

UNIT-2

KNOWLEDGE REPRESENTATION

SYLLABUS

- Introduction to KR, Knowledge agent, Predicate logic, WFF,
- Inference rule & theorem proving forward chaining, backward chaining,
- resolution;
- Propositional knowledge,
- Boolean circuit agents.
- Rule Based Systems,
- Forward reasoning: Conflict resolution,
- backward reasoning: Use of Back tracking,
- Structured KR: Semantic Net - slots,
- inheritance,
- Frames- exceptions and defaults attached predicates,
- Conceptual Dependency formalism and other knowledge representations.

Knowledge Representation

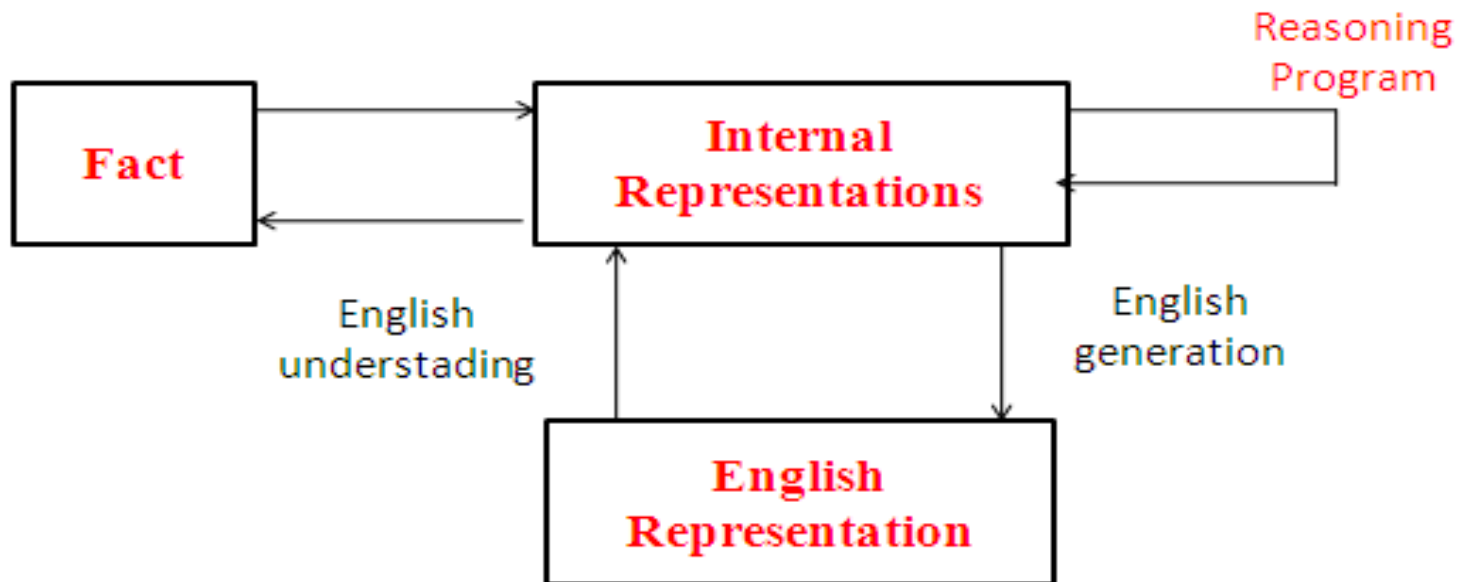
- A knowledge representation is a study of ways of how knowledge is actually represented and how effectively it resembles the representation of knowledge in human brain.
- A knowledge representation system should provide ways of representing complex knowledge.
- How to represent the knowledge in a machine?
- So to represent the knowledge we need a language. There must be a method to use this knowledge.
- Syntax and semantic must be well defined in order to represent the language.

Characteristics of Knowledge Representation

- i. The representation scheme should have a set of well-defined syntax and semantic. This will help in representing various kinds of knowledge.
- ii. The knowledge representation scheme should have a good expressive capacity. A good expressive capability will catalyze the inference mechanism in its reasoning process.
- iii. From the computer system point of view, the representation must be efficient. By this we mean that it should use only limited resources without compromising on the expressive power.

Fact and Fact Representation

- Fact: Truth in some relevant world. These are the things we want to represent.
- Representation of fact in some chosen formalism. These are the things we will actually be able to manipulate.



Representation schemes

- i. Propositional logic
- ii. Semantic Networks
- iii. Frame
- iv. Conceptual dependency
- v. Scripts

Propositional Logic

Proposition

- Propositions are elementary atomic sentences.
- Propositions may be either true or false but may take on no other value.
- There are two kinds of proposition
 - Simple
 - compound
- Some examples of simple propositions are
 - It is raining.
 - My car is painted silver.
 - John and sue have five children.
 - Snow is white.
 - People live on the moon.

Proposition continued....

- Compound propositions are formed from atomic formulas using the logical connectives not, or, ifthen, and, if and only if.
- For example, the following are compound formulas.
 - It is raining and the wind is blowing.
 - The moon is made of green cheese or it is not.
 - If you study hard you will be rewarded.
 - The sum of 10 and 20 is not 50.

Logic

- One of the prime activities of the human intelligence is reasoning.
- The activity of reasoning involves construction, organization and manipulation of statements to arrive at new conclusions.
- Thus logic can be defined as a scientific study of the process of reasoning.
- Logic is a formal language.
- Logic is basically classified in two main categories
 - Propositional logic
 - Predicate logic

Propositional logic

- Propositional logic is a representational language that makes the assumption that the world can be represented solely in terms of propositions that are true or false.
- One of the main concerns of propositional logic is the study of rules by which new logic variables can be produced as functions of some given logic variables.

Syntax for Propositional logic

1. Logic constants: true, false
2. Propositional atoms: indivisible propositions A,B,C,... P,Q,R.
3. Connectives: negation, conjunctive, disjunctive, implication, equivalence
4. Sentences (well-formed formulas): wff

Connectives

\wedge (and)	This is called <i>conjunction</i> and is used for constructing a sentence like $P \wedge Q$
\vee (or)	This is called <i>disjunction</i> and is used for constructing a sentence like $P \vee Q$
\neg (negation)	This is called <i>not</i> and is used in constructing sentence like $\neg P$
\Rightarrow (implication)	This is used in constructing sentences like $P \Rightarrow Q$ and is equivalent to $\neg P \vee Q$
\Leftrightarrow (equivalent)	These are used in constructing a sentence like $P \Leftrightarrow Q$ which implies P is equivalent to Q

Well-formed formulas (wff)

- WFF consist of atomic symbols joined with connectives.
- Examples: P , $P \wedge \sim Q$, $P \vee Q$, $P \rightarrow Q$

Two Normal (Canonical) Forms

All wffs can be expressed in the following to normal forms

1. CNF (Conjunctive Normal Form)

$$\text{e.g.: } (A \vee \neg B) \wedge (B \vee \neg D \vee \neg C)$$

Clause 1

clause 2

2. DNF (Disjunctive Normal Form)

$$\text{e.g.: } (A \wedge \neg B) \vee (B \wedge \neg D \wedge \neg C)$$

models

models

Semantics for Propositional logic

The semantics or meaning of a sentence is just the value true or false. The terms used for semantics of a language are given below.

Valid	A sentence is valid if it is true for all interpretations. Valid sentences are also called tautologies.
Model	An interpretation of a formula or sentence under which the formula is true is called a model of that formula.
Unsatisfiable (contradiction)	It is said to be unsatisfiable if it is false for every interpretation.
Satisfiable	It is said to be satisfiable if it is true for some interpretation.
Equivalence	Two sentences are equivalent if they have the same truth value under every interpretation.

Example

Write the syntax for propositional logic

If the road is closed, then the traffic is blocked.

“the road is closed” is represented by a proposition, P .

“then the traffic is blocked” is represented by a proposition, Q .

The sentence is represented as

$$P \Rightarrow Q.$$

Truth tables for logical connectives

A	B	$\sim A$	$\sim B$	$A \vee B$	$A \& B$	$A \rightarrow B$	$A \leftrightarrow B$
T	T	F	F	T	T	T	T
F	T	T	F	T	F	T	F
T	F	F	T	T	F	F	F
F	F	T	T	F	F	T	T

Show that

$P \rightarrow Q$ is equivalent to $\neg P \vee Q$

Show that

$P \leftrightarrow Q$ is equivalent to

$$(P \rightarrow Q) \& (Q \rightarrow P)$$

Show that

$P \rightarrow Q$ is equivalent to $\sim P \vee (P \wedge Q)$

Truth table for equivalent sentences

P	Q	$\neg P$	$(\neg P \vee Q)$	$(P \rightarrow Q)$	$(Q \rightarrow P)$	$(P \rightarrow Q) \& (Q \rightarrow P)$
true	true	false	true	true	true	true
true	false	false	false	false	true	false
false	true	true	true	true	false	false
false	false	true	true	true	true	true

Some Equivalence Laws

Idempotency

$$P \vee P = P$$
$$P \& P = P$$

Associativity

$$(P \vee Q) \vee R = P \vee (Q \vee R)$$
$$(P \& Q) \& R = P \& (Q \& R)$$

Commutativity

$$P \vee Q = Q \vee P$$
$$P \& Q = Q \& P$$
$$P \leftrightarrow Q = Q \leftrightarrow P$$

Distributivity

$$P \& (Q \vee R) = (P \& Q) \vee (P \& R)$$
$$P \vee (Q \& R) = (P \vee Q) \& (P \vee R)$$

De Morgan's
laws

$$\neg(P \vee Q) = \neg P \& \neg Q$$
$$\neg(P \& Q) = \neg P \vee \neg Q$$

Conditional
elimination

$$P \rightarrow Q = \neg P \vee Q$$

Bi-conditional
elimination

$$P \leftrightarrow Q = (P \rightarrow Q) \& (Q \rightarrow P)$$

Question 1

- Determine whether each of the following sentences is (a) Satisfiable (b) Contradictory and (c) Valid.

S1: $(P \ \& \ Q) \vee \sim(P \vee Q)$

S2: $(P \vee Q) \rightarrow (P \ \& \ Q)$

S3: $(P \ \& \ Q) \rightarrow (P \vee \sim Q)$

S4: $(P \vee Q) \ \& \ (P \vee \sim Q) \vee P$

S5: $P \rightarrow Q \rightarrow \sim P$

S6: $P \vee Q \ \& \ \sim P \vee \sim Q \ \& \ P$

S1

P	Q	$P \& Q$	$P \vee Q$	$\sim(P \vee Q)$	$(P \& Q) \vee \sim(P \vee Q)$
T	T	T	T	F	T
F	T	F	T	F	F
T	F	F	T	F	F
F	F	F	F	T	T

S1: $(P \& Q) \vee \sim(P \vee Q)$ satisfiable

S2

P	Q	$P \vee Q$	$P \& Q$	$(P \vee Q) \rightarrow (P \& Q)$
T	T	T	T	T
F	T	T	F	F
T	F	T	F	F
F	F	F	F	T

S2: $(P \vee Q) \rightarrow (P \& Q)$ satisfiable

S3

P	Q	$\sim Q$	$P \& Q$	$P \vee \sim Q$	$(P \& Q) \rightarrow (P \vee \sim Q)$
T	T	F	T	T	T
F	T	F	F	F	T
T	F	T	F	T	T
F	F	T	F	T	T

S3: $(P \& Q) \rightarrow (P \vee \sim Q)$ valid

S4

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

S4: $(P \vee Q) \& (P \vee \neg Q) \vee P$ satisfiable

S5

P	Q	$\sim P$	$P \rightarrow Q$	$P \rightarrow Q \rightarrow \sim P$
T	T	F	T	F
F	T	T	T	T
T	F	F	F	T
F	F	T	T	T

S5: $P \rightarrow Q \rightarrow \sim P$ satisfiable

S6

P	Q	$\sim P$	$\sim Q$	$P \vee Q$	$(\sim P \vee \sim Q)$	$(P \vee Q) \& (\sim P \vee \sim Q)$	$(P \vee Q) \& (\sim P \vee \sim Q) \& P$
T	T	F	F	T	F	F	F
F	T	T	F	T	T	T	F
T	F	F	T	T	T	T	T
F	F	T	T	F	T	F	F

S6: $P \vee Q \& \sim P \vee \sim Q \& P$ satisfiable

Rules of inference

- The inference rules of propositional logic provide the means to perform logical proofs or deductions.
- Rules are
 - i. Modus ponens
 - ii. Modus tollens
 - iii. Chain rule
 - iv. Substitution
 - v. Simplification
 - vi. Conjunction
 - vii. Transposition

Inference Rules cont.....

Modus ponens. From P and $P \rightarrow Q$ infer Q . This is sometimes written as

$$\frac{P \quad P \rightarrow Q}{Q}$$

For example

given: (joe is a father)

and: (joe is a father) \rightarrow (joe has a child)

conclude: (joe has a child)

Inference Rules cont.....

Modus tollens

- $\alpha \rightarrow \beta$
 $\underline{\neg\beta}$
 $\neg\alpha$

Example:

Given: The machine is defective(α)->the production is less (β)

And : The production is not less ($\neg\beta$)

Conclude: The machine is not defective ($\neg\alpha$)

Inference Rules cont.....

Chain rule. From $P \rightarrow Q$, and $Q \rightarrow R$, infer $P \rightarrow R$. Or

$$\frac{P \rightarrow Q \quad Q \rightarrow R}{P \rightarrow R}$$

For example,

given: (programmer likes LISP) \rightarrow (programmer hates COBOL)

and: (programmer hates COBOL) \rightarrow Programmer likes recursion)

conclude: (programmer likes LISP) \rightarrow (programmer likes recursion)

Inference Rules cont.....

Substitution. If s is a valid sentence, s' derived from s by consistent substitution of propositions in s , is also valid. For example, the sentence $P \vee \sim P$ is valid; therefore $Q \vee \sim Q$ is also valid by the substitution rule.

Inference Rules cont.....

Simplification. From $P \& Q$ infer P .

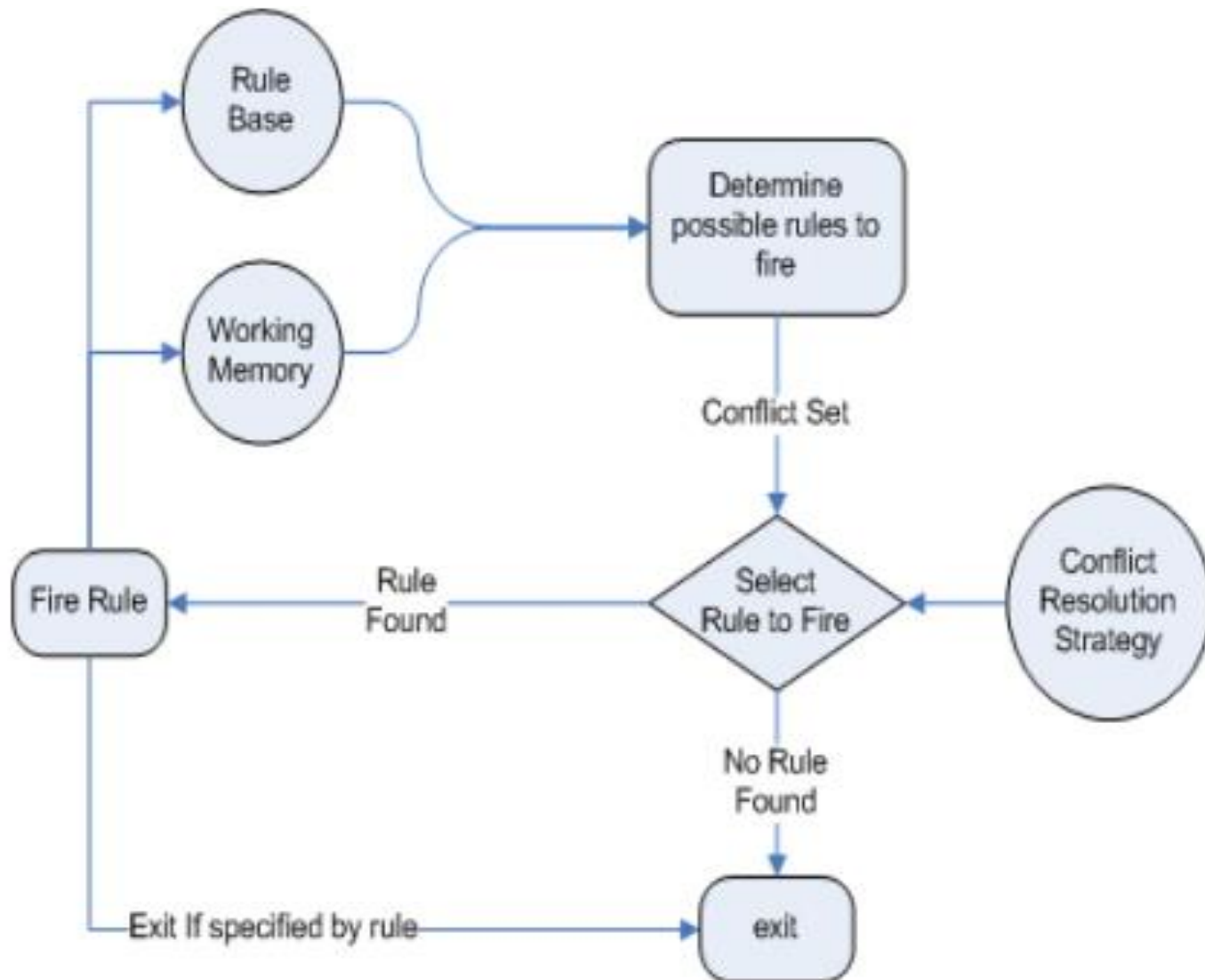
Conjunction. From P and from Q , infer $P \& Q$.

Transposition. From $P \rightarrow Q$, infer $\sim Q \rightarrow \sim P$.

Forward Chaining

- An inference engine using forward chaining searches the inference rules until it finds one where the IF clause is known to be true.
- When found it can conclude, or infer, the THEN clause, resulting in the addition of new information to its dataset. In other words, it starts with some facts and applies rules to find all possible conclusions. Therefore, it is also known as Data Driven Approach.
- The standard definition of a forward-chaining system is that the system operates by repeating the following sequence of operations
 1. Examine the rules to find one who's If part is satisfied by the current contents of Working Memory.
 2. Fire the rule by adding to Working Memory the facts that are specified in the rules Then part. (The Then part may perform other actions as well, but that can be ignored for now.) This control cycle continues until no rules have satisfied If parts.

Flowchart for forward chaining



Backward chaining

- An inference engine using backward chaining would search the inference rules until it finds one which has a THEN clause that matches a desired goal. If the IF clause of that inference rule is not known to be true, then it is added to the list of goals (in order for goal to be confirmed it must also provide data that confirms this new rule) . In other words, this approach starts with the desired conclusion and works backward to find supporting facts. Therefore, it is also known as Goal-Driven Approach.
- Backward-chaining systems try to satisfy the goals in the goal stack. They do this by finding rules that can conclude the information needed by the goal, and trying to make the If parts of those rules satisfied . In more detail, the standard backward-chaining control cycle is shown in figure.

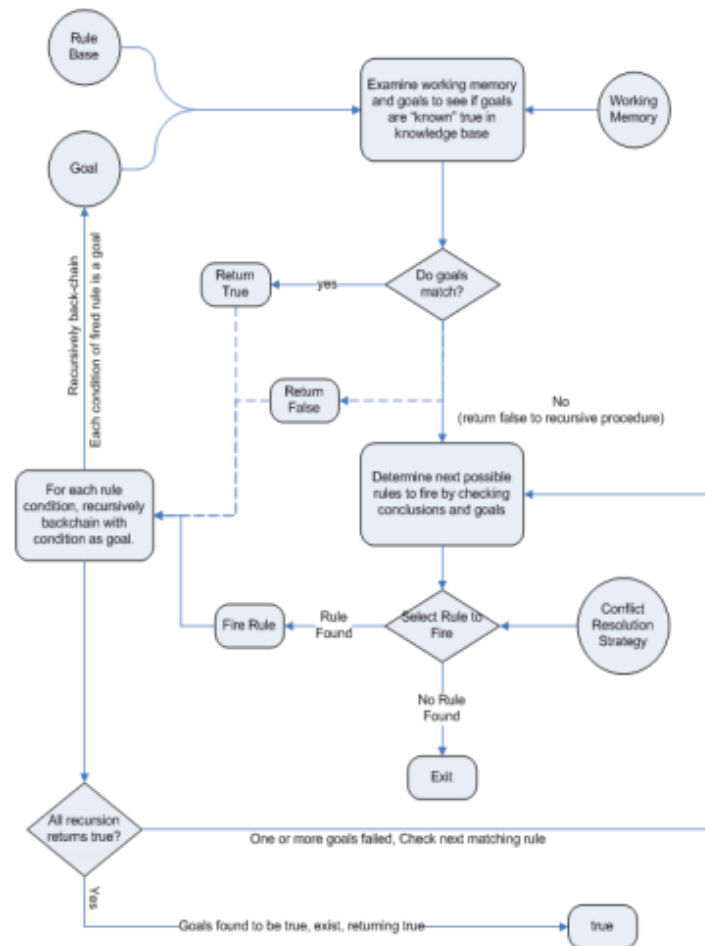
Steps for backward chaining

1. Check the conclusions of the rules to find all rules that can satisfy the top goal on the stack.
2. Process these rules one at a time:
 - a. Evaluate the conditions in the rules If part one at a time:
 - i. If the condition is currently unknown (that is, if there is not enough information currently known to determine whether the condition is satisfied) push a goal to make that condition known, and recursively invoke the system.
 - ii. If the condition is known to be unsatisfied, continue with the loop at Step 2.
 - iii. If it was not possible to determine whether the condition was satisfied, continue with the loop at Step 2.

Steps for backward chaining

- b. If all the conditions in the selected rule are satisfied, add to Working Memory the facts specified in the Then part of the rule, pop the goal off the stack, and return from this invocation of the system. The system terminates with success when the goal stack is empty. It terminates with failure if the system runs out of rules to try in Step 2.

Flowchart for backward chaining



First Order Predicate Logic (FOPL)

- FOPL was developed to extend the expressiveness of Propositional logic.
- Propositional logic works fine in situations where the result is either true or false. However, there are many real life situations that cannot be treated this way.
- Predicate logic is the area of logic that deals with predicates and quantifiers
- Predicate – refers to a property that the subject of a statement can have

E.G. “x is greater than 3”

x \rightarrow is the subject and

“is greater than 3” \rightarrow is the predicate

Limitations of Propositional Logic

- The propositional logic has its limitations that you cannot deal properly with general statements of the form
- “All men are mortal”
- You can not derive from the conjunction of this and “Socrates is a man” that “Socrates is mortal”

Example:

- If All men are mortal = P
- Socrates is a Man = Q
- Socrates is mortal = R
- Then $(P \ \& \ Q) \rightarrow R$ is not valid

Syntax of FOPL

- Connectives
- Quantifiers
- Constants
- Variables
- Predicates
- Functions

Syntax of FOPL

- **Connectives:** There are five connective symbols
 - \sim : not or negation
 - $\&$: and or conjunction
 - \vee : or or inclusive disjunction
 - \rightarrow : implication
 - \leftrightarrow : equivalence or if and only if
- **Quantifiers:** There are two quantifier symbols are
 - \exists : existential quantification
 - Example: $\exists x$ means **there exist** x
 - \forall : universal quantification
 - Example: $\forall x$ means **for all** x

Syntax of FOPL...

- **Constants:** constants are fixed-values terms that belong to a given domain of discourse. They are denoted by numbers, words and small letter.
- **Variables:** variables are terms that can assume different values over a given domain.

Syntax of FOPL...

- **Predicates:** a predicate is defined as a relation that binds two atoms together. Ex: Rabbit likes carrots, is represented as
 - **LIKES**(rabit, carrots)
 - Here **LIKES** is a predicate that links two atoms “rabbit” and “carrots”
- **Functions:** it is also possible to have a function as an argument, e.g. “ Ravi’s father is Rani’s father” is represented as-
 - FATHER(father(Ravi), Rani)
 - Here FATHER is a predicate and father(Ravi) is a function to indicate Ravi’s father.

Syntax of FOPL...

- Constants, variables, and functions are referred to as terms.
- Predicates are referred to as atomic formulas or atoms.
- When we want to refer to an atom or its negation, we often use the word literal.
- In addition to above symbols, left and right parentheses , square brackets, braces, and the period are used for punctuation in symbolic representation.

Translating English to FOPL

1. Bhaskar likes aeroplanes.
2. Ravi's father is rani's father.
3. Plato is a man
4. Ram likes mango.
5. Sima is a girl.
6. Rose is red.
7. John owns gold
8. Ram is taller than mohan
9. My name is khan
10. Apple is fruit.

Translating English to FOPL

11. Ram is male.
12. Tuna is fish.
13. Dashrath is ram's father.
14. Kush is son of ram.
15. Kaushaliya is wife of Dashrath.
16. Clinton is tall.
17. There is a white alligator.
18. All kings are person.
19. Nobody loves john.
20. Every one has a father.

Translating English to FOPL

- | | |
|------------------------------------|-------------------------------|
| 1. Bhaskar likes aeroplanes. | Likes (bhaskar, aeroplanes). |
| 2. Ravi's father is rani's father. | Father(father(ravi), rani)). |
| 3. Plato is a man. | Man (plato). |
| 4. Ram likes mango. | Likes(ram, mango). |
| 5. Sima is a girl. | Girl(sima) . |
| 6. Rose is red. | Red (rose). |
| 7. John owns gold. | Owns(john, gold). |
| 8. Ram is taller than mohan. | Taller(ram, mohan). |
| 9. My name is khan. | Name (khan) or Name(my, khan) |
| 10. Apple is fruit. | Fruit(apple). |

Translating English to FOPL

11. Ram is male.

Male (ram).

12. Tuna is fish.

Fish (tuna).

13. Dashrath is ram's father.

Father (dashrath, ram).

14. Kush is son of ram.

Son(kush, ram).

15. Kaushaliya is wife of Dashrath.

Wife(kaushaliya, dashrath).

16. Clinton is tall.

Tall(clinton).

17. There is a white alligator.

Alligator (white).

18. All kings are person.

$\forall x: \text{Kings}(x) \rightarrow \text{Person}(x).$

19. Nobody loves john.

$\forall x: \neg \text{Loves}(x, \text{john}).$

20. Every one has a father.

$\forall x: \exists y: \text{Father}(y, x)$

Translate into predicate logic

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. All Romans were either loyal to Caesar or hated him.
6. Everyone is loyal to someone.
7. People only try to assassinate rulers they aren't loyal to.
8. Marcus tried to assassinate Caesar.
9. All men are people.

Solution

1. Marcus was a man.
 - $\text{Man}(\text{Marcus})$.
2. Marcus was a Pompeian.
 - $\text{Pompeian}(\text{marcus})$
3. All Pompeian were Romans.
 - $\forall x: \text{Pompeian}(x) \rightarrow \text{Roman}(x)$
4. Caesar was a ruler.
 - $\text{Ruler}(\text{Caesar})$
5. All Romans were either loyal to Caesar or hated him.
 - $\forall x: \text{Roman}(x) \rightarrow \text{LoyalTo}(x, \text{Caesar}) \vee \text{Hate}(x, \text{Caesar})$

Solution

6. Everyone is loyal to someone.

$$\forall x: \exists y: \text{LoyalTo}(x,y)$$

7. People only try to assassinate rulers they aren't loyal to.

$$\forall x: \forall y: \text{Person}(x) \wedge \text{Ruler}(y) \wedge \text{TryAssassinate}(x,y) \rightarrow \neg \text{LoyalTo}(x,y)]$$

8. Marcus tried to assassinate Caesar.

$$\text{TryAssassinate}(\text{Marcus}, \text{Caesar})$$

9. All men are people.

$$\forall x: \text{Men}(x) \rightarrow \text{People}(x)$$

Translate into predicate logic

- i. Hari likes all kind of food.
- ii. Bananas are food.
- iii. Apples are food.
- iv. Anything anyone eats and isn't killed by food.
- v. Hari eats everything ram eats.

Solution

- i. $\forall x: \text{Food}(x) \rightarrow \text{Likes}(\text{hari}, x).$
- ii. $\text{Food}(\text{bananas}) .$
- iii. $\text{Food}(\text{apples}) .$
- iv. $\forall x: \exists y: \text{Eats}(y,x) \wedge \neg \text{Killedby}(y,x) \rightarrow \text{Food}(x)$
- v. $\forall x: \text{eats}(\text{ram}, x) \rightarrow \text{eats}(\text{hari}, x) .$

Translating English to FOPL

1. Every gardener likes the sun.
2. Not Every gardener likes the sun.
3. You can fool some of the people all of the time.
4. You can fool all of the people some of the time.
5. You can fool all of the people at same time.
6. You can not fool all of the people all of the time.
7. Everyone is younger than his father.

Solution

1. $(\forall x): \text{gardener}(x) \Rightarrow \text{likes}(x, \text{Sun})$
2. $\sim((\forall x) : \text{gardener}(x) \Rightarrow \text{likes}(x, \text{Sun}))$
3. $(\exists x):(\forall t) : \text{person}(x) \wedge \text{time}(t) \Rightarrow \text{can-be-fooled}(x, t)$
4. $(\forall x):(\exists t) : \text{person}(x) \wedge \text{time}(t) \Rightarrow \text{can-be-fooled}(x, t)$
5. $(\exists t):(\forall x) : \text{person}(x) \wedge \text{time}(t) \Rightarrow \text{can-be-fooled}(x, t)$
6. $\sim((\forall x):(\forall t): \text{person}(x) \wedge \text{time}(t) \Rightarrow \text{can-be-fooled}(x, t))$
7. $(\forall x) : \text{person}(x) \Rightarrow \text{younger}(x, \text{father}(x))$

Translating English to FOPL

1. All purple mushrooms are poisonous.
2. No purple mushroom is poisonous.
3. There are exactly two purple mushrooms.
4. Clinton is not tall.
5. X is above Y if X is directly on top of Y or there is a pile of one or more other objects directly on top of one another starting with X and ending with Y.
6. no one likes everyone

Solution

1. $(\forall x): (\text{mushroom}(x) \wedge \text{purple}(x)) \Rightarrow \text{poisonous}(x)$
2. $\sim(\exists x): \text{purple}(x) \wedge \text{mushroom}(x) \wedge \text{poisonous}(x)$
 $(\forall x): (\text{mushroom}(x) \wedge \text{purple}(x)) \Rightarrow \sim\text{poisonous}(x)$
3. $(\exists x): (\exists y): \text{mushroom}(x) \wedge \text{purple}(x) \wedge \text{mushroom}(y) \wedge \text{purple}(y) \wedge \sim(x=y)$
4. $\sim\text{tall}(\text{Clinton})$
5. $(\forall z): (\text{mushroom}(z) \wedge \text{purple}(z)) \Rightarrow ((x=z) \vee (y=z))$
 $(\forall x): (\forall y): \text{above}(x,y) \Leftrightarrow (\text{on}(x,y) \vee (\exists z) (\text{on}(x,z) \wedge \text{above}(z,y)))$
6. $\sim(\exists x): (\forall y): \text{likes}(x,y) \text{ or } (\forall x): (\exists y): \sim\text{likes}(x,y)$

Skolemization

- Skolemization is the process of replacement of existential quantified variable with Skolem function and deletion of the respective quantifiers.
- Skolem function is arbitrary functions which can always assume a correct value required of an existentially quantified variable.
- Example
 - $\exists x : \text{President}(x)$
 - Can be transformed into the formula
 - $\text{President}(P1)$
 - Where $P1$ is a function with no arguments that somehow produces a value that satisfies president.

Example 1

Everybody loves somebody.

$$\forall x: \exists y: (\text{Person}(x) \wedge \text{Person}(y)) \rightarrow \text{Loves}(x,y)]$$

Converted to

$$\forall x: (\text{Person}(x) \wedge \text{Person } f(x)) \rightarrow \text{Loves}(x,f(x))]$$

Where $f(x)$ specifies the person that x loves.

Clausal Form

- A formula is said to be in clausal form if it is of the form:

$$\forall x_1 \forall x_2 \dots \forall x_n [C_1 \wedge C_2 \wedge \dots \wedge C_k]. \quad \square$$

- All first-order logic formulas can be converted to clausal form.

Equivalent Logical Expressions

- i. $\sim(\sim F) = F$ (Double Negation)
- ii. $F \& G = G \& F, F \vee G = G \vee F$ (Commutativity)
- iii. $(F \& G) \& H = F \& (G \& H), (F \vee G) \vee H = F \vee (G \vee H)$ (Associativity)
- iv. $F \vee (G \& H) = (F \vee G) \& (F \vee H), F \& (G \vee H) = (F \& G) \vee (F \& H)$ (Distributivity)
- v. $\sim(F \& G) = \sim F \vee \sim G, \sim(F \vee G) = \sim F \& \sim G$ (De Morgan)
- vi. $F \rightarrow G = \sim F \vee G$
- vii. $F \leftrightarrow G = (\sim F \vee G) \& (\sim G \vee F)$
- viii. $\forall x F[x] \vee G = \forall x (F[x] \vee G)$
- ix. $\exists x F[x] \vee G = \exists x (F[x] \vee G)$
- x. $\forall x F[x] \& G = \forall x (F[x] \& G)$
- xi. $\exists x F[x] \& G = \exists x (F[x] \& G)$

Equivalent Logical Expressions...

xii. $\sim(\forall x) F[x] = \exists x (\sim F[x])$

xiii. $\sim(\exists x) F[x] = \forall x (\sim F[x])$

xiv. $\forall x F[x] \ \& \ \forall x G[x] = \forall x (F[x] \ \& \ G[x])$

xv. $\exists x F[x] \ \& \ \exists x G[x] = \exists x (F[x] \ \& \ G[x])$

Conversion to Clausal form or Conjunctive Normal Form (CNF)

1. Eliminate logical implications, \Rightarrow , using the fact that $A \Rightarrow B$ is equivalent to $\neg A \vee B$.
2. Reduce the scope of each negation to a single term, using the following facts:
 $\neg(\neg P) = P$
 $\neg(A \vee B) = \neg A \wedge \neg B$
 $\neg(A \wedge B) = \neg A \vee \neg B$
 $\neg \forall x: P(x) = \exists x: \neg P(x)$
 $\neg \exists x: P(x) = \forall x: \neg P(x)$
3. Standardize variables so that each quantifier binds a unique variable.
4. Move all quantifiers to the left, maintaining their order.
5. Eliminate existential quantifiers, using Skolem functions (functions of the preceding universally quantified variables).
6. Drop the prefix; assume universal quantification.
7. Convert the matrix into a conjunction of disjunctions. $[(a \ \&b) \text{ or } c] = (a \text{ or } c) \ \& \ (b \text{ or } c)$
8. Create a separate clause corresponding to each conjunction.
9. Standardize apart the variables in the clauses.

Example of Clausal Conversion

- $\exists x \forall y (\forall z P(f(x), y, z) \rightarrow (\exists u Q(x, u) \& \exists v R(y, v)))$
- **Step 1:** Eliminate logical implications
 - $\exists x \forall y (\sim(\forall z) P(f(x), y, z) \vee (\exists u Q(x, u) \& (\exists v) R(y, v)))$
- **Step 2:** Reduce the scope of each negation to a single term
 - $\exists x \forall y (\exists z \sim P(f(x), y, z) \vee (\exists u Q(x, u) \& (\exists v) R(y, v)))$
- **Step 3:**
 - It is not require here because quantifiers have different variable assignments.
- **Step 4:** Move all quantifiers to the left
 - $\exists x \forall y \exists z \exists u \exists v (\sim P(f(x), y, z) \vee (Q(x, u) \& R(y, v)))$
- **Step 5:** Skolem functions
 - $\forall y (\sim P f(x) , y, g(y)) \vee Q(a, h(y)) \& R(y, l(y))$
- **Step 6:** Drop the prefix
 - $(\sim P f(x) , y, g(y)) \vee Q(a, h(y)) \& R(y, l(y))$

Example of Clausal Conversion

- **Step 7:** Convert the matrix into a conjunction of disjunctions
 - $((\sim P f(x), y, g(y)) \vee Q(a, h(y)) \wedge (\sim P f(x), y, g(y)) \vee R(y, l(y)))$
- **Step 8:**
 - $(\sim P f(x), y, g(y)) \vee Q(a, h(y))$
 - $(\sim P f(x), y, g(y)) \vee R(y, l(y))$
- **Step 9:** Standardize apart the variables in the clauses

Translate into Clausal form

- i. Hari likes all kind of food.
- ii. Bananas are food.
- iii. Apples are food.
- iv. Anything anyone eats and isn't killed by food.
- v. Hari eats everything ram eats.

Convert it into clausal form

- i. $\forall x: \text{Food}(x) \rightarrow \text{Likes}(\text{hari}, x).$
- ii. $\text{Food}(\text{bananas}) .$
- iii. $\text{Food}(\text{apples}) .$
- iv. $\forall x: \exists y: \text{Eats}(y, x) \wedge \neg \text{Killedby}(y, x) \rightarrow \text{Food}(x)$
- v. $\forall x: \text{Eats}(\text{ram}, x) \rightarrow \text{Eats}(\text{hari}, x) .$

Solution

- i. $\neg \text{Food}(x) \vee \text{Likes}(\text{hari}, x).$
- ii. $\text{Food}(\text{bananas}) .$
- iii. $\text{Food}(\text{apples}) .$
- iv. $\neg \text{Eats}(y,x) \vee \text{Killedby}(y,x) \vee \text{Food}(x)$
- v. $\neg \text{Eats}(\text{ram}, x) \vee \text{eats}(\text{hari}, x) .$

Convert it into FOPL

- i. All lectures are determined.

$$\forall x: \text{Lecturer}(x) \rightarrow \text{Determined}(x)$$

- ii. Any one who is determined and intelligent will give good service.

$$\forall x: \text{Determined}(x) \wedge \text{Intelligent}(x) \rightarrow \text{Givegoodservice}(x)$$

- iii. Mary is an intelligent lecturer.

$\text{Lecturer}(\text{mary})$

$\text{Intelligent}(\text{mary})$

Convert FOPL into clausal form

- i. All lectures are determined.
 $\forall x: \text{Lecturer}(x) \rightarrow \text{Determined}(x)$
 $\neg \text{Lecturer}(x) \vee \text{Determined}(x)$
- i. Any one who is determined and intelligent will give good service.
 $\forall x: \text{Determined}(x) \wedge \text{Intelligent}(x) \rightarrow \text{Givegoodservice}(x)$
 $\neg \text{Determined}(x) \vee \neg \text{Intelligent}(x) \vee \text{Givegoodservice}(x)$
- iii. Mary is an intelligent lecturer.
 $\text{Lecturer}(\text{mary})$
 $\text{Intelligent}(\text{mary})$
(Both are same)

Horn clause

- It is a clause with at most one positive literal.
- Example
 - P
 - $\sim P \vee Q$
 - $\sim P \vee \sim Q \vee R$

Clauses of this type were first investigated by the logician Alfred Horn.
There are three types of Horn clauses

- i. A single atom: often called a “fact”.
- ii. An implication: often called a “rule”- whose antecedent consists of a conjunction of positive literals and whose consequent consists of a single positive literal.
- iii. A set of negative literals: written in implication form with an antecedent consisting of a conjunction of positive literals and an empty consequent. This form is obtained, for example, when one negates a wff to be proved consisting of a conjunction of positive literals. Such a clause is therefore often called a “goal”.

Horn clause

- In a horn clause, generally one condition is followed by zero, or more conditions. It is represented as

Conclusion..

Condition-1,

Condition-2,

Condition-3,

.

Condition-n,

The conclusion is true if and only if condition-1, condition-2.....and so on until condition –n is true.

In simple and easy terms, a Horn clause consists of a set of statements joined by logical AND's.

Example of FOPL

1. Marcus was a man.

➤ $\text{Man}(\text{Marcus})$.

2. Marcus was a Pompeian.

➤ $\text{Pompeian}(\text{marcus})$

3. All Pompeian were Romans.

➤ $\forall x: \text{Pompeian}(x) \rightarrow \text{Roman}(x)$

4. Caesar was a ruler.

➤ $\text{Ruler}(\text{Caesar})$

5. All Romans were either loyal to Caesar or hated him.

➤ $\forall x: \text{Roman}(x) \rightarrow \text{LoyalTo}(x, \text{Caesar}) \vee \text{Hate}(x, \text{Caesar})$

Example of FOPL

6. Everyone is loyal to someone.

$$\forall x: \exists y: \text{LoyalTo}(x,y)$$

7. People only try to assassinate rulers they aren't loyal to.

$$\forall x: \forall y: \text{Person}(x) \wedge \text{Ruler}(y) \wedge \text{TryAssassinate}(x,y) \rightarrow \neg \text{LoyalTo}(x,y)]$$

8. Marcus tried to assassinate Caesar.

$$\text{TryAssassinate}(\text{Marcus}, \text{Caesar})$$

9. All men are people.

$$\forall x: \text{Men}(x) \rightarrow \text{People}(x)$$

Prove this

- Given the above sentences, can we make a conclusion as follows:

“Marcus was not loyal to Caesar ?”

or:

$\neg \text{loyalto}(\text{Marcus}, \text{Caesar})$

Solution:

In order to prove the goal, we need to use the rules of inference to transform it into another goal that can in turn be transformed, and so on, until there are no unsatisfied goals remaining.

Solution

$\neg \text{loyalto}(\text{Marcus}, \text{Caesar})$
 \uparrow (7, substitution)
 $\text{person}(\text{Marcus}) \wedge$
 $\text{ruler}(\text{Caesar}) \wedge$
 $\text{tryassassinate}(\text{Marcus}, \text{Caesar})$
 \uparrow (4)
 $\text{person}(\text{Marcus})$
 $\text{tryassassinate}(\text{Marcus}, \text{Caesar})$
 \uparrow (8)
 $\text{person}(\text{Marcus})$

Fig. 5.2 *An Attempt to Prove $\neg \text{loyalto}(\text{Marcus}, \text{Caesar})$*

Unification

- Any substitution that makes two or more expression equal is called a unifier for the expression.
- Two formulas unify if they can be made identical.
- A unification is a function that assigns bindings to variables.
- A binding is either a constant, a functional expression or another variable.

Unification process

- The basic idea of unification is very simple.
- To attempt to unify two literals, we first check if their initial predicate symbols are the same. If so, we can proceed.
- Otherwise, there is no way they can be unified, regardless of their arguments. For example, the two literals

tryassassinate (Marcus, Caesar)

hate(Marcus, Caesar)

cannot be unified.

If the predicate symbols match, then we must check the arguments, one pair at a time. If the first matches, we can continue with the second, and so on.

Example

$P(x, x)$	$P(A, A)$	$\{x/A\}$
$P(x, x)$	$P(A, B)$	fail
$P(x, y)$	$P(A, B)$	$\{x/A, y/B\}$
$P(x, y)$	$P(A, A)$	fail
$P(x, y)$	$P(A, z)$	$\{x/A, y/z\}$
$P(f(x), y)$	$P(f(A), B)$	$\{x/A, y/B\}$
$P(x, y)$	$P(f(x), B)$	fail
$P(x, y)$	$P(z, f(z))$	$\{x/z, y/f(z)\}$

Unification algorithm

- $\text{Unify}(L1, L2)$ returns a list representing the composition of the substitutions that were performed during the match.
- The empty list, NIL, indicates that a match was found without any substitutions.
- The list consisting of the single value FAIL indicates that the unification procedure failed.
- The final substitution produced by the unification process will be used by the resolution procedure.

Unification algorithm

Algorithm: *Unify*($L1$, $L2$)

- I. If $L1$ or $L2$ are both variables or constants, then:
 - (a) If $L1$ and $L2$ are identical, then return NIL.
 - (b) Else if $L1$ is a variable, then if $L1$ occurs in $L2$ then return {FAIL}, else return ($L2/L1$).
 - (c) Else if $L2$ is a variable, then if $L2$ occurs in $L1$ then return {FAIL} , else return ($L1/L2$).
 - (d) Else return {FAIL}.
2. If the initial predicate symbols in $L1$ and $L2$ are not identical, then return {FAIL}.
3. If $L1$ and $L2$ have a different number of arguments, then return {FAIL}.
4. Set $SUBST$ to NIL. (At the end of this procedure, $SUBST$ will contain all the substitutions used to unify $L1$ and $L2$.)
5. For $i \leftarrow 1$ to number of arguments in $L1$:
 - (a) Call Unify with the i th argument of $L1$ and the i th argument of $L2$, putting result in S .
 - (b) If S contains FAIL then return {FAIL}.
 - (c) If S is not equal to NIL then:
 - (i) Apply S to the remainder of both $L1$ and $L2$.
 - (ii) $SUBST := \text{APPEND}(S, SUBST)$.
6. Return $SUBST$.

Solve

- (Unification) For each pair of atomic sentences, give the most general unifier if it exists, otherwise say “fail”:
 - a. $R(A, x), R(y, z)$
 - b. $P(A, B, B), P(x, y, z)$
 - c. $Q(y, G(A, B)), Q(G(x, x), y)$
 - d. $\text{Older}(\text{Father}(y), y), \text{Older}(\text{Father}(x), \text{John})$
 - e. $\text{Knows}(\text{Father}(y), y), \text{Knows}(x, x)$

Solution

- (Unification) For each pair of atomic sentences, give the most general unifier if it exists, otherwise say “fail”:
 - a. $R(A, x), R(y, z)$ $y/A, x/z$
 - b. $P(A, B, B), P(x, y, z)$ $x/A, y/B, z/B$
 - c. $Q(y, G(A, B)), Q(G(x, x), y)$ fail
 - d. $\text{Older}(\text{Father}(y), y), \text{Older}(\text{Father}(x), \text{John})$
 $x/y, y/\text{John}$
 - e. $\text{Knows}(\text{Father}(y), y), \text{Knows}(x, x)$ fail

Resolution Principle

- Given two clauses C_1 and C_2 with no variables in common, if there is a literal l_1 in C_1 and which is a complement of a literal l_2 in C_2 , both l_1 and l_2 are deleted and a disjuncted C is formed the remaining reduced clauses. The new clauses C is called the resolve of C_1 and C_2 .
- Resolution is the process of generating these resolvents from the set of clauses.

Example: $(\sim P \vee Q)$ and $(\sim Q \vee R)$

- We can write

$$(\sim P \vee Q), (\sim Q \vee R)$$

$$\sim P \vee R$$

Refutation

- Resolution produces proofs by refutation.
- In other words, to prove a statement, resolution attempts to show that the negation of the statement produces a contradiction with the known statements.

Example of Resolution

- Consider the following clauses:-

A : $P \vee Q \vee R$

B: $\sim P \vee Q \vee R$

C: $\sim Q \vee R$

- Solution

A: $P \vee Q \vee R$ (Given in the problem)

B : $\sim P \vee Q \vee R$ (Given in the problem)

D : $Q \vee R$ (Resolvent of A and B)

C: $\sim Q \vee R$ (Given in the problem)

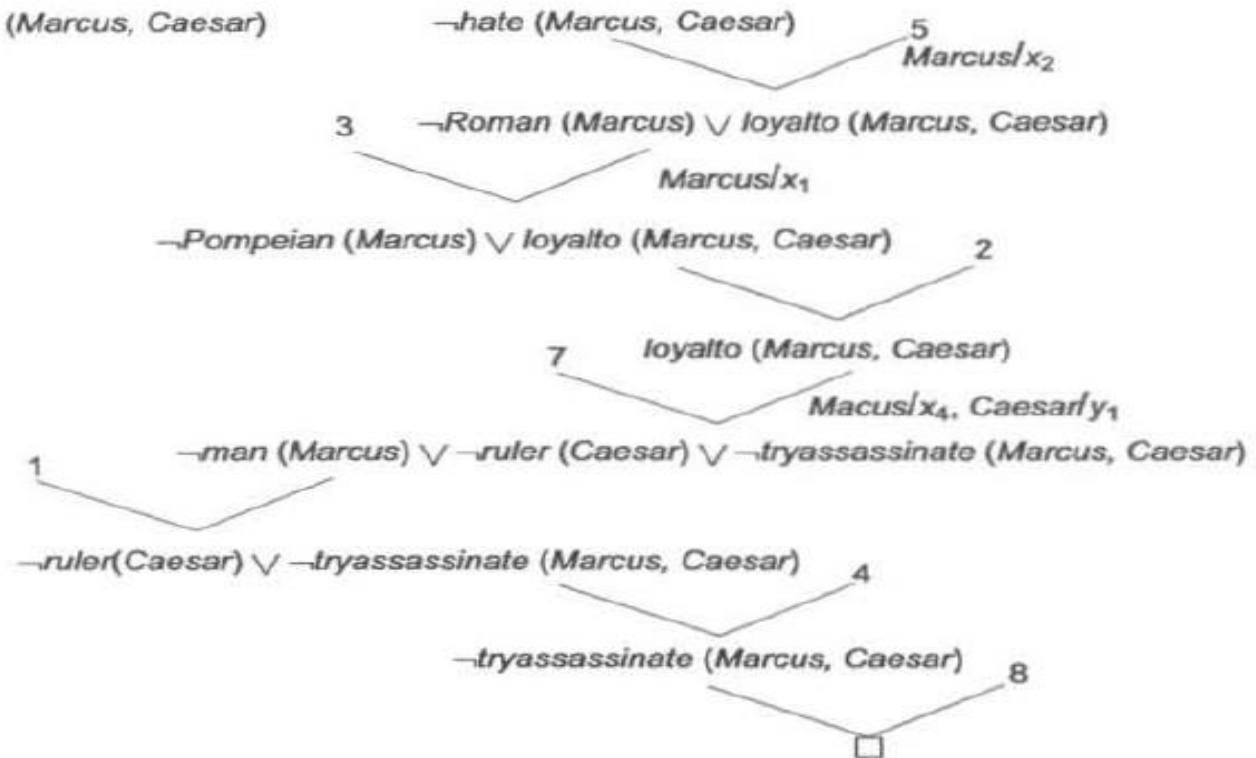
E: R (Resolvent of C and D)

Axioms in clause form:

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $\neg Pompeian(x_1) \vee Roman(x_1)$
4. $ruler(Caesar)$
5. $\neg Roman(x_2) \vee loyalto(x_2, Caesar) \vee hate(x_2, Caesar)$
6. $loyalto(x_3, fl(x_3))$
7. $\neg man(x_4) \vee \neg ruler(y_1) \vee \neg tryassassinate(x_4, y_1) \vee loyalto(x_4, y_1)$
8. $tryassassinate(Marcus, Caesar)$

(a)

Prove: $hate(Marcus, Caesar)$



Example of resolution

STT	Clauses
1	man(Marcus)
2	Pompiean(Marcus)
3	\neg Pompiean(X1) v Roman(X1)
4	Ruler(Caesar)
5	\neg Roman(X2) v loyalto(X2, Caesar) v hate(X2, Caesar)
6	Loyalto(X3, f1(X3))
7	\neg man(X4) v \neg ruler(Y1) v \neg tryassassinate(X4, Y1) v loyalto(X4, Y1)
8	Tryassassinate(Marcus, Caesar)

Prove: "Marcus hate Caesar," or
hate(Marcus, Caesar).

Prepared by- Agniwesh Mishra, Rungta
College of Engg. & Tech., Bhilai

Solution

#	Clauses	Note
1	man(Marcus)	P
2	Pompiean(Marcus)	P
3	\neg Pompiean(X1) v Roman(X1)	P
4	Ruler(Caesar)	P
5	\neg Roman(X2) v loyalto(X2, Caesar) v hate(X2, Caesar)	P
6	Loyalto(X3, f1(X3))	P
7	\neg man(X4) v \neg ruler(Y1) v \neg tryassassinate(X4, Y1) v \neg loyalto(X4, Y1)	P
8	Tryassassinate(Marcus, Caesar)	P
9	\neg hate(Marcua, Ceasar).	P
10	\neg Roman(Marcus) v loyalto(Marcus, Caesar)	5,9: X2= Marcus
11	\neg Pompiean(Marcus) v loyalto(Marcus, Caesar)	3,10: X1 = Marcus

Solution

STT	Clauses	Ghi chú
12	loyalto(Marcus, Ceasar)	2,11:
13	$\neg \text{man}(\text{Marcus}) \vee \neg \text{ruler}(\text{Ceasar}) \vee$ $\neg \text{tryassassinate}(\text{Marcus}, \text{Ceasar})$	12,7:X4=Marcus, y1=Ceasar
14	$\neg \text{ruler}(\text{Ceasar}) \vee \neg \text{tryassassinate}(\text{Marcus},$ $\text{Ceasar})$	13,1
15	$\neg \text{tryassassinate}(\text{Marcus}, \text{Ceasar})$	14,4
16	\square	15,8

Example

Assume the following facts:

- i. “Steve only likes easy courses.
- ii. Science courses are hard.
- iii. All the courses in Humanities Department are easy.
- iv. HM101 is a course in Humanities”.

Convert the above statements into appropriate wffs so that the resolution can be performed to answer the question. “ what course would steve like?”

Solution

First we will convert it into FOPL (First order predicate logic)

i. “Steve only likes easy courses.

$\forall x: \text{easy}(x) \rightarrow \text{likes}(\text{steve}, x)$

ii. Science courses are hard.

$\forall x: \text{science}(x) \rightarrow \sim \text{easy}(x)$

iii. All the courses in Humanities Department are easy.

$\forall x: \text{humanities}(x) \rightarrow \text{easy}(x)$

iv. HM101 is a course in Humanities”.

$\text{humanities}(\text{HM101})$

The conclusion is encoded as $\text{likes}(\text{steve}, x)$.

Solution continued....

- First we put our premises in the clause form and the negation of conclusion to our set of clauses (we use numbers in parentheses to number the clauses):

(1) $\sim \text{easy}(x) \vee \text{likes}(\text{steve}, x)$

(2) $\sim \text{science}(x) \vee \sim \text{easy}(x)$

(3) $\sim \text{humanities}(x) \vee \text{easy}(x)$

(4) $\text{humanities}(\text{HM101})$

(5) $\sim \text{likes}(\text{steve}, x)$

Solution continued....

St	Clauses	Note
1	$\sim \text{easy}(x) \vee \text{likes}(\text{steve}, x)$	P
2	$\sim \text{science}(x) \vee \sim \text{easy}(x)$	P
3	$\sim \text{humanities}(x) \vee \text{easy}(x)$	P
4	$\text{humanities}(\text{HM101})$	P
5	$\sim \text{likes}(\text{steve}, x)$	P
6	$\sim \text{easy}(x)$	1 & 5
7	$\sim \text{humanities}(x)$	3 & 6
8	NIL	4 & 7, $x = \text{HM101}$

Solution continued....

- A resolution proof may be obtained by the following sequence of resolutions
- (6) 1&5 yields resolvent $\sim\text{easy}(x)$.
- (7) 3&6 yields resolvent $\sim\text{humanities}(x)$.
- (8) 4&7 yields empty clause; the substitution $x/\text{HM101}$ is produced by the unification algorithm which says that the only wff of the form $\text{likes}(\text{steve}, x)$ which follows from the premises is $\text{likes}(\text{steve}, \text{HM101})$. Thus, resolution gives us a way to find additional assumptions.

Example

Problem Statement:

1. Ravi likes all kind of food.
2. Apples and chicken are food
3. Anything anyone eats and is not killed is food
4. Ajay eats peanuts and is still alive
5. Rita eats everything that Ajay eats.

Prove by resolution that Ravi likes peanuts using resolution.

Solution

Step 1: Converting the given statements into Predicate/Propositional Logic

- i. $\forall x : \text{food}(x) \rightarrow \text{likes}(\text{Ravi}, x)$
- ii. $\text{food}(\text{Apple}) \wedge \text{food}(\text{chicken})$
- iii. $\forall a : \forall b : \text{eats}(a, b) \wedge \text{killed}(a) \rightarrow \text{food}(b)$
- iv. $\text{eats}(\text{Ajay}, \text{Peanuts}) \wedge \text{alive}(\text{Ajay})$
- v. $\forall c : \text{eats}(\text{Ajay}, c) \rightarrow \text{eats}(\text{Rita}, c)$
- vi. $\forall d : \text{alive}(d) \rightarrow \sim \text{killed}(d)$
- vii. $\forall e : \sim \text{killed}(e) \rightarrow \text{alive}(e)$

Conclusion: $\text{likes}(\text{Ravi}, \text{Peanuts})$

Solution continued...

Step 2: Convert into CNF

- i. $\sim \text{food}(x) \vee \text{likes}(\text{Ravi}, x)$
- ii. $\text{Food}(\text{apple})$
- iii. $\text{Food}(\text{chicken})$
- iv. $\sim \text{eats}(a, b) \vee \text{killed}(a) \vee \text{food}(b)$
- v. $\text{Eats}(\text{Ajay}, \text{Peanuts})$
- vi. $\text{Alive}(\text{Ajay})$
- vii. $\sim \text{eats}(\text{Ajay}, c) \vee \text{eats}(\text{Rita}, c)$
- viii. $\sim \text{alive}(d) \vee \sim \text{killed}(d)$
- ix. $\text{Killed}(e) \vee \text{alive}(e)$

Conclusion: likes (Ravi, Peanuts)

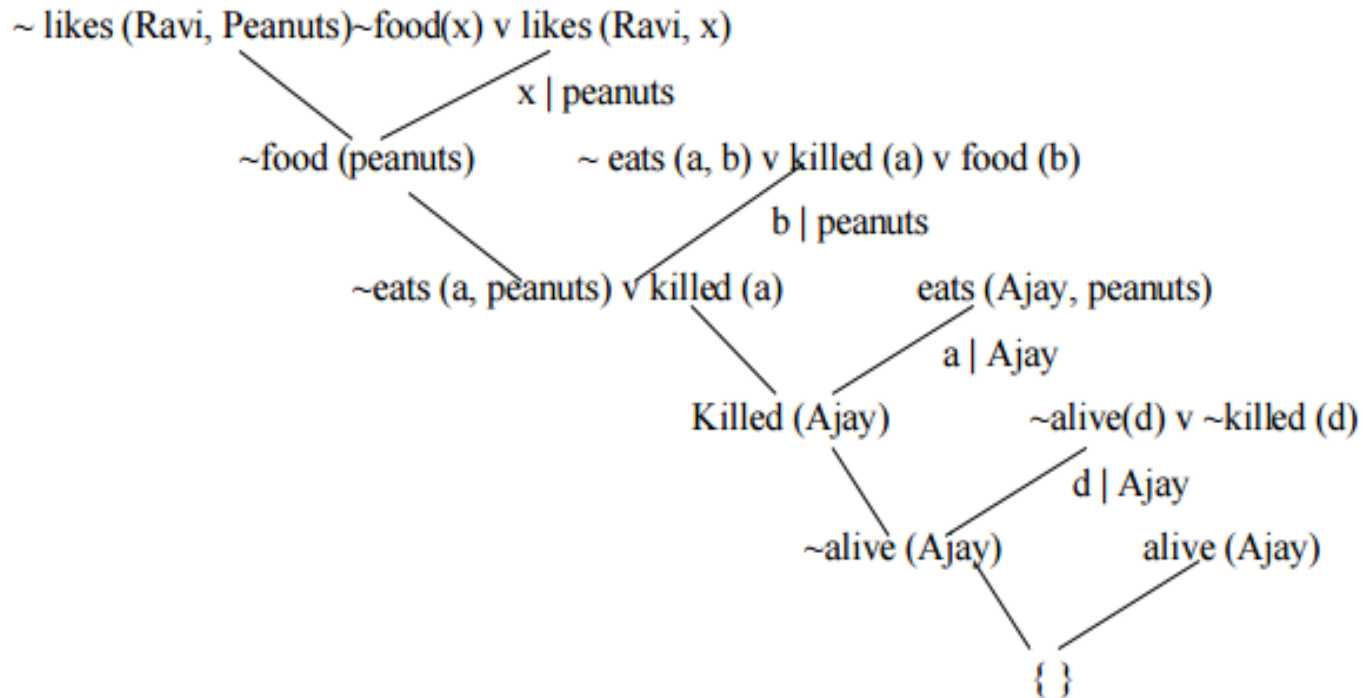
Solution continued...

Step 3: Negate the conclusion

\sim likes (Ravi, Peanuts)

Step 4: Resolve using a resolution tree

Solution continued...



Hence we see that the negation of the conclusion has been proved as a complete contradiction with the given set of facts. Hence the negation is completely invalid or false or the assertion is completely valid or true.

Structured Knowledge

Type of Knowledge Structure

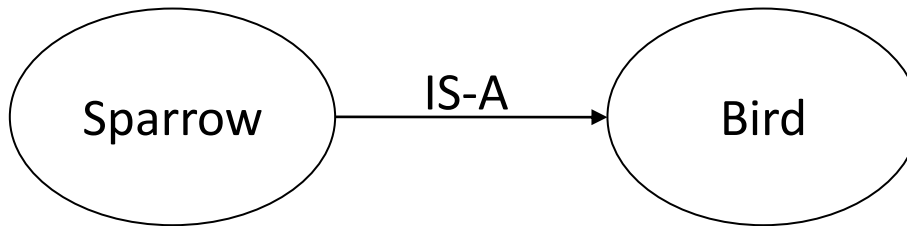
- Weak Slot - Filler Structure
 - Semantic Nets
 - Frame
- Strong Slot - Filler Structure
 - Scripts
 - Conceptual Dependency

Semantic Nets

- Semantic network or a semantic net is a structure for representing knowledge as a pattern of interconnected nodes and arcs.
- It is also representation of knowledge.
- Node in the semantic net represent either
 - Entities,
 - Attributes,
 - State or Events.
- Arcs in the net give the relationship between the nodes.
- Labels on the arc specify what type of relationship actually exists.

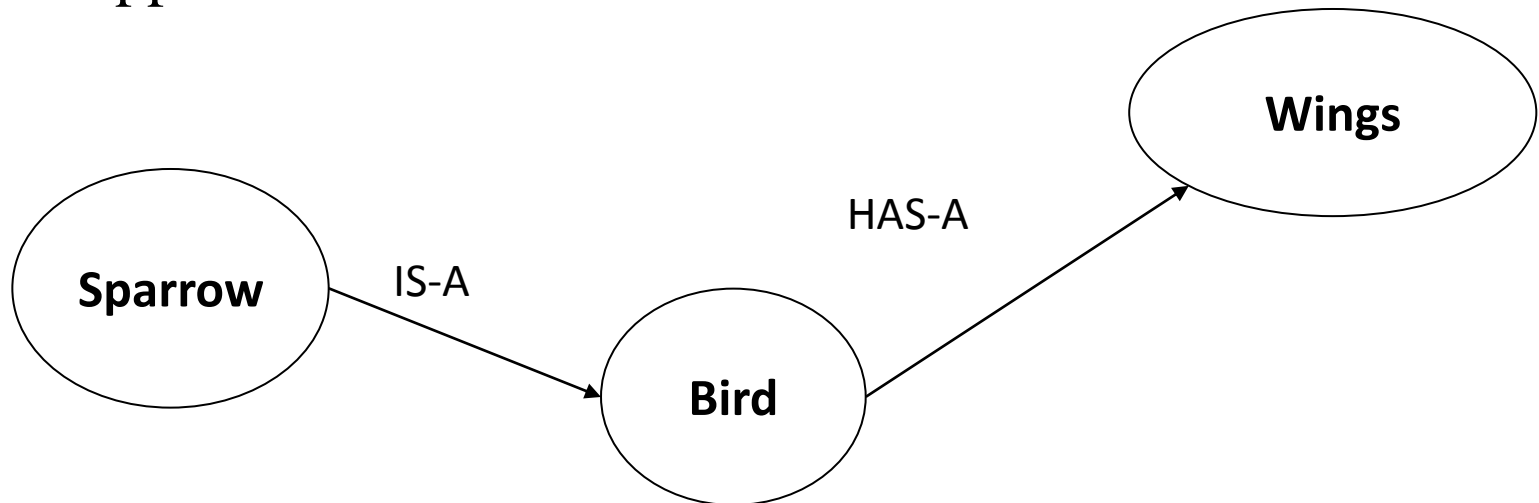
Example: Semantic networks...

- “A sparrow is a bird”
 - Two concepts: “sparrow” and “bird”
 - sparrow is a kind of bird, so connect the two concepts with a *IS-A relation*
 - This is an higher-lower relation or abstract-concrete relation



Example: Semantic networks...

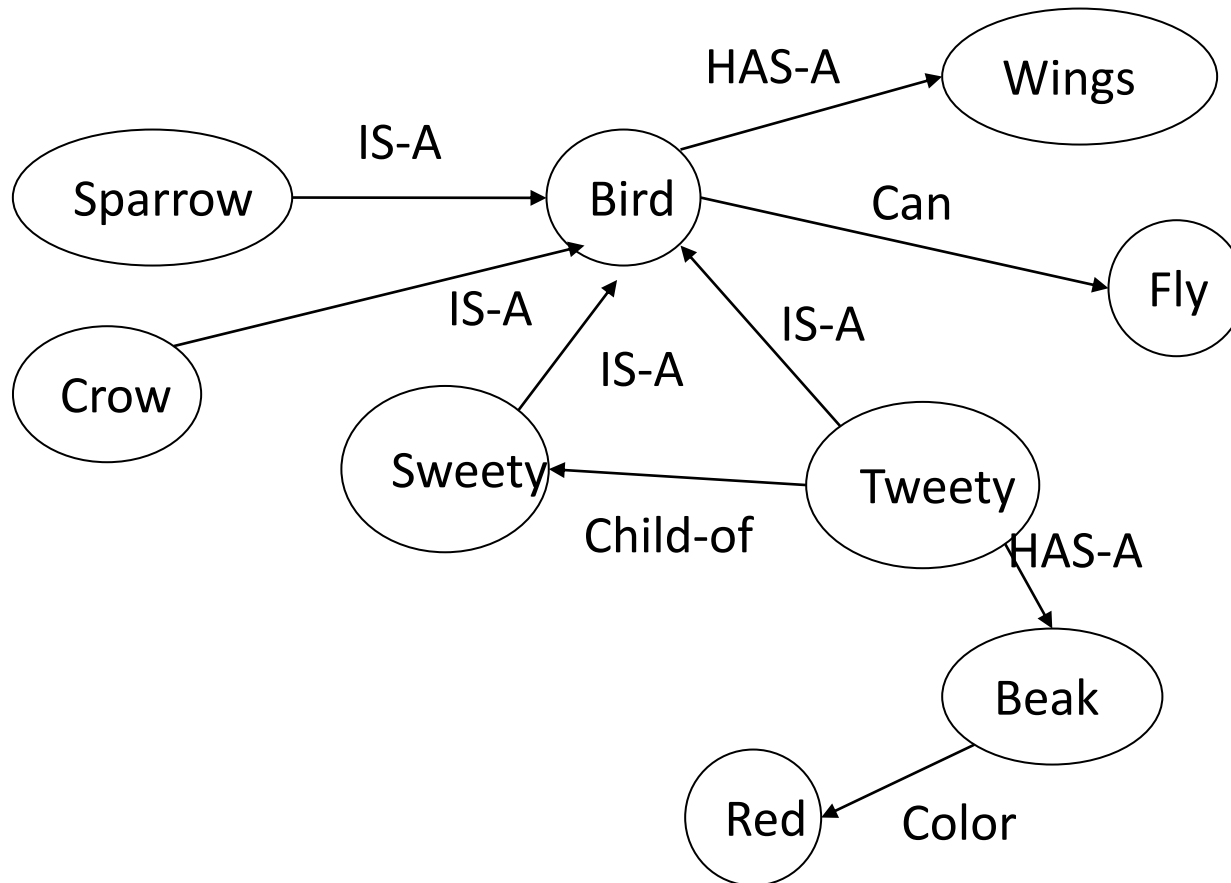
- “A bird has wings”
 - This is a different relation: the part-whole relation
 - Represented by a HAS-A link or PART-OF link
 - The link is from whole to part, so the direction is the opposite of the IS-A link



Example: Semantic Networks...

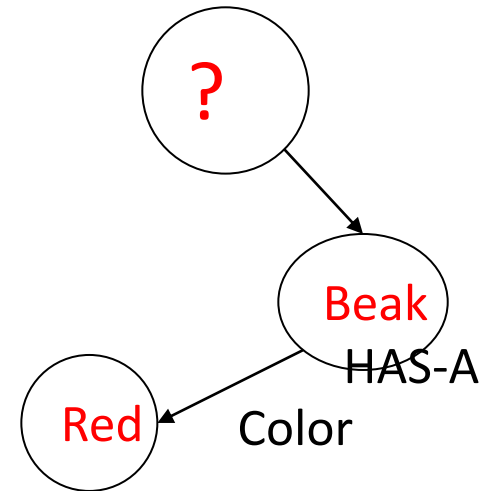
- Tweety and Sweety are birds
- Tweety has a red beak
- Sweety is Tweety's child
- A crow is a bird
- Birds can fly
- Sparrow is a bird.
- Sparrow has a wing.

Example: Semantic networks...



Semantic networks can answer queries

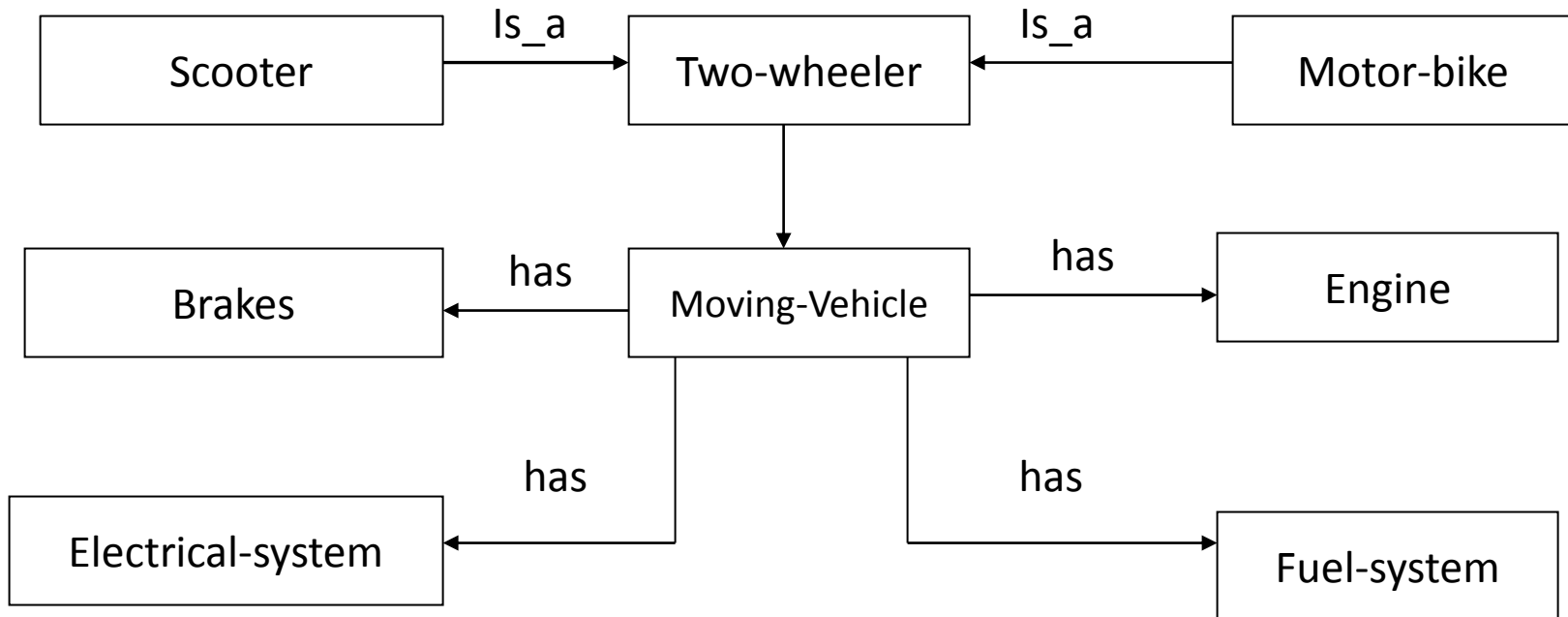
- *Query*: “Which birds have red beaks?”
 - *Answer*: Tweety
 - *Method*: Direct match of subgraph
-
- *Query*: “Can Tweety fly?”
 - *Answer*: Yes
 - *Method*: Following the IS-A link from “Tweety” to “bird” and the property link of “bird” to “fly”



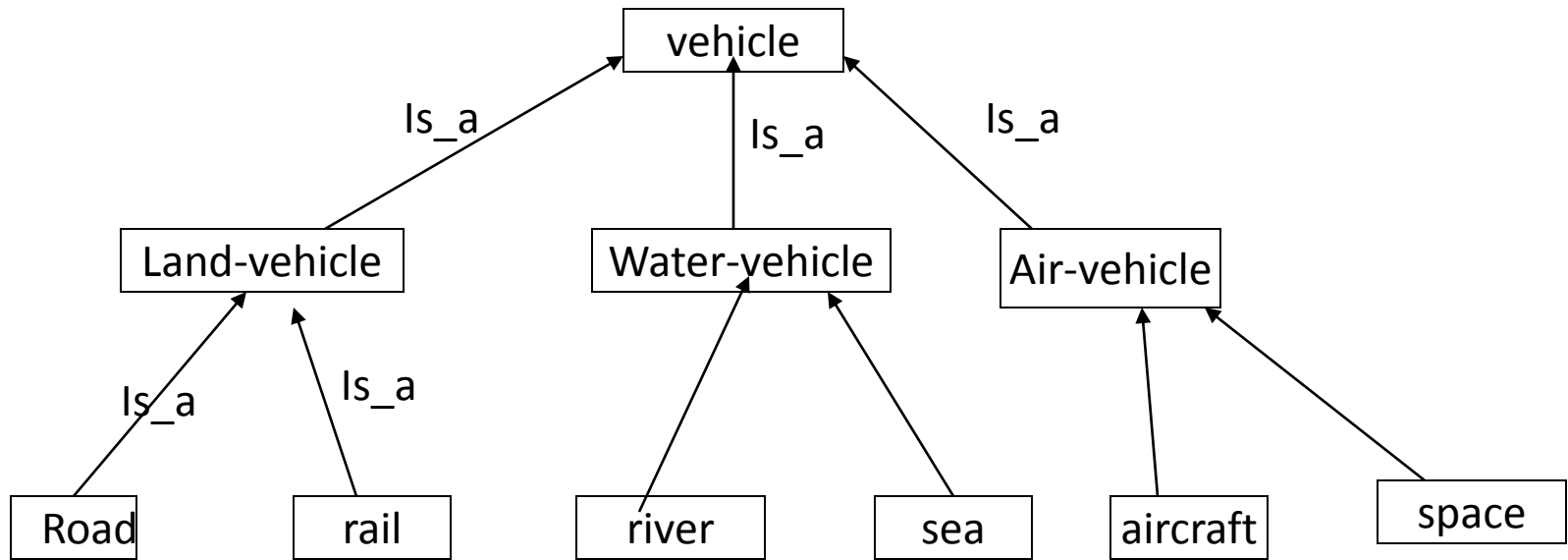
Example: Semantic Networks..

- Scooter is a two wheeler.
- Motor-bike is a two wheeler.
- Motor-bike is a moving-vehicle.
- Moving –vehicle has engine.
- Moving-vehicle has electrical system.
- Moving-vehicle has fuel system.

Example: Semantic Networks...



Hierarchical Structure



Give semantic network representation for the following facts –

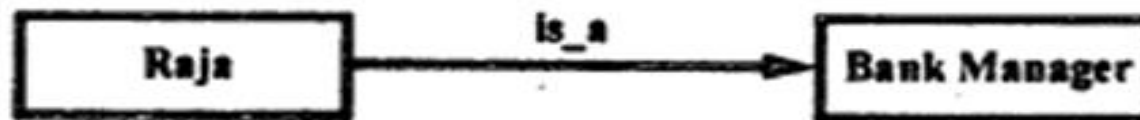
- (i) Raja is a bank manager**
- (ii) Raja works in SBI located in M.I.T.S. campus.**
- (iii) Raja is 26 years old**
- (iv) Raja has blue eyes**
- (v) Raja is taller than Piyush.**

Ans. These facts can be represented in semantic network as follows –

- (i) Raja is a bank manager.**

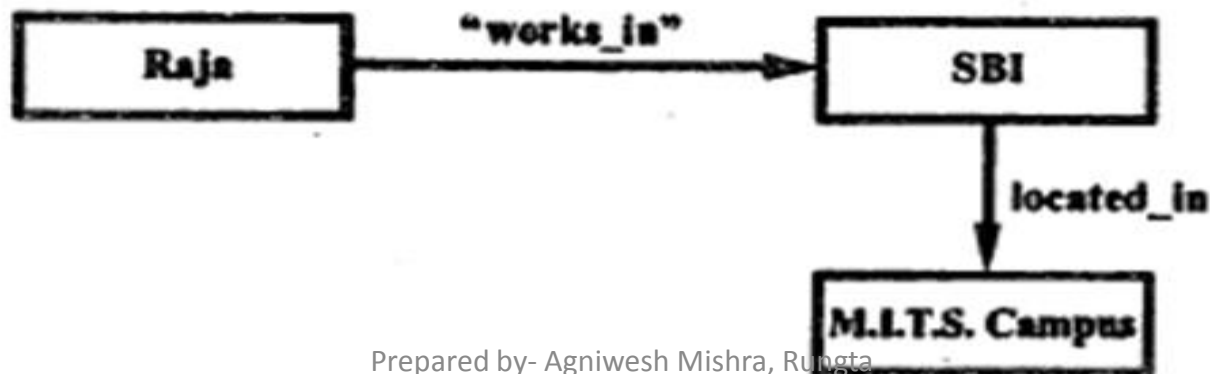
Here link between nodes is “is_a”.

So, it can be represented as –



- (ii) Raja works in SBI located in M.I.T.S. campus.**

Here we will have two links “works_in and located_in.



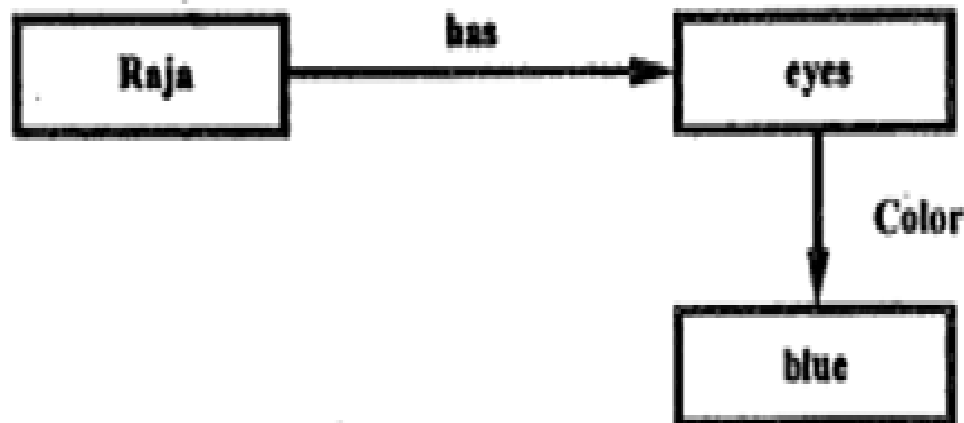
(iii) Raja is 26 years old.



In this only one link “age” shows the relation between Raja and 26 that 26 is age of Raja.

Indirectly one can say that Raja is 26 years old.

(iv) Raja has blue eyes.



This shows that Raja has eyes and color of eyes is blue, so Raja has blue eyes.

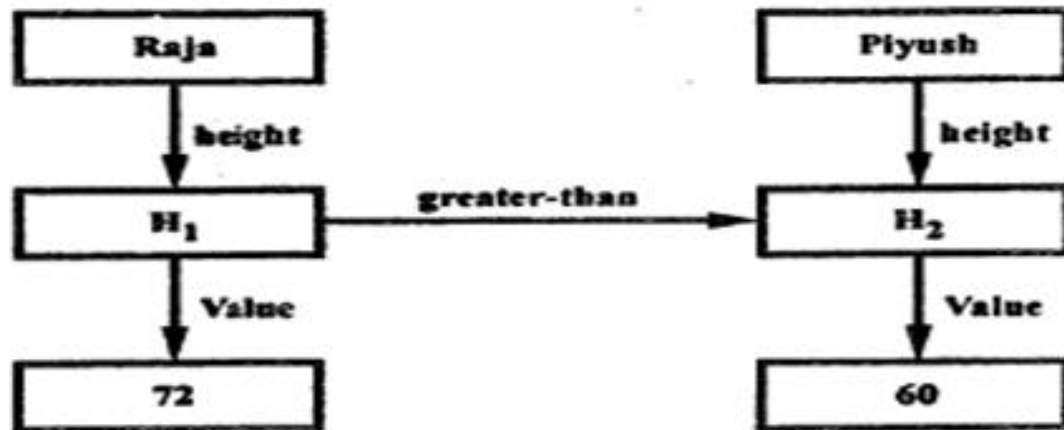
(v) Raja is taller than Piyush.



Here, we have little bit different representation that Raja and Piyush have height H_1 and H_2 respectively.

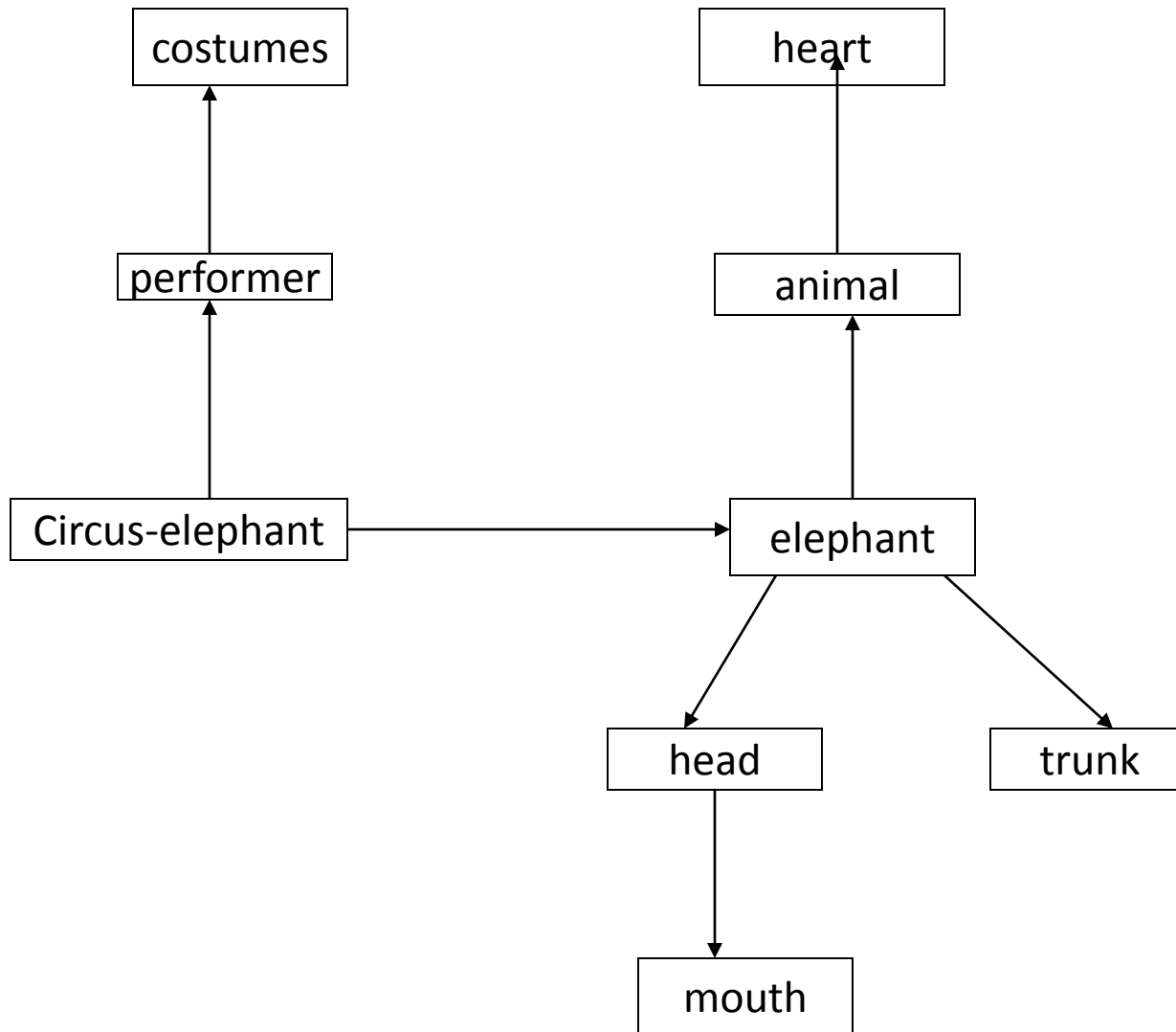
Next, we show that height of Raja is greater than height of Piyush. So, Raja is taller than Piyush.

If we would have statement that Raja has height 6 feet and Piyush has height 5 feet, i.e., if we wish to represent values too then representation would be as follows –



Represent following information in Semantic net

- (is_a circus-elephant elephant)
- (has elephant head)
- (has elephant trunk)
- (has head mouth)
- (is_a elephant animal)
- (has animal heart)
- (is_a circus-elephant performer)
- (has performer costumes)



Semantic networks

- Advantages of semantic networks
 - Simple representation, easy to read
 - Associations possible
 - Inheritance possible
- Disadvantages of semantic networks
 - A separate inference procedure (interpreter) must be build
 - The validity of the inferences is not guaranteed
 - For large networks the processing is inefficient

Partitioned Semantic Networks

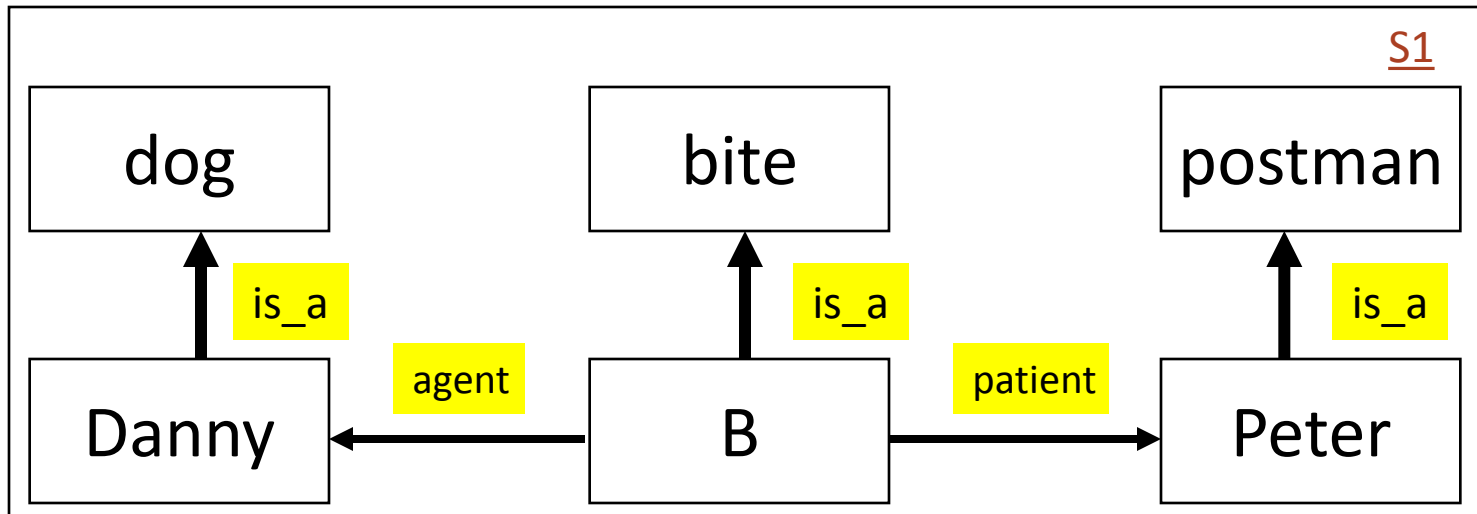
- Hendrix developed the so-called **partitioned semantic network** to represent the difference between the description of an individual object or process and the description of a set of objects. The set description involves **quantification**.
- Hendrix partitioned a semantic network whereby a semantic network, loosely speaking, can be **divided** into one or more networks for the description of an individual.

Partitioned Semantic Networks

- The central idea of partitioning is to allow groups, nodes and arcs to be bundled together into units called **spaces** – fundamental entities in partitioned networks, on the same level as nodes and arcs (Hendrix).
- Every node and every arc of a network belongs to (or lies in/on) one or more spaces.
- Some spaces are used to encode 'background information' or generic relations; others are used to deal with specifics called 'scratch' space.

Partitioned Semantic Networks

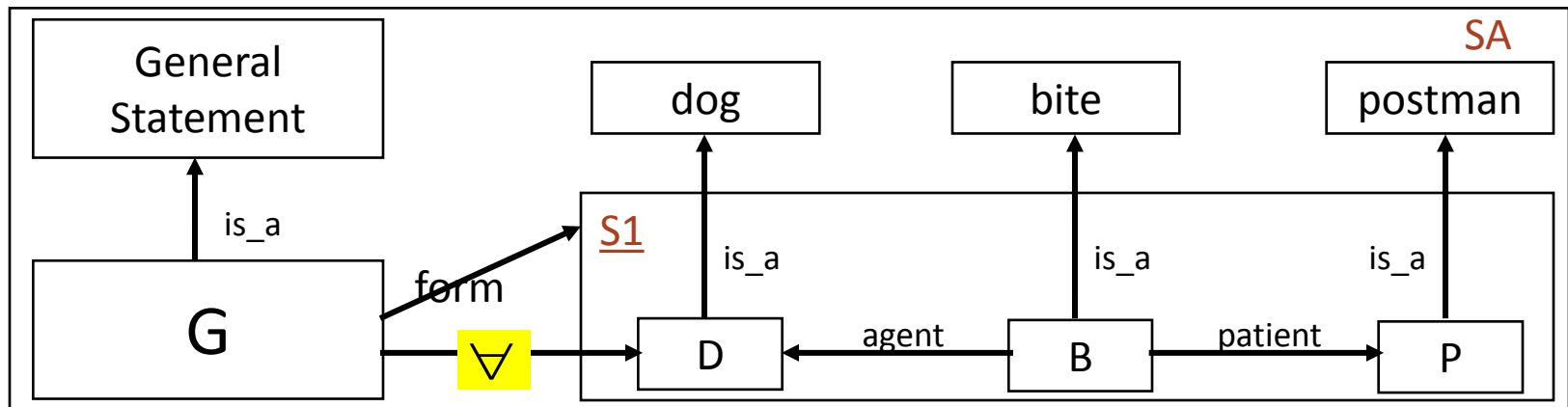
- Suppose that we wish to make a specific statement about a dog, Danny, who has bitten a postman, Peter:
 - " Danny the dog bit Peter the postman"
- Hendrix's Partitioned network would express this statement as an ordinary semantic network:



Partitioned Semantic Networks

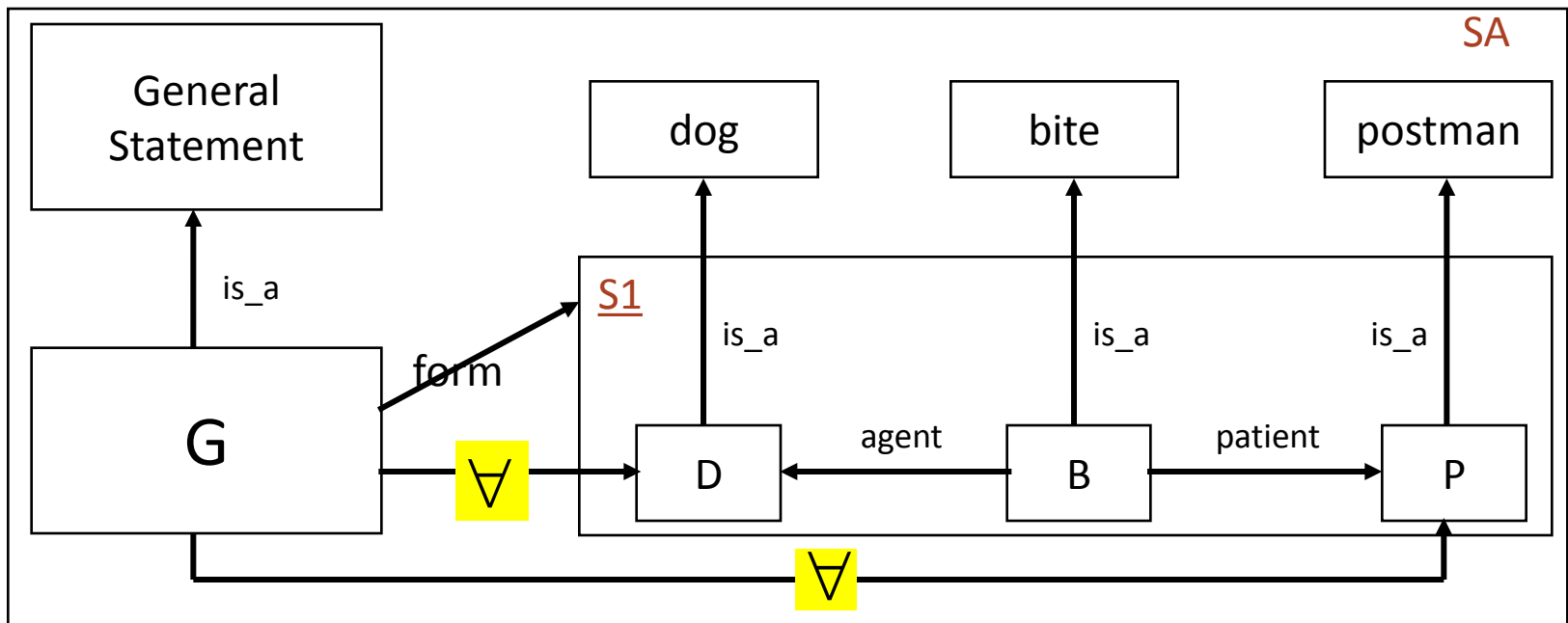
- Suppose that we now want to look at the statement:
 - "Every dog has bitten a postman"
- Hendrix partitioned semantic network now comprises two partitions SA and S1. Node G is an **instance** of the special class of general statements about the world comprising link statement, **form**, and one **universal quantifier**

∇



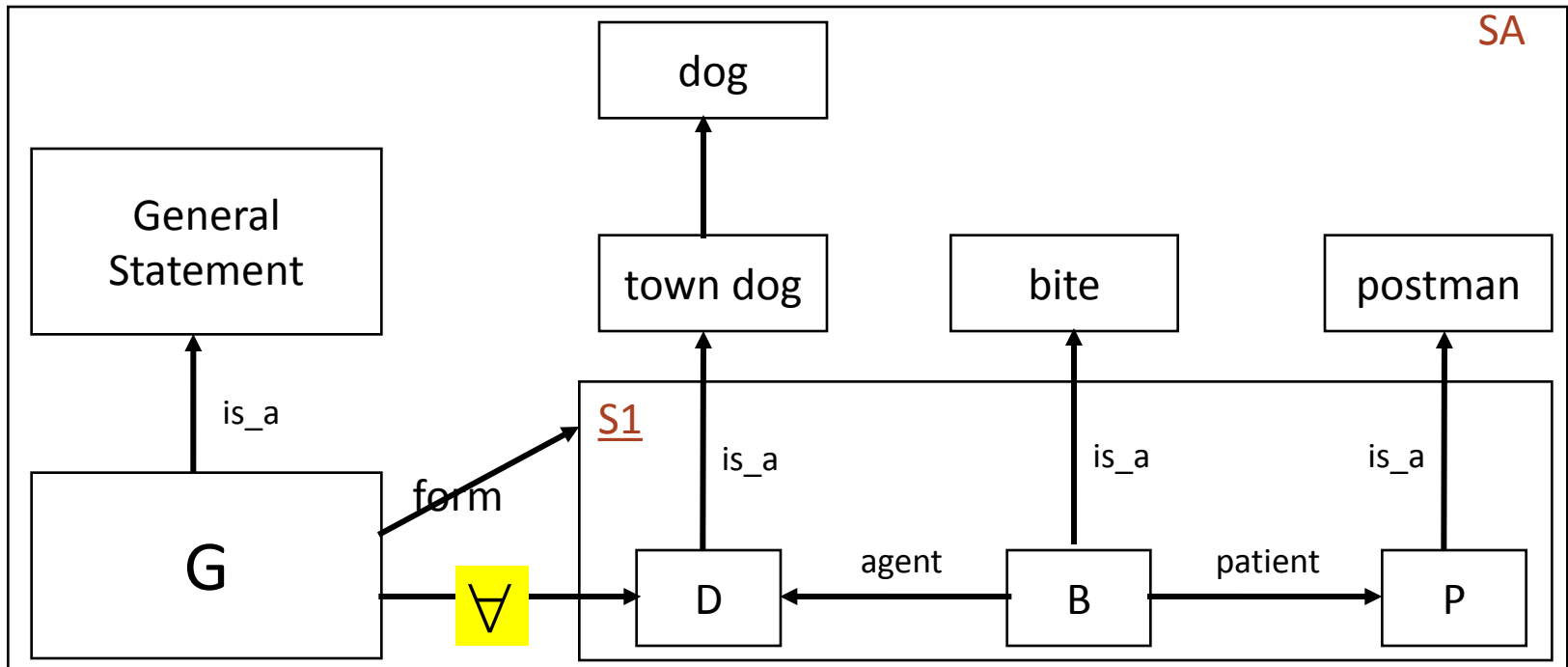
Partitioned Semantic Networks

- Suppose that we now want to look at the statement:
 - "Every dog has bitten every postman"



Partitioned Semantic Networks

- Suppose that we now want to look at the statement:
 - "Every dog in town has bitten the postman"



NB: 'ako' = 'A Kind Of'

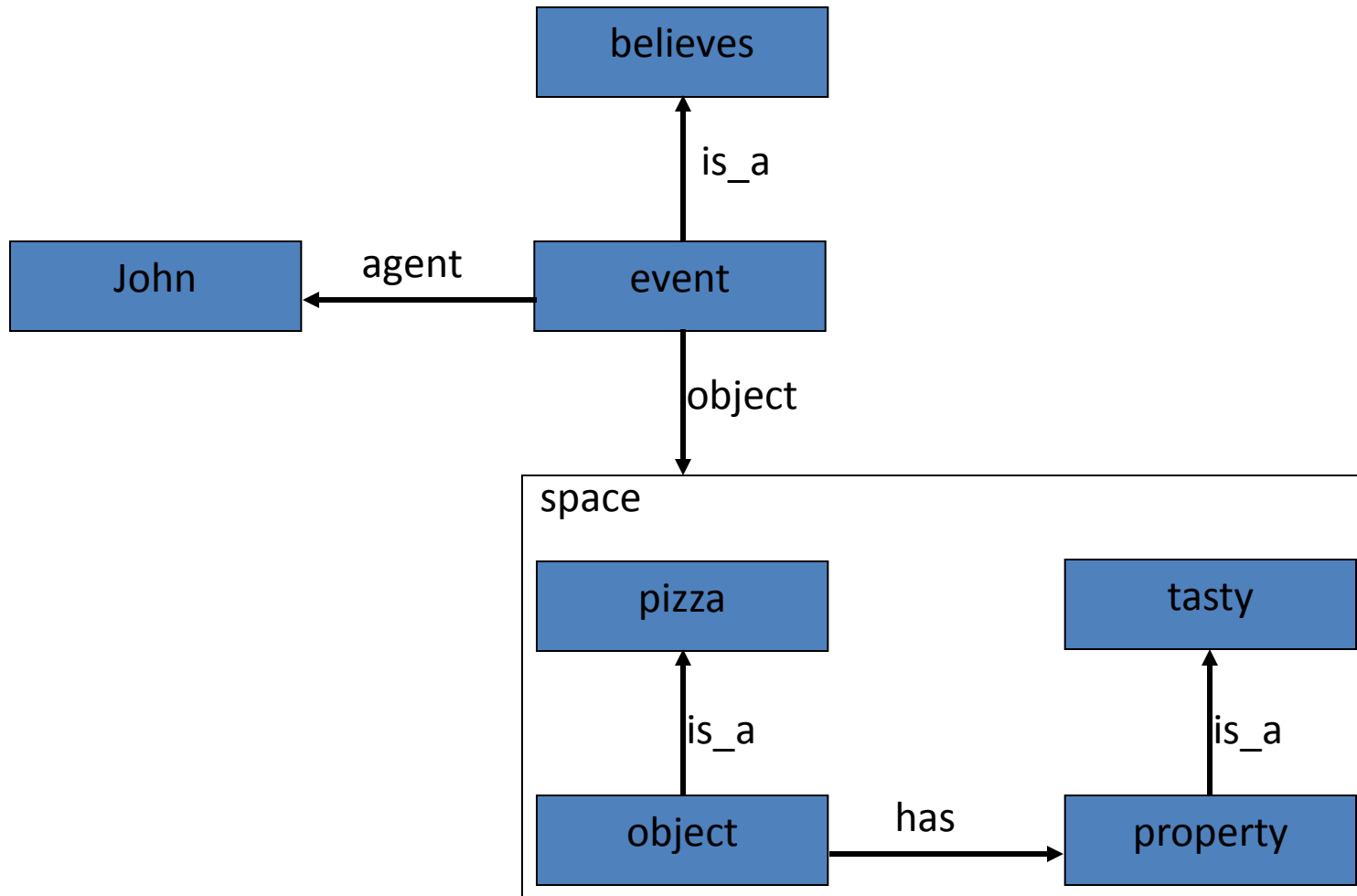
Partitioned Semantic Networks

- The partitioning of a semantic network renders them more
 - **logically adequate**, in that one can distinguish between individuals and sets of individuals,
 - and indirectly more **heuristically adequate** by way of controlling the search space by delineating semantic networks.
- Hendrix's partitioned semantic networks-oriented formalism has been used in building natural language front-ends for data bases and for programs to deduct information from databases.

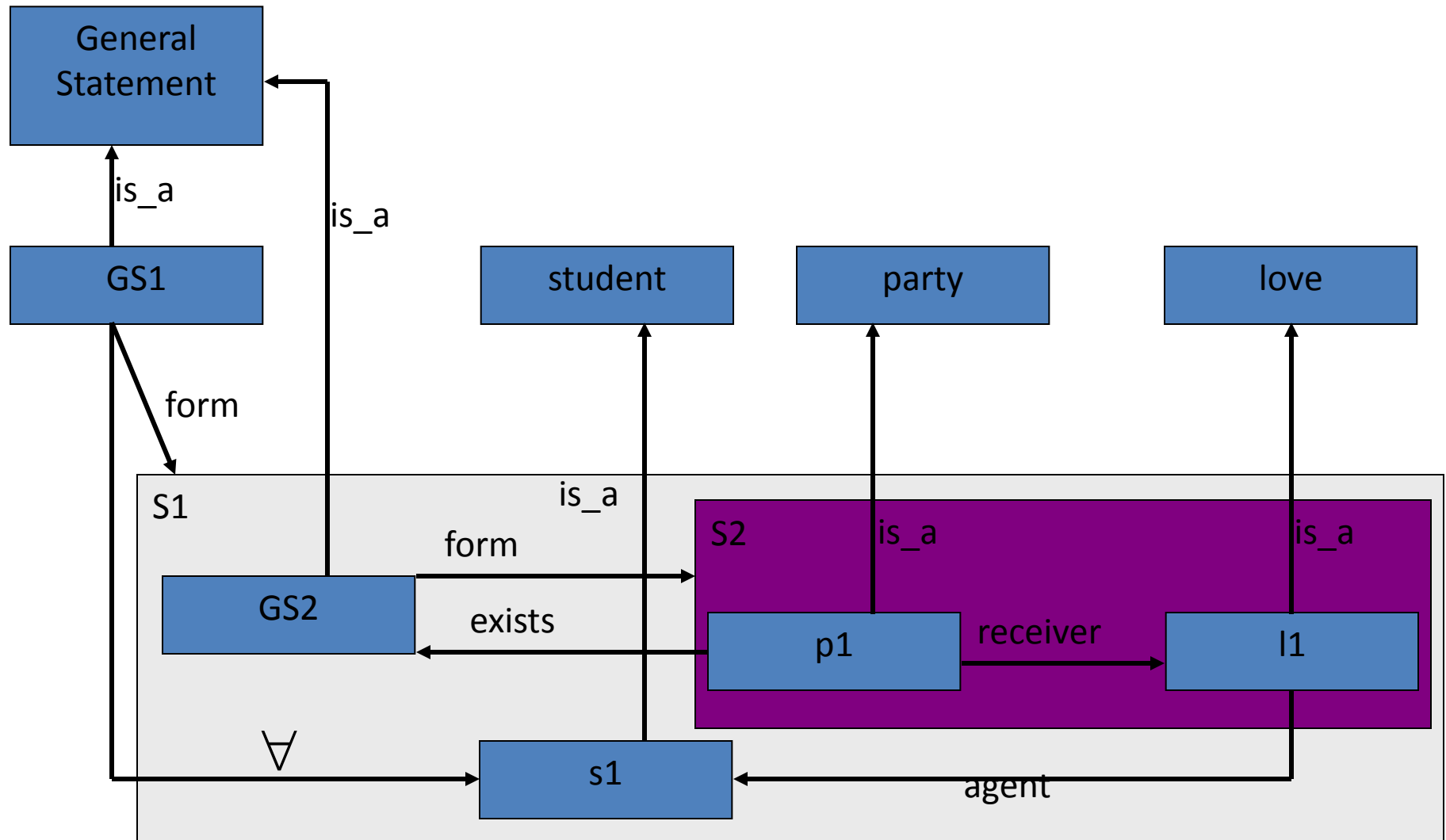
Exercises

- Try to represent the following two sentences into the appropriate semantic network diagram:
 - "John believes that pizza is tasty"
 - "Every student loves to party"
 - John gave Mary the book

Solution 1: "John believes that pizza is tasty"

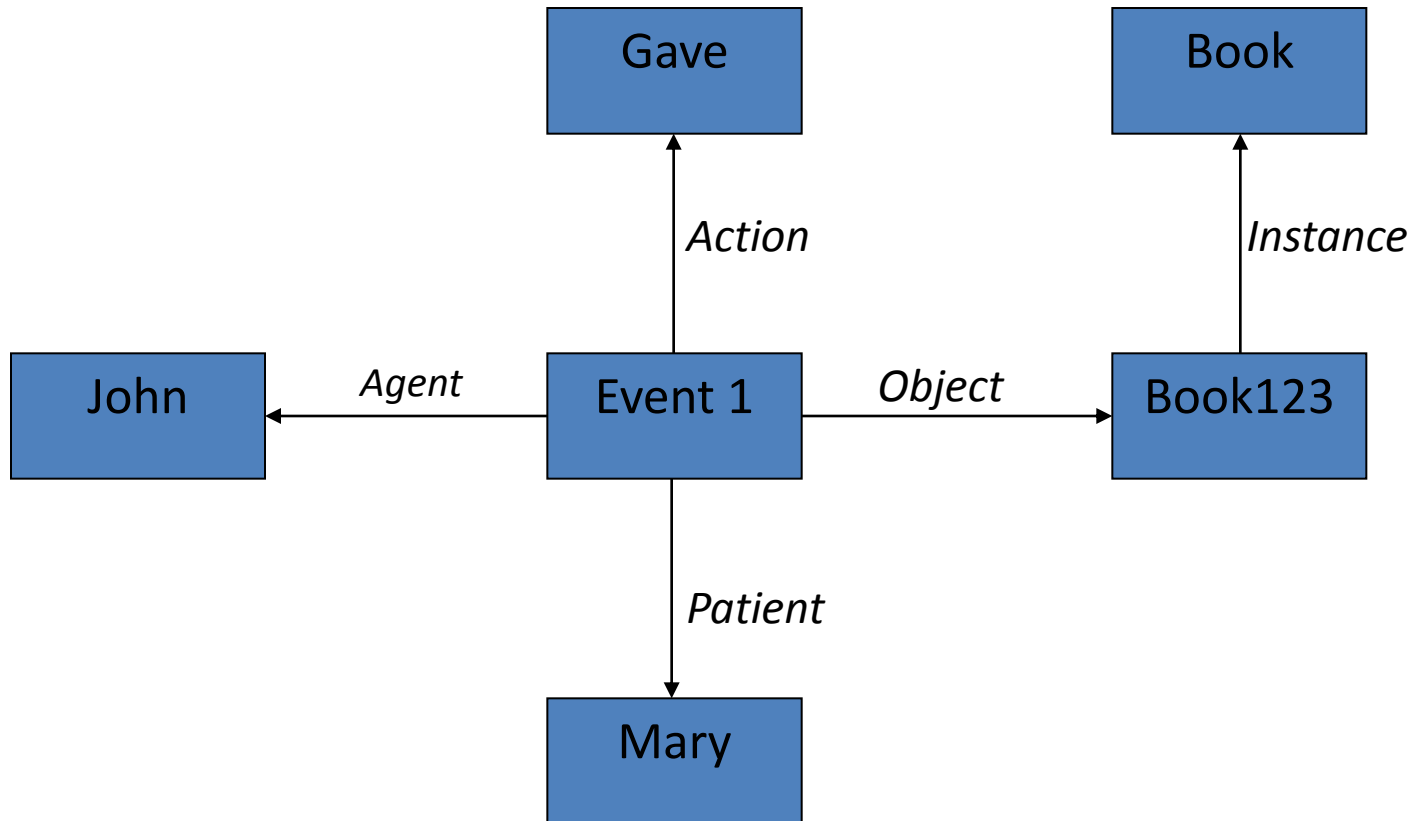


Solution 2: "Every student loves to party"



Solution 3

- John gave Mary the book



Frame

- Marvin Minsky proposed (1975) frames as a means of **common-sense knowledge**.
- Minsky proposed that knowledge is organized into small packets called frames.
- The contents of the frame are certain slots which have values .
- A Frame can be defined as static data structure that has slots for various objects and a collection of frames consists of expectation for a given situation.
- All frames of a given situation constitute the system, whenever one encounters a situation, a series of related frames are activated and reasoning is done.

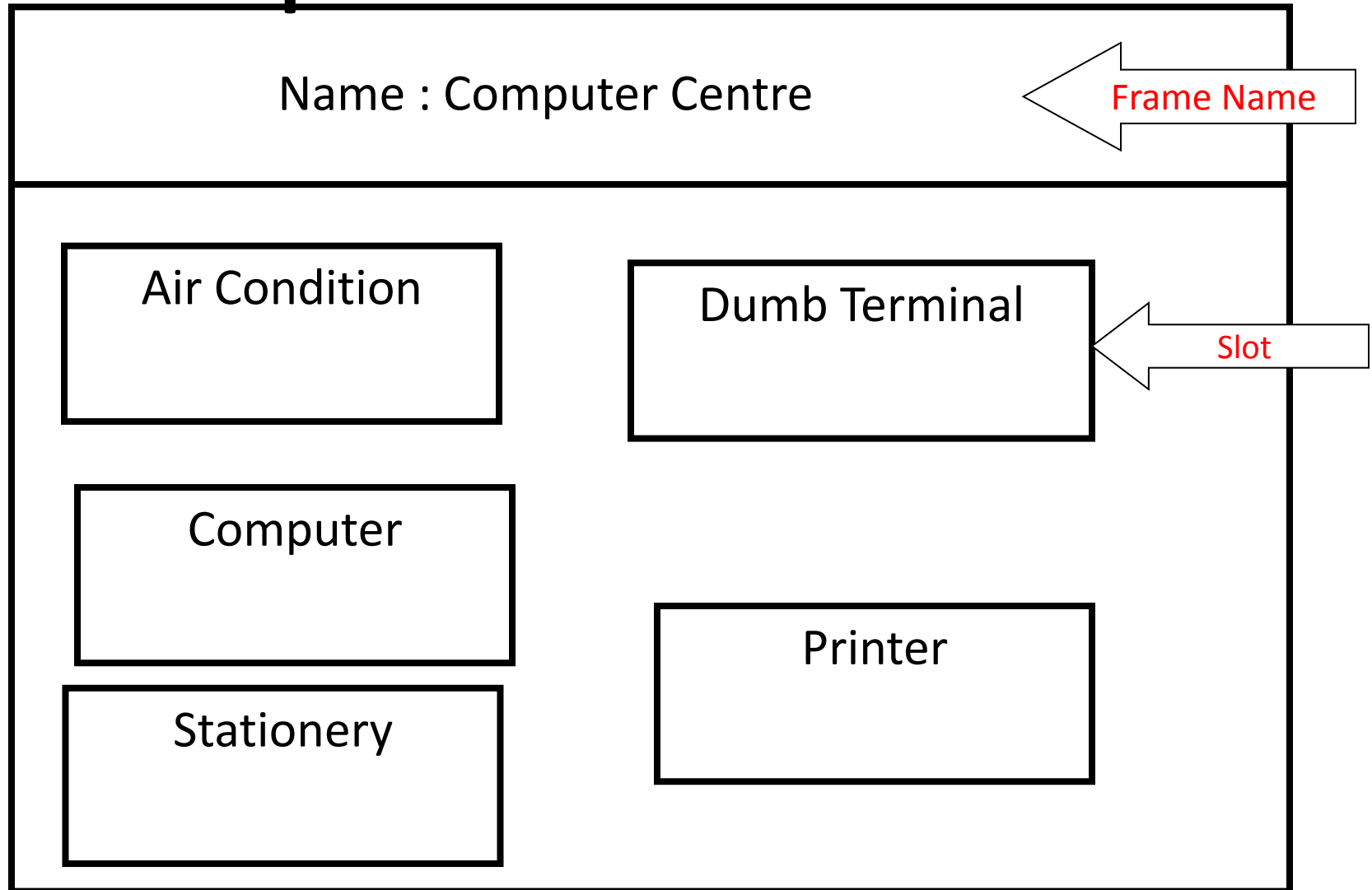
Type of Frame

- Frame is used to represent two type of knowledge
 - Declarative/factual/situational/ frame
 - Procedural/action frame

Declarative Frame

- A frame that contains only descriptive type of knowledge called declarative/factual Frame type .

Example of Declarative Frame



Procedural Frame

- Apart from the declarative part in a frame, it is possible to attached slots which explain how to perform things. Or we say that, it is possible to have procedural knowledge representation in a frame.

The action-frame (Procedural knowledge embedded) has the following slots-

- **Actor Slot-** Which holds information about who is performing the activity.
- **Object Slot-** This frame has information about the item to be operated on.
- **Source Slot-** Source slot holds information from where the action has to begin
- **Destination Slot-** Holds information about the place where the action has to end.
- **Task Slot-** This generate the necessary sub-frames required to perform the operation.

Name : Cleaning the jet of carburetor

Actor

Expert

Object

Carburetor

Source

Scooter

Destination

Scooter

Task1

Remove Carburetor

Task2

Clean Nozzle

Task3

Fix Carburetor


Slots can contain these information

- Frame identification information (name)

 **Example:**


A frame which stores knowledge about cars can have a name "Car"

- Relationship of this frame to other frames

 **Example:**


A super-class of a frame "Car" is a frame "Vehicle"

- Knowledge about an attribute of an object and its value

 **Example:**

A frame "Car" can have an attribute "Number of wheels" with value 4

- Frame default information (These are slot values that are taken to be true when no evidence to the contrary has been found)

 **Example:**

A frame "Car" can have a slot "Number of doors" with value 4, however, there are cars with only 2 doors

Frames: structure

In general a frame has the following structure:

Name of a frame	
Names of slots	Values of slots



Example:

Car	
Engine	
Number of doors	4

Frames: types

- Slot values can point out another frame.
- By relating frames through slot values a frame system can be acquired.


Three types of frames can be found in a frame system:

1. **Class frame:** Such a frame includes slots describing an attributes of a class of objects. Typically slots of such frames have default information or unspecified values.

Class: Vehicle	
Reg.No.	
Model	
Producer	
Owner	


Frames: types

2. A sub-class frame

 **Example:**

Car	
Class: Vehicle	
Reg.No.	
Model	
Producer	
Owner	
Number of doors	4
Engine	

3. An instance frame

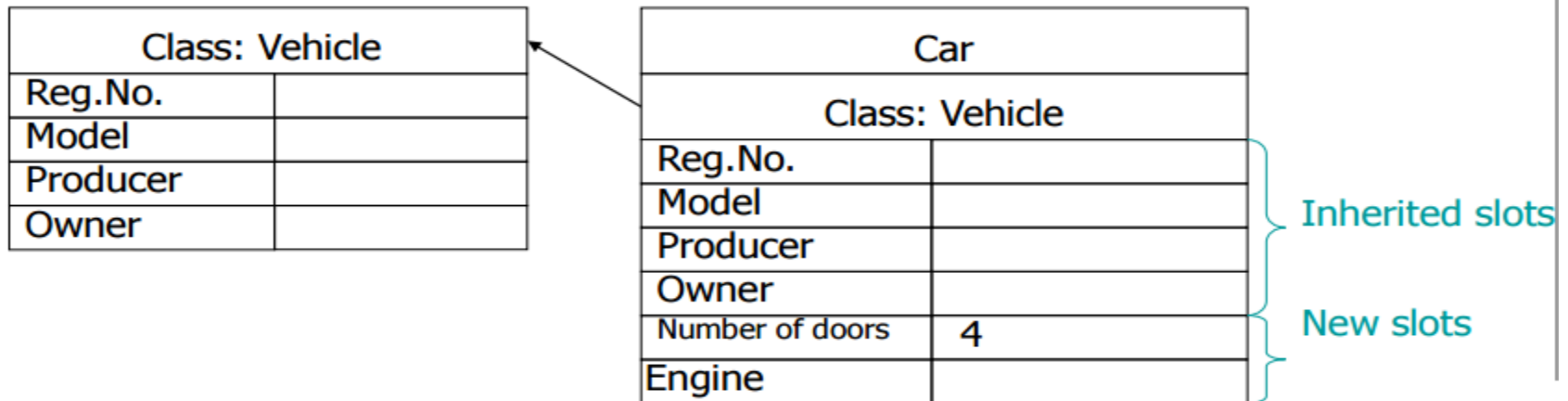
 **Example:**

John's car	
Class: Car	
Reg.No.	LA657
Model	850
Producer	BMW
Owner	John
Number of doors	2
Engine	5.0

Frames: relationships

- Three types of relationships can relate frames in a frame system:
 - “Is-a” relationship. Relates a sub-class frame with a class frame or an instance frame with a sub-class or class frame. In this case a sub-class frame or an instance frame inherits all slots from a class frame, but it can include also a new slots.

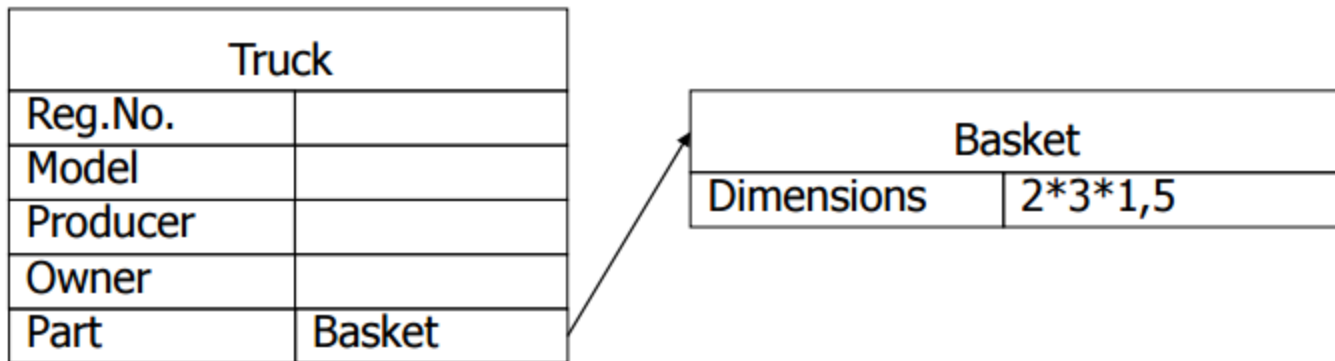
✓ **Example:**



Frames: relationships

2. "a-part-of" relationship. Relates whole with its constituent parts.

✓ **Example:**




Frames: relationships

3. Semantic relationship

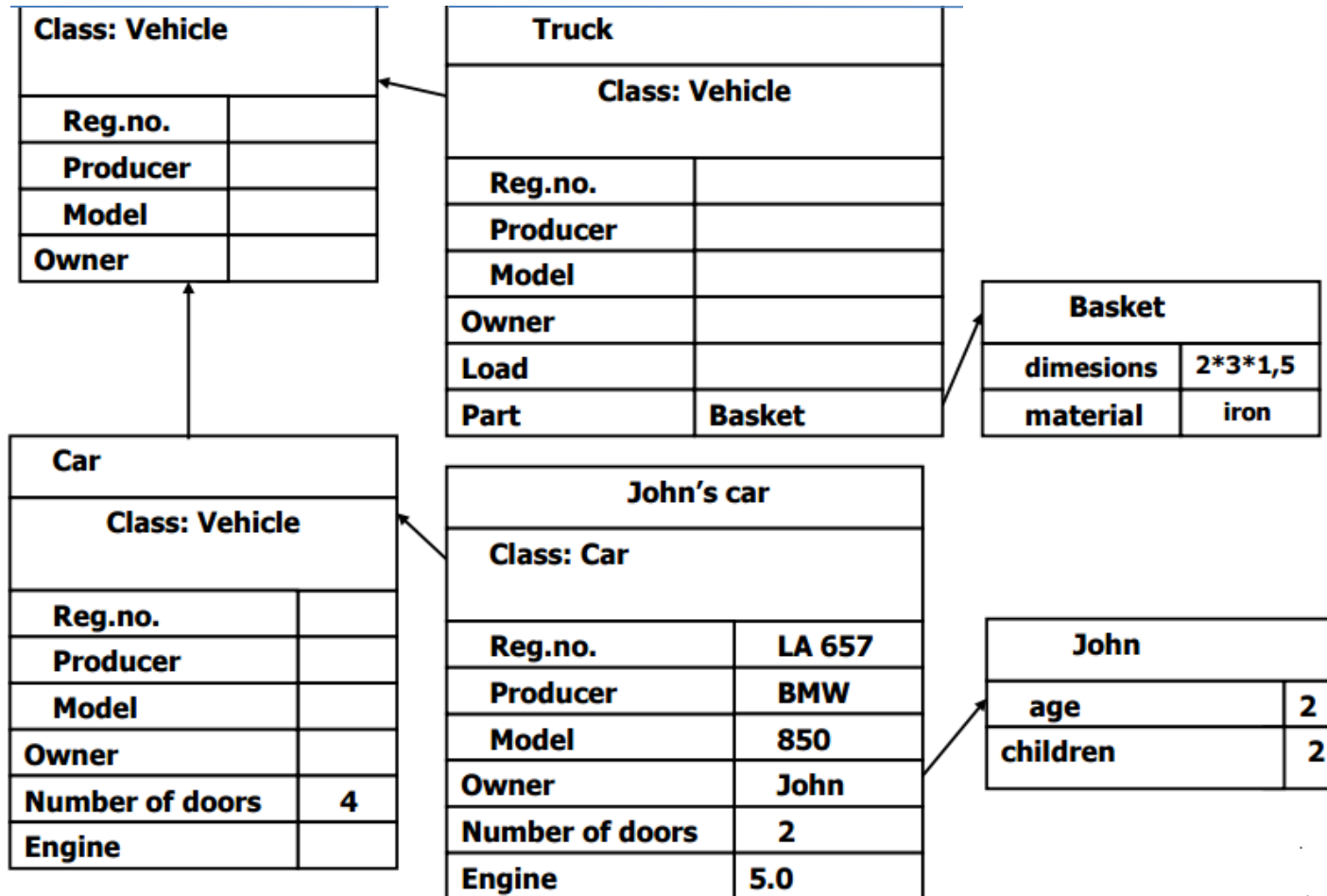
✓ **Example:**

John's car	
Class: Car	
Reg.No.	LA657
Producer	BMW
Model	850
Owner	John
Number of doors	2
Engine	5.0



John	
Age	22

Frames: an example



Frames: facets

- Frames can incorporate facets, which represent extended knowledge about slot values
- Facets can include:
 - Type of a value
 - Default value
 - Constraints on a value
 - Minimum and maximum values
 - Actions on values
- They allow controlling of slot values.

Name of a frame		
Slots	Slot values	Facets

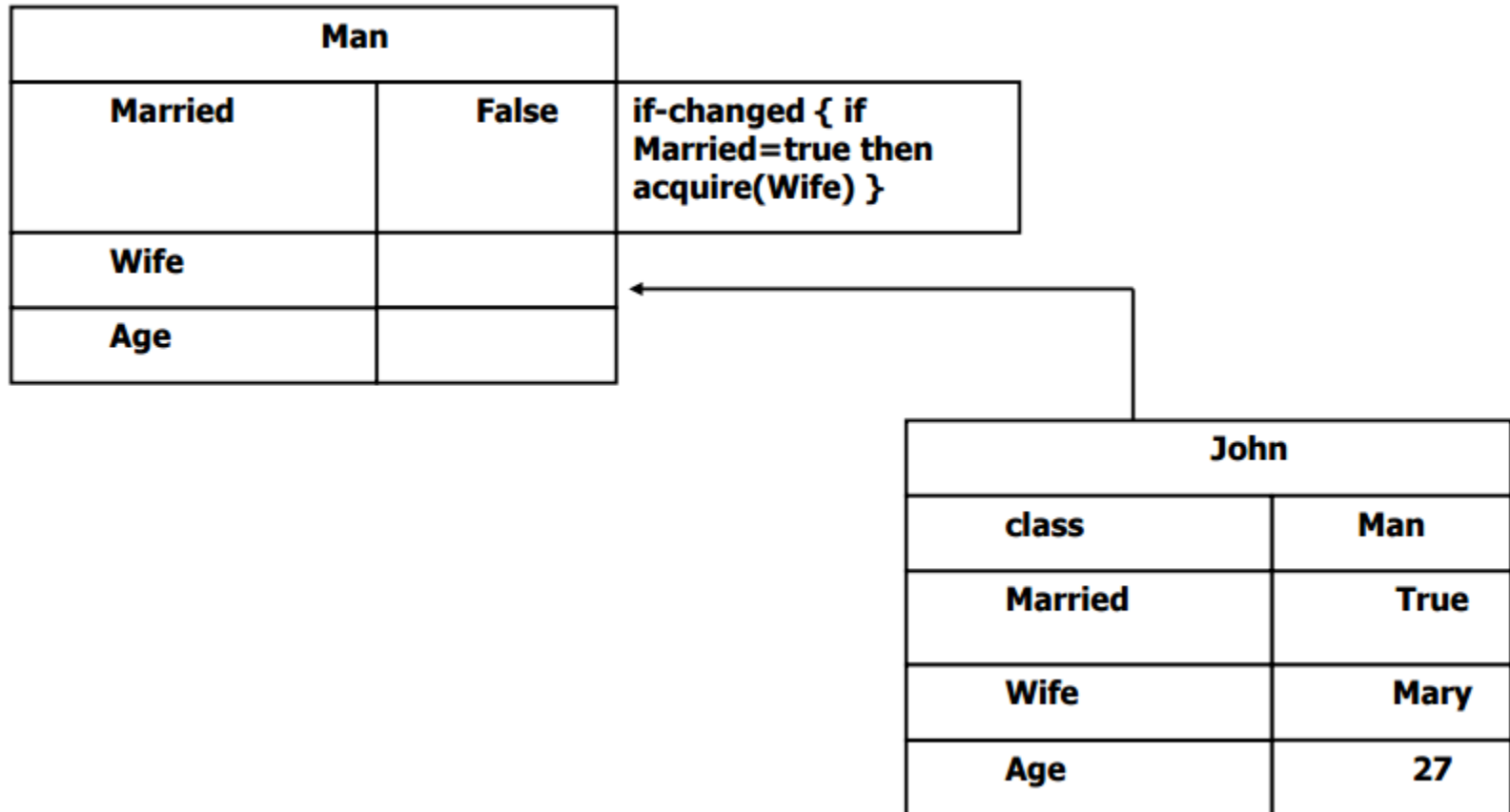
Frames: methods

- Methods called demons can be attached to slots.
- Demons are invoked automatically when a slot is accessed.

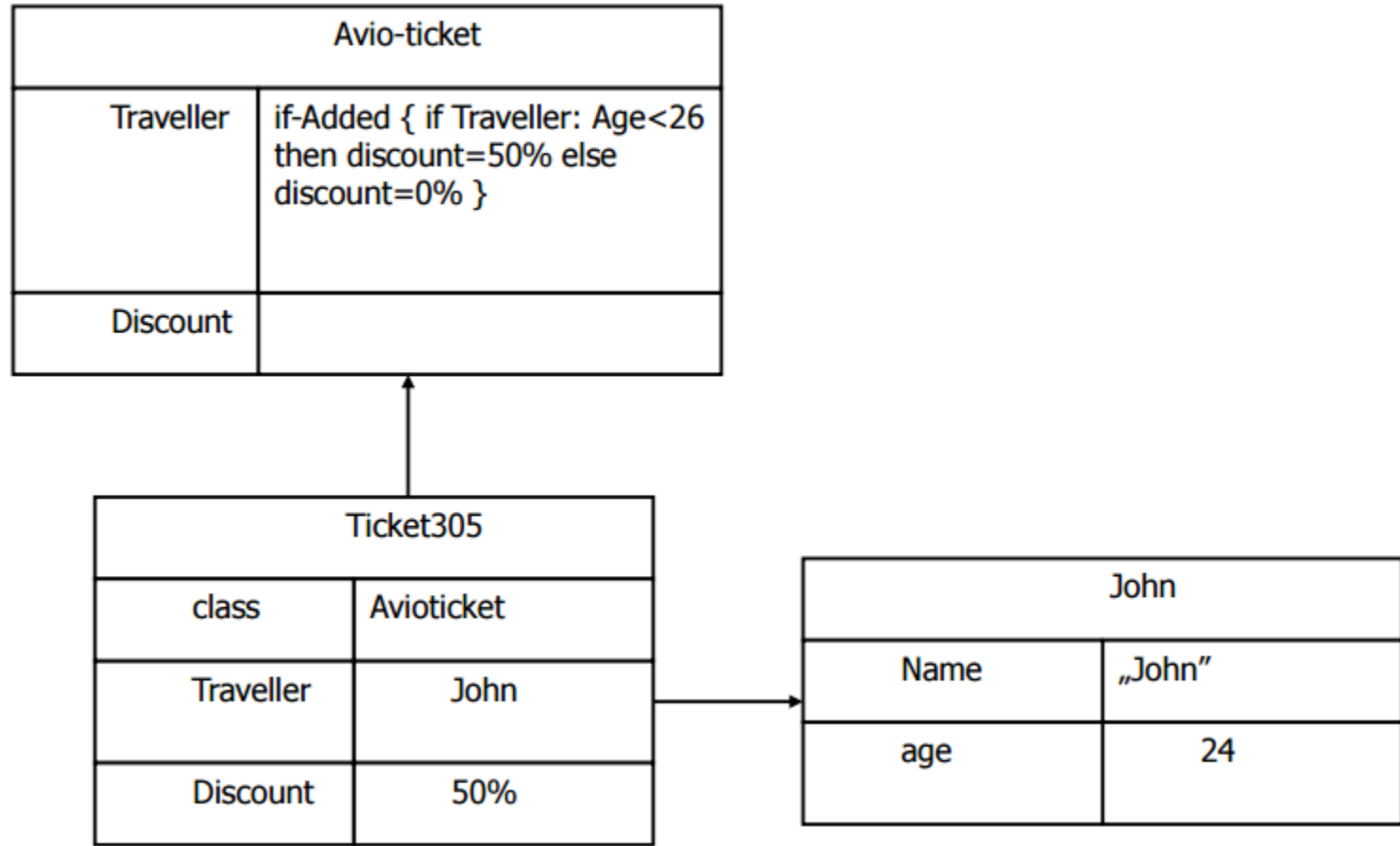
Standard demons are the following:

- IF-NEEDED is invoked when it is necessary to acquire a slot value
- IF-CHANGED is invoked when a value of a slot is changed
- IF-ADDED is invoked when a value is added to a slot
- IF-REMOVED is invoked when a value of a slot is deleted

Frames: an example of demons



Frames: an example of demons



Frame Structure

- (<frame name>
 (<slot1> (<facet1> <value1>....<valuek1>)
 » (<facet2> <value1>....<valuek2>)
 .
 .
 .
 (<slot2>(<facet1> <value1>....<valuek1>)
 .
 .
 .)

Simple Frame Example

- (bob
 (PROFESSION (VALUE professor))
 (AGE (VALUE 42))
 (WIFE (VALUE sandy))
 (CHILDREN (VALUE sue joe))
 (ADDRESS (STREET (VALUE wolfstr.))
 (CITY (VALUE BONN))
 (STATE (VALUE nrw))
 (ZIP (VALUE 52100))))

Frame systems

- Frame interpreter
 - Each frame system needs an inference mechanism
 - Takes care of inheritance, the invoking of demons and the message passing
- Advantages of frame systems
 - The knowledge can be structured
 - Flexible inference by using procedural knowledge
 - Layered representation and inheritance is possible
- Disadvantages of frame systems
 - The design of the interpreter is not easy
 - The validity of the inferences is not guaranteed
 - Hard to maintain consistency between the knowledge

Conceptual dependency

- It is a theory of how to represent the kind of knowledge about events that is usually contained in natural language sentences.
- Extension to semantic networks to define a complete set of primitives to use as relations in semantic networks
- The goal is to represent the knowledge in a way that
 - Facilitates drawing inferences from the sentences.
 - Is independent of the language in which the sentences were originally stated.
- The theory was first described by Roger Schank in 1970s.
- Four primitive conceptual categories
 - ACT action
 - PP object
 - AA modifiers of actions
 - PA modifiers of objects

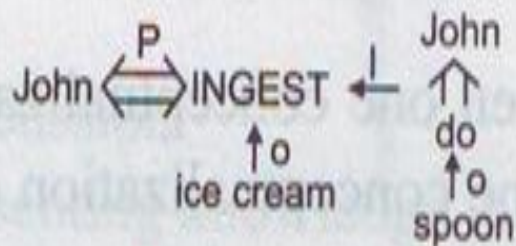
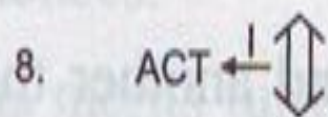
Primitive ACTs

- Primitive ACTs represent basic actions
- All actions can be reduced to one or more primitive ACT (with modifiers)
- 12 primitive ACTs
 1. **ATRANS** transfer a relationship *give*
 2. **PTRANS** transfer a physical location of an object *go*
 3. **PROPEL** apply physical force to an object *push*
 4. **MOVE** move body part by owner *kick*
 5. **GRASP** grab an object by an actor *grasp*
 6. **INGEST** ingest an object by an animal *eat*
 7. **EXPEL** expel from an animal's body *cry*
 8. **MTRANS** transfer mental information *tell*
 9. **MBUILD** mentally make new information *decide*
 10. **CONC** conceptualize or think about an idea *think*
 11. **SPEAK** produce sound *say*
 12. **ATTEND** focus sense organ *listen*

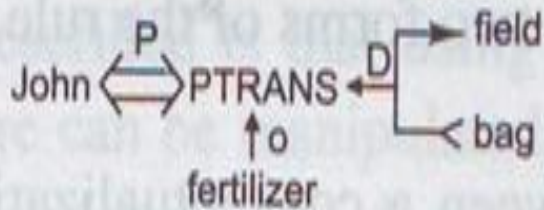
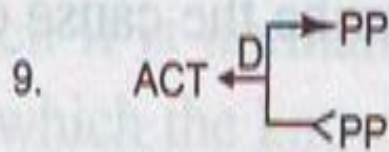
1. $PP \rightleftarrows ACT$ John $\overset{p}{\rightleftarrows} PTRANS$ John ran.
2. $PP \rightleftarrows PA$ John \rightleftarrows height (> average) John is tall.
3. $PP \rightleftarrows PA$ John \rightleftarrows doctor John is a doctor.
4. $PP \uparrow PA$ boy \uparrow nice A nice boy.
5. $PP \uparrow \uparrow PP$ dog $\uparrow \uparrow$ John Poss-by John's dog.
6. $ACT \xleftarrow{o} PP$ John $\overset{p}{\rightleftarrows} PROPEL \xleftarrow{o} cart$ John pushed the cart.
7. $ACT \xleftarrow{o} \begin{cases} \rightarrow PP \\ \leftarrow PP \end{cases}$ John $\overset{p}{\rightleftarrows} ATRANS \xleftarrow{o} \begin{cases} \rightarrow John \\ \leftarrow Mary \end{cases}$

$\uparrow o$
book

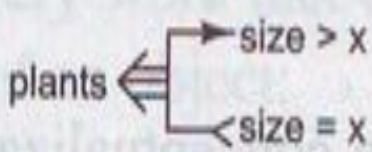
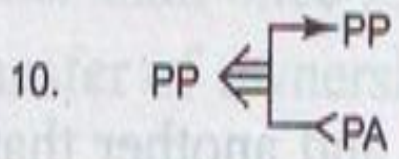
 John took the book from Mary.



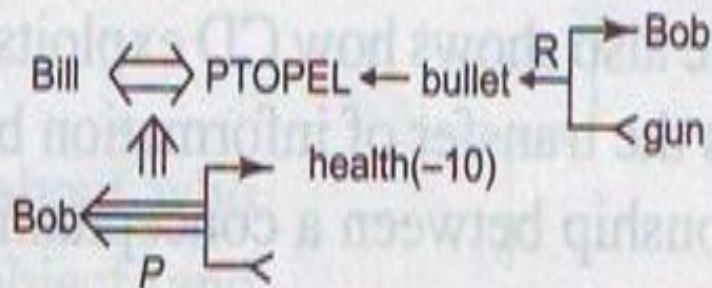
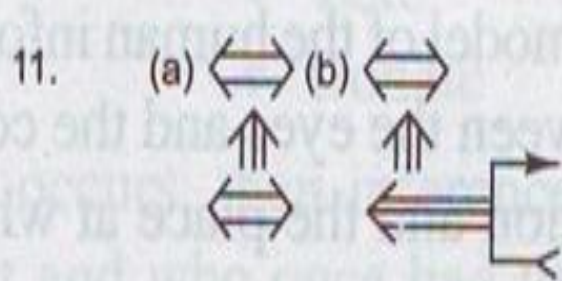
John ate ice cream with a spoon.



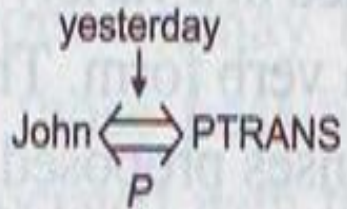
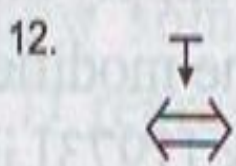
John fertilized the field.



The plants grew.

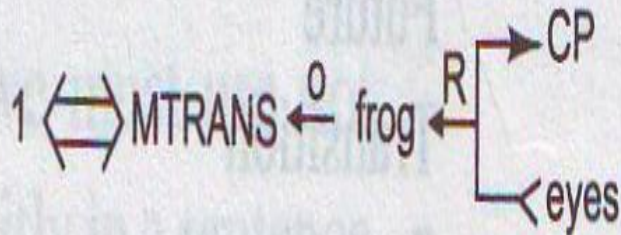
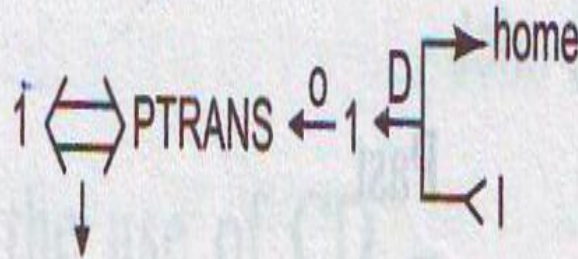


Bill shot Bob.



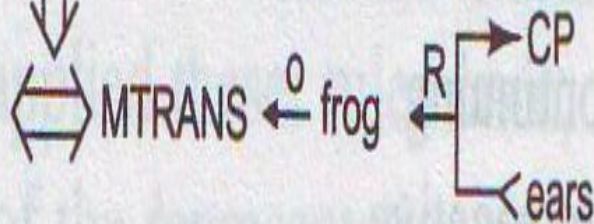
John ran yesterday.

13.



While going home, I saw a frog.

14.



I heard a frog in the woods.

Explanation

- **Rule 1:** describes the relationship between an actor and the event he or she causes. The letter p above the dependency indicates past tense.
- **Rule 2:** describes the relationship between a PP and a PA.
- **Rule 3:** describes the relationship between two PPs, one of which belongs to the set defined by the other.
- **Rule 4:** describes the relationship between a PP and an attribute that has already been predicated of it.
- **Rule 5:** describes the relationship between two PPs, one of which provides a particular kind of information about the other.
- **Rule 6:** describes the relationship between an ACT and PP that is the object of that ACT.
- **Rule 7:** describes the relationship between an ACT and the source and , the recipient of the ACT.

Explanation

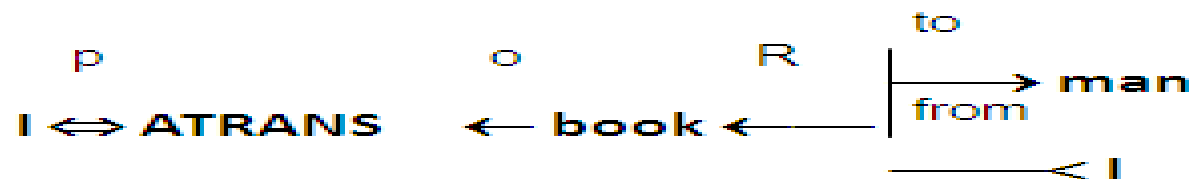
- **Rule 8:** describes the relationship between an ACT and the instrument with which it is performed.
- **Rule 9:** describes the relationship between an ACT and its physical source and destination.
- **Rule 10:** represents the relationship between a PP and a state in which it started and another in which it ended.
- **Rule 11:** describes the relationship between one conceptualization and another that causes it.
- **Rule 12:** describes the relationship between a conceptualization and the time at which the event it describes occurred.
- **Rule 13:** describes the relationship between one conceptualization and another that is the time of the first.
- **Rule 14:** describes the relationship between a conceptualization and the place at which it occurred.

Primitive conceptual tenses

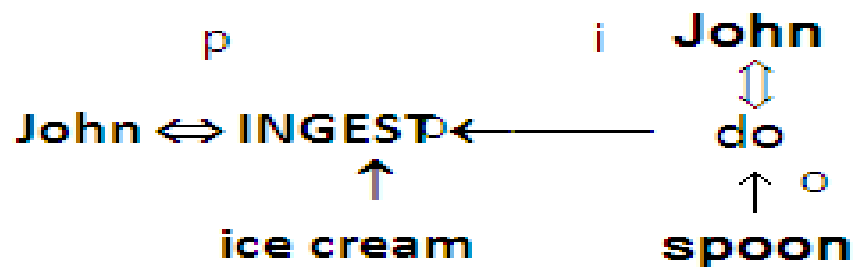
1. p	Past
2. f	Future
3. t	Transition
4. k	Continuing
5. t_s	Start transition
6. t_k	Finish transition
7. ?	Interrogative
8. /	Negative
9. Nil	Present
10. Delta	Timeless
11. C	Condition

Example

“I gave the man a book”

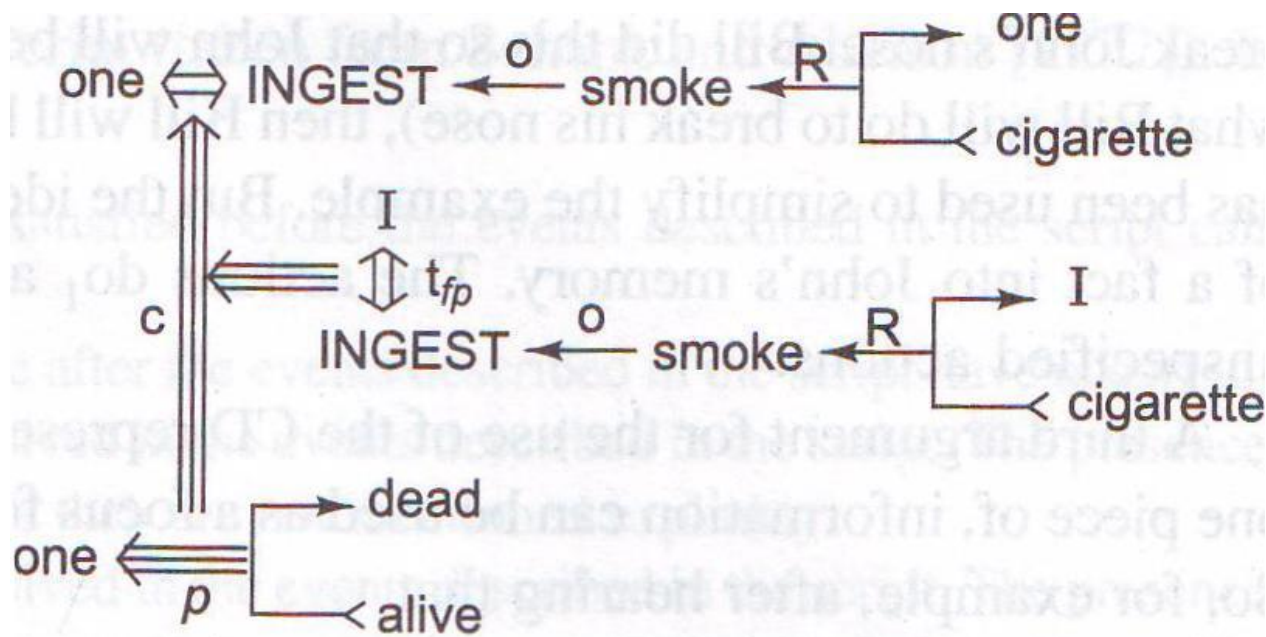


“John ate ice scream with a spoon”



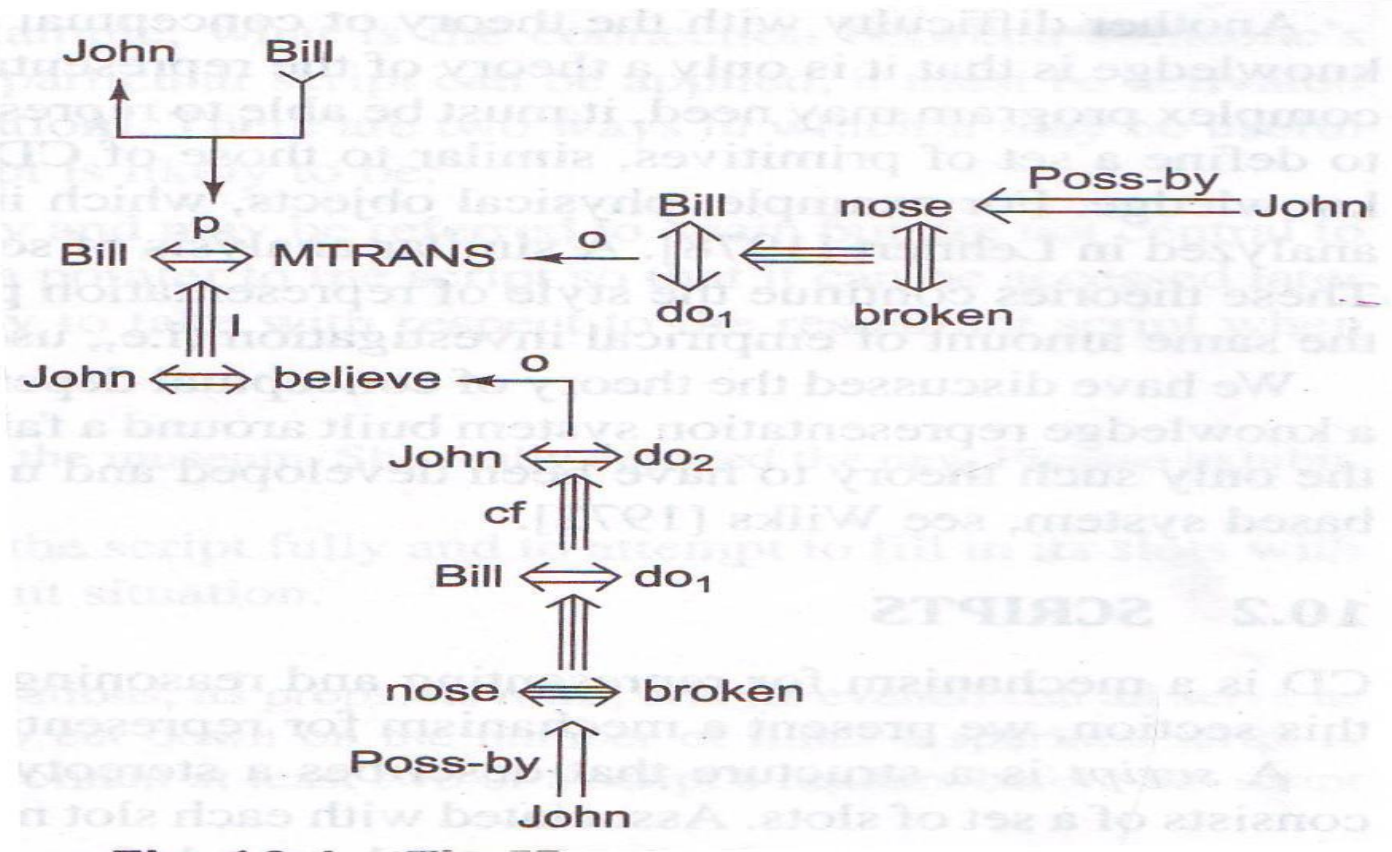
Example

“Since, smoking can kill one, I stooped”.



Example

Bill threatened John with a broken nose.



Conceptual Dependency

Advantages:

- Fewer inference rules are needed.
- Many inferences are already contained in the representation.
- Holes in the representation can serve as an attention focuser.

Disadvantages:

- ✓ Requires knowledge to be decomposed into fairly low-level primitives is only a theory of the representation of events.
- ✓ can't produce them automatically from NL
 - needs to be built by hand

Solve

1. Write a conceptual dependency structure for the following:
 - John begged Marry for a pencil. (rule 7)
 - While going I saw snake. (rule 13)
2. Formulate CD structure for the following sentence:”
“Since, smoking can kill one, I stooped”. (slide 152)
3. Express the following sentence as CD structure
 - Sam gave Mary a box of candy. (rule 7)
 - Bill and Ram is a programmer. (rule 3)
 - Charlie drive a car but not bike.

Scripts

- Developed by Roger Schank, late 1970s
- We need large amounts of background knowledge to understand even the simplest conversation
 - “Sue went out to lunch. She sat at a table and called a waitress, who brought her a menu. She ordered a sandwich.”
 - questions:
 - why did the waitress bring a menu to Sue?
 - who was the “she” who ordered a sandwich?
 - who paid?
- **Claim:** people organize background knowledge into structures that correspond to typical situations (scripts)
- **Script:** A typical scenario of what happens in...
 - a restaurant
 - a soccer game
 - a classroom

Scripts

- A script is a knowledge representation structure that is extensively used for describing stereo type sequences of action.
- It is special case of frame structure.
- It represent events that takes place in day – to – day activities.
- Script do have slots and with each slots, we associate info about the slot.

Components of scripts

1. Entry conditions

- Preconditions:
 - facts that must be true to call the script
- Eg.: an open restaurant, a hungry customer that has some money

2. Results

- Postconditions:
 - facts that will be true after the script has terminated
- Eg.: customer is full and has less money; restaurant owner has more money

Components of scripts

3. Props

- Typical things that support the content of the script
- Eg.: waiters, tables, menus

4. Roles

- Actions that participants perform
- Represented using conceptual dependency
- Eg.: waiter takes orders, delivers food, presents bill

5. Scenes

- A temporal aspect of the script
- Eg.: entering the restaurant, ordering, eating, ...

6. Track

- represents a specific instance of a generic pattern.
- Restaurant is a specific instance of a hotel. This slot permits one to inherit the characteristics of the generic node.

Food Market Example

- i. SCRIPT-NAME : food market
- ii. TRACK : supermarket
- iii. PROPS : shopping cart
 market items
 checkout statnds
 cashier
 money
- iv. ROLES : shopper
 daily attendant
 food attendant
 checkout clerk
 other shoppers
- v. ENTRY
 CONDITION : shopper needs food market open
- vi. RESULTS:

Food Market Example...

- Scene1 : Enter Market
Shopper PTRANS Shopper into market
Shopper PTRANS Shopping –cart to shopper
- Scene2 : Shop for Items
Shopper MOVE shopper through aisles
Shopper ATTEND eyes to display items
Shopper PTRANS items to shopping cart
- Scene3 :Check out
Shopper MOVE shopper to checkout stand
Shopper WAIT shopper turn
Shopper ATTEND eyes to charges
Shopper ATRANS money to cashier
Sacker ATRANS bags to shopper
- Scene4 : Exit Market
Shopper PTRANS shopper to exit market

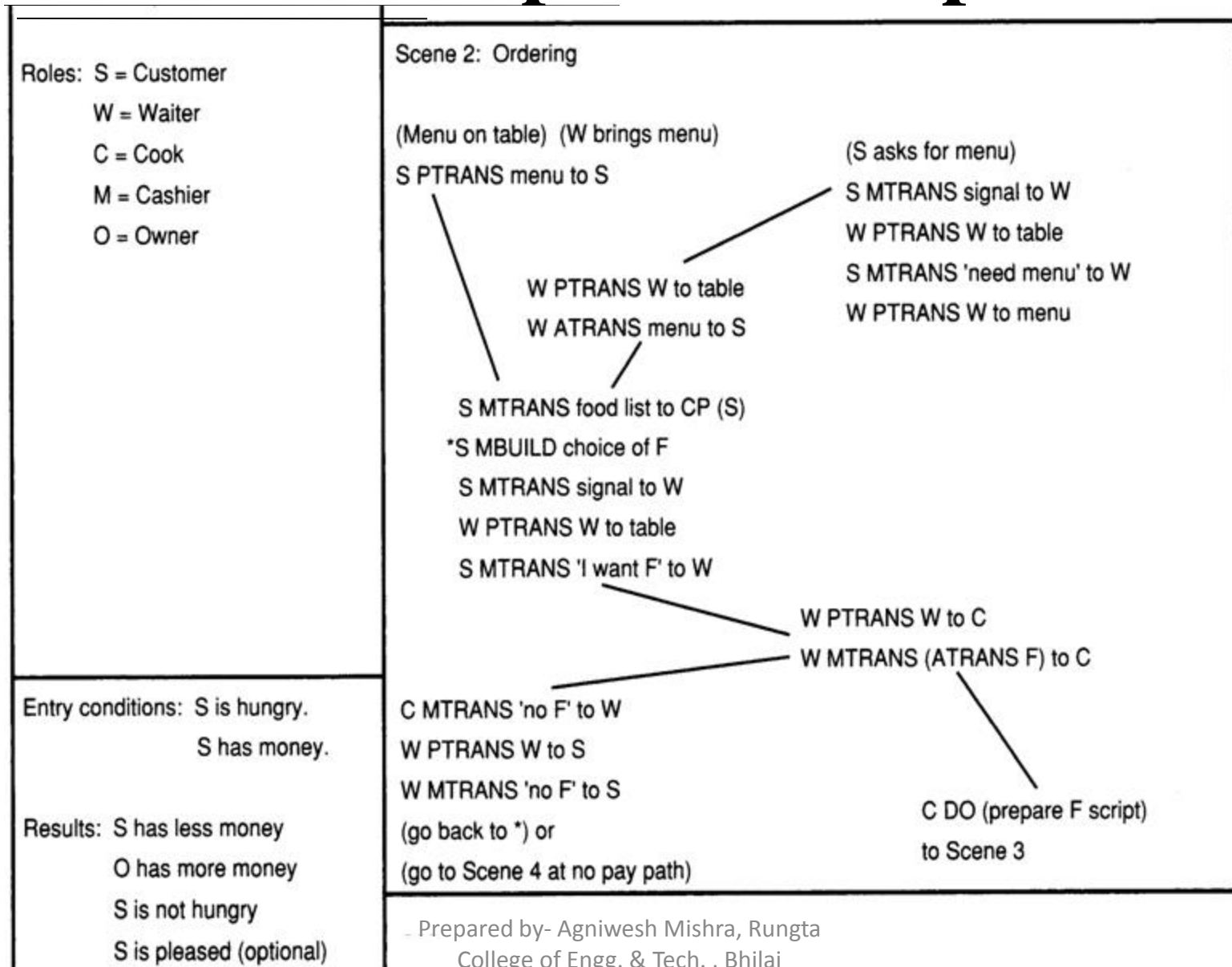
Food Market Example...

- Results
 - : Shopper has less money
 - Shopper has grocery items
 - Market has less grocery items
 - Market has more money

Example of a script

Script: RESTAURANT	Scene 1: Entering
Track: Coffee Shop	
Props: Tables	S PTRANS S into restaurant
Menu	S ATTEND eyes to tables
F = Food	S MBUILD where to sit
Check	S PTRANS S to table
Money	S MOVE S to sitting position

Example of a script



Example of a script

Scene 3: Eating

C ATRANS F to W

W ATRANS F to S

S INGEST F

(Option: Return to Scene 2 to order more;
otherwise, go to Scene 4)

Scene 4: Exiting

S MTRANS to W

(W ATRANS check to S)

W MOVE (write check)

W PTRANS W to S

W ATRANS check to S

S ATRANS tip to W

S PTRANS S to M

S ATRANS money to M

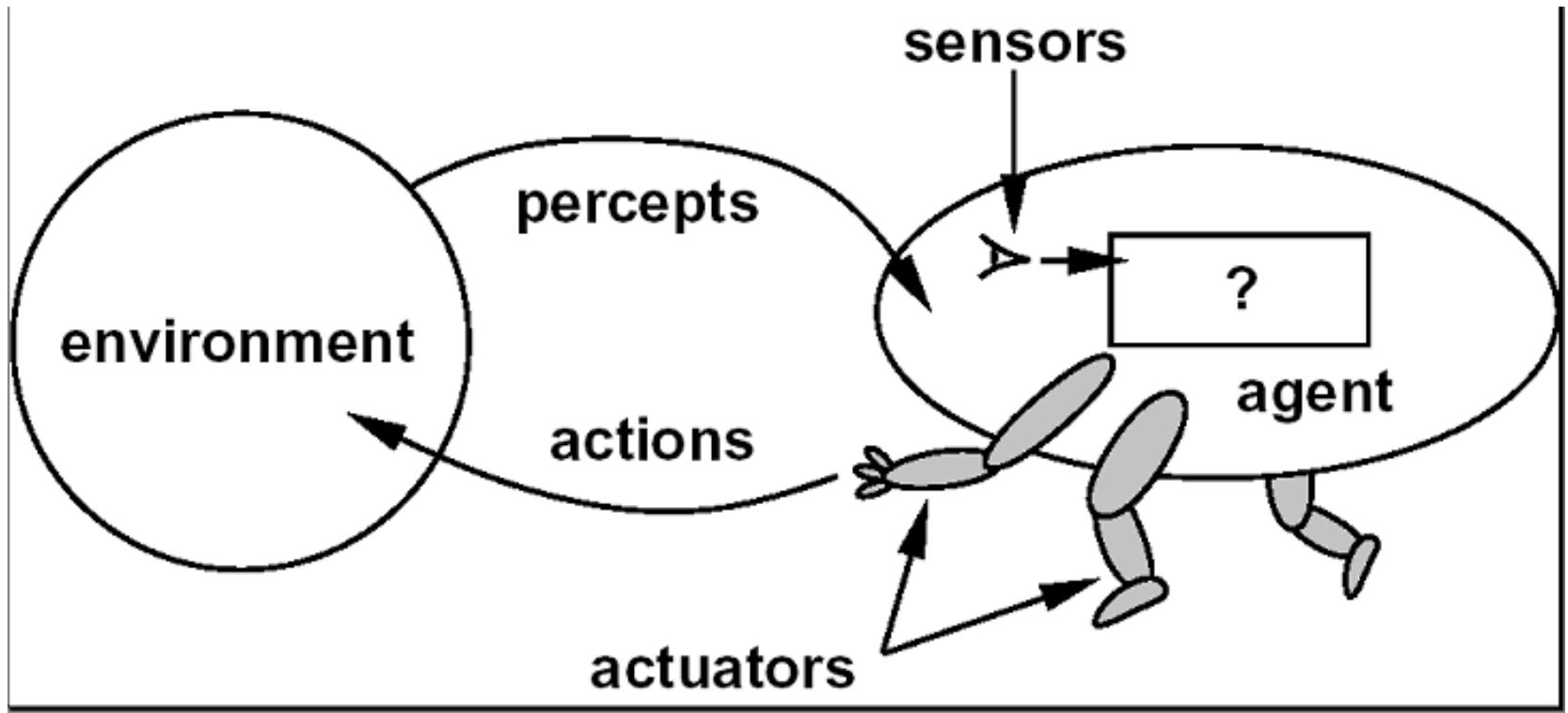
S PTRANS S to out of restaurant

(No pay path)

Agents

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
- **A human agent** has eyes, ears and other organs for sensors and hands, legs, mouth and other body parts for actuators.
- **A robotics agent** might have cameras and infrared range finders for sensors and various motors for actuators.
- **A software agent** receives keystrokes, file contents , and network packets as sensory inputs and acts on the environment by displaying on the screen , writing files, and sending network packets.

Agents



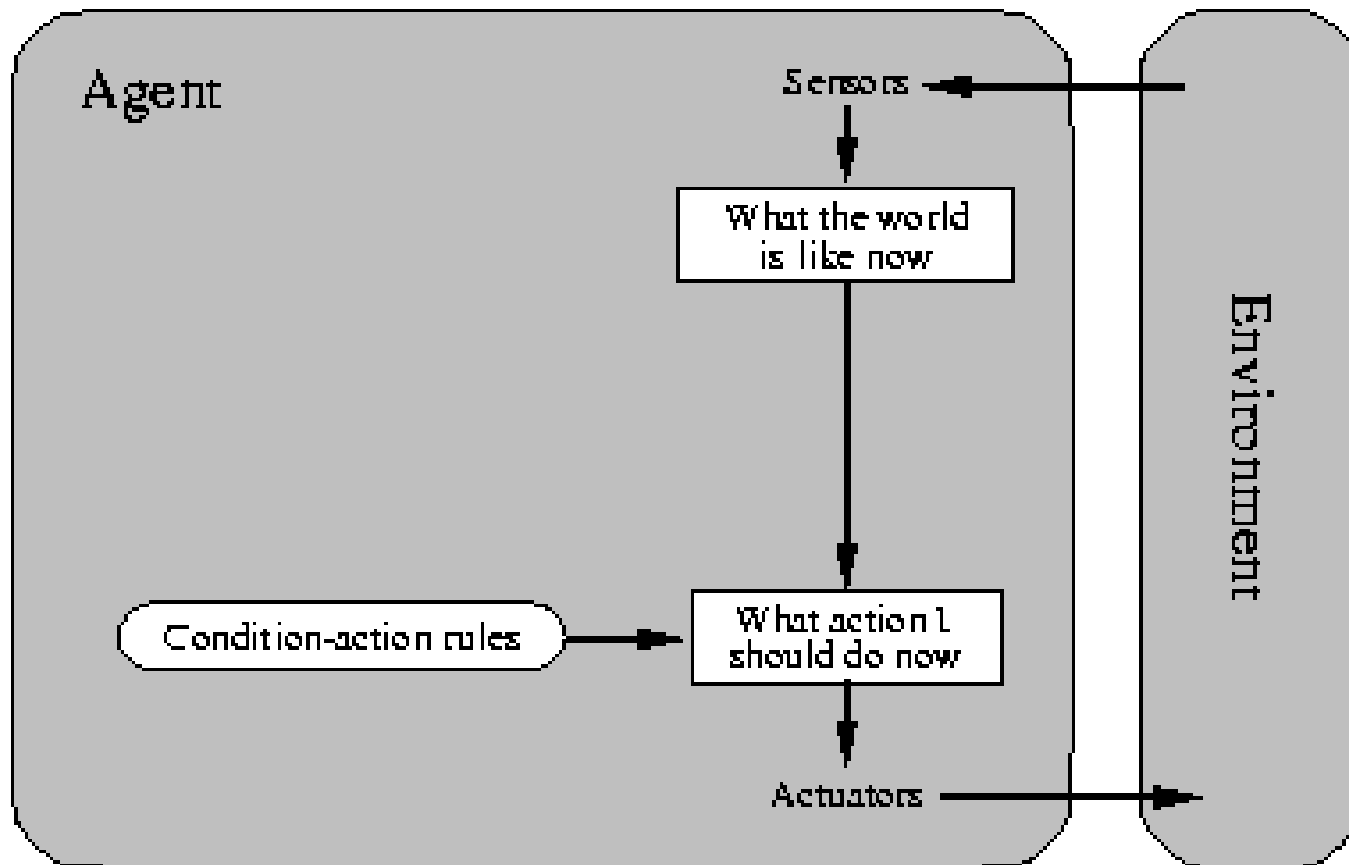
Structure of Agents

- Agent = architecture + program
- architecture
 - device with sensors and actuators
 - e.g., A robotic car, a camera, a PC, ...
- program
 - implements the agent function on the architecture

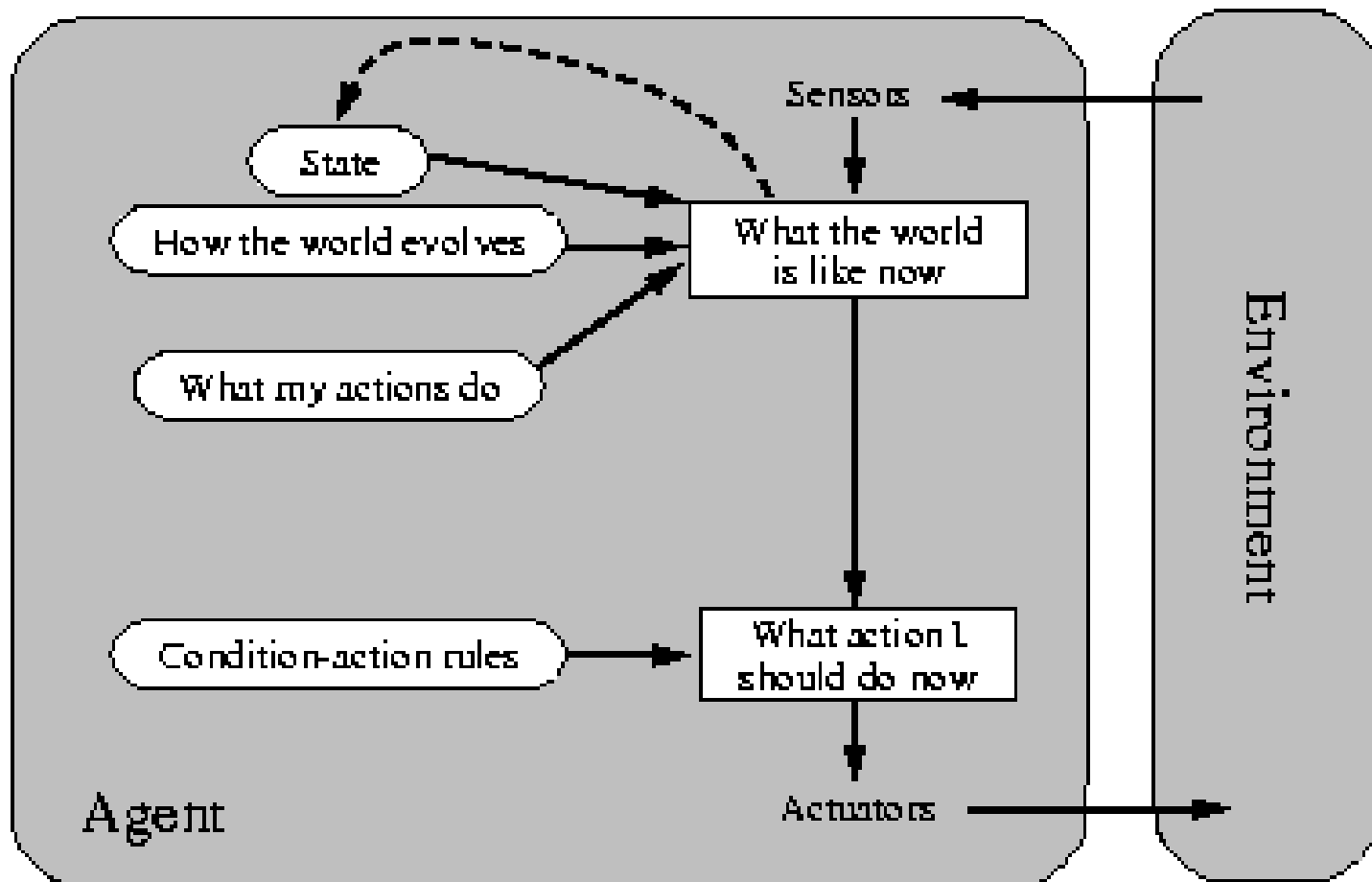
Types of Agents

- Reflex Agent
- Reflex Agent with State
- Goal-based Agent
- Utility-Based Agent
- Learning Agent

Reflex Agent



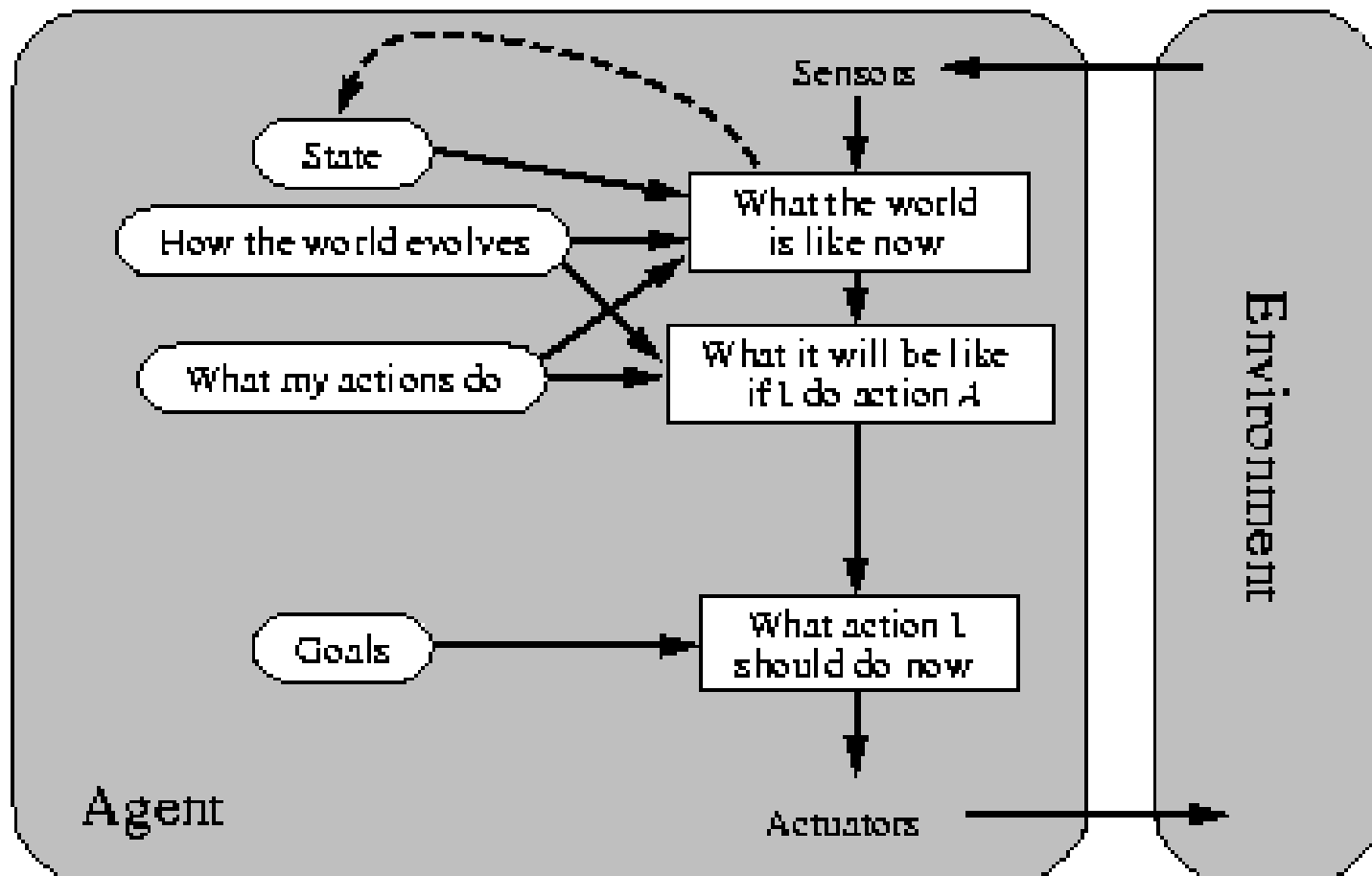
Reflex Agent with State



State Management

- Reflex agent with state
 - Incorporates a **model** of the world
 - Current state of its world depends on percept history
 - Rule to be applied next depends on resulting state
- $\text{state}' \leftarrow \text{next-state}(\text{state}, \text{percept})$
 $\text{action} \leftarrow \text{select-action}(\text{state}', \text{rules})$

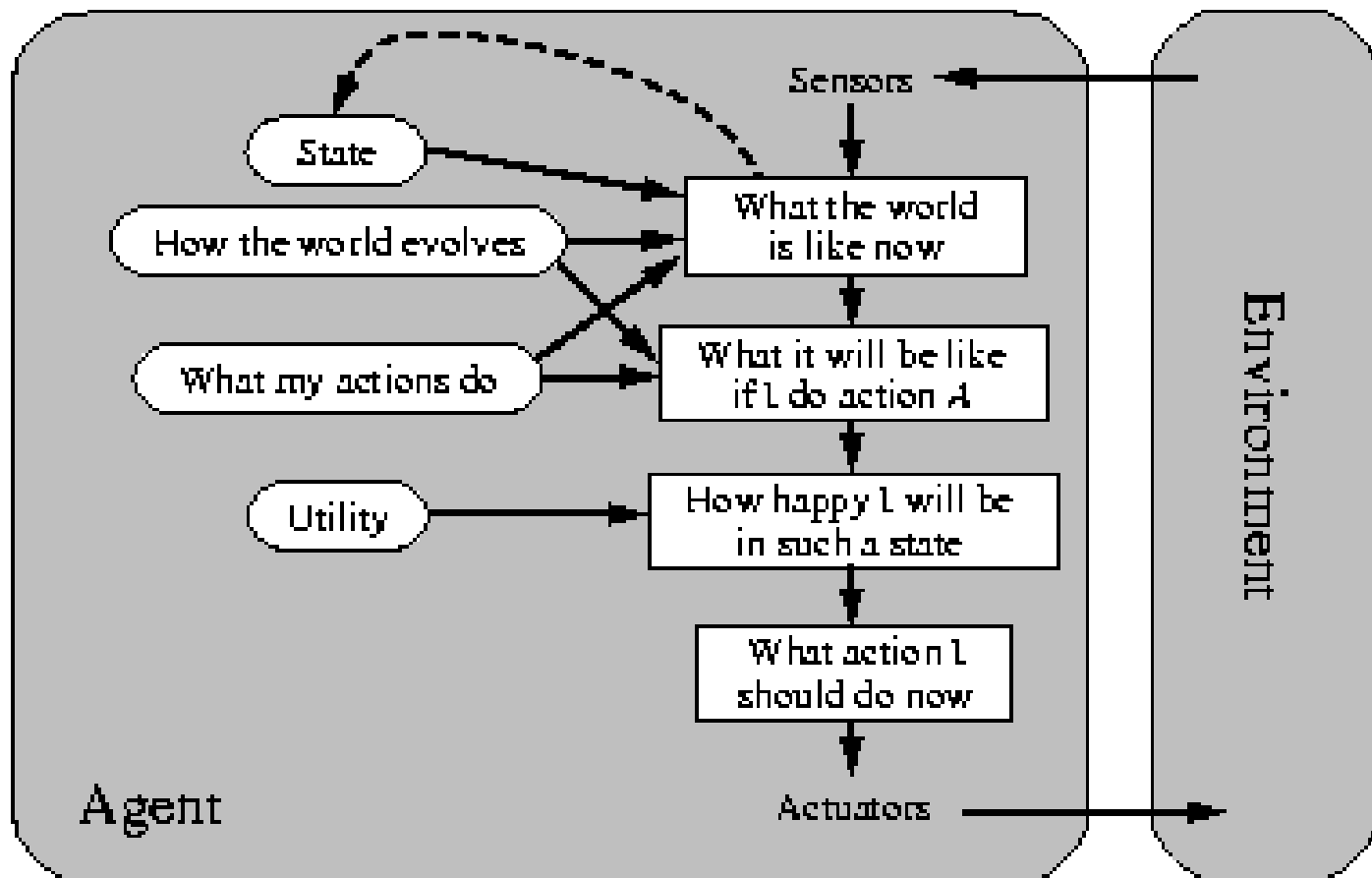
Goal-based Agent



Incorporating Goals

- Rules and “foresight”
 - Essentially, the agent’s rule set is determined by its goals
 - Requires knowledge of future consequences given possible actions
- Can also be viewed as an agent with more complex state management
 - Goals provide for a more sophisticated next-state function

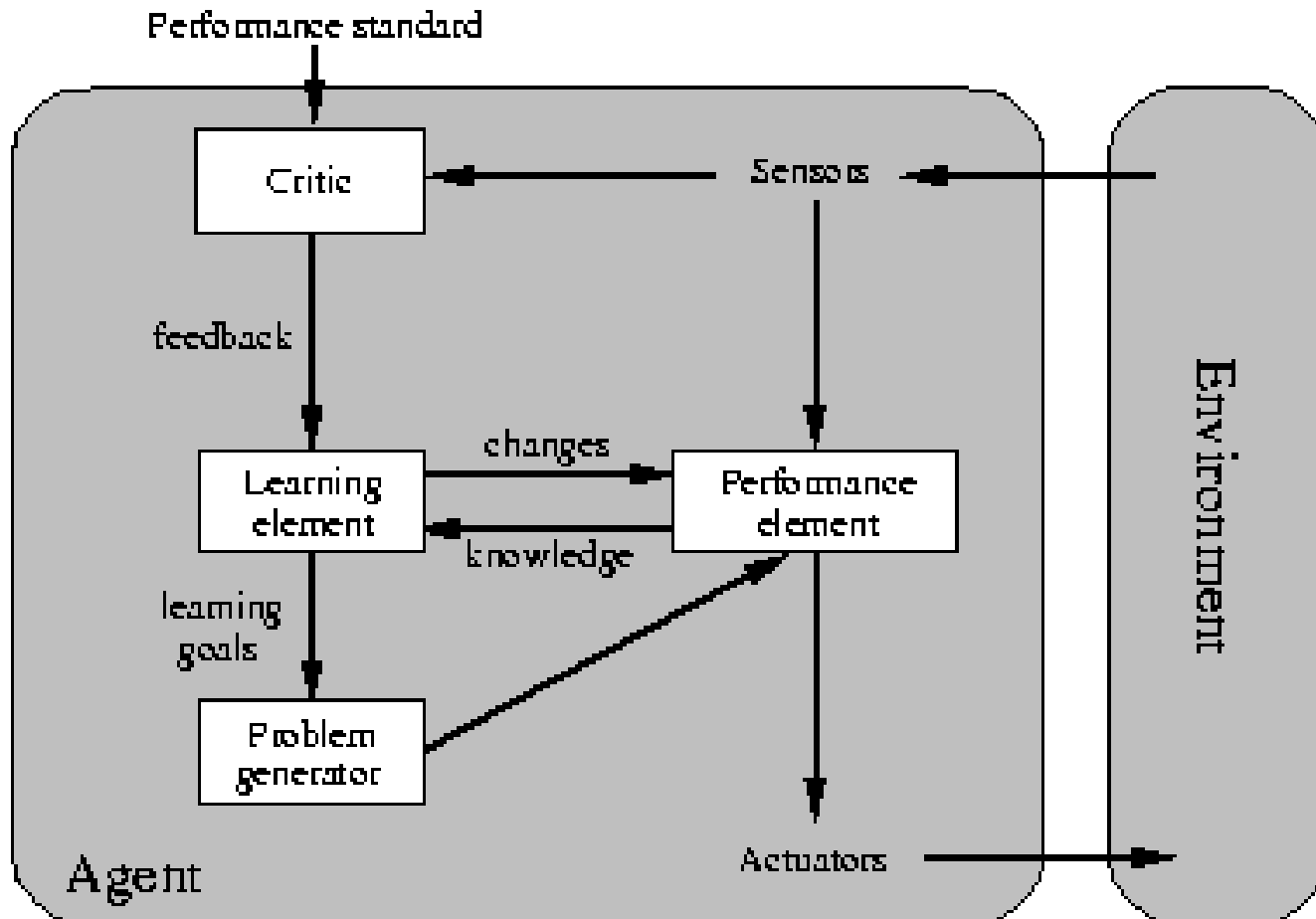
Utility-based Agent



Incorporating Performance

- May have multiple action sequences that arrive at a goal
- Choose action that provides the best level of “happiness” for the agent
- Utility function maps states to a measure
 - May include tradeoffs
 - May incorporate likelihood measures

Learning Agent



Incorporating Learning

- Can be applied to any of the previous agent types
 - Agent \leftrightarrow Performance Element
- Learning Element
 - Causes improvements on agent/ **performance element**
 - Uses feedback from **critic**
 - Provides goals to **problem generator**

Knowledge-based agents

- Knowledge-based agents can benefit from knowledge expressed in very general forms, combining and recombining information to suit myriad purposes.
- Often, this process can be quite far removed from the needs of the moment—as when a mathematician proves a theorem or an astronomer calculates the earth's life expectancy.
- A knowledge-based agent can combine general knowledge with current percepts to infer hidden aspects of the current state prior to selecting actions.
- For example, a physician diagnoses a patient—that is, infers a disease state that is not directly observable—prior to choosing a treatment.
- Some of the knowledge that the physician uses is in the form of rules learned from textbooks and teachers, and some is in the form of patterns of association that the physician may not be able to consciously describe. If it's inside the physician's head, it counts as knowledge.

Knowledge-based agents continue....

- Understanding natural language also requires inferring hidden state, namely, the intention of the speaker.
- When we hear, “John saw the diamond through the window and coveted it,” we know “it” refers to the diamond and not the window—we reason, perhaps unconsciously, with our knowledge of relative value.
- Similarly, when we hear, “John threw the brick through the window and broke it,” we know “it” refers to the window.
- Reasoning allows us to cope with the virtually infinite variety of utterances using a finite store of commonsense knowledge.

Knowledge-based agents continue....

- Problem-solving agents have difficulty with this kind of ambiguity because their representation of contingency problems is inherently exponential.
- Our final reason for studying knowledge-based agents is their flexibility.
- They are able to accept new tasks in the form of explicitly described goals, they can achieve competence quickly by being told or learning new knowledge about the environment, and they can adapt to changes in the environment by updating the relevant knowledge.

Knowledge-based agents continue....

- The central component of a knowledge-based agent is its **knowledge base, or KB**.
- **Informally**, a knowledge base is a set of **sentences**. (Here “sentence” is used as a technical term.
- **It is** related but is not identical to the sentences of English and other natural languages.) Each sentence is expressed in a language called a **knowledge representation language** and **represents** some assertion about the world.
- There must be a way to add new sentences to the knowledge base and a way to query what is known.
- The standard names for these tasks are TELL and ASK, respectively. Both tasks may involve **inference—that is, deriving new sentences from old**. In **logical agents**, which are the main subject of study in this chapter, inference must obey the fundamental requirement that when one ASKs a question of the knowledge base, the answer should follow from what has been told (or rather, TELLED) to the knowledge base previously.
- Later in the chapter, we will be more precise about the crucial word “follow.” For now, take it to mean that the inference process should not just make things up as it goes along.

A generic knowledge-based agent

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 \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

- Above figure shows the outline of a knowledge-based agent program. Like all our agents, it takes a percept as input and returns an action.
- The agent maintains a knowledge base, KB, BACKGROUND which may initially contain some **background knowledge**. **Each time the agent program is** KNOWLEDGE called, it does two things. First, it TELLS the knowledge base what it perceives. Second, it ASKS the knowledge base what action it should perform.
- In the process of answering this query, extensive reasoning may be done about the current state of the world, about the outcomes of possible action sequences, and so on.
- Once the action is chosen, the agent records its choice with TELL and executes the action. The second TELL is necessary to let the knowledge base know that the hypothetical *action has actually been executed*.

A generic knowledge-based agent

- The knowledge-based agent is not an arbitrary program for calculating actions. It is amenable to a description at the **KNOWLEDGE LEVEL knowledge level, where we need specify only what the agent knows and what its goals are**, in order to fix its behavior.
- For example, an automated taxi might have the goal of delivering a passenger to Marin County and might know that it is in San Francisco and that the Golden Gate Bridge is the only link between the two locations. Then we can expect it to cross the Golden Gate Bridge *because it knows that that will achieve its goal. Notice that this analysis* IMPLEMENTATION is independent of how the taxi works at the **implementation level**.
- **It doesn't matter whether** LEVEL its geographical knowledge is implemented as linked lists or pixel maps, or whether it reasons by manipulating strings of symbols stored in registers or by propagating noisy signals in a network of neurons.

A generic knowledge-based agent

- *One can build a knowledge-based agent simply by TELLing it what it needs to know. The agent's initial program, before it starts to receive percepts, is built by adding one by one the sentences that represent the designer's knowledge of the environment.*
- Designing the representation language to make it easy to express this knowledge in the form of sentences simplifies the construction problem DECLARATIVE enormously.
- This is called the **declarative approach to system building**.
- **In contrast, the procedural approach encodes desired behaviors directly as program code; minimizing the** role of explicit representation and reasoning can result in a much more efficient system.
- We will see agents of both kinds in Section 7.7. In the 1970s and 1980s, advocates of the two approaches engaged in heated debates. We now understand that a successful agent must combine both declarative and procedural elements in its design.

A generic knowledge-based agent

- In addition to TELLing it what it needs to know, we can provide a knowledge-based agent with mechanisms that allow it to learn for itself.
- These mechanisms, create general knowledge about the environment out of a series of percepts.
- This knowledge can be incorporated into the agent's knowledge base and used for decision making. In this way, the agent can be fully autonomous.
- All these capabilities—representation, reasoning, and learning—rest on the centuries long development of the theory and technology of logic.
- Before explaining that theory and technology, however, we will create a simple world with which to illustrate them.

Boolean circuit agents