ≡ **(a∧b)∨c ≡ c∨(a∧b)**

The second statement becomes:

pass(Y,E)∨¬study(Y)∧pass(Y,E)∨¬lucky(Y)

- ¬study(john)∧lucky(john)
- ¬lucky(Z)∨win(Z,lottery)

8. Split ∧

- ┐pass(X,ai_exam)∨┐win(X,lottery)∨ happy(X)
- pass(Y,E)∨┐study(Y)
- pass(Y,E)∨┐lucky(Y)
- ┐study(john)
- lucky(john)
- ┐lucky(Z)∨win(Z,lottery)

9. Standardize Variables

- ┐pass(X,ai_exam)∨┐win(X,lottery)∨ happy(X)
- pass(Y,E)∨┐study(Y)
- pass(M,G)∨┐lucky(M)
- ┐study(john)
- lucky(john)
- ┐lucky(Z)∨win(Z,lottery)

C. Add the negation of what is to be proved to the clause form.

- ך happy(john).

added it to the other six clauses and shown below:

- ךpass(X,ai_exam)vךwin(X,lottery)v happy(X)
- pass(Y,E)vךstudy(Y)
- pass(M,G)vךlucky(M)
- ךstudy(john)
- lucky(john)
- ךlucky(Z)vwin(Z,lottery)
- ך happy(john).

D. Resolve the clauses to producing new clauses and producing a contradiction by generating the empty clause.

- There are two types of resolution, the first one is *backward resolution* and the second is *forward resolution*.

d_1) Backward Resolution

The proving for ***happy(john)*** using **Backward Resolution** is shown as follows:

1. ¬pass(X,ai_exam)∨¬win(X,lottery)∨happy(X)
2. pass(Y,E)∨¬study(Y)
3. pass(M,G)∨¬lucky(M)
4. ¬study(john)
5. lucky(john).
6.¬lucky(Z)∨win(Z,lottery).
7.¬happy(john).

7: ¬happy(john) **1**:¬pass(X,ai_exam)∨¬win(X,lottery)∨ happy(X) {X=john}

8: ¬pass(john, ai_exam)∨¬win(john,lottery) **6**: ¬lucky(Z)∨win(Z,lottery){Z=john}

9: ¬pass(john, ai_exam)∨¬lucky(john) **5**: lucky(john)

10: ¬pass(john, ai_exam) **3**: pass(M,G)∨¬lucky(M) {M=john, G=ai_exam}

11: ¬lucky(john) **5**: lucky(john)

12: □ {the empty clause}

∴ John is happy

d_2) Forward Resolution

The proving for ***happy(john)*** using **Forward Resolution** is shown as follows:

1. ¬pass(X,ai_exam)∨¬win(X,lottery)∨happy(X)
2. pass(Y,E)∨¬study(Y)
3. pass(M,G)∨¬lucky(M)
4. ¬study(john)
5. lucky(john).
6.¬lucky(Z)∨win(Z,lottery).
7.¬happy(john).

1:¬pass(X,ai_exam)∨¬win(X,lottery)∨happy(X) **6**: ¬lucky(Z)∨win(Z, lottery){Z=X}

8: ¬ pass(X,ai_exam)∨happy(X) ∨ ¬lucky(X) **5**: lucky(john) {X=john}

9:¬ pass(john, ai_exam)∨happy(john) **3**: pass(M,G)∨¬lucky(M) {M=john, G=ai_exam}

10: happy(john) ∨ ¬lucky(john) **5**: lucky(john)

11 : happy(john) 7: ¬happy(john)

12: □ {the empty clause}

∴ John is happy

Example2: Given the following Predicate logic statements, prove ∃W ¬ s(W) using Backward resolution:

(1) ∀X [(∀Y s(Y) ∧ v(X, Y)) ⇒ ((∃Z ¬ t(X, Z)) ∧ v(X, X))]
(2) ∀X∀Y s(Y) ⇒ t(X,Y) ∧ v(X,Y)

B. Convert the predicate logic statements to clause form:

Solution:

(1) **∀X [(∀Y s(Y) ∧ v(X, Y)) ⇒ ((∃Z ¬ t(X, Z)) ∧ v(X, X))]**

1-1) **Remove ⇒**: ∀X [¬(∀Y s(Y) ∧ v(X,Y)) ∨ ((∃Z ¬ t(X,Z)) ∧ v(X,X))]

1-2) **Reduce ¬:** ∀X [(∃Y ¬s(Y) ∨ ¬v(X,Y)) ∨ ((∃Z ¬ t(X,Z)) ∧ v(X,X))]

1-3) **Standardize Variables:** Nothing to do here.

1-4) **Move quantifiers:** ∀X ∃Y ∃Z [(¬s(Y) ∨ ¬v(X,Y)) ∨ (¬t(X,Z) ∧v(X,X))]

1-5) **Remove ∃:** ∀X [(¬s(f1(X)) ∨ ¬v(X,f1(X))) ∨ (¬ t(X,f2(X)) ∧ v(X,X))]

1-6) **Remove ∀:** (¬s(f1(X)) ∨ ¬v(X,f1(X))) ∨ (¬ t(X,f2(X)) ∧ v(X,X))

1-7) **CNF:** (¬s(f1(X)) ∨ ¬v(X,f1(X)) ∨ ¬ t(X,f2(X))) ∧(¬s(f1(X)) ∨ ¬v(X,f1(X)) ∨ v(X,X))

1-8) **Split ∧:** ¬s(f1(X)) ∨ ¬v(X,f1(X)) ∨ ¬ t(X,f2(X))
¬s(f1(X)) ∨ ¬v(X,f1(X)) ∨v(X,X)

1-9) **Standardize Variables:**

¬s(f1(X)) ∨ ¬v(X,f1(X)) ∨ ¬ t(X,f2(X))
¬s(f3(X3)) ∨ ¬v(X3,f3(X3)) ∨ v(X3,X3)

(2) ∀N∀M s(M) ⇒ t(N, M) ∧ v(N, M)

2-1) **Remove ⇒:** ∀N∀M ¬s(M) ∨ (t(N, M) ∧ v(N, M))

2-2) 2-3)2-4)2-5) Nothing to do here

2-6) **Remove ∀:** ¬s(M) ∨ (t(N, M) ∧ v(N, M))

2-7) **CNF:** (¬s(M) ∨ t(N, M))∧ (¬s(M) ∨ v(N, M))

2-8) **Split ∧:** ¬s(M) ∨ t(N, M)
¬s(M) ∨ v(N, M)

2-9) **Standardize Variables:**
¬s(M) ∨ t(N, M)
¬s(M1) ∨ v(N1, M1)

After applying the **clause form** method, the **two** sentences become **four** clauses as follows:

- **¬s(f1(X)) ∨ ¬v(X,f1(X)) ∨ ¬ t(X,f2(X))**
- **¬s(f3(X3)) ∨ ¬v(X3,f3(X3)) ∨ v(X3,X3)**
- **¬s(M) ∨ t(N, M)**
- **¬s(M1) ∨ v(N1, M1)**

***C.* Add the negation of what is to be proved ∃W ¬ s(W)**

Thus, **∃W ¬ s(W)** become **s(W)** because **¬(∃W ¬ s(W)) = ∀W s(W) = s(W).** Now we have a new clause (5. **s(W).**)

added to the other four clauses and shown below:

- **¬s(f1(X)) ∨ ¬v(X,f1(X)) ∨ ¬ t(X,f2(X))**
- **¬s(f3(X3)) ∨ ¬v(X3,f3(X3)) ∨ v(X3,X3)**
- **¬s(M) ∨ t(N, M)**
- **¬s(M1) ∨ v(N1, M1)**
- **s(W)**

D. Backward Resolution

1. ¬s(f1(X)) ∨ ¬v(X,f1(X)) ∨ ¬ t(X,f2(X))
2. ¬s(f3(X3)) ∨ ¬v(X3,f3(X3)) ∨ v(X3,X3)
3. ¬s(M) ∨ t(N, M)
4. ¬s(M1) ∨ v(N1, M1)
5. s(W)

5: s(W) 2: ¬s(f3(X3)) ∨ ¬v(X3,f3(X3)) ∨ v(X3,X3) {f3(X3)=W}

6: ¬v(X3,W) ∨ v(X3,X3) 4: ¬s(M1) ∨ v(N1, M1) { N1=X3, M1=W}

7: v(X3,X3) ∨ ¬s(W) 1: ¬s(f1(X)) ∨ ¬v(X,f1(X)) ∨ ¬ t(X,f2(X)) {X=X3, f1(X)=X3}

8: ¬s(W) ∨ ¬s(X3) ∨ ¬ t(X3,f2(X)) 3: ¬s(M) ∨ t(N, M) { N=X3,M=f2(X)}

9: ¬s(W) ∨ ¬s(X3) ∨ ¬s(f2(X)) 5. s(W) {f2(X)=W}

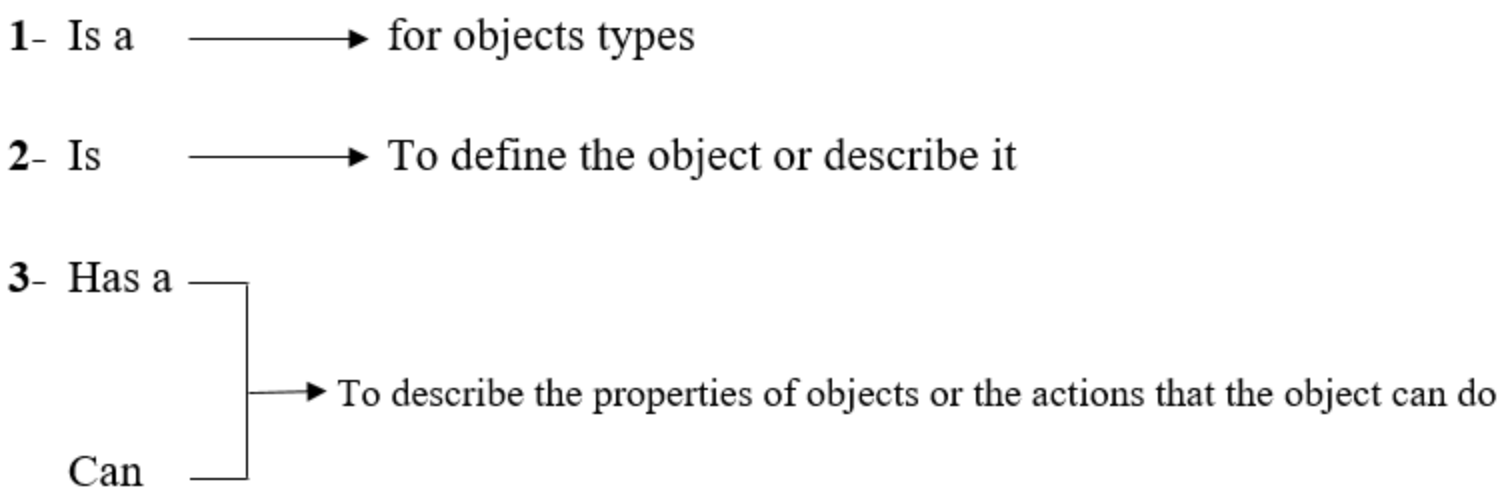
10: ¬s(W) ∨ ¬s(X3) 5. s(W) {X3=W}

11: ¬s(W) 5. s(W)

12: ( ) { Empty clause}

3- Semantic Network

- The semantic network consists of a set of nodes and arrows; each node is represented as a rectangle to describe the objects, concepts and the events. The arrows are used to connect the nodes, these arrows are divided into three types:-

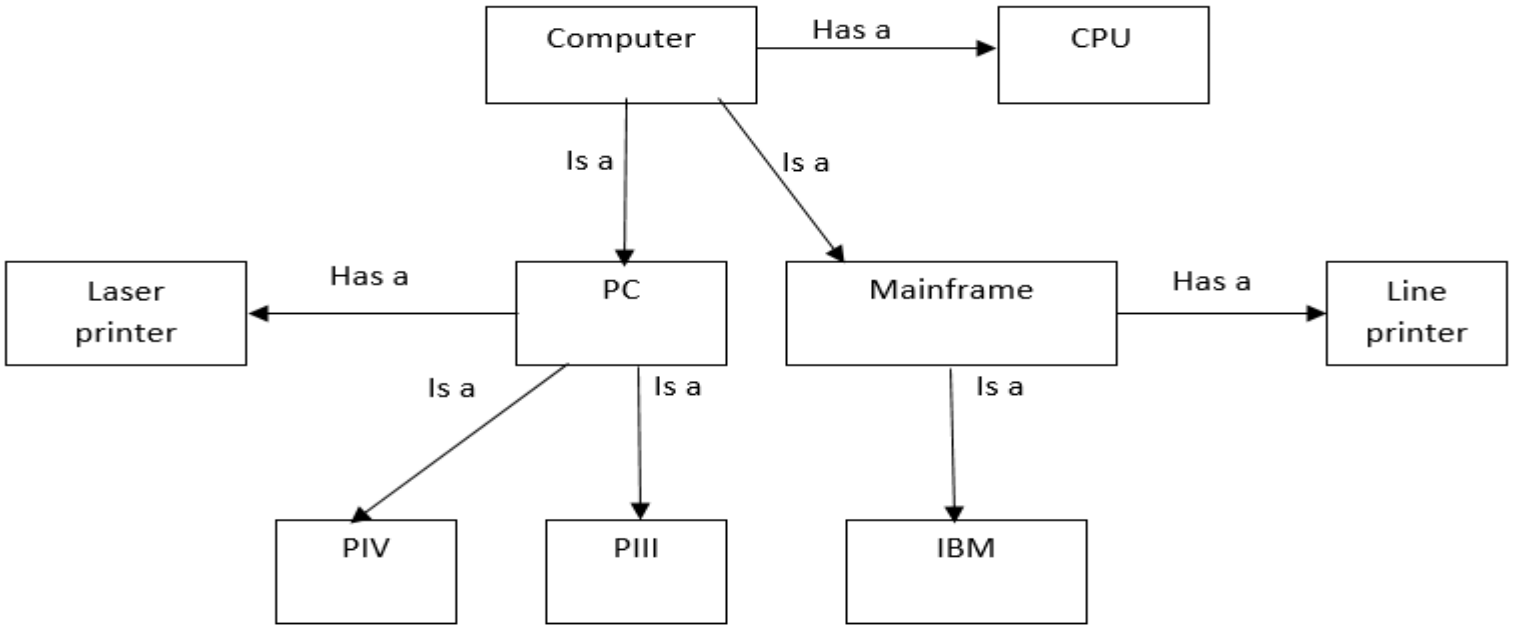


لتمثيل الأفعال والأحداث والكائنات

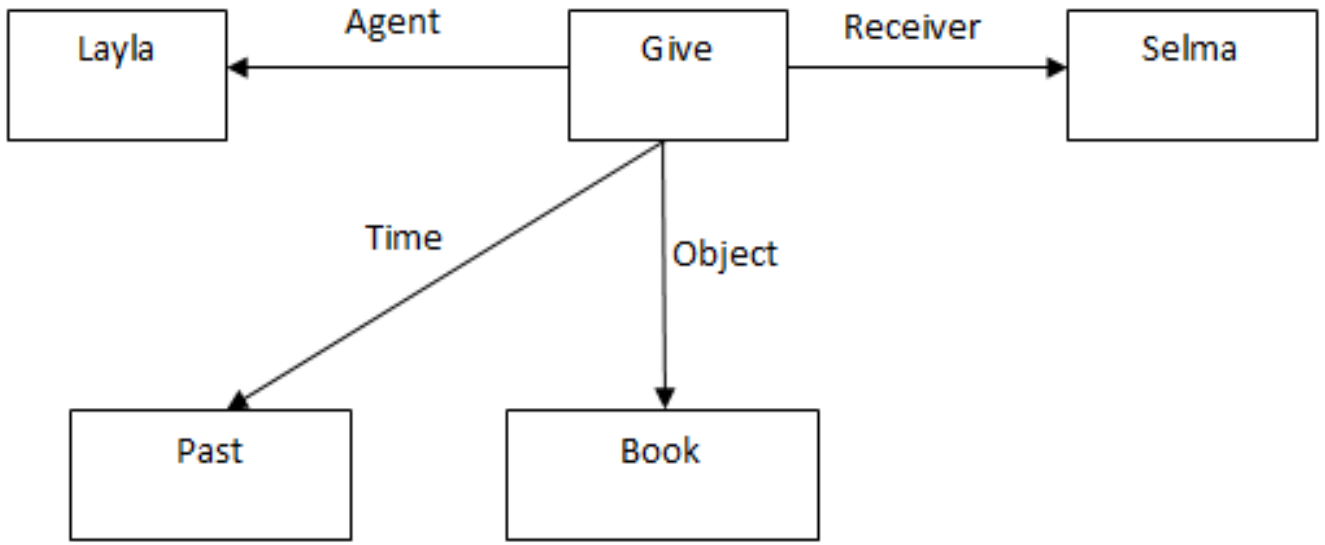
لتمثيل العلاقة بين الكائنات

في وصف اللغات الطبيعية فان arrows تخرج من الفعل لتوضح او لتشير الى الفاعل (agent) والمستقبل (Receiver) والكائن (object) كما تشير الى وقت حدوث الفعل أي في الماضي , الحاضر او المستقبل.

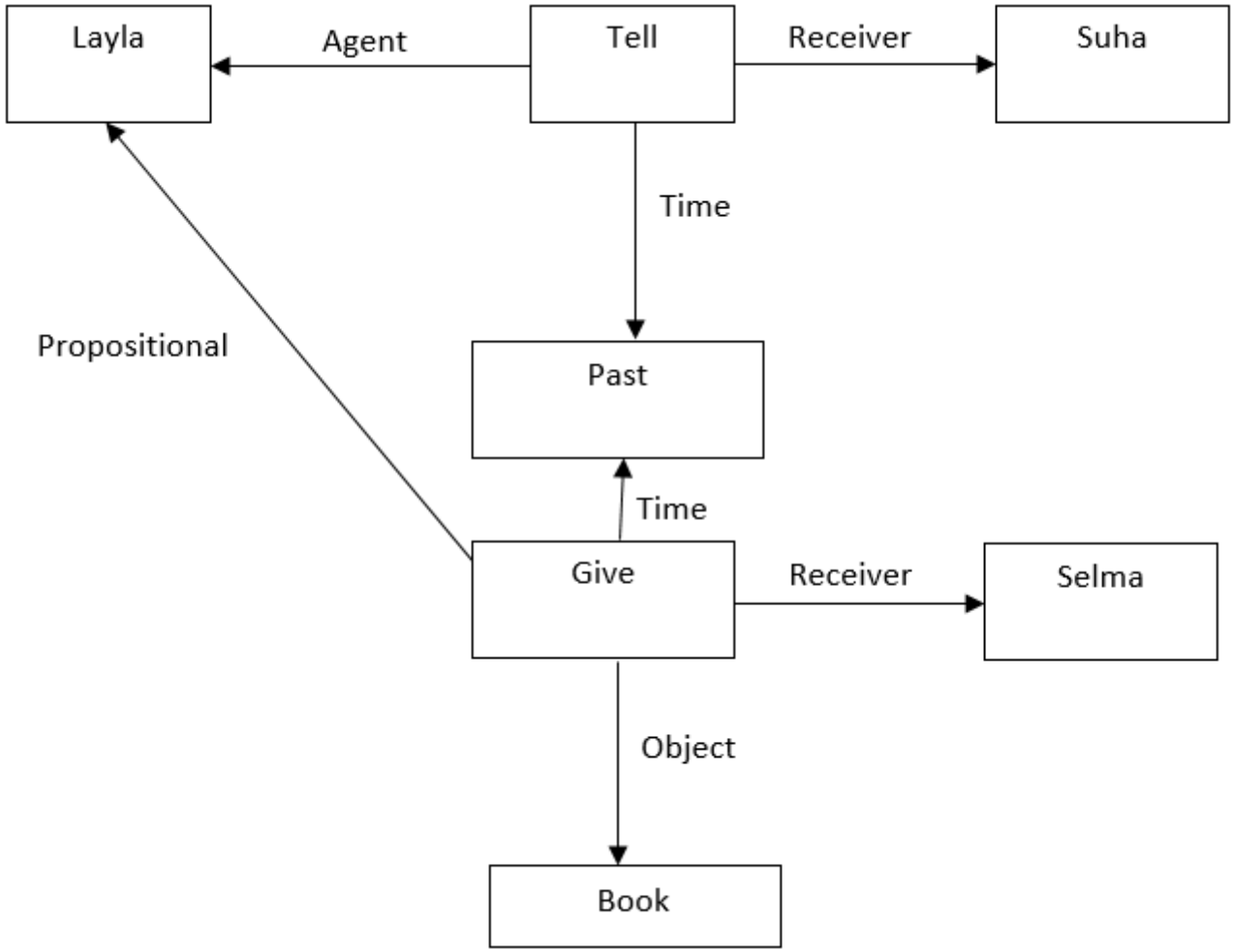
Example 1: Computer has many parts like a CPU and the computer divided into two types, the first one is the mainframe and the second is the personal computer. Mainframe has a line printer with large sheet but the personal computer has a laser printer. IBM as example to the mainframe but PIII and PIV as examples to the personal computer.



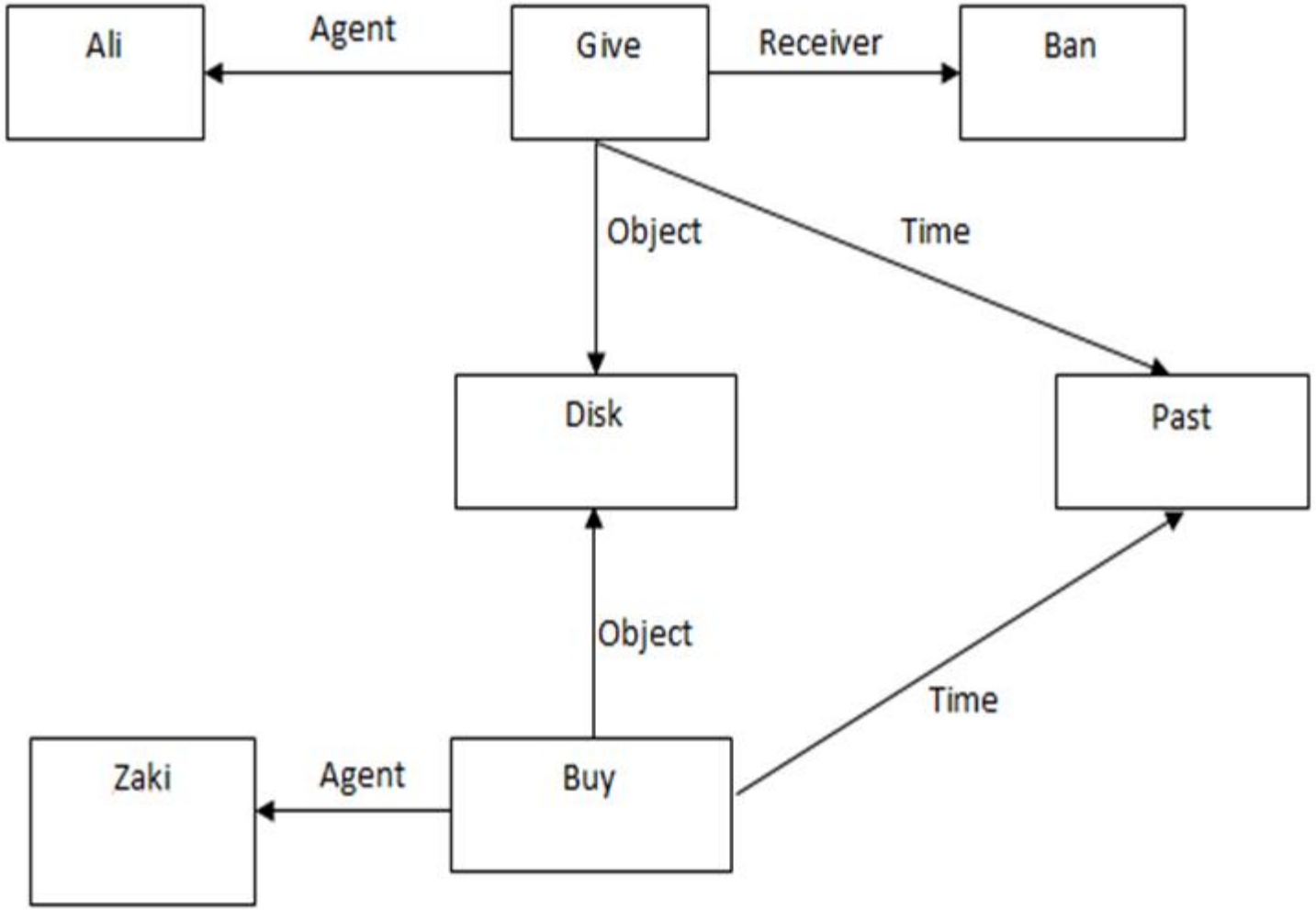
Example 2: Layla gave Selma a book



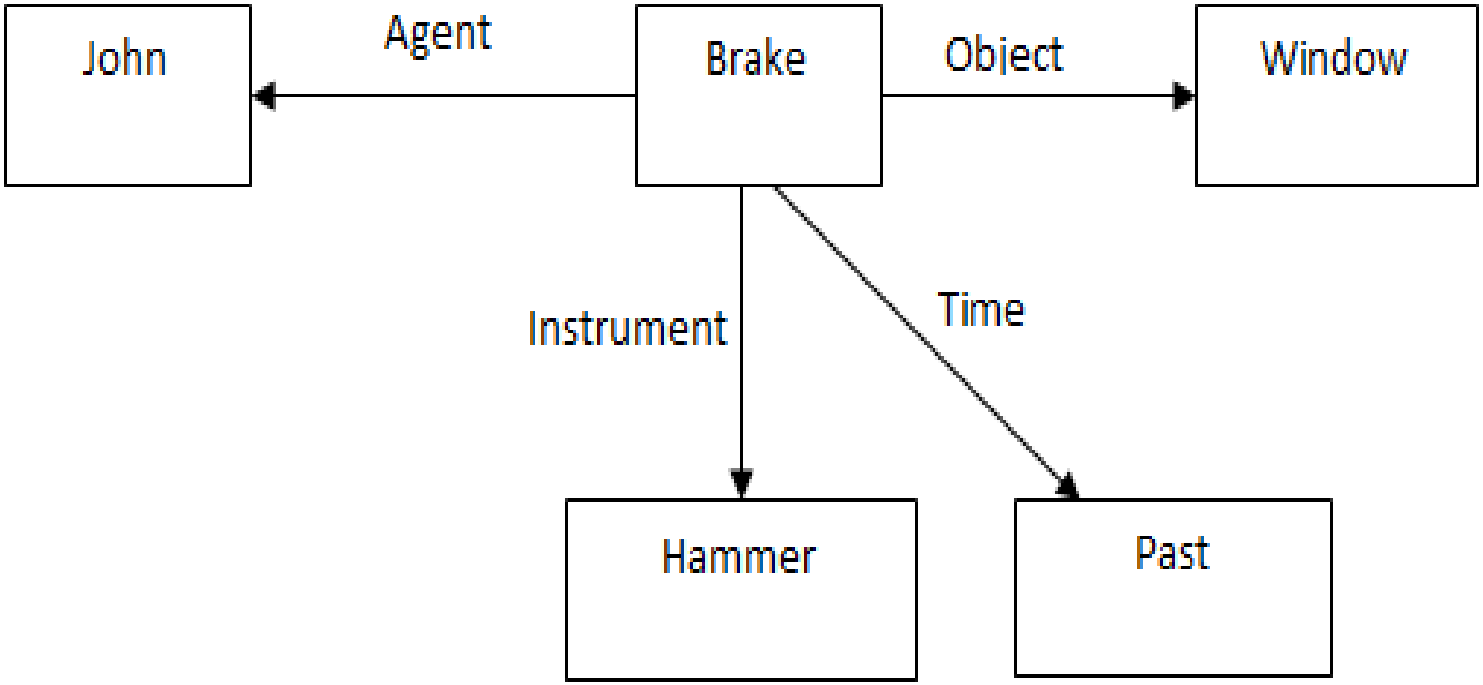
Example 3: Layla told Suha that she gave Selma a book



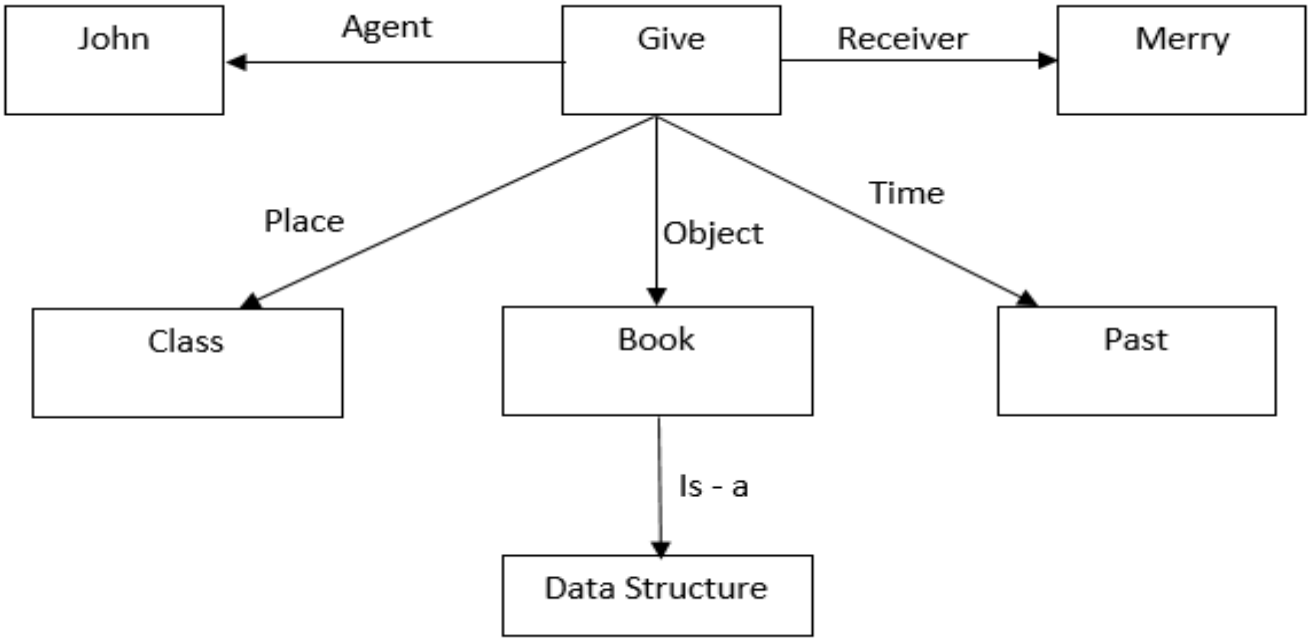
Example 4: Ali gave Ban a disk which is Zaki bought



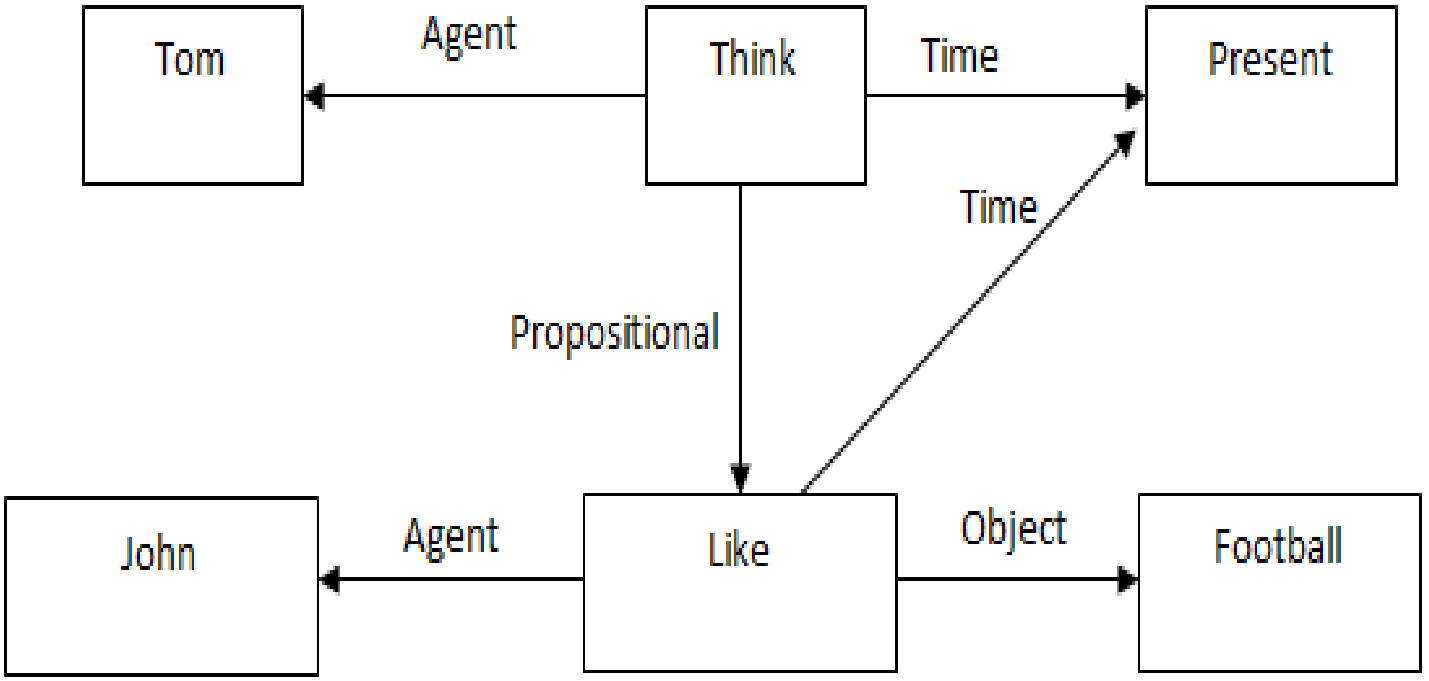
Example 5: John broke the window with the hammer



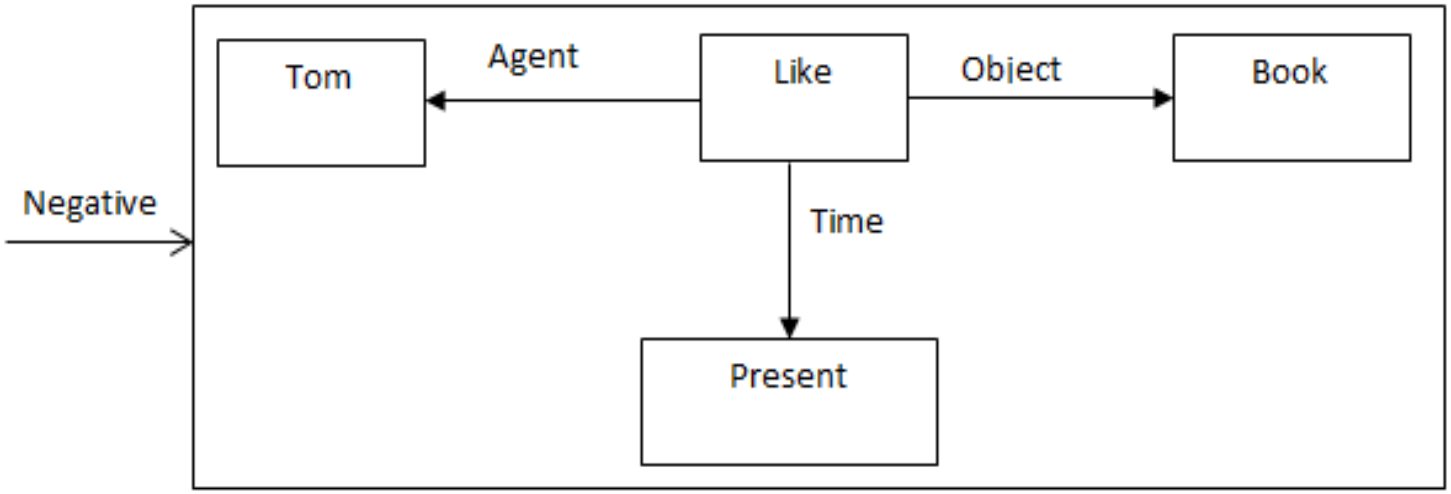
Example 6: John gave merry a book of data structure in the class.



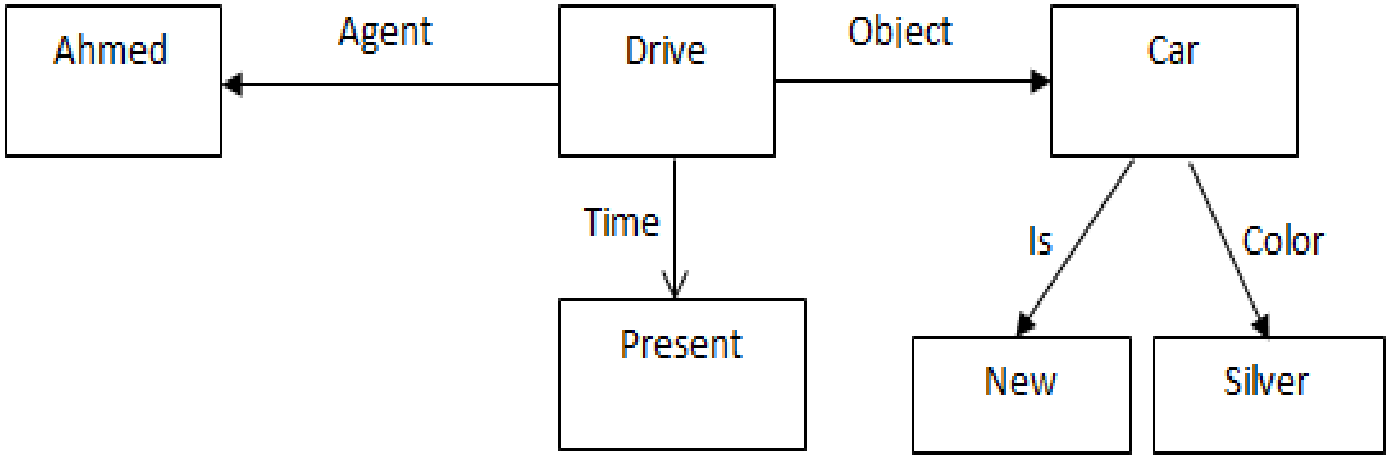
Example 7: Tom thinks that John likes football.



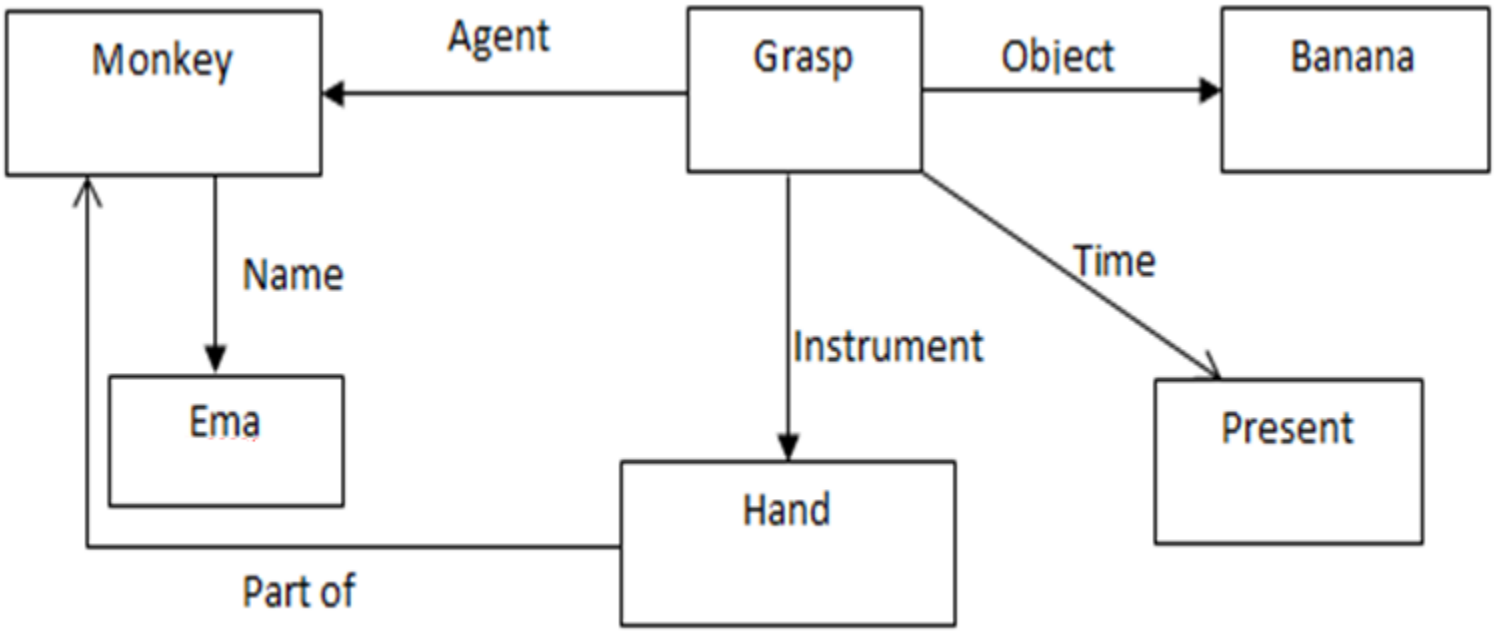
Example 8: Tom doesn't like a book.



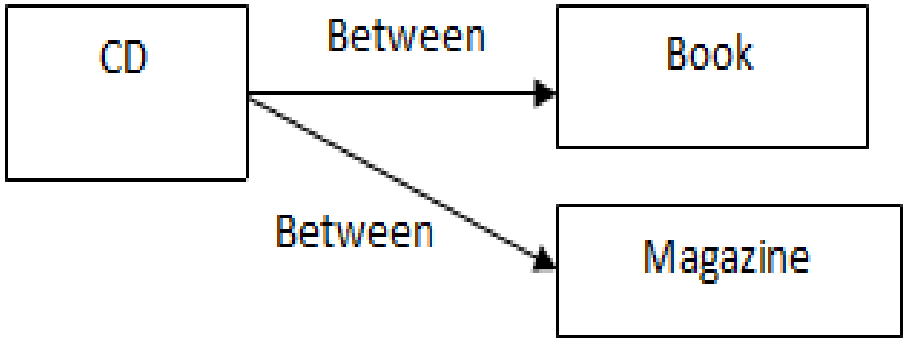
Example 9: Ahmed drives a new silver car.



Example 10: Monkey Ema grasps the banana with hand



Example 11: The CD is between book and magazine.



4- Conceptual Graph

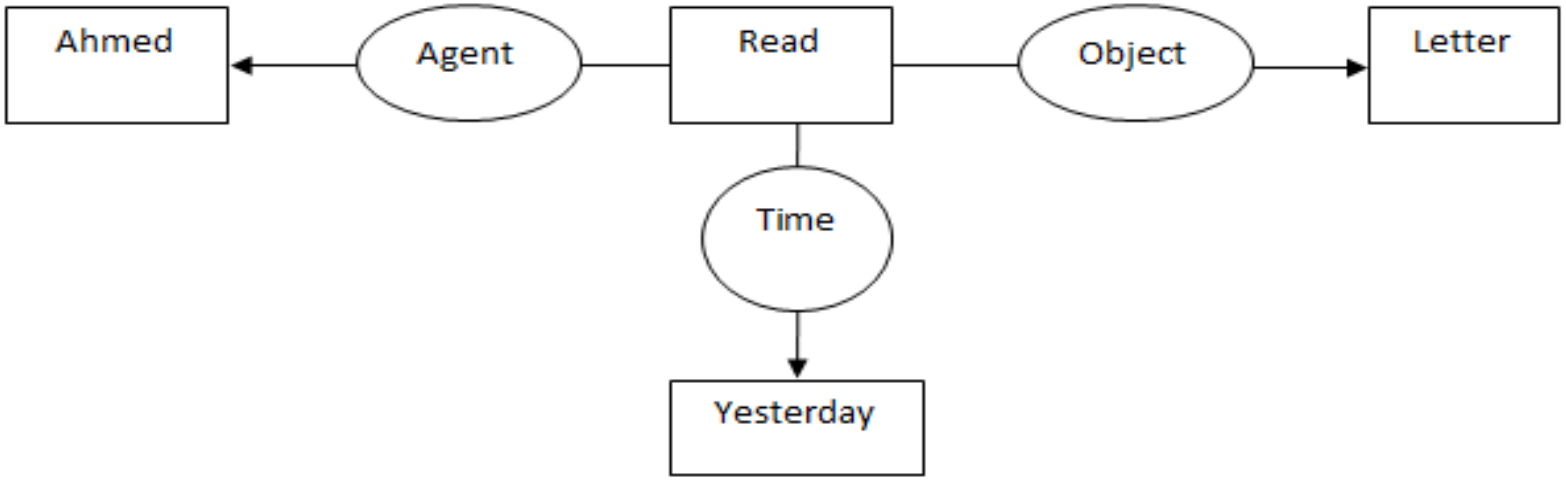
Conceptual Graph is a logical formalism that includes classes, relations, individuals and quantifiers. This formalism is based on semantic networks. The main feature is a standardized graphical representation that like in the case of semantic networks allows human to get a quick overview of what the graph means.

The ***conceptual graph*** is a bipartite oriented graph where concepts are represented as a rectangle and conceptual relations are represented as an ellipse. Oriented edges then link these vertices and denote the existence and orientation of relation. A relation can have more than one edge, in which case edges are numbered.

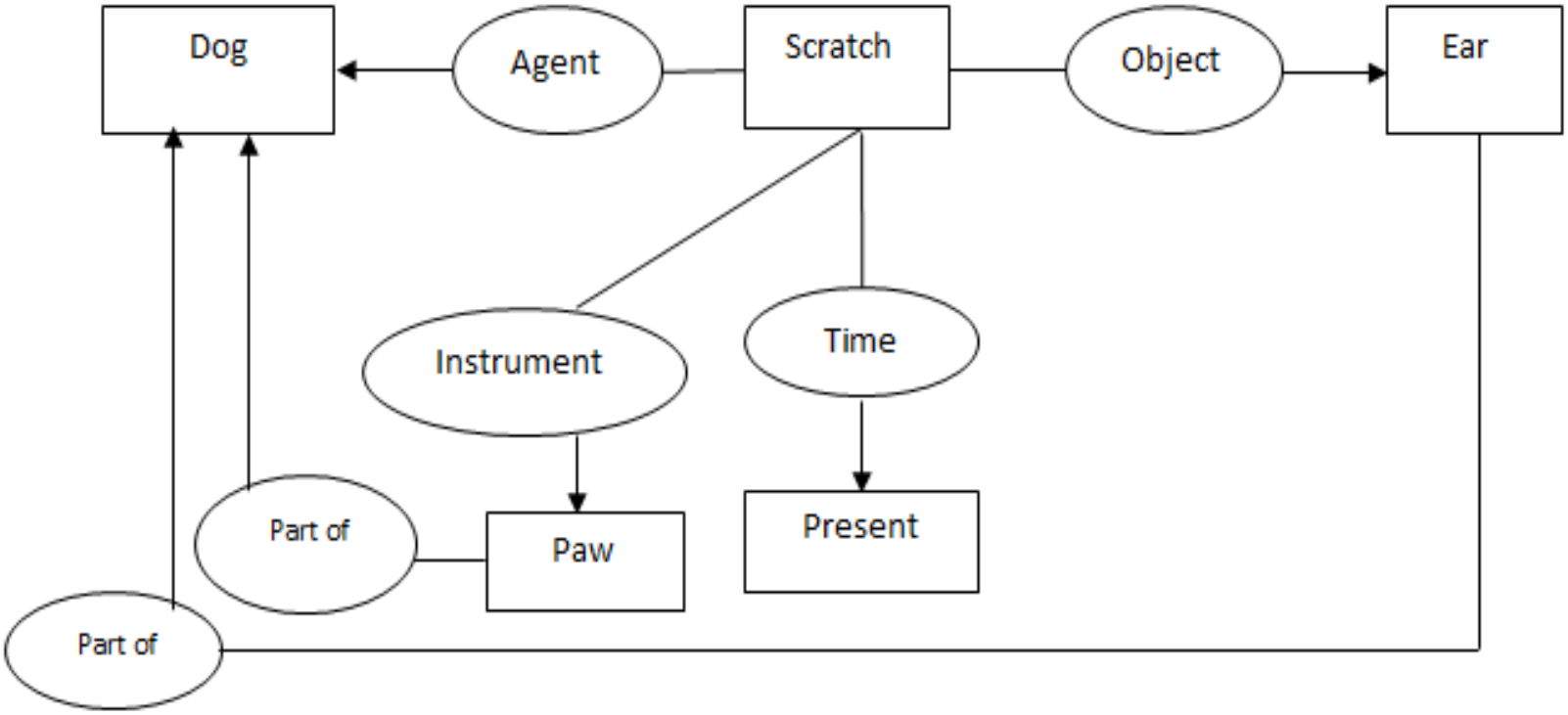
يستخدم لتمثيل الأسماء والصفات والأفعال والثوابت

يستخدم لتمثيل أدوات التعريف والعلاقات

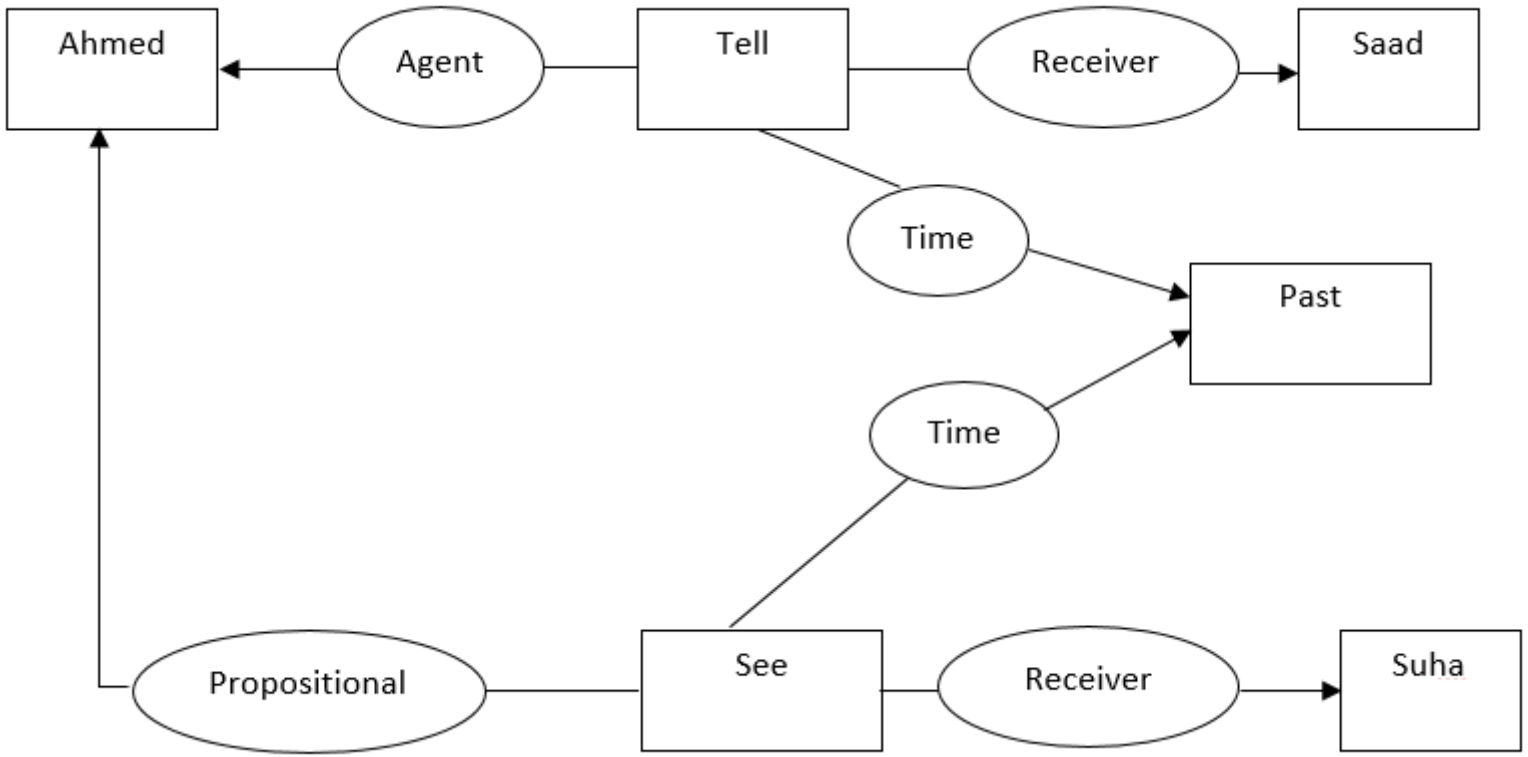
Example 1: Ahmed read a letter yesterday



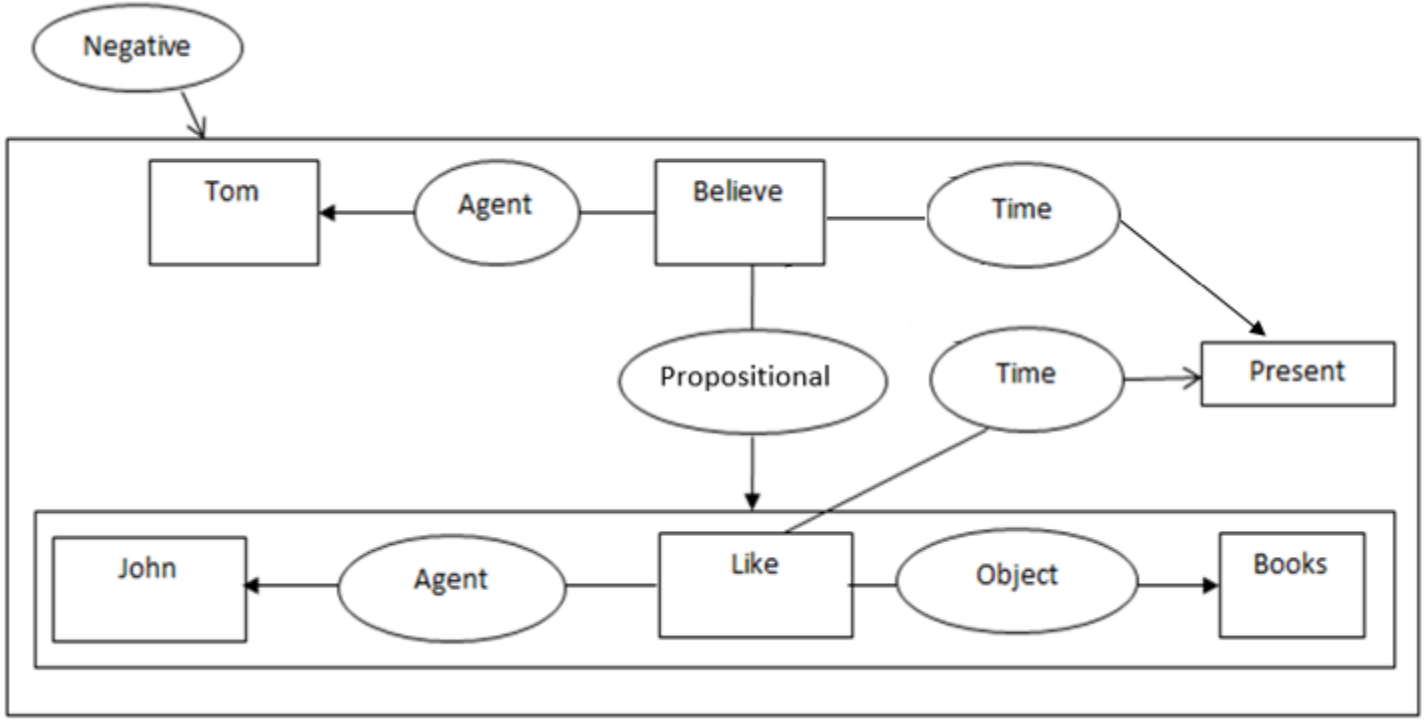
Example 2: The dog scratches its ear with its paw



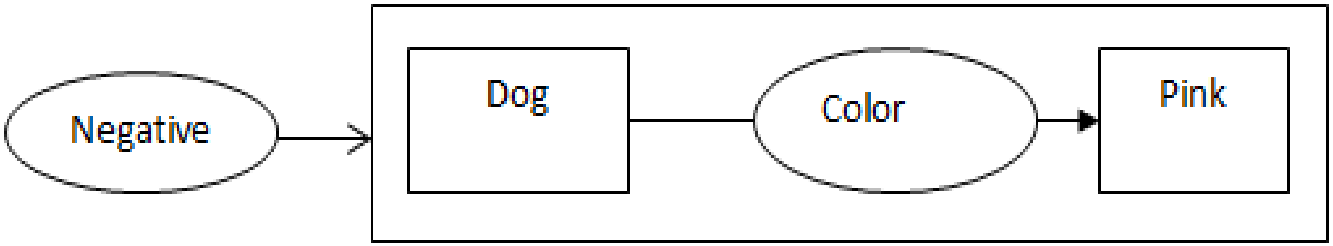
Example 3: Ahmed told Saad that he saw Suha



Example 4: Tom doesn't believe that John likes books

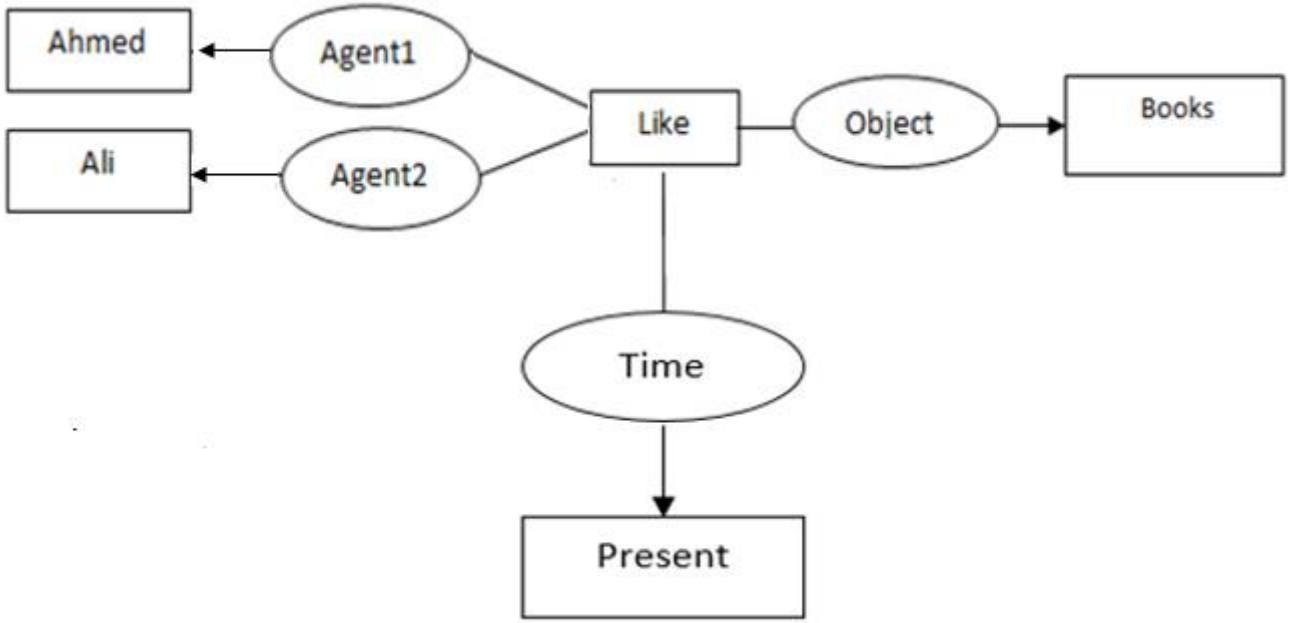


Example 5: There are no pink dogs



Example 6: The dog has brown color.

Example 7: Ahmed and Ali like books.

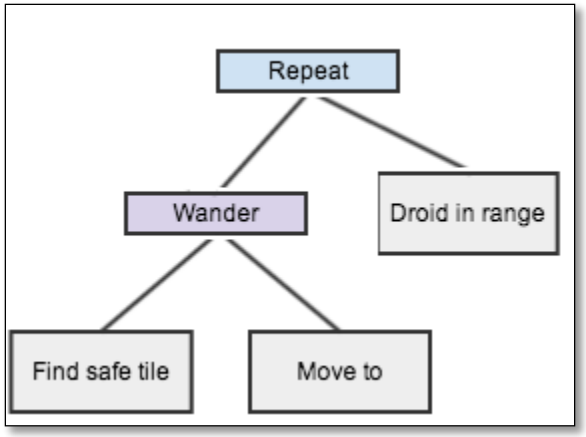


5- Frame Representation

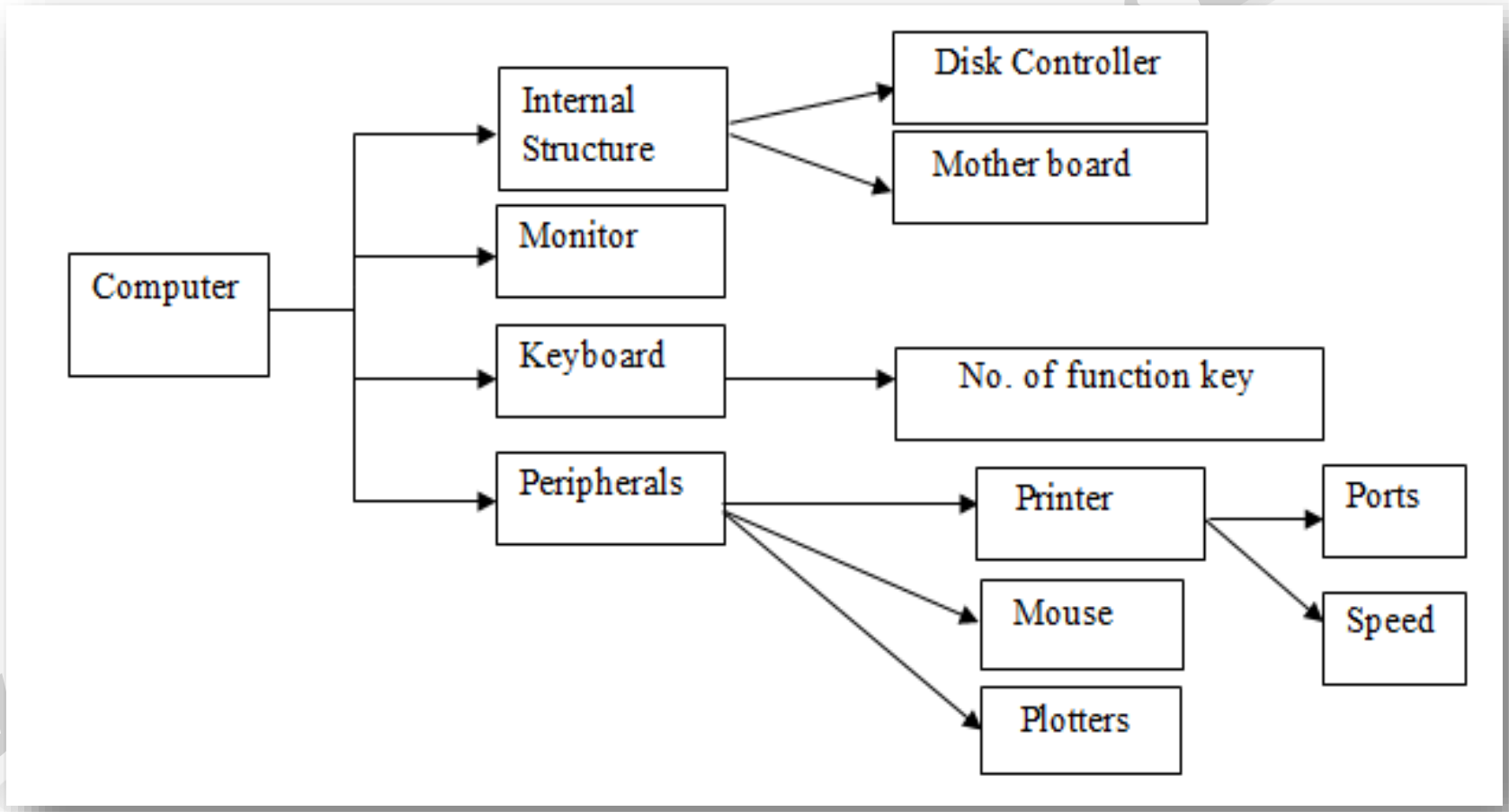
The frame representation consists of two parts:

1. Frame-list.
2. Slot-list.

- **Frame-list (node-name, parent, [child]).**
- **Slot-list (node-name, parent).**



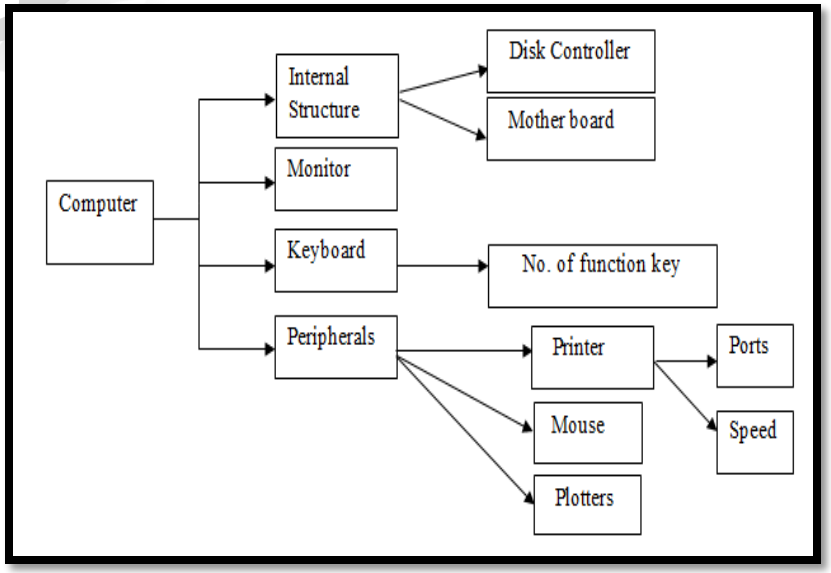
Example: ***Using the frame representation method to represent the following figure as knowledge; then illustrate, how many frame and slot in this figure?***



Frame-list (node-name, parent, [child]).
Slot-list (node-name, parent).

- Frame -list (computer,_ ,[Internal structure, monitor, keyboard , Peripherals]).
- Frame-list (Internal structure, computer, [disk controller, motherboard]).
- Frame-list (keyboard, computer, [No. of function key]).
- Frame-list (peripherals, computer, [printer, mouse, plotters]).
- Frame- list (printer, peripherals, [ports, speed]).

- Slot-list (monitor, computer).
- Slot-list (Disk controller, internal structure).
- Slot-list (motherboard, internal structure).
- Slot-list (No. of function key, keyboard).
- Slot-list (mouse, peripherals).
- Slot-list (plotters, peripherals).
- Slot-list (ports, printer).
- Slot-list (speed, printer).



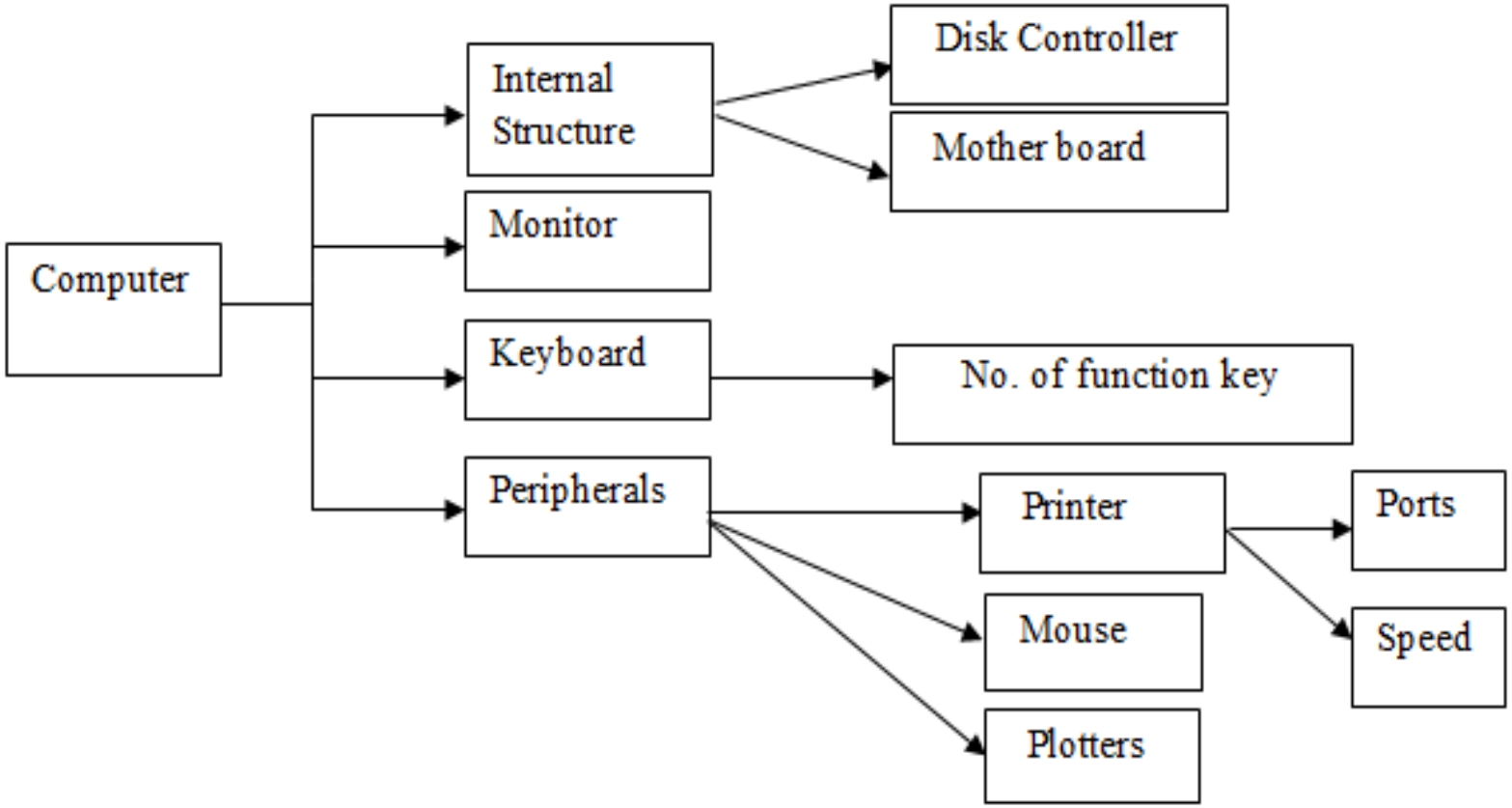
(There are 5 frames and 8 slots)

Example2: ***you have the following facts; represent them as a tree, then using the frame representation method to represent the tree as knowledge.***

Facts:

Hardware(computer, internal structure).
Hardware(computer, monitor).
Hardware(computer, keyboard).
Hardware(computer, peripherals).
Hardware(internal structure, disk controller).
Hardware(internal structure, motherboard).
Hardware(keyboard, No. of function key).
Hardware(peripherals, printer).
Hardware(peripherals, mouse).
Hardware(peripherals, plotters).
Hardware(printer, ports).
Hardware(printer, speed).

- Sol :


Computer
Internal Structure
Disk Controller
Mother board
Monitor
Keyboard
No. of function key
Peripherals
Printer
Ports
Speed
Mouse
Plotters

- Frame -list (computer,_ ,[Internal structure, monitor, keyboard , Peripherals]).
- Frame-list (Internal structure, computer, [disk controller, motherboard]).
- Frame-list (keyboard, computer, [No. of function key]).
- Frame-list (peripherals, computer, [printer, mouse, plotters]).
- Frame- list (printer, peripherals, [ports, speed]).

- Slot-list (monitor, computer).
- Slot-list (Disk controller, internal structure).
- Slot-list (motherboard, internal structure).
- Slot-list (No. of function key, keyboard).
- Slot-list (mouse, peripheral).
- Slot-list (plotters, peripheral).
- Slot-list (ports, printer).
- Slot-list (speed, printer).

7- Script Representation

A script is a structured representation describing a stereotyped sequence of event in a particular context.

A script is **composed** of the following **components**:

- **Roles** are the actions that the individual participants perform.
- **Props** or the “things” that make up the content of the script.
- **Entry conditions** that must be true for the script to be called.
- **Results** or facts those are true once the script has terminated.
- **Scenes** which present temporal aspects of the script.
- **Track** specific variation on more general pattern in the script.

Example1: Using script representation to represent someone having a meal at a restaurant.

Track **“**having a meal at a restaurant”

Roles
- Customer
- Waiter
- Cashier
- Owner

Props
- Tables
- Menu
- Food
- Money
- Bill

Entry conditions
- Customer is hungry
- Customer has money
- Owner has Food

Results

- Customer has less money
- Customer is not hungry
- Owner has more money
- Owner has less food

Scenes

Scene1: Entering

- Customer enters the restaurant.
- Customer looks at tables.
- Customer decides where to sit.
- Customer goes there and seated.

Scene2: Ordering

- Customer asks for menu.
- Waiter brings it.
- Customer chooses what to eat.
- Customer orders that item.

Scene3: Eating

- Waiter brings the Food.
- Customer eats it.

Scene4: Paying

- Customer asks for the bill.
- Waiter brings it.
- Customer pays for it
- Waiter hands the money to the cashier.
- Waiter brings the balance amount.
- Customer leaves the restaurant

- ملاحظة: اي شخص من Role لم يظهر في Scene يجب ان يظهر في Result و العكس صحيح.

Example2: Using script representation to represent robbing a bank.

Track "robbing a bank"

Roles

- Robber.
- Cashier.
- Bank Manager.
- Policeman.

Props

- Gun.
- Loot.
- Bag.
- Getaway car.

Entry Conditions

- Robber is poor.
- Robber is destitute.

Results

- Robber has more money.
- Bank Manager is angry.
- Cashier is in a state of shock.
- Policeman is shot.

Senses

Sense1: getting a gun

- Robber goes to gun shop.
- Robber chooses the gun.
- Robber buys gun.

Sense2: Holding up the bank

- Robber goes into bank.
- Robber sees Cashier, Bank Manager, and Policeman.
- Robber moves to Cashier position.
- Robber grasps the gun.
- Robber moves the gun to point to cashier.
- Robber tells Cashier to give him the loot.
- Policeman tells Robber to hold it: hand up.
- Robber shoots from gun.
- Policeman shoots from a gun.
- Cashier gathers the loot to Robber.
- Cashier puts the loot in bag.
- Robber comes out of the bank from the exit gate
- Bank Manager raises the alarm.

Sense3: getaway

- Robber runs away by a getaway car.