Knowledge representation

- Knowledge representation is a study of ways of how knowledge is actually picturized and how effectively it resembles the representation of knowledge in human brain.

Some widely known representation schemes:

- Semantic nets
- Frames
- Scripts
- Conceptual dependency

Semantic networks

- A semantic network is a structure for representing knowledge as a pattern of interconnected nodes and arc.
- It is a graphical representation of knowledge.
- Nodes in the semantic net represent either

-- Entities

-- attributes

-- states or

-- Events

- Arcs in the net gives the relationship between the nodes and labels on the arc specify the type of relationship.

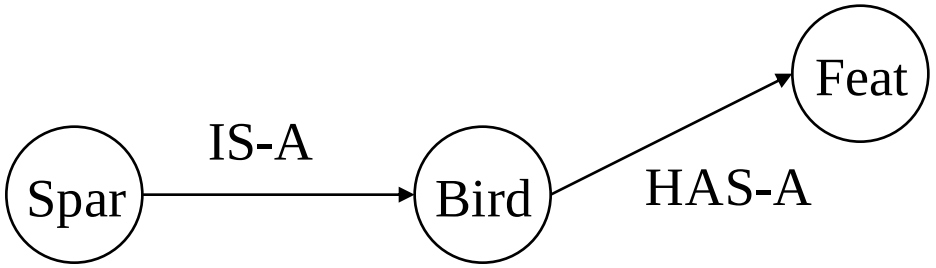
Semantic networks

- "A sparrow is a bird"
  - Two concepts: "sparrow" and "bird"

  - A sparrow is a kind of bird, so connect the two concepts with a *IS-A relation*

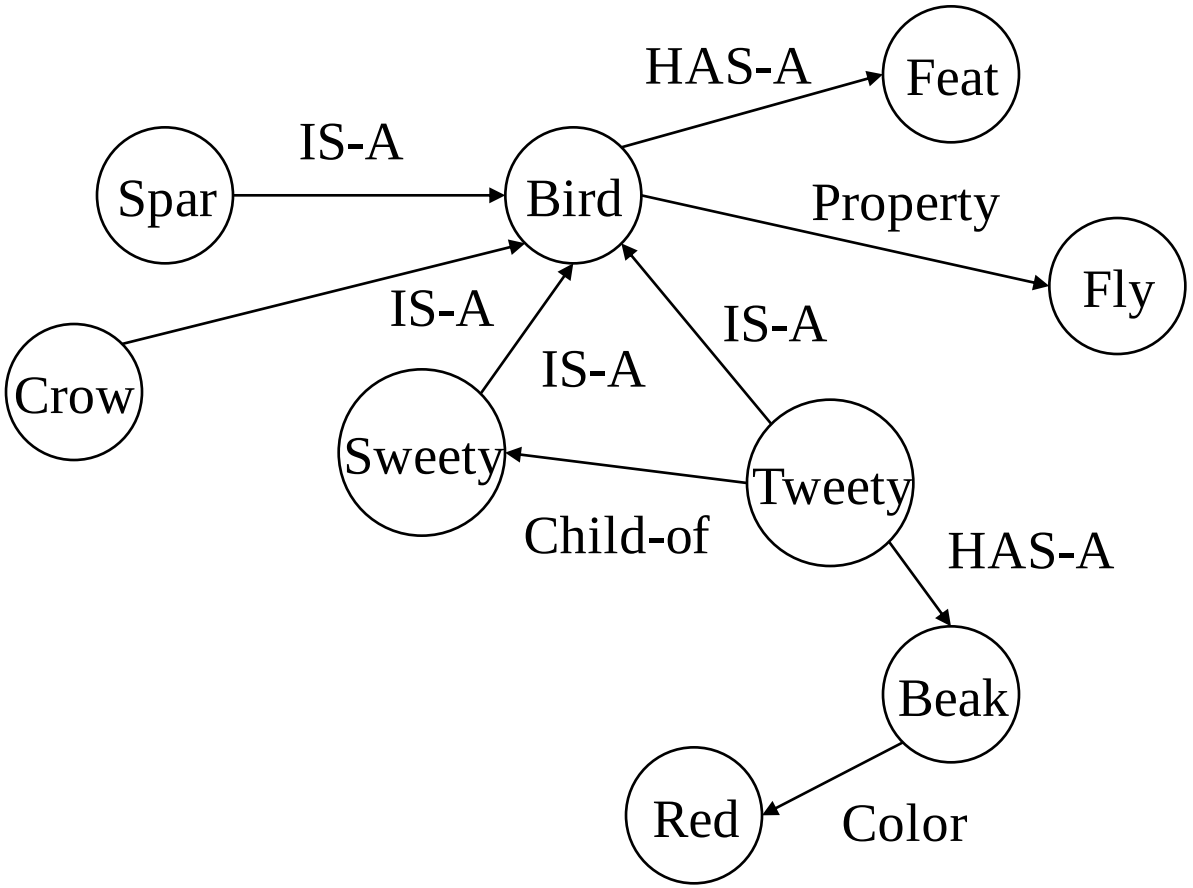
Semantic networks

- "A bird has feathers"
  - 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



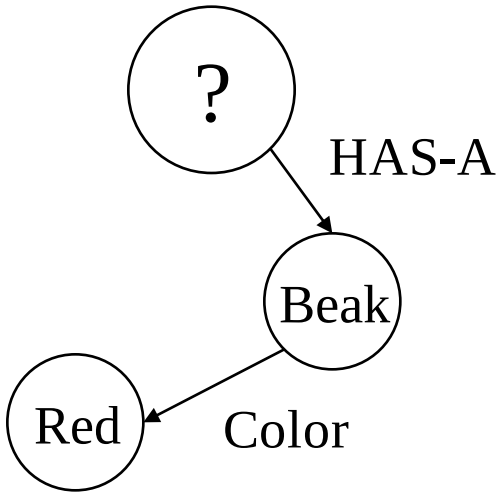
- Tweety and Sweety are birds
- Tweety has a red beak
- Sweety is Tweety’s child
- A crow is a bird
- Birds can fly

Semantic networks



Semantic networks

- Semantic networks can answer queries
  - *Query*: "Which birds have red beaks?"
  - *Answer*: Tweety
  - *Method*: Direct match of subgraph

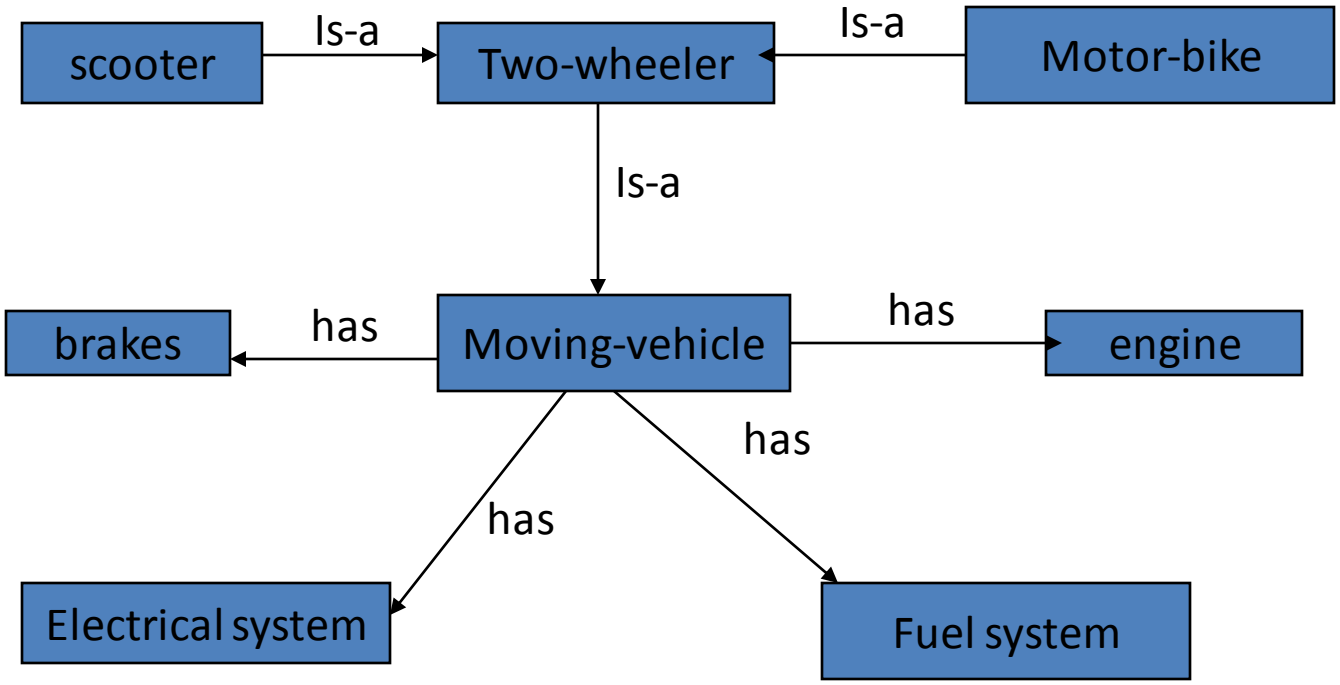


  - *Query*: "Can Tweety fly?"
  - *Answer*: Yes
  - *Method*: Following the IS-A link from "Tweety" to "bird" and the property link of "bird" to "fly"
  - This process is called inheritance

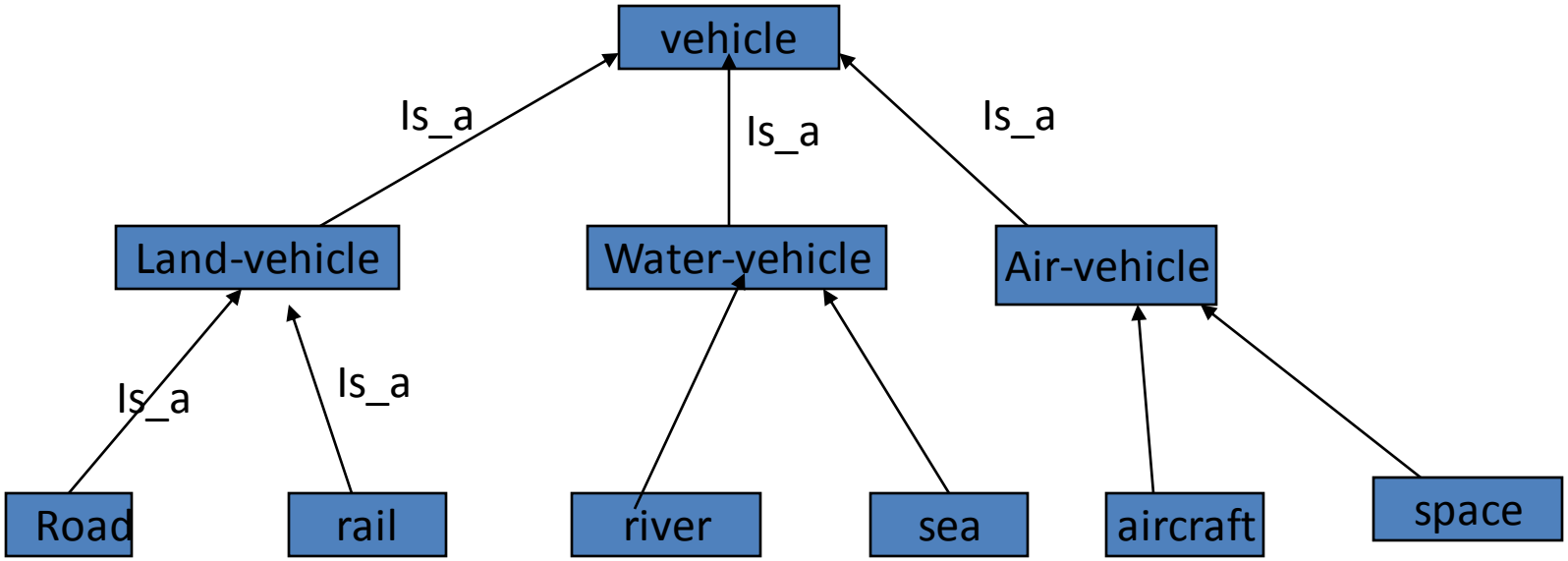
Convert following into Semantic Net

1. Motor-bike is a two wheeler.
2. Two wheeler is a moving vehicle.
3. Moving-vehicle has a brake.
4. Moving vehicle has a engine.
5. Moving vehicle has electrical system.
6. Moving vehicle has fuel- system.

A simple semantic net



Hierarchical Structure



Q.42. 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 –

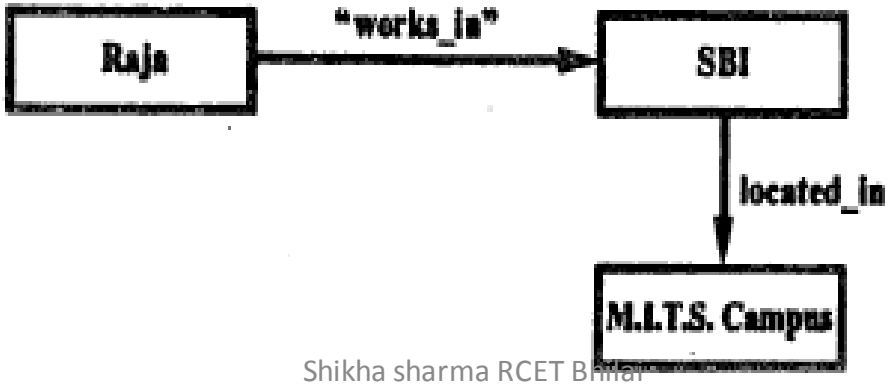
(i) Raja is a bank manager.

Here link between nodes is "is_a".

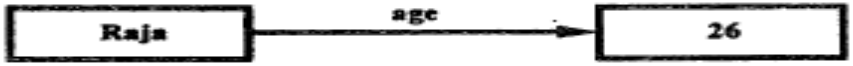
So, it can be represented as –

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

Here we will have two links "works_in and located_in.



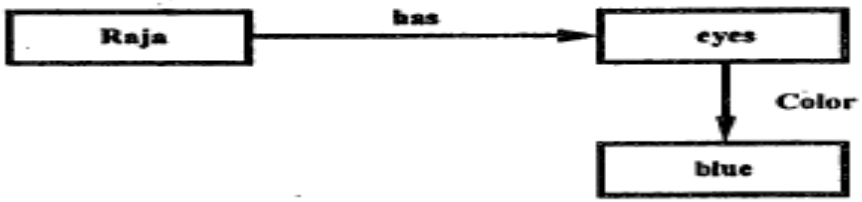
(iii) Raja is 26 years old.



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.



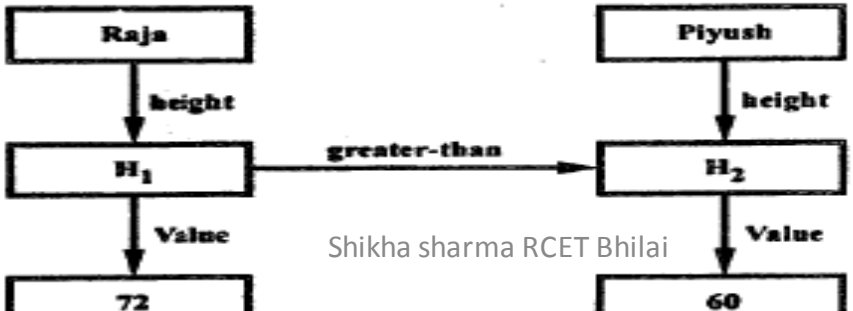
This shows that Raja has eyes and color of eyes is blue, so Raja has blue eyes.

(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.

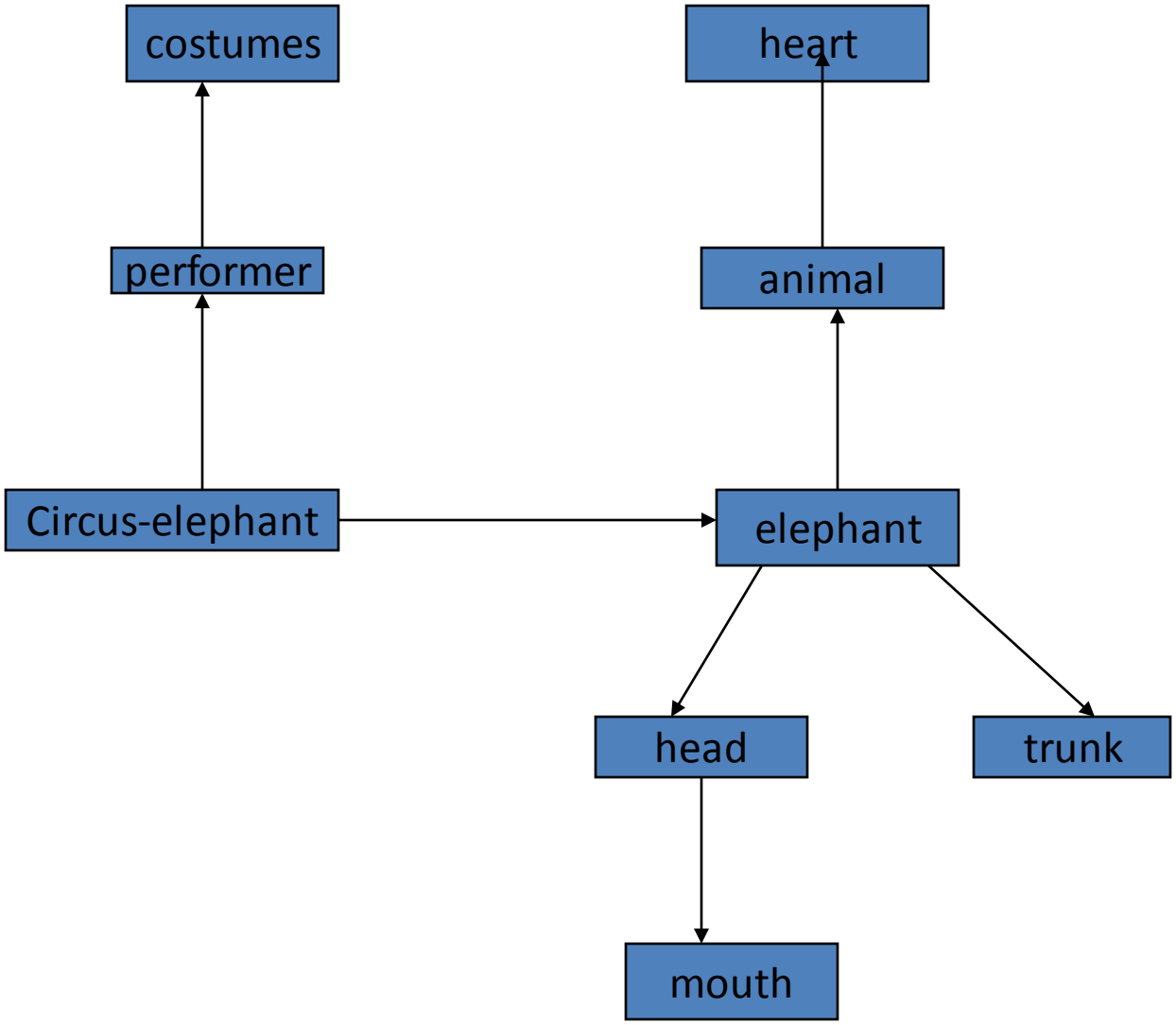
Next, we show that height of Raja is greater than height of Piyush. So, Raja is taller than Piyush.

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 SN

- (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)


costumes
heart
performer
animal
Circus-elephant
elephant
head
trunk
mouth

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

Frame systems

- Frame theory
  - When humans encounter something new, a basic structure called a *frame* is selected from memory
  - A frame is a fixed framework in which all kinds of information is stored
  - For more details about the information in a frame, a different frame is selected
  - A frame is connected to other frames, so this is a network of frames

Frame

- The term *Frame* was introduced in Minsky's paper: ``A Framework for Representing Knowledge''.
- A basic idea of frames is that people make use of stereo typed information about typical features of objects, images, and situations;
- such information is assumed to be structured in large units representing the stereotypes, and these units are referred to as ``frames''.

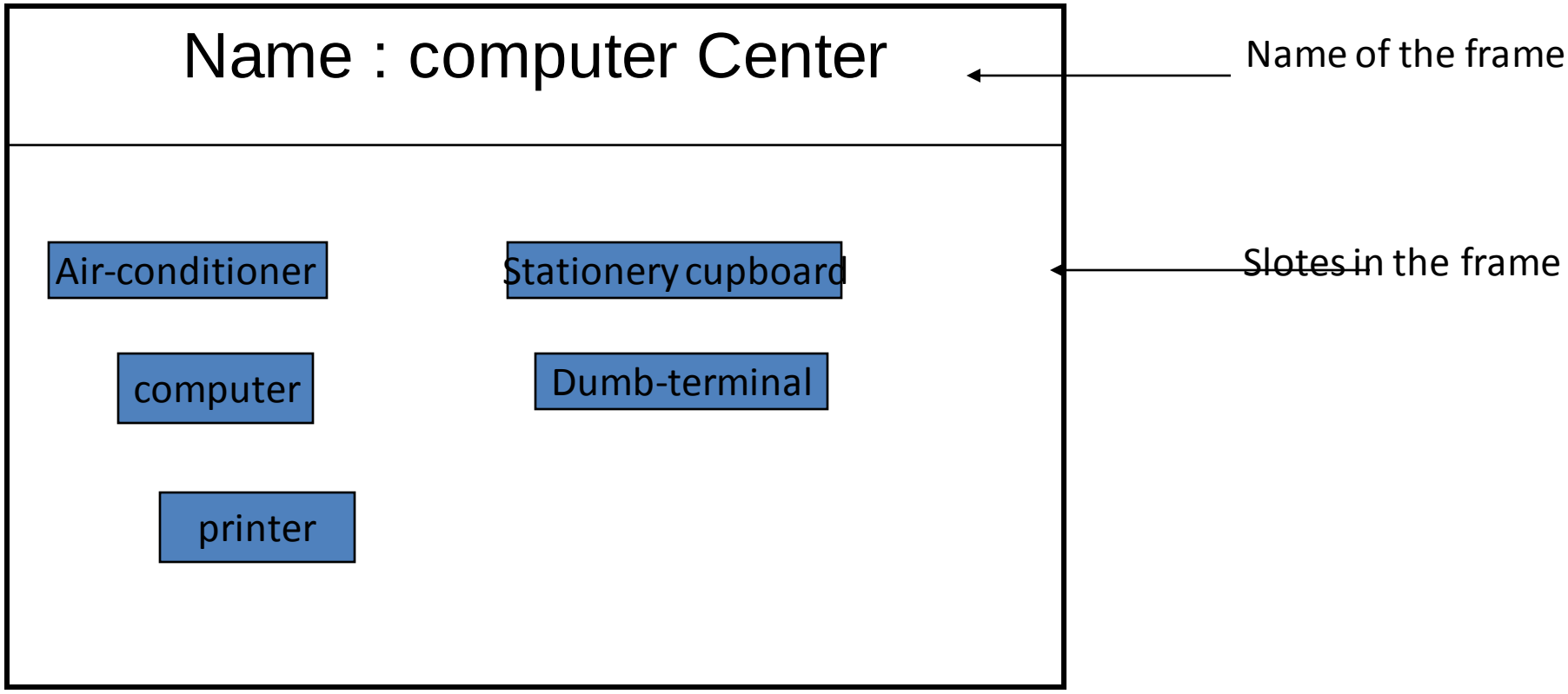
Typical Features of Frames

- A frame can represent an *individual* object or a *class* of similar objects.

- Instead of properties, a frame has *slots*. A slot is like a property, but can contain more kinds of information

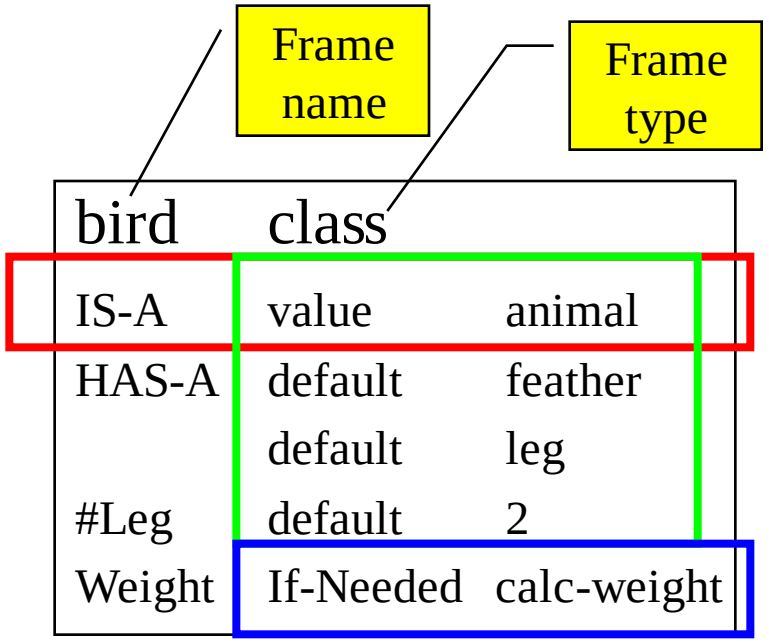
- Data type information; constraints on possible slot fillers, Documentation.
- Frames can inherit slots from *parent* frames. For example, FIDO (an individual dog) might inherit properties from DOG (its parent class) or MAMMAL (a parent class of DOG).

A sample frame of a computer center

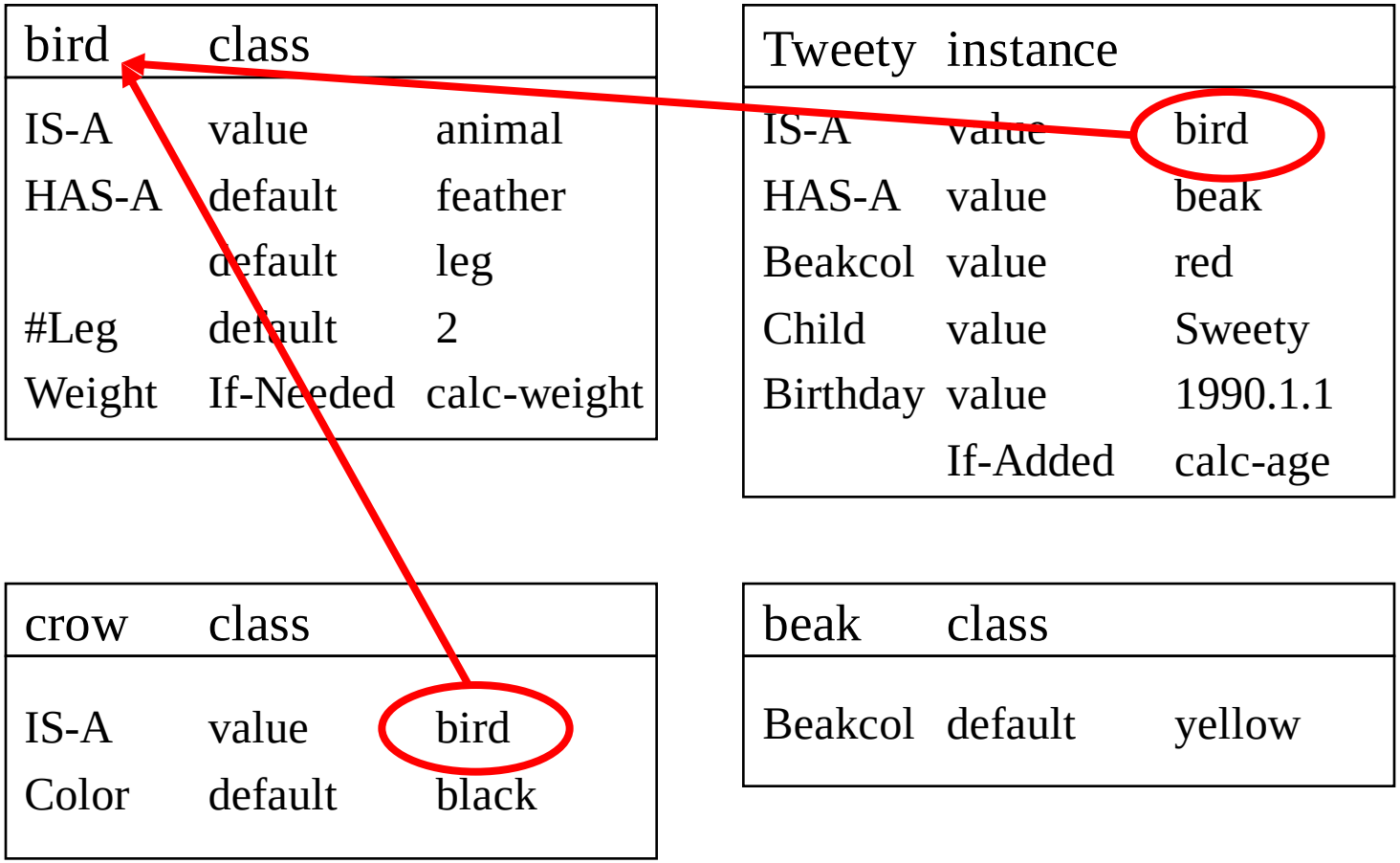


Frame systems

- Frame
  - *Frame name*: represents an object or a concept, so similar to node in the semantic network
  - *Frame type*: shows if this a concept (class) or an object (instance)
- Slot
  - Consists of slot name and facets
  - *Slot name*: property or relation name
- Facet
  - A facet gives information about the slot, i.e. the value and name
  - *Value*: the value of the property
  - *Default*: connecting frames can have a different value for this property
- Demon
  - Perform a certain action if a condition is satisfied



Frame systems



Frame systems

bird	class	
IS-A	value	animal
HAS-A	default	feather
	default	leg
#Leg	default	2
Weight	If-Needed	calc-weight

Tweety	instance	
IS-A	value	bird
HAS-A	value	beak
Beakcol	value	red
Child	value	Sweety
Birthday	value	1990.1.1
	If-Added	calc-age

crow	class	
IS-A	value	bird
Color	default	black

beak	class	
Beakcol	default	yellow

Frame systems

- Inference in frame systems
  - *Query*: "How many legs has a crow?"
  - *Answer*: 2
  - *Inference*
    - No information about this in the "crow" frame
    - Try to find it in the "bird" frame
    - Default value is 2
  - Also called inheritance

  - As soon as the birthday of Tweety is added, the "calc-age" procedure is invoked

  - *Query*: "What is the weight of Tweety?"
  - The answer is obtained by the procedure "calc-weight" in bird

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

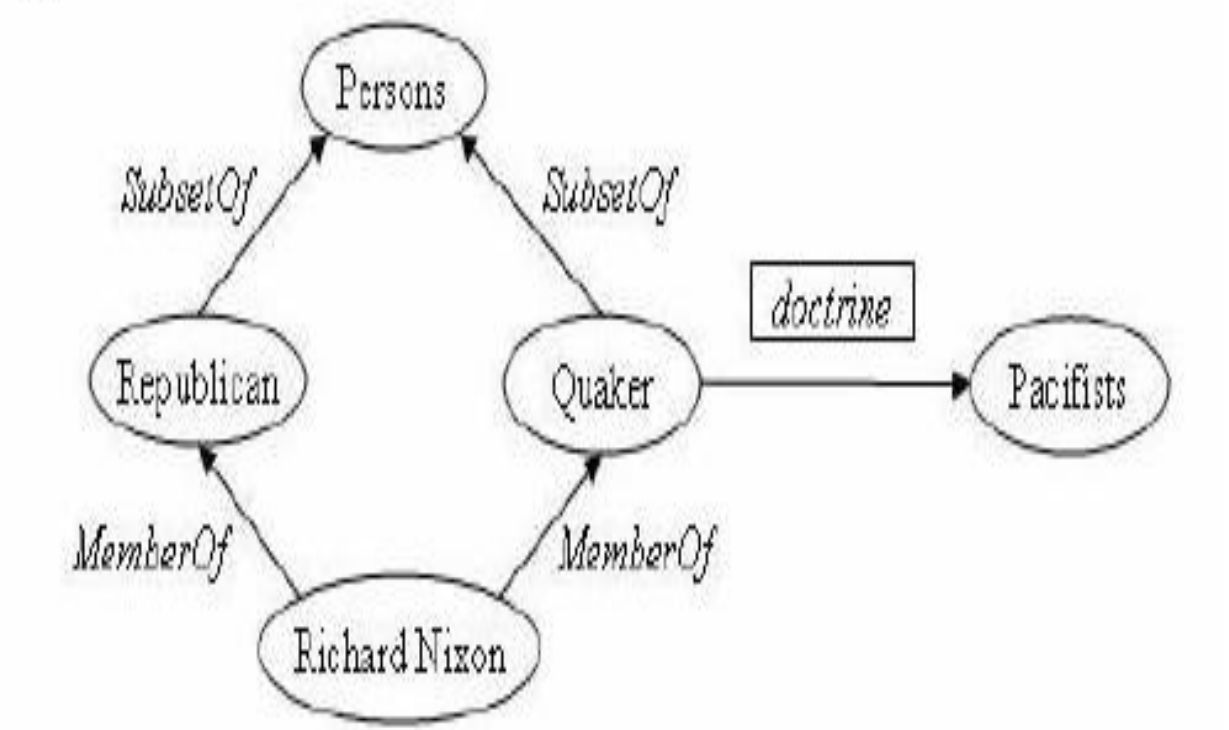
Construct semantic network representations

1Richard Nixon is a Quaker and a Republican. Quakers and Republicans are Persons.

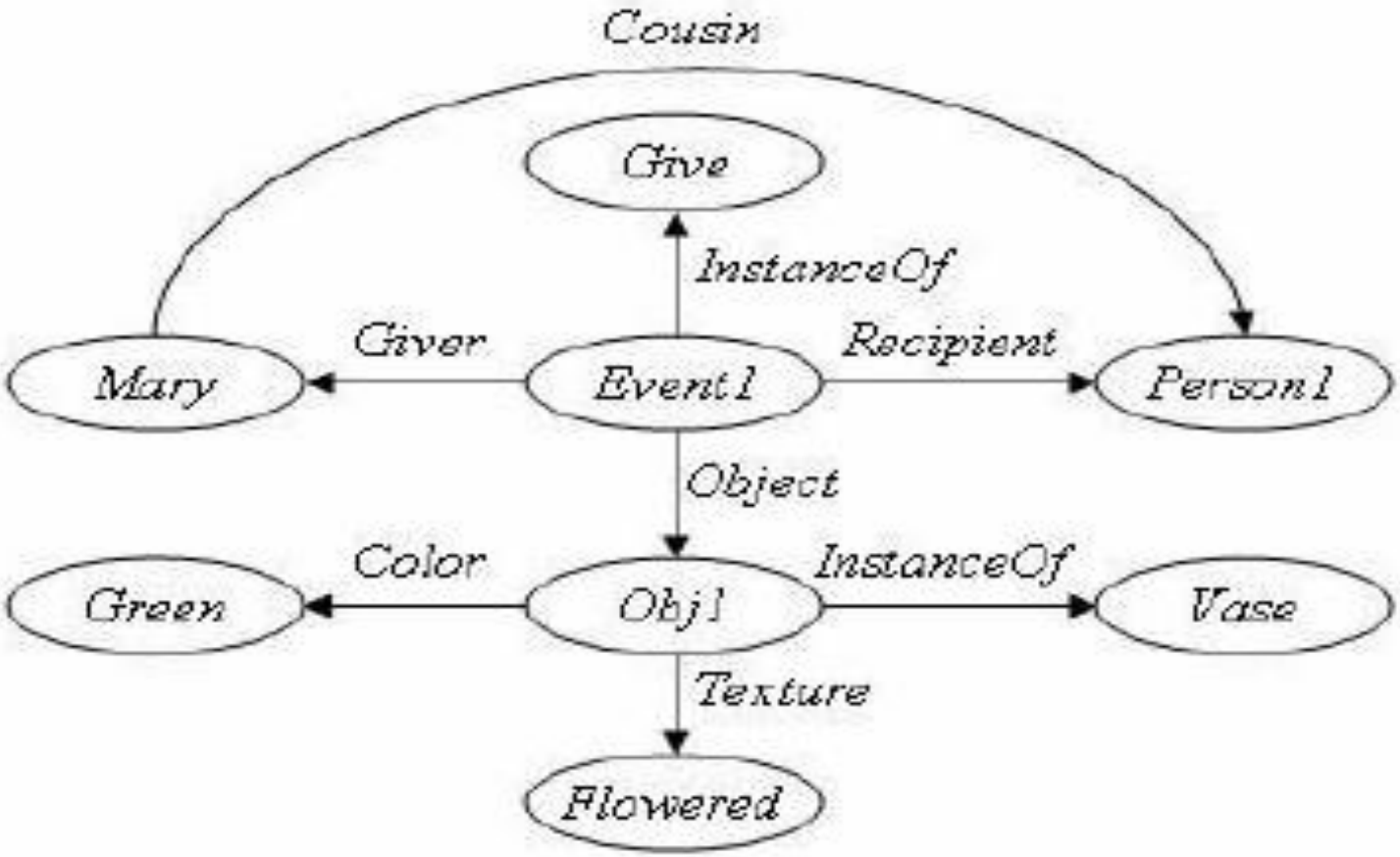
Every Quaker follows the doctrine of pacifism.

b. Mary gave the green flowered vase to her cousin.

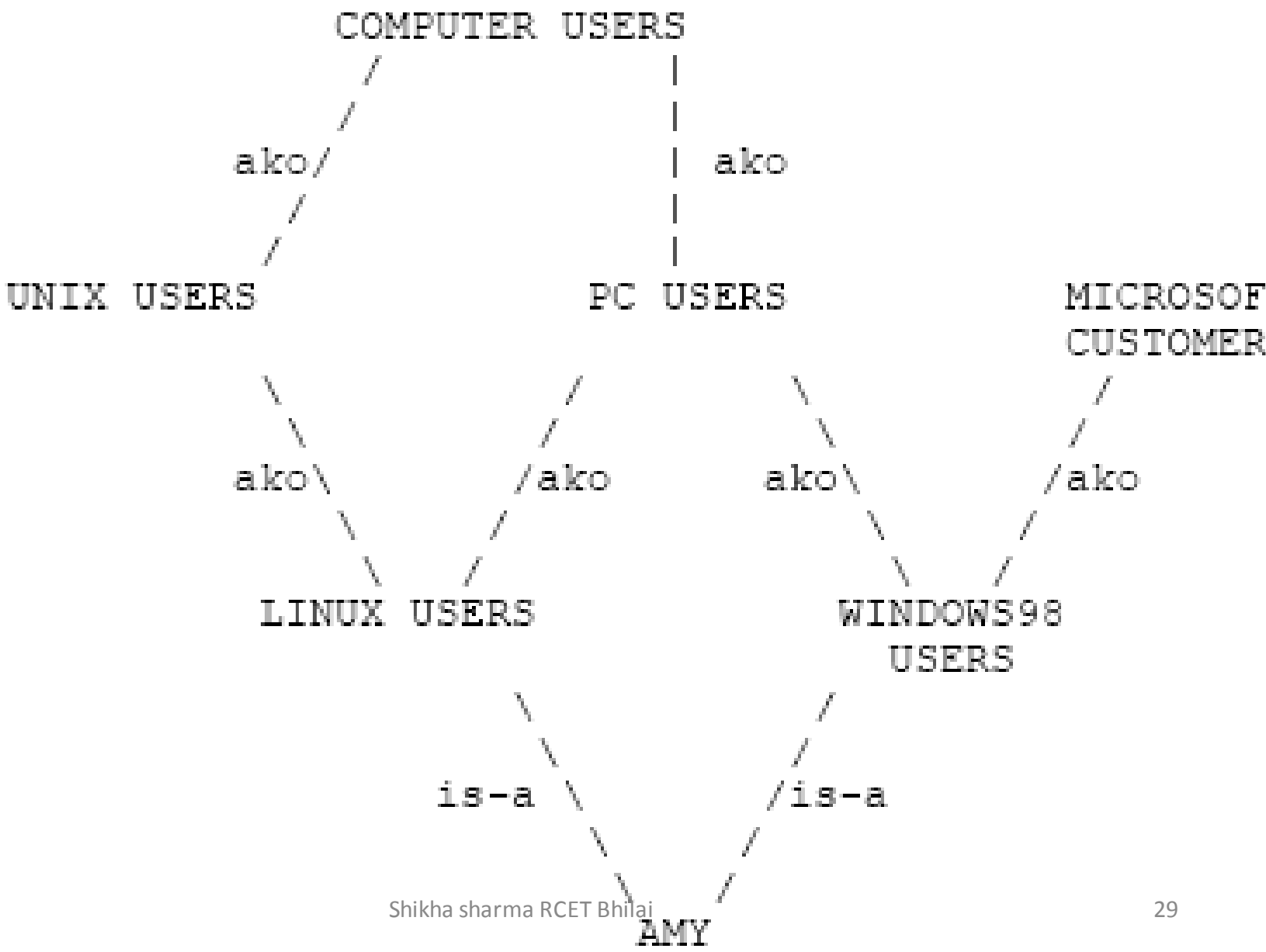
1.a


Persons
SubsetOf
SubsetOf
doctrine
Republican
Quaker
Pacifists
MemberOf
MemberOf
Richard Nixon

1.b


Cousin
Give
InstanceOf
Giver
Recipient
Mary
Event1
Person1
Object
Color
InstanceOf
Green
Obj1
Vase
Texture
Flowered

2. Consider the following hierarchy of frames.



a. Give the class-precedence list for Amy that would be obtained by applying the topological-sorting algorithm to the above graph.

b. Suppose that each of the classes *Unix users, PC users and Computer Users contains a favorite programming language slot. The default value for this slot is:*

- Fortran, for the *Computer Users class.*
- C, for the *Unix Users class.*
- C++, for the *PC Users class.*

What is the value obtained for Amy's favorite programming language according to the class-precedence list you constructed above?

2.a.

Node	Fish-hook pairs
Amy	Amy-Linux Users, Linux Users-Windows98 Users
Linux Users	Linux Users-Unix Users, Unix Users-PC Users
Windows98 Users	Windows98 Users-PC Users, PC Users-Microsoft Customers
Unix Users	Unix Users-Computer Users
PC Users	PC Users-Computer Users

Class Precedence list :

Amy	
Linux Users	
Unix Users	- Use C
Windows98 Users	
PC Users	- Use C++
Computer Users	- Use Fortran
Microsoft Customers	

1. Amy's favorite programming language is C

Scripts

- Due to 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)

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

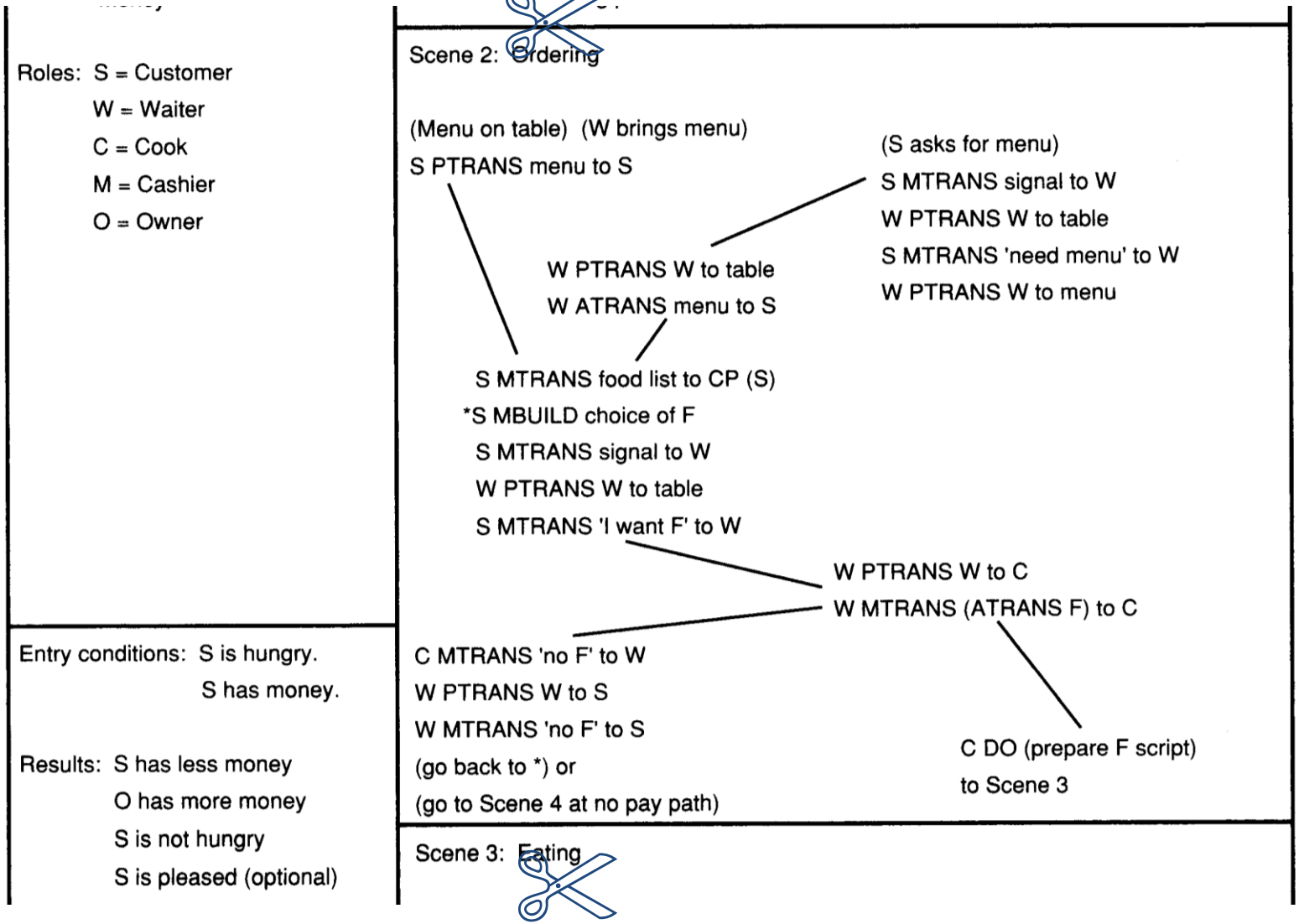
Components of scripts (2)

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, …

Example of a script (1)

Script: RESTAURANT Track: Coffee Shop Props: Tables Menu F = Food Check Money Roles: S = Customer	Scene 1: Entering S PTRANS S into restaurant S ATTEND eyes to tables S MBUILD where to sit S PTRANS S to table S MOVE S to sitting position
	Scene 2: Ordering

Example of a script (2)



Example of a script (3)

C has more money

S is not hungry

S is pleased (optional)

(go to Scene 4 at no pay path)

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)

So?

- Story: "*Sue went out to lunch. She sat at a table and called a waitress, who brought her a menu. She ordered a sandwich.*"
- A system can now:
  - retrieve the restaurant script
  - check entry conditions
    - unify {S / Sue}
    - infer that (typically) Sue is hungry and Sue has money
  - unify people and things in the story with the roles and props in the script
    - {W / waitress, F / sandwich}

So?

- answer questions
  - why did the waitress bring a menu to Sue?
    - because S MTRANS “need menu” to W …
    - … Sue tells “need menu” to waitress
  - who was the “she” who ordered a sandwich?
    - S MTRANS “I want F” to W …
    - … Sue tells “I want a sandwich” to the waitress
  - who paid?
    - S ATRANS money to M …
    - … Sue gives money to the cashier

Conceptual dependency graphs

- Extension to semantic networks to define a complete set of primitives to use as relations in semantic networks
- (very ambitious) Goal: model formally the deep semantic structure of natural language
- Four primitive concepts
  - ACT action
  - PP object
  - AA modifiers of actions
  - PA modifiers of objects

Primitive ACTs

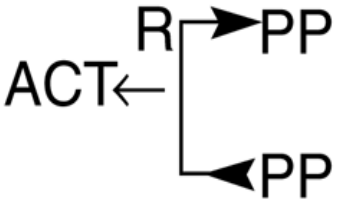
- Primitive ACTs represent basic actions
- All actions can be reduced to one or more primitive ACT (with modifiers)
- 12 primitive ACTs
  1. ATRANS transfer a relationship (*give)*
  2. PTRANS transfer a physical location of an object (*go)*
  3. PROPEL apply physical force to an object (*push)*
  4. 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)*

Syntax and semantics of CDGs (1)

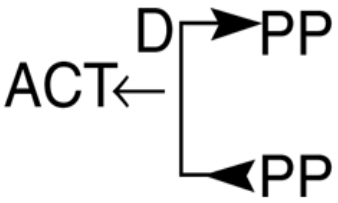
PP ⇔ ACT — indicates that an actor acts.

PP ⇔ PA — indicates that an object has a certain attribute.

ACT ←(O) PP — indicates the object of an action.

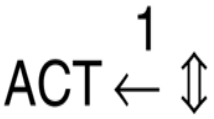


indicates the recipient and the donor of an object within an action.



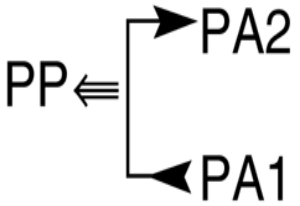
indicates the direction of an object within an action.

Syntax and semantics of CDGs (2)



indicates the instrumental conceptualization for an action.

indicates that conceptualization X caused conceptualization Y. When written with a C this form denotes that X COULD cause Y.



indicates a state change of an object.

PP1 ← PP2

indicates that PP2 is either PART OF or the POSSESSOR OF PP1.

Conceptual Dependency

Primitive conceptual tenses:

p	**Past**
f	**Future**
t	**Transition**
k	**Continuing**
....	

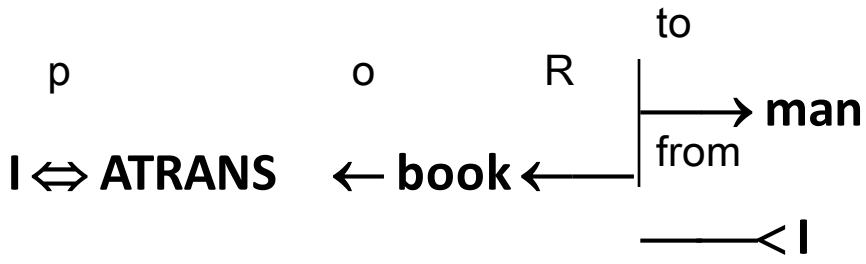
Conceptual Dependency

Primitive dependencies

	p	
PP ⇔ ACT	John ⇔ PTRANS	"John ran"
PP ⇔ PA	John ⇔ doctor	"John is a doctor"
PP ↑ PA	boy ↑ nice	"A nice boy"
ACT ←o PP	John ⇔ PROPEL ←o cart	"John pushed the cart"
....		

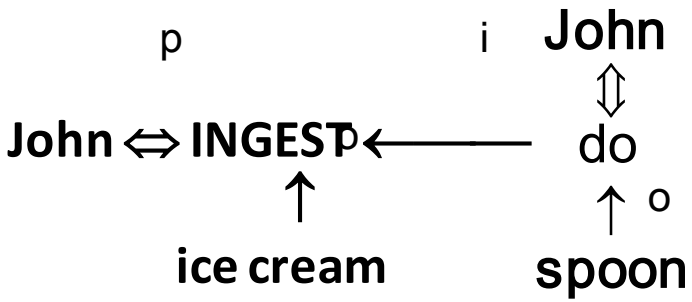
Conceptual Dependency

“I gave the man a book”



Conceptual Dependency

“John ate ice scream with a spoon”



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.

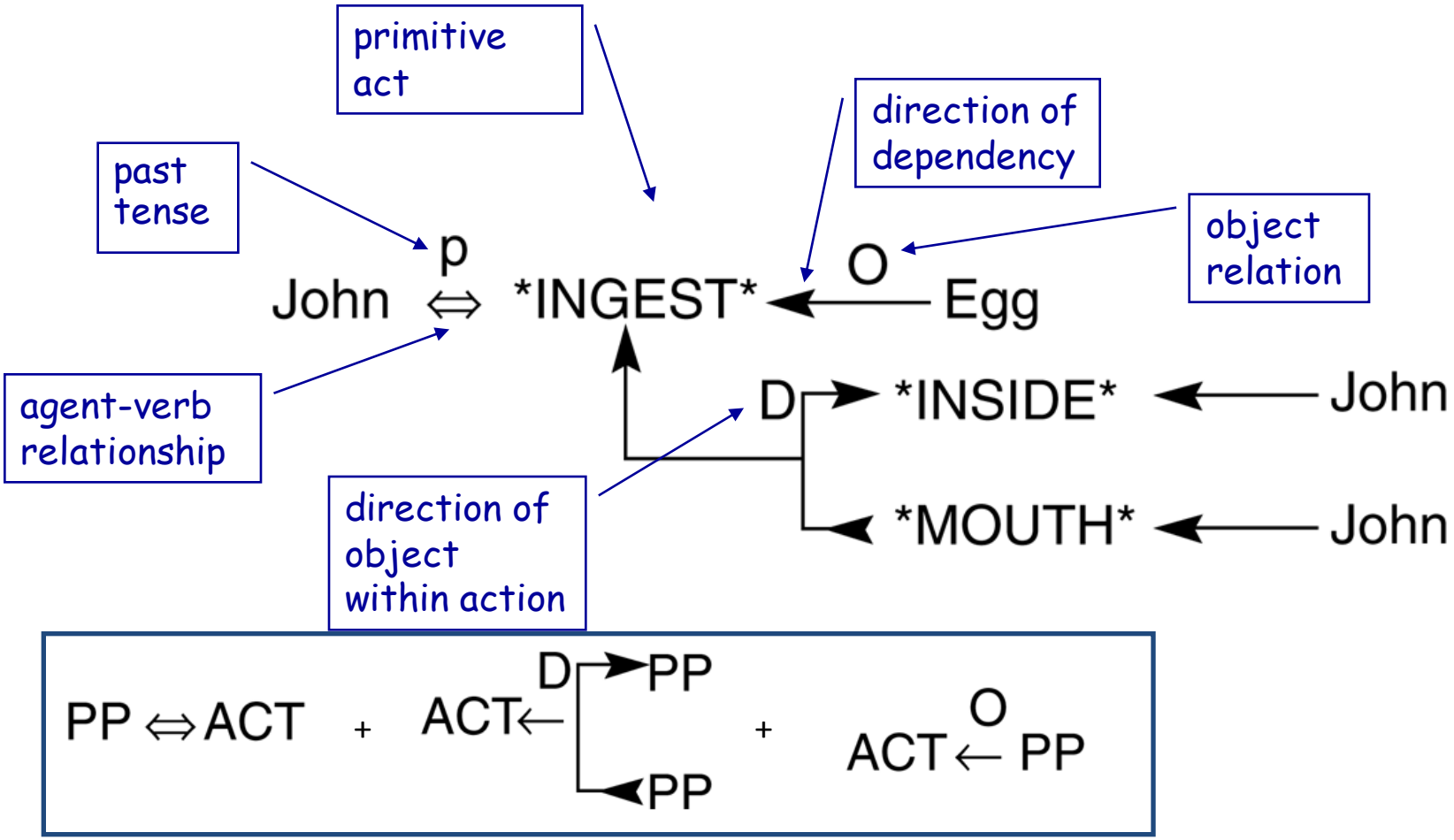
Conceptual Dependency

Disadvantages:

Requires knowledge to be decomposed into fairly low-level primitives.

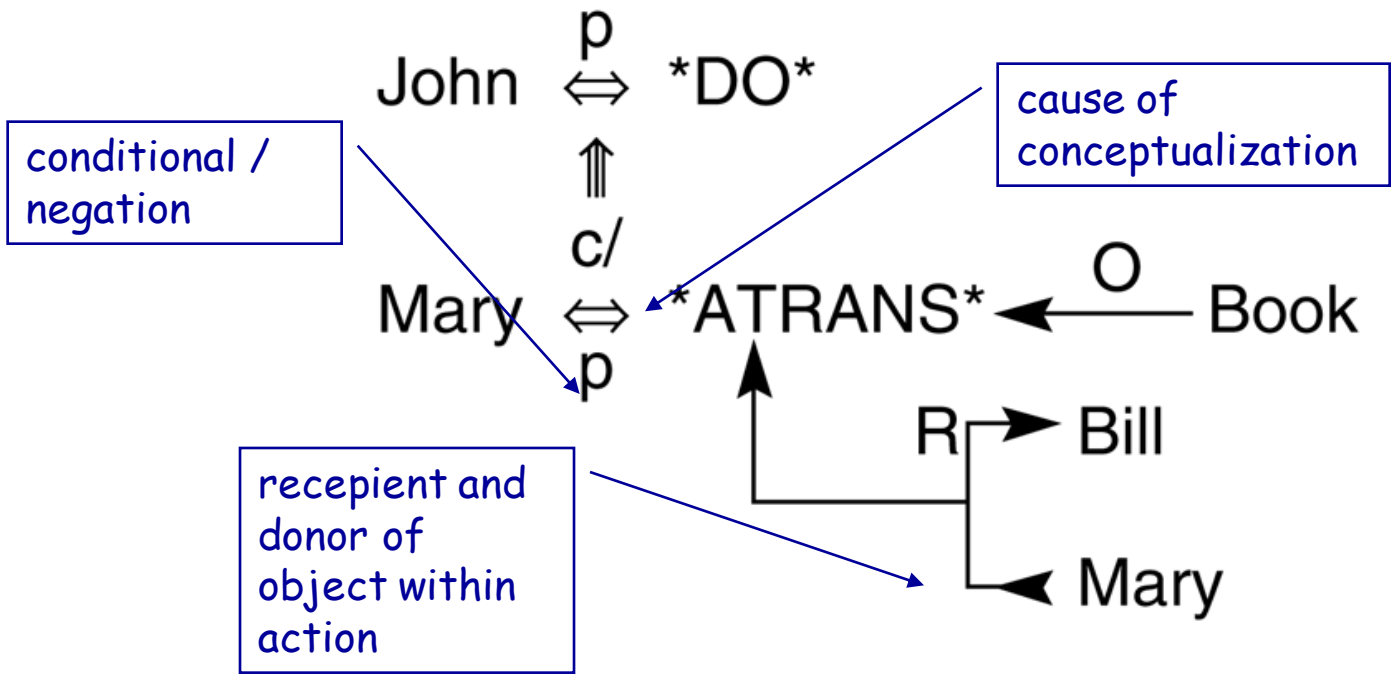
Is only a theory of the representation of events.

Examples of CDGs



"John ate the egg"

Examples of CDGs (2)



"John prevented Mary from giving a book to Bill"

Problems with CDGs

- can't produce them automatically form NL
  - needs to be built by hand
- not necessarily what humans do
  - (use primitive acts to reason…)

Artificial Intelligence

Logic

- One of the prime activity of 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 and the system of rates and procedure that help in the reasoning process.
- Basically the logic process takes some function called premises and produces some outputs called conclusions.

Classification of Logic

1. Propositional Logic
2. Predicate Logic

Propositional Logic

- This is the simplest form of logic
- It takes only two values , i. e. either the proposition is true or it is false.
- Examples

Kinds of proposition

- Atomic or Simple Proposition → in which simple or atomic sentences.
- Molecular or Compound Propositions → combining one or more atomic proposition using a set of logical connectives.

Commonly used Propositional Logical Connectives

NAME	CONNECTIVE
CONJUCTION	AND
DISJUNCTION	OR
NEGATION	NOT
MATERIAL CONDITION	IMPLIES
JOINT DENIAL	NAND
DISJOINT DENIAL	NOR

SEMANTICS OF LOGICAL PROPOSITIONS

- Example →
- The machine is defective.
- The production is less.

Properties of statements

- Satisfiable
- Contradiction
- Valid
- Equivalence
- Logical Consequence

- **A sentence is**
  - **satisfiable** if it is true under some interpretation
  - **valid** if it is true under all possible interpretations
  - **Inconsistent/contradiction** if there does not exist any interpretation under which the sentence is true

- **logical consequence:** S |= X if all models of S are also models of X OR
- A sentence is LC of another if it is satisfied by all interpretations which satisfy the first.
- Example→ P is a LC of (P & Q) since any interpretation for which (P & Q) is true , P is also true.

Consider the following statement

- P V ~P
- P and ~ (~P)

First Order Predicate Logic
FOPL

- The predicate logic is logical extension of propositional logic.
- FOPL was developed by logician as a means for formal reasoning , primarily in the areas of mathematics.
- It is used in representing different kind of knowledge.
- FOPL is flexible enough to permit the accurate representation of natural language.

- It is commonly used in program design.
- It provides a way of deducing new statements from old ones.

First Order (Predicate) Logic (FOL)

- First-order logic is used to model the world in terms of
  - **objects** which are things with individual identities

    e.g., individual students, lecturers, companies, cars ...
  - **properties** of objects that distinguish them from other objects

    e.g., mortal, blue, oval, even, large, ...
  - **classes** of objects (often defined by properties)

    e.g., human, mammal, machine, ...
  - **functions** which are a subset of the relations in which there is only one ``value'' for any given ``input''.

    e.g., father of, best friend, second half, one more than ...

Syntax of FOL

- **Predicates:** P(x[1], ..., x[n])
  - **predicate name**; (x[1], ..., x[n]): **argument list**
  - Examples: human(x),
  - father(x, y)

  A predicate, like a membership function, defines a set (or a class) of objects
- **Terms** (arguments of predicates must be terms)
  - **Constants** are terms (e.g., Fred, a, Z, “red”, etc.)
  - **Variables** are terms (e.g., x, y, z, etc.), a variable is **instantiated** when it is assigned a constant as its value

Examples:

- **Predicates:**
  - parent(x, y), child (x, y), father(x, y), daughter(x, y), etc.
  - spouse(x, y), husband(x, y), wife(x,y)
  - ancestor(x, y), descendent(x, y)
  - relative(x, y)
- **Facts:**
  - husband(Joe, Mary), son(Fred, Joe)
  - spouse(John, Nancy), male(John), son(Mark, Nancy)
  - father(Jack, Nancy), daughter(Linda, Jack)
  - daughter(Liz, Linda)

- A **well-formed formula** (**wff**) is a sentence containing no "free" variables. i.e., all variables are "bound" by universal or existential quantifiers.

  $(\forall x)P(x,y)$ has x bound as a universally quantified variable, but y is free.

- **Quantifiers**

  Universal quantification ∀ (or ***for all***)

  – (∀x)P(x) means that P holds for **all** values of x in the domain associated with that variable.

  – E.g., (∀x) dolphin(x) => mammal(x)

  (∀x) human(x) => mortal(x)

  – Universal quantifiers often used with "implication (=>)“ (∀x) student(x) => smart(x) (All students are smart)

  – Often associated with English words “all”, “everyone”, “always”, etc.

Existential quantification ∃

- (∃x)P(x) means that P holds for **some** value(s) of x in the domain associated with that variable.
- E.g., (∃x) mammal(x) ^ lays-eggs(x)

  (∃x) taller(x, Fred)

- Existential quantifiers usually used with "^ (and)" to specify a list of properties about an individual.

  (∃x) student(x) ^ smart(x) (there is a student who is smart.)
- Often associated with English words "someone", "sometimes", " at least " etc.

Scopes of quantifiers

- Each quantified variable has its scope
  - (∀x)[human(x) => (∃y) [human(y) ^ father(y, x)]
  - All occurrences of x within the scope of the quantified x refer to the same thing.
  - Use different variables for different things
- Switching the order of universal quantifiers does not change the meaning:
  - (∀x)(∀y)P(x,y) <=> (∀**y**)(∀x)P(x,y), can write as (∀x,y)P(x,y)
- Similarly, you can switch the order of existential quantifiers.
  - (∃**x**)(∃y)P(x,y) <=> (∃y)(∃**x**)P(x,y)
- Switching the order of universals and existential does change meaning:
  - Everyone likes someone: (∀x)(∃y)likes(x,y)
  - Someone is liked by everyone: (∃y)(∀x) likes(x,y)

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.

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 jane.
20. Every body has a father.

INFERENCE RULE

- If we want to prove something, we apply some manipulation procedures on the given statements to deduce new statements.
- If we are totally sure that the given statement are true , then the newly derived statements are also true.

1. Modus Ponens
2. Chain Rule
3. Substitution
4. Simplification
5. Transposition
6. Resolution
7. Unification

Modus Pones

- If a has property P and all objects that have property P also have property Q, we conclude that a has property Q.

$$\frac{P(a) \qquad (\forall x)\, P(x) \rightarrow Q(x)}{Q(a)}$$

- Assertion : Leo is a lion
- Implication : All lions are ferocious.
- Conclusion : Leo is ferocious.

Lion(Leo)

$(\forall x)$ Lion (x) $\rightarrow$ ferocious(x)

ferocious(Leo)

Chain Rule

- If P→Q and Q→R then P →R
- Example

Given : (programmer likes LISP)→(programmer hates COBOL)

And : (programmer hates COBOL) → (programmer likes Prolog)

Conclude: (programmer likes LISP)→ (programmer likes Prolog)

Substitution

- If S is a valid statement then S’ is derived from is also valid.
- Example :- if P V ˜P is valid then Q V ˜ Q is also valid.

- Simplification: - P and Q → P

- Transposition :- P → Q

  infer ~ Q → ~ P

Translating English to FOL

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.

1. (∀x) gardener(x) => likes(x, Sun)
2. ~((∀x) gardener(x) => likes(x, Sun))
3. (∃x)(∀t) person(x) ^ time(t) => can-be-fooled(x, t)
4. (∀x)(∃t) person(x) ^ time(t) => can-be-fooled(x ,t)
5. (∃t)(∀x) person(x) ^ time(t) => can-be-fooled(x, t)
6. ~((∀x)(∀t) person(x) ^ time(t) => can-be-fooled(x, t))
7. (∀x) person(x) => younger(x, father(x))

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

1. (∀x) (mushroom(x) ^ purple(x)) => poisonous(x)
2. ~(∃x) purple(x) ^ mushroom(x) ^ poisonous(x)
   (∀x) (mushroom(x) ^ purple(x)) => ~poisonous(x)
3. (∃x)(Ey) mushroom(x) ^ purple(x) ^ mushroom(y) ^ purple(y) ^ ~(x=y) ^
4. ~tall(Clinton)
5. (∀z) (mushroom(z) ^ purple(z)) => ((x=z) v (y=z))
   (∀x)(∀y) above(x,y) <=> (on(x,y) v (∃z) (on(x,z) ^ above(z,y)))
   6. ~ (∃x)(∀y)likes(x,y) or (∀x)(∃y)~likes(x,y)

Unification and Resolution in FOL

UNIFICATION

- A unification of two terms is a join with respect to a specialization order.

Answering Questions by Matching

Data Base of Ground Instances

Person(John)
Person(Mary)
Dog(Fido)
Likes(John,Mary)
Not(Likes (Mary, John))
Likes(Fido, Mary)
Likes(Fido, John)
Likes(Fido, Fido)
Owners(Fido, John, Mary)

Possible Queries

Person(x)
Likes(Mary,x)
Likes(x,x)
Likes(x,y)
Not(Likes(z, John))
Owners(Fido,x,y)

Simple Matching

- Token match on predicates
- Variables only for query arguments→ const-const or const-variable match

Full Unification (variables on both sides)

- **Variable-Variable Matching**

P(x)	P(y)
Q(x,x)	Q(y,z)
P(f(x),z)	P(y, Fido)

- **Unifiers: Variable Substitutions**

P(x)	P(y)	{y/x}
Q(x,x)	Q(y,z)	{y/x, z/x}
P(f(x),z)	P(y,Fido)	{y/f(x), z/Fido}

- **Consistent Variable Assignments**

P(Mary,John)	P(y,y)	#
R(x,y,y)	R(y,y,z)	{y\z, x\y}
W(P(x),y,z)	W(Q(x),y,Fido)	#

- **Advantages of Full Unification**
  - Query and data => both fully allow variables
  - Permits full FOL Resolution (next)

Unification

Q(x) *P(y)*	→ FAIL
P(x) *P(y)*	→ *x/y*
P(Marcus) *P(y)*	→ Marcus/*y*
P(Marcus) *P(Julius)*	→ FAIL
P(x,x) *P(y,y)*	→ *(y/x)*
P(y,z)	→ (*z/y , y/x)*

Finding General Substitutions

- Given:

  Hate(x,y)

  Hate(Marcus,z)

  We Could Produce:

  (*Marcus/x, z/y)*

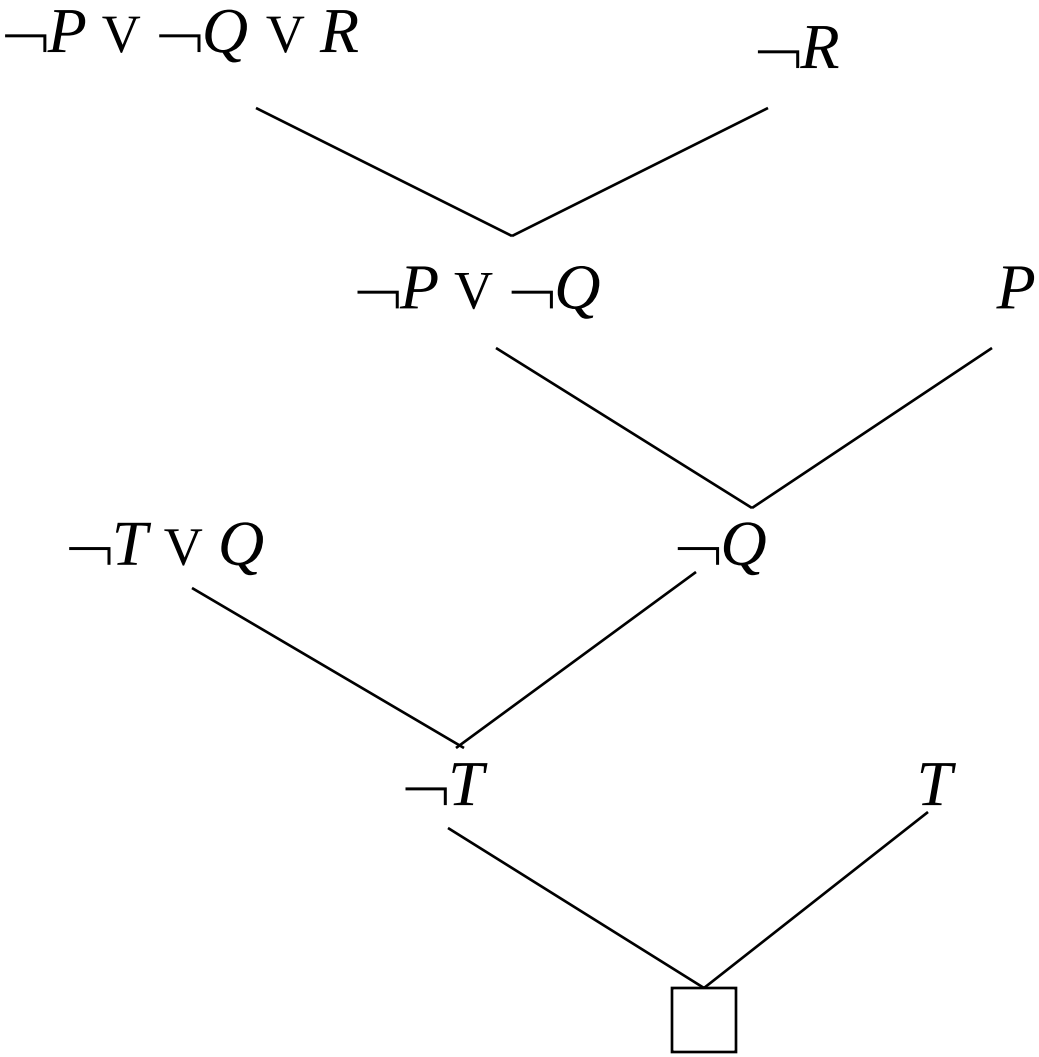
  (Marcus/x, y/z)

  (Marcus/x, Caesar/y, Caesar/z)

  (Marcus/x, Polonius/y, Polonius/z)

RESOLUTION

Resolution in Propositional Logic



A Predicate Logic Example

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.

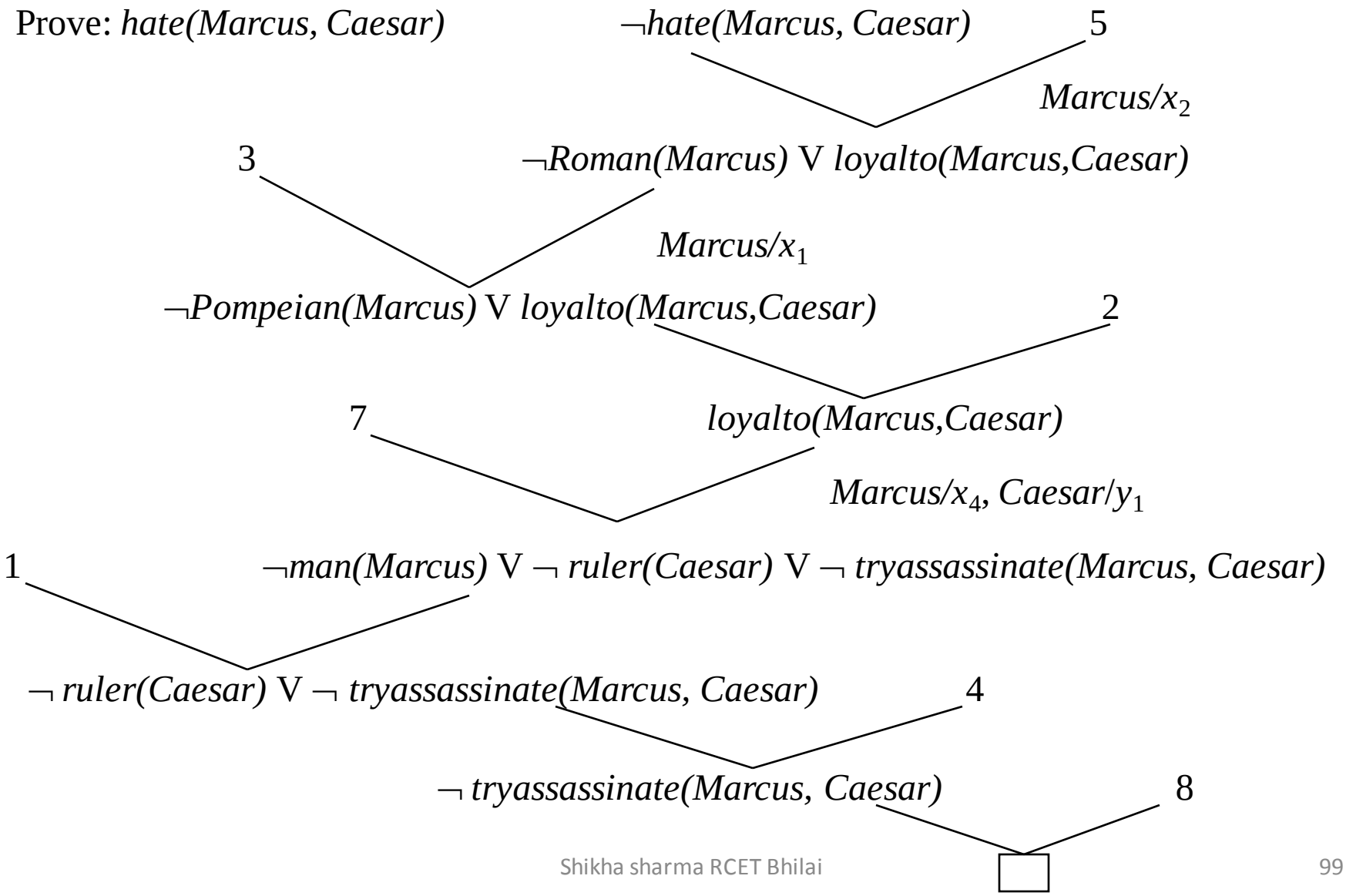
8. Marcus tried to assassinate Caesar.

9. All men are people.

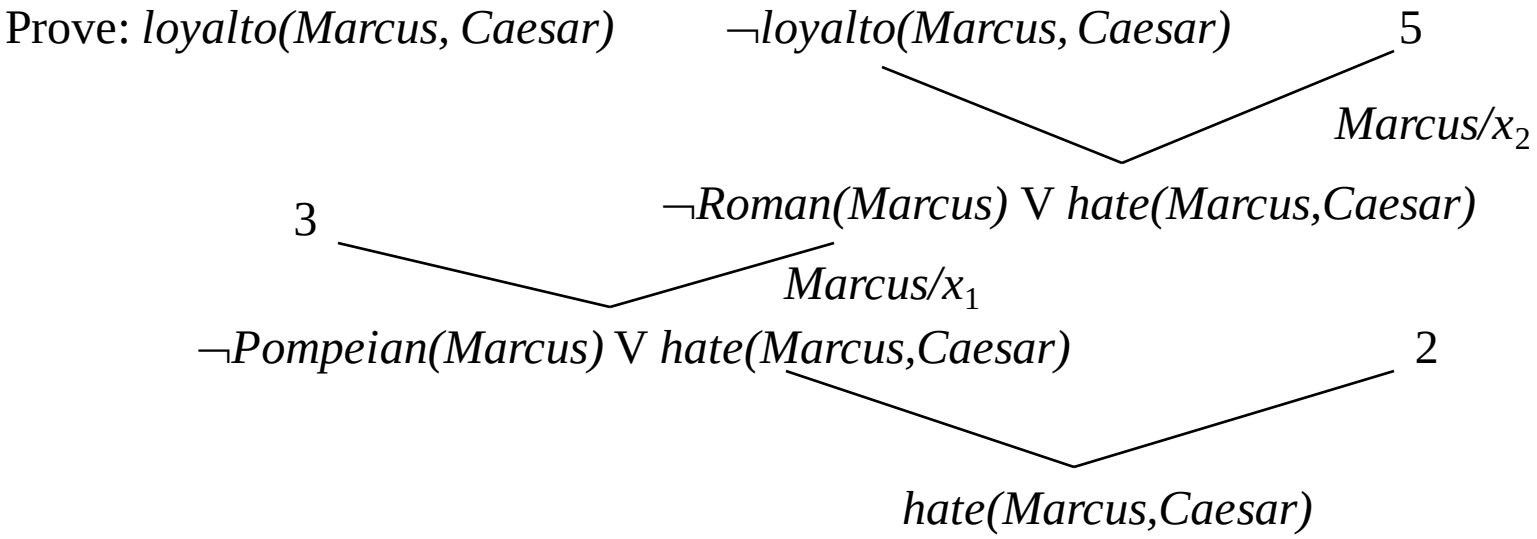
CNF

- Axioms in clause form:
  1. *man(Marcus)*
  2. *Pompeian(Marcus)*
  3. $\neg$ *Pompeian*(x_1) v Roman(x_1)
  4. *Ruler*(*Caesar*)
  5. $\neg$ *Roman*(x_2) v *loyalto*(x_2, *Caesar*) v *hate*(x_2, *Caesar*)
  6. *loyalto*(x_3, x_3)
  7. $\neg$ *person*(x_4) v $\neg$ *ruler*(y_1) v $\neg$ *tryassassinate*(x_4, y_1) v *loyalto* (x_4, y_1)
  8. *tryassassinate*(*Marcus, Caesar*)
9. $\neg$ man(x5) v *person*(x_4)

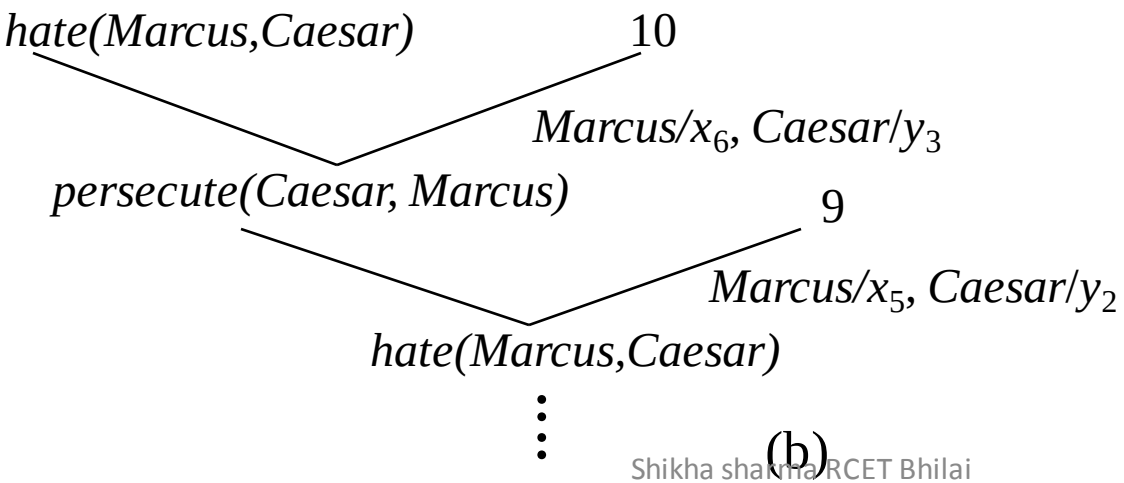
Resolution Proof cont.



An Unsuccessful Attempt at Resolution



(a)



(b)

Conversion to Clause Form

- Problem:

  $\forall x$: [*Roman*(*x*)Λ *know*(*x,Marcus*)]$\rightarrow$ [*hate*(*x,Caesar*)$\vee$ ($\forall y$:$\exists z$: *hate*(*y,z*) $\rightarrow$ *thinkcrazy*(*x,y*))]
- Solution:
  - Flatten
  - Separate out quantifiers
- Conjunctive Normal Form: $\neg$*Roman*(*x*) v $\neg$*know*(*x,Marcus*) v *hate*(*x,Caesar*) v $\neg$ *hate*(*y,z*) v *thinkcrazy*(*x,z*)
- Clause Form
  - Conjunctive normal form
  - No instances of Λ

Algorithm: Convert to Clause Form

1. Eliminate $\rightarrow$, using: $a \rightarrow b = \neg a \vee b$.
2. Reduce the scope of each $\neg$ to a single term, using:
   - $\neg (\neg p) = p$
   - deMorgan's laws: $\neg(a \wedge b) = \neg a \vee \neg b$
     $\neg(a \vee b) = \neg a \wedge \neg b$
   - $\neg \forall x\, P(x) = \exists x \neg P(x)$
   - $\neg \exists x\, P(x) = \forall x \neg P(x)$
3. Standardize variables.
4. Move all quantifiers to the left of the formula without changing their relative order.
5. Eliminate existential quantifiers by inserting Skolem functions.
6. Drop the prefix.
7. Convert the expression into a conjunction of disjuncts, using associativity and distributivity.
8. Create a separate clause for each conjunct.
9. Standardize apart the variables in the set of clauses generated in step 8, using the fact that: $(\forall x: P(x) \wedge Q(x)) = \forall x: P(x) \wedge \forall x: Q(x)$

Skolem Functions in FOL

- **Objective**
  - Want all variables universally quantified
  - Notational variant of FOL w/o existentials
  - Retain implicitly full FOL expressiveness
- **Skolem's Theorem**

  Every existentially quantified variable can be replaced by a unique *Skolem function* whose arguments are all the universally quantified variables on which the existential depends, without changing FOL.
- **Examples**
  - "Everybody likes something"
    $\forall(x) \exists (y)$ [Person(x) & Likes(x,y)]
    $\forall(x)$ [Person(x) & Likes(x, S1(x))]
    Where S1(x) = "that which x likes"
  - "Every philosopher writes at least one book"
    $\forall(x) \exists(y)$[Philosopher(x) & Book(y)) => Write(x,y)]
    $\forall(x)$[(Philosopher(x) & Book(S2(x))) => Write(x,S2(x))]

Examples of Conversion to Clause Form

- Example:

  $\forall x$: [*Roman*(x) Λ *know*(x, *Marcus*)] $\rightarrow$[*hate*(x,*Caesar*) $\vee$ ($\forall y$: $\exists z$: *hate*(y,z) $\rightarrow$ *thinkcrazy*(x,y))]

  – Eliminate $\rightarrow$

    $\forall x$: $\neg$[*Roman*(x) Λ *know*(x, *Marcus*)] $\vee$ [*hate*(x,*Caesar*) $\vee$ ($\forall y$:

    $\neg\exists z$: *hate*(y,z) $\vee$ *thinkcrazy*(x,y))]

  – Reduce scope of $\neg$.

    $\forall x$: [$\neg$*Roman*(x) $\vee$ $\neg$ *know*(x, *Marcus*)] $\vee$ [*hate*(x,*Caesar*) $\vee$ ($\forall y$: $\forall z$: $\neg$*hate*(y,z) $\vee$ *thinkcrazy*(x,y))]

  – “Standardize” variables:

    $\forall x$: *P*(x) $\vee$ $\forall x$: Q(x) converts to $\forall x$: *P*(x) $\vee$ $\forall y$: Q(y)

  – Move quantifiers. $\forall x$: $\forall y$: $\forall z$: [$\neg$*Roman*(x) $\vee$ $\neg$ *know*(x, *Marcus*)] $\vee$ [*hate*(x,*Caesar*) $\vee$ ($\neg$*hate*(y,z) $\vee$ *thinkcrazy*(x,y))]

Examples of Conversion to Clause Form

- Eliminate existential quantifiers.

  $\exists y$: *President*(y) will be converted to *President*($S1$)

  $\forall x$: $\exists y$: *father-of*(y,x) will be converted to $\forall x$: *father-of*($S2(x)$,x))

- Drop the prefix.

  [$\neg$*Roman*(x) $\neg$ *know*(x,*Marcus*)] $\vee$ [*hate*(x, *Caesar*) $\vee$ ($\neg$ *hate*(y,z) $\vee$ *thinkcrazy*(x,y))]

- Convert to a conjunction of disjuncts.

  $\neg$ *Roman*(x) $\vee$ $\neg$ *know*(x,*Marcus*) $\vee$ *hate*(x,*Caesar*) $\vee$ $\neg$ *hate*(y,z) $\vee$ *thinkcrazy*(x,y)

Algorithm: Propositional Resolution

1. Convert all the propositions of *F* to clause form.
2. Negate *P* and convert the result to clause form. Add it to the set of clauses obtained in step 1.
3. Repeat until either a contradiction is found or no progress can be made:

   a) Select two clauses. Call these the parent clauses.

   b) Resolve them together. The *resolvent* will be the disjunction of all of the literals of both of the parent clauses with the following exception: If there are any pairs of literals *L* and $\neg L$ such that one of the parent clauses contains *L* and the other contains $\neg L$, then select one such pair and eliminate both *L* and $\neg L$ from the resolvent.

   c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

A Few Facts in Propositional Logic

Given Axioms	**Clause Form**	
P	P	(1)
$(P \wedge Q) \rightarrow R$	$\neg P \vee \neg Q \vee R$	(2)
$(S \vee T) \rightarrow Q$	$\neg S \vee Q$	(3)
	$\neg T \vee Q$	(4)
T	T	(5)

Algorithm: Resolution

1. Convert all the propositions of *F* to clause form.
2. Negate *P* and convert the result to clause form. Add it to the set of clauses obtained in 1.
3. Repeat until either a contradiction is found, no progress can be made, or a predetermined amount of effort has been expended.
   a) Select two clauses. Call these the parent clauses.
   b) Resolve them together. The resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitutions performed and with the following exception: If there is one pair of literals $T1$ and $\neg T2$ such that one of the parent clauses contains $T1$ and the other contains $\neg T2$ and if $T1$ and $T2$ are unifiable, then neither $T1$ nor $\neg T2$ should appear in the resolvent. If there is more than one pair of complementary literals, only one pair should be omitted from the resolvent.
   c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

Using Resolution with Equality and Reduce

- Axioms in clause form:

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $Born(Marcus, 40)$
4. $\neg man(x_1) \vee mortal(x_1)$
5. $\neg Pompeian(x_2) \vee died(x_2, 79)$
6. $erupted(volcano, 79)$
7. $\neg mortal(x_3) \vee \neg born(x_3, t_1) \vee \neg gt(t_2 — t_1, 150) \vee dead(x_3, t_2)$
8. $Now=2002$
9. $\neg alive(x_4, t_3) \vee \neg dead(x_4, t_3)$
10. $\neg dead(x_5, t_4) \vee alive(x_5, t_4)$
11. $\neg died(x_6, t_5) \vee \neg gt(x_6, t_5) \vee dead(x_6, t_6)$

Prove: $\neg alive(Marcus, now)$

Issues with Resolution

- Requires full formal representation in FOL (for conversion to clause form)
- Resolution defines a search space (which clauses will be resolved against which others define the operators in the space) → search method required
- Worst case: resolution is exponential in the number of clauses to resolve. Actual: exponential in average resolvable set (= branching factor)
- Can we define heuristics to guide search for BestFS, or A* or B*? (Not in the general case)

More on definitions

- A **definition** of P(x) by S(x)), denoted $(\forall x)$ P(x) <=> S(x), can be decomposed into two parts
  - Necessary description: "P(x) => S(x)" (**only if**, for P(x) being true, S(x) is necessarily true)
  - Sufficient description "P(x) <= S(x)" (**if**, S(x) being true is sufficient to make P(x) true)
- Examples: define father(x, y) by parent(x, y) and male(x)
  - parent(x, y) is a necessary (**but not sufficient**) description of father(x, y)

    father(x, y) => parent(x, y), parent(x, y) => father(x, y)
  - parent(x, y) ^ male(x) is a necessary and sufficient description of father(x, y)

    parent(x, y) ^ male(x) <=> father(x, y)
  - parent(x, y) ^ male(x) ^ age(x, 35) is a sufficient (**but not necessary**) description of father(x, y) because

    father(x, y) => parent(x, y) ^ male(x) ^ age(x, 35)

1. Every woman that can be burnt is a witch.

2 Everything that is made of wood can be burnt.

3 Everything that oats is made of wood.

4 Everything that weighs the same as something that oats, does oat too.

5 This girl is a woman.

6 This girl weighs the same as this duck.

7 This duck oats.

? Is the girl a witch?

1(all x)(Burns(x) ^Woman(x)) Witch(x)).

2 (all x)(Is made of wood(x)) Burns(x)).

3 (all x)(Floats(x)) Is made of wood(x)).

4 (all x)(8y)((Floats (x) ^ Sameweight(x; y))) Floats(y)).

5 Woman(Girl).

6 Same weight(Duck; Girl).

7 Floats(Duck).

? Witch(Girl)

:1Burns(x1) _ :Woman(x1) _ Witch(x1).
2 :Ismadeofwood(x2) _ Burns(x2).
3 :Floats(x3) _ Ismadeofwood(x3).
4 :Floats(x4) _ :Sameweight(x4; y1) _ Floats(y1).
5 Woman(Girl).
6 Sameweight(Duck; Girl).
7 Floats(Duck).

We shall insert in KB the complement of the proposition that we want to prove. If the initial proposition is true according to our KB then we will end up in a refutation.

? :Witch(Girl)

First order logic

- Advantages of using logic
  - No need to make a difference between knowledge representation and the inference method
  - Soundness (a false statement can not be derived) and completeness (all true statements can be derived)
  - Has a logical make-up
- Disadvantage of using logic
  - The derivation takes a lot of effort
  - Difficult to use different layers of representation

Logic programming

- Logic programming
  - New programming paradigm
  - Viewing the set of clauses *K* as a program
- PROLOG
  - Using a set of true first order logic clauses as base
  - Has the advantage of knowledge presentation in logic
  - Program “What” instead of “How” (like in C)

Logic programming

- Horn clauses
  - *First order logic*: a clause is a set of positive and negative literals (atoms and atom negation)
  - *Horn clauses*: a maximum of one positive literal per clause
  - Using Horn clauses decreases expressiveness but considerably improves efficiency

Logic programming

- Fact → unit clause
  - *Formal description*: Atom
  - *Example*: Raven(Veety). Likes(Jim, Betty).
- Rule → definite clause
  - *Formal description*: Atom :- $Atom_1$, …, $Atom_n$
  - A :- B_1, B_2, …, B_n
    (When all the *body* conditions B_i are true, then the *head* A is true)
  - *Example*: Black(x) :- Raven(x).
    Mother(x,y) :- Parent(x,y), Female(x).
- Query → goal clause
  - *Formal description*: :- $Atom_1$, …, $Atom_n$
  - :- B_1, B_2, …, B_n (Are the goals B_1, B_2, …, B_n true?)
  - *Example*: :- Black(Veety).
    :- Father(x, Jim).

Logic programming

- Horn clauses
  - Number of atoms in the head is 0 or 1
  - Number of atoms in the body is 0 or more
  - *Empty clause*: both head and body have 0 clauses
- Procedural meaning
  - The procedural call of the program
- Declarative meaning
  - Logical meaning

Logic programming

Horn clause		Procedural meaning	Declarative meaning
fact	A.	Definition of procedure A	A is TRUE
rule	A :- B_1, B_2, …, B_n	For invoking the procedure A, the procedures B_1, B_2, …, B_n have to be called in order	IF B_1, B_2, …, B_n THEN A
query	:- B_1, B_2, …, B_n	Start the calculation	Negation of B_1, B_2, …, B_n
empty clause	□	Termination of the calculation	Contradiction

Logic programming

- PROLOG uses proof by contradiction
  - *SLD resolution*: unification of atoms in the body of goal clause with the heads of unit clauses or definite clauses
  - *Unification*: manipulating two predicates to make them appear the same
  - If there are no candidates for unification, backtrack to the previous goal and try a different unification candidate
  - If the empty clause can be derived, the program terminates successfully