Chapter 5
Structured knowledge representation

Basanta Joshi, PhD

basanta@ioe.edu.np

Lecture notes can be downloaded from

www.basantajoshi.com.np

Knowledge Definition

- "The fact or condition of *knowing something* with familiarity gained through experience or association." (Webster' s Dictionary, 1988)(Knowing something via seeing, hearing, touching, feeling, and tasting.)
- "The fact or condition of *being aware of* something" . (Ex. Sun is hot, balls are round, sky is blue,…)

Knowledge Storing

Natural language for people

- ➢Symbols for computer: a number or character string that represents an object or idea (Internal representation of the knowledge).
- ➢The core concepts: mapping from facts to an internal computer representation and also to a form that people can understand.

Building a Knowledge Base (KB)

- KB is designed for grouping the various knowledge together in one place. The KB is the central repository of information containing the facts we know about objects and their relationships.
- ***Knowledge engineering*** (knowledge acquisition): mapping the set of knowledge in a particular problem domain and converting it into a knowledge base.
- ***Domain expert***: who through years of experience, has gathered the knowledge about how things work and relate to one another, and know how to solve problems in his or her specialty.

Building a Knowledge Base (KB)

Knowledge engineer: who can take that domain knowledge and represent it in a form for use by the reasoning system. As an intermediary between the human expert and the expert system, the knowledge engineer must have good people skills as well as good technical skills.

A combination of questionnaires, interviews, and first-hand observations are used to give the knowledge engineer the deep understanding required to transform the expert’s knowledge into facts and rules for the knowledge base.

Building a Knowledge Base (KB)

- Neural networks could be trained to perform classification and prediction tasks without going through the expensive knowledge acquisition process.
- Neural networks may not be easily converted to a symbolic form, they most definitely are a knowledge base, because they encode the knowledge implicit in the training data.

Knowledge Representation

- Simple facts or complex relationships
- Mathematical formulas or rules for natural language syntax
- Associations between related concepts
- Inheritance hierarchies between classes of objects
- Knowledge is not a "one-size-fits-all" proposition.

Properties for Knowledge Representation Systems

- The following properties should be possessed by a knowledge representation system.
- **Representational Adequacy**
- -- the ability to represent the required knowledge;
- **Inferential Adequacy**
- - the ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original;
- **Inferential Efficiency**
- - the ability to direct the inferential mechanisms into the most productive directions by storing appropriate guides;
- **Acquisitional Efficiency**
- - the ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.
- To date no single system optimises all of the above

Knowledge Representation Methods

Effective knowledge representation methods:

- Easy to use.
- Easily modified and extended (changing the knowledge manually or through automatic machine learning techniques).

Knowledge Representation Methods

- Procedural method
- Declarative method
- Relational method
- Hierarchical method
- Complex network graph

Procedural representation

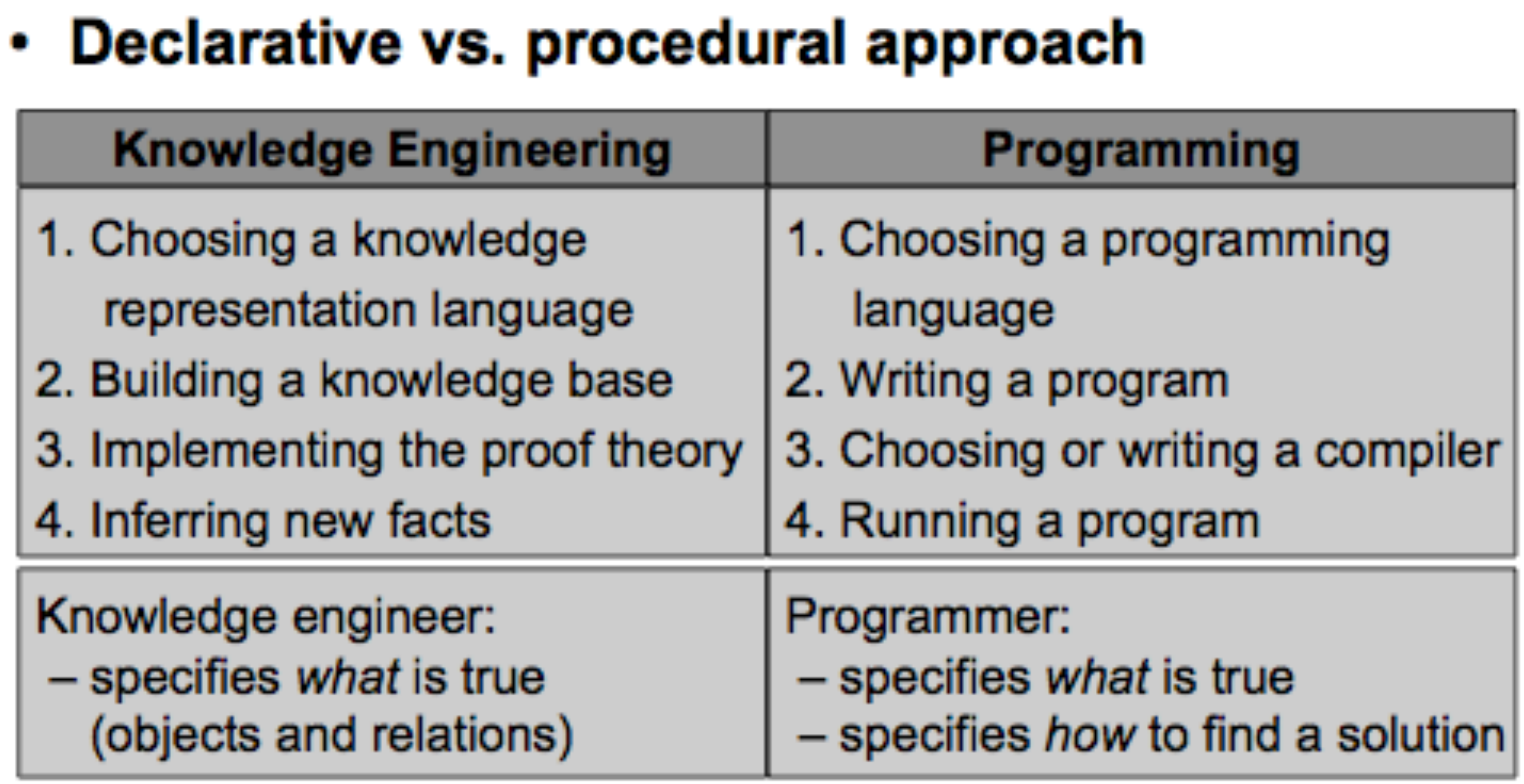
- Encode facts and define the sequence of operations step by step. (Hardcoded logic)
- The weakness: the knowledge and the manipulation of that knowledge are inextricably linked.

Declarative method

Overcoming the weakness of procedural representation

- The states, facts, rules, and relationships are separated declared.
- The separation of knowledge from the algorithm used to manipulate or reason with that knowledge provides advantages over procedural codes.
- Because the knowledge is explicitly represented, it can be more easily modified.
- Separating the control logic and reasoning algorithms from the knowledge allows us to write optimized and reusable inferencing procedures.

Procedural vs Declarative method



- **Declarative vs. procedural approach**

Knowledge Engineering	**Programming**
1. Choosing a knowledge representation language 2. Building a knowledge base 3. Implementing the proof theory 4. Inferring new facts	1. Choosing a programming language 2. Writing a program 3. Choosing or writing a compiler 4. Running a program
Knowledge engineer: – specifies *what* is true (objects and relations)	Programmer: – specifies *what* is true – specifies *how* to find a solution

Hierarchical representation

Used to represent inheritable knowledge.

- **Inheritable knowledge**: it centers on relationships and shared attributes between kinds or classes of objects.
- **Hierarchical knowledge** is best used to represent "**isa**" relationships, where a **general or abstract type** (ex, ball) is linked to more **specific types** (rubber, golf, baseball, football) which *inherit the basic properties of the general type.*

Hierarchical representation

- The strength of object inheritance allows for compact representation of knowledge and allows reasoning algorithm to process at different levels of abstraction or granularity.
- The use of categories or types gives structure to the world by grouping similar objects together.
- Using categories or clusters simplifies reasoning by limiting the number of distinct things we have to deal with. (reduce complexity)

Hierarchical representation - Capturing knowledge

- Knowing what to expect based on the elapsed time from one event to another is often the hallmark of *intelligent behavior*.
- Time concepts such as *before, after, and during* are crucial to common-sense reasoning and planning. ***Temporal logic*** is usually used to represent and reason about time.

Predicate Logic

The use of *formal logic* as a primary knowledge representation harkens back to the beginnings of AI research.

Mathematical deduction based on logic, was a well-known method of *generating new knowledge* from existing knowledge.

Predicate Logic

- **Formal logic** is a language with its own syntax, which defines how to make sentences, and corresponding semantics, which describe the meaning of the sentences.
- Sentences can be constructed using proposition symbols (P, Q, R) and Boolean connectives, such as conjunction (And), disjunction (Or), implication (P implies Q).
- **Ex**: if P and Q then R, the preceding rule, P and Q, is called the *premise or antecedent*, and R is the *conclusion or consequent*.

Predicate Logic

Modus Ponens: where given a rule, ***A implies B***, if A is true, we can infer that B is also true.

Predicate logic introduces the concept of quantifiers, which allow us to refer to sets of objects. *Using objects, attributes, and relations, we can represent almost any type of knowledge*.

Two quantifiers: $\forall$: universal (all objects of this type have this attribute), $\exists$: existential (there exists some object that has the specified attribute)

Predicate Logic

Ex: “Minnesota is cold in the winter” can be represented in three single parameter:

Place (Minnesota), Temperature(cold) and Season(winter).

Or it can be represented a single relation:

Cold (Minnesota, winter), Winter(Minnesota, cold).

Resolution

Resolution is an algorithm for proving facts true or false by virtue of contradiction. If we want to prove a theorem X is true, we have to show that the negation of X is not true.

Example of Resolution

Suppose that we know the following two facts:

1.not feathers(Tweety) or bird(Tweety)

2.feathers(Tweety)

Sentence 1 states that either Tweety does not have feathers or else Tweety is a bird. Sentence 2 states that Tweety has feathers. To prove that Tweety is a bird, we first add an assumption that is the negation of that predicate, giving sentence 3:

3.not bird(tweety)

Example of Resolution

In sentence 1 and 2, not feathers(Tweety) and feathers(Tweety) cancel each other out. Resolving sentences 1 and 2 produces the resolvant, sentence 4, which is added to our fact set:

4. Bird(Tweety)

It is clear that sentences 3 and 4 cannot both be true, either Tweety is a bird ot it is not. Thus, we have a contradition. WE have just proved that our first assumption, *not bird(Tweety)*, is false, and the alternative, *bird(Tweety)*, must be true (Winston, 1993).

Resolution

If the clauses to be resolved are selected in systematic ways, then resolution is guaranteed to find a contradiction if one exists, although it may take a long time to find.

Unification

Unification is a technique for taking two sentences in predicate logic and finding a substitution that makes them look the same.

- A variable can be replaced by a constant.
- A variable can be replaced by another variable.
- A variable can be replaced with a predicate, as long as the predicate does not contain that variable.

Example of unification

Given the following set of predicates, let's explore how they can be unified:

- hates(X,Y)
- hates(George, broccoli)
- hates(Alex, spinach)

We could unify sentence 2 with 1 by binding George to variable X, and broccoli to variable Y. Similarly, we could bind Alex to X and spinach to Y. Note that if the predicate names were different, we could not unify these predicates.

Example of unification

If we introduce a few more predicates, we can explore more complex unifications:

4.hates(X, vegetable(Y))

5.hates(George, vegetable(Y))

6.hates(Z, broccoli)

We could unify sentence 6 with sentence 1 by replacing variable X with variable Z and variable Y with the constant broccoli. Sentence 4 and 5 could be unified with George bound to X, and broccoli to variable Y.

Unification using in Prolog

A generalized version of the unification algorithm, called *match*, is used in Prolog (Programming in logic).

Facts are represented in Prolog by clauses, which look like standard predicates, and declare things which are unconditionally true.

Rules are clauses where the conclusion may be true, provided that all of the clauses in the condition part are true.

Unification using in Prolog

Prolog provides a built-in inferencing procedure, based on resolution, for processing rules and answering questions posed as goal clauses (Bratko, 1986).

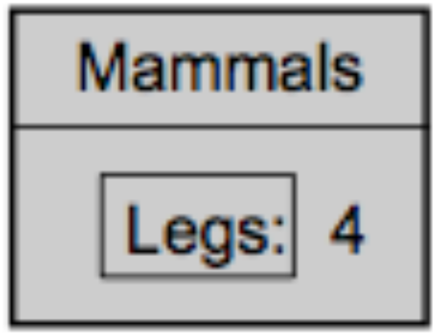
Nowadays, most commercial implementations of rule-based systems are written in c and c++.

A "GUI" for logic

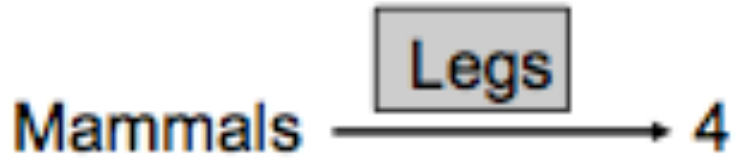
- **Knowledge representation languages**
  - Mathematical representations, i.e. logic
  - Graphical representations
    - Existential graphs (Peirce, 1896), then semantic networks
    - Frame systems (Minsky)
  - *Different syntaxes but same semantics and proof theory*

    (e.g. that of first-order logic)
    - e.g.
      Logic: $\forall x\ Mammal(x) \Rightarrow Legs(x,4)$ or $Rel(Legs,Mammals,4)$

Frame:



Semantic network:



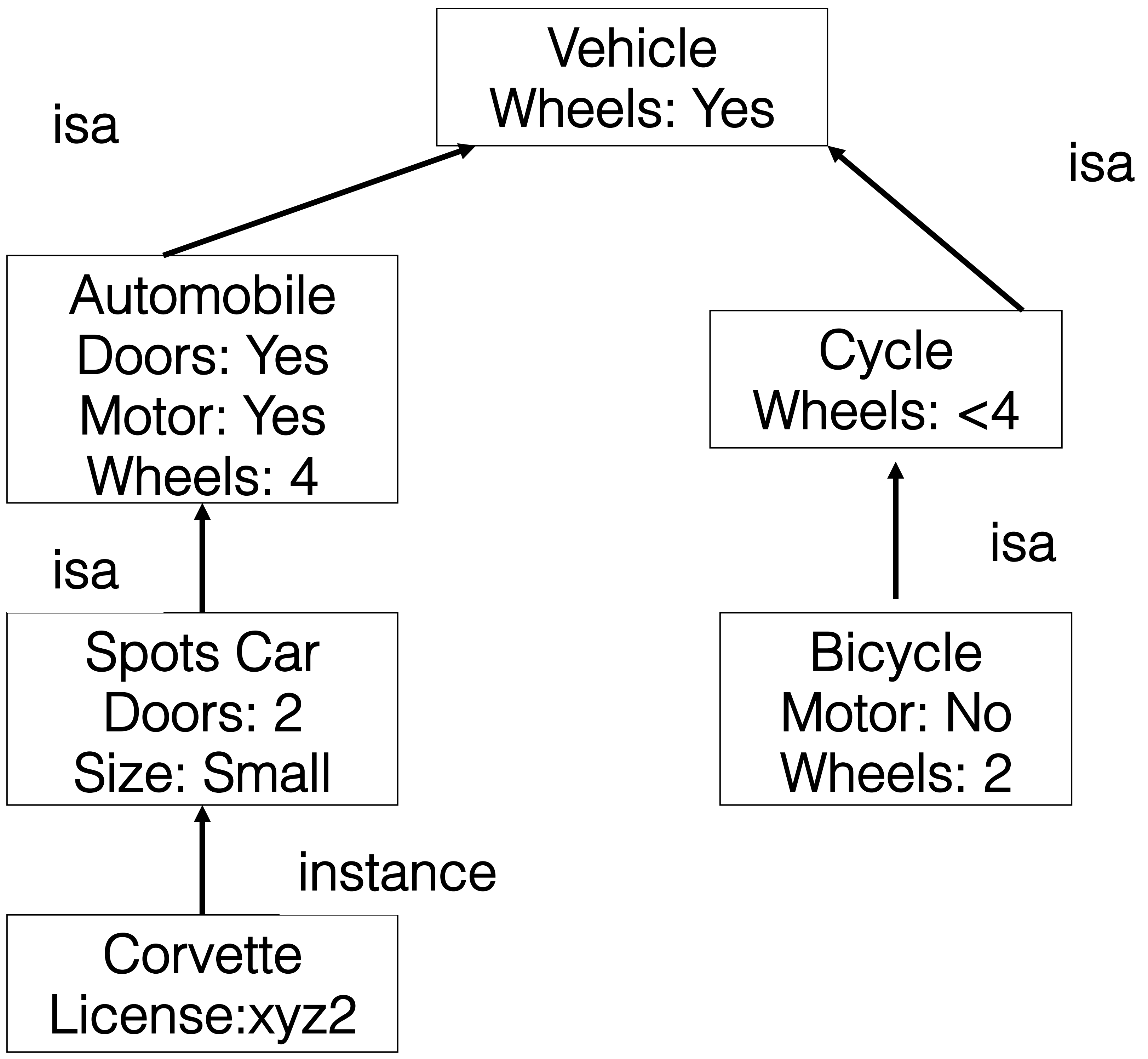
Issues in Knowledge Representation

- There are several issues that must be considered when representing various kinds of real-world knowledge.
  - Important Attributes
  - Relationship among Attributes
  - Inverses
  - Existence in an Isa hierarchy
  - Technique for reasoning about values
  - Single-valued attributes

Frames

- A frame is a collection of attributes, which defines the state of an object and its relationship to other frames (objects). But a frame is much more than just a record or data structure containing data.
- In AI, frames are called slot-filler data representations. The slots are the data values, and the fillers are attached procedures which are called before, during (to compute the value of), or after the slot's value is changed.
- Frames are often linked into a hierarchy to represent has-part ans isa relationships.

Example of Frames



Frame v.s. to OOP

- A Frame sounds very much like an object,
  - whose data members are the slots, and
  - whose methods are the attached procedures or daemons.
- In some sense, any Java program is a frame-based mechanism for knowledge representation.
- It makes use of inheritance for isa relationships, and containment or references for has-part relationships.

Semantic Nets

- Semantic nets are used to define the meaning of a concept by its relationships to other concepts.
- A graph data structure is used, with nodes used to hold concepts, and links with natural language labels used to show the relatjonships.
- A modern implementation of a semantic net is the Knowledge Utility (KnU) developed by IBM. (http://www.ibm.aqui.com)

Semantic Nets

- **Syntax and semantics**
  - Focus on categories of and relations between objects
    - e.g. "cats are mammals":

$\text{Cats} \xrightarrow{\textit{subset}} \text{Mammals}$

in logic: $\text{Cats} \subset \text{Mammals}$

$\forall x\ \text{Cat}(x) \Rightarrow \text{Mammal}(x)$

"Felix is a cat":

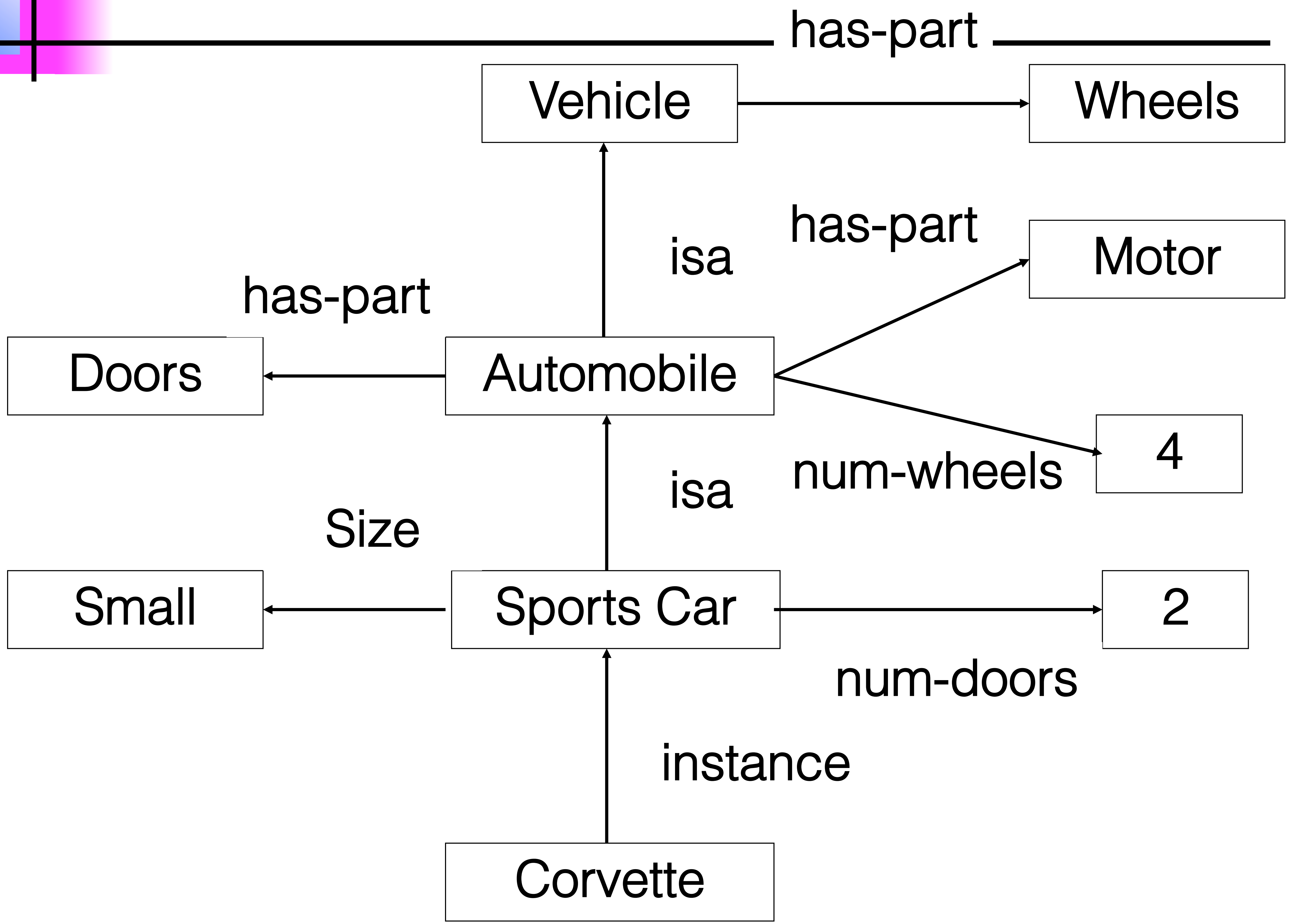
Cat(Felix).

$\text{Felix} \xrightarrow{\textit{member}} \text{Cats}$

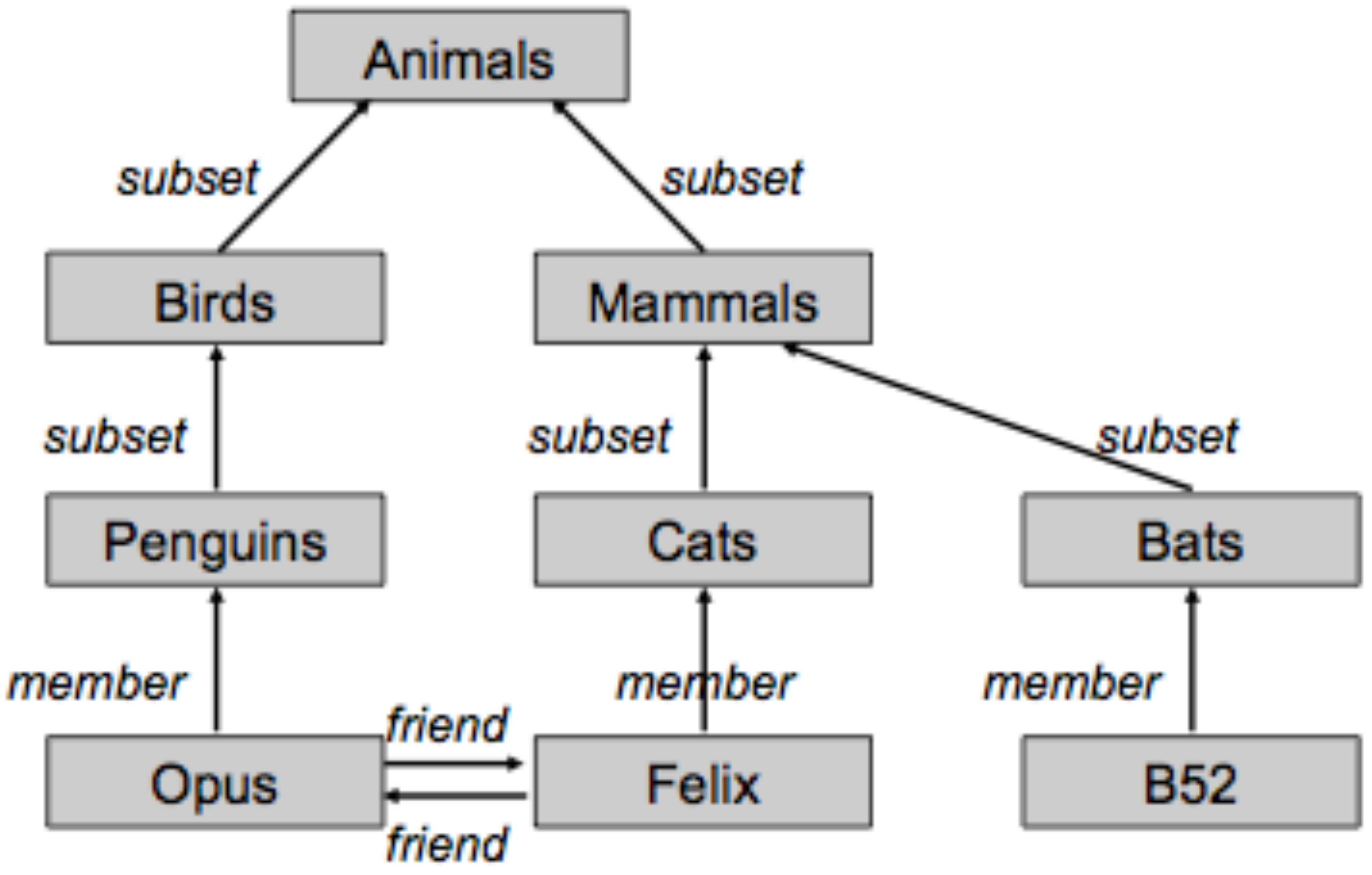
note: "a cat *IsA* mammal"

- **Proof theory**
  - "Link following" inference strategy *yields* *<u>inheritance</u>*
    - e.g. $\text{Felix} \rightarrow \text{Cat} \rightarrow \text{Mammal}$ infers $\text{Felix} \xrightarrow{\textit{member}} \text{Mammal}$

Example of Semantic Net

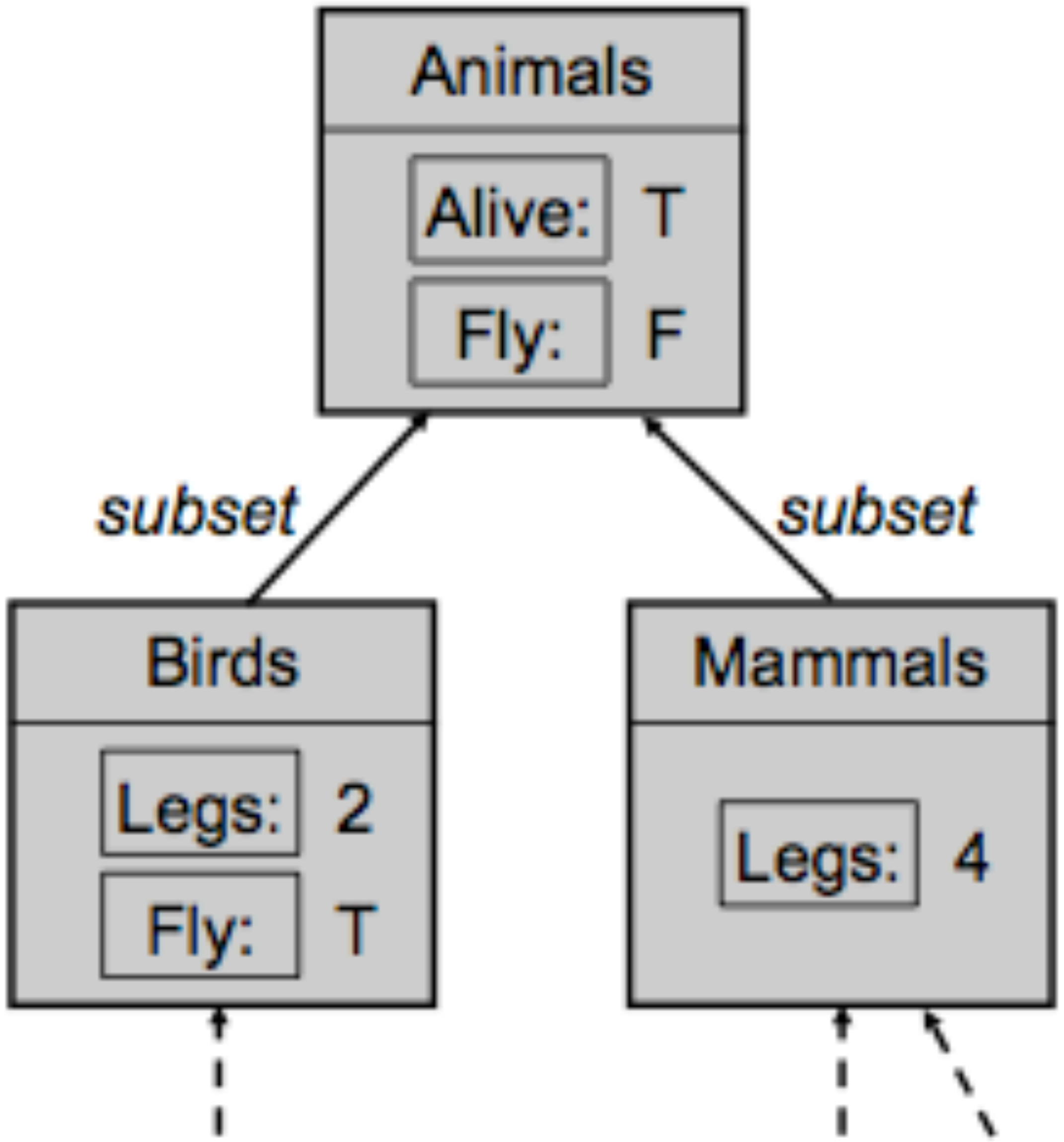


Example of Semantic Network / Frame-based Representation



Details of the Knowledge Base

Frame-based representation **First-order logic equivalent**



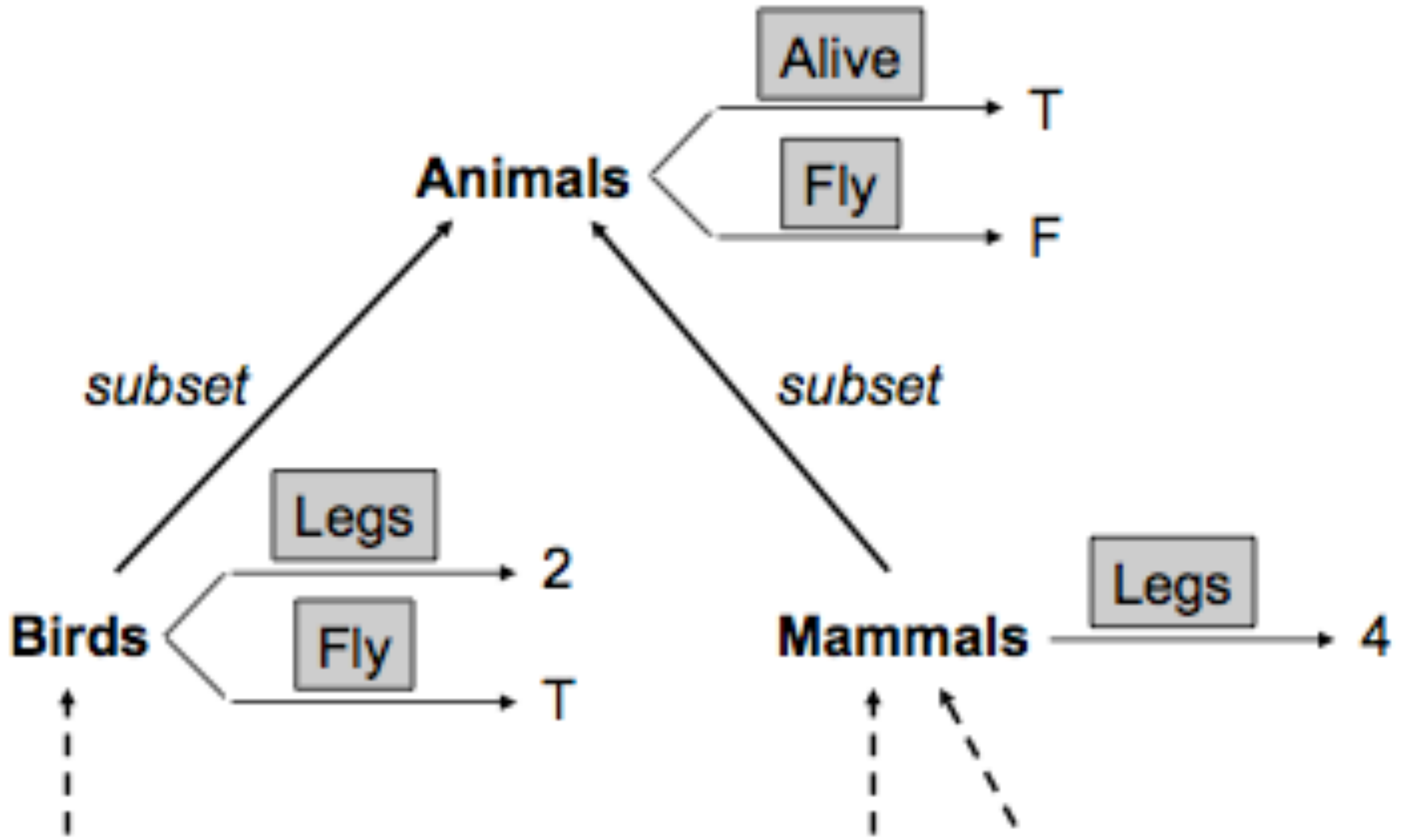
Rel(Alive,Animals,T)
Rel(Fly,Animals,F)

Subset(Birds,Animals)
Subset(Mammals,Animals)

Rel(Fly,Birds,T)
Rel(Legs,Birds,2)
Rel(Legs,Mammals,4)

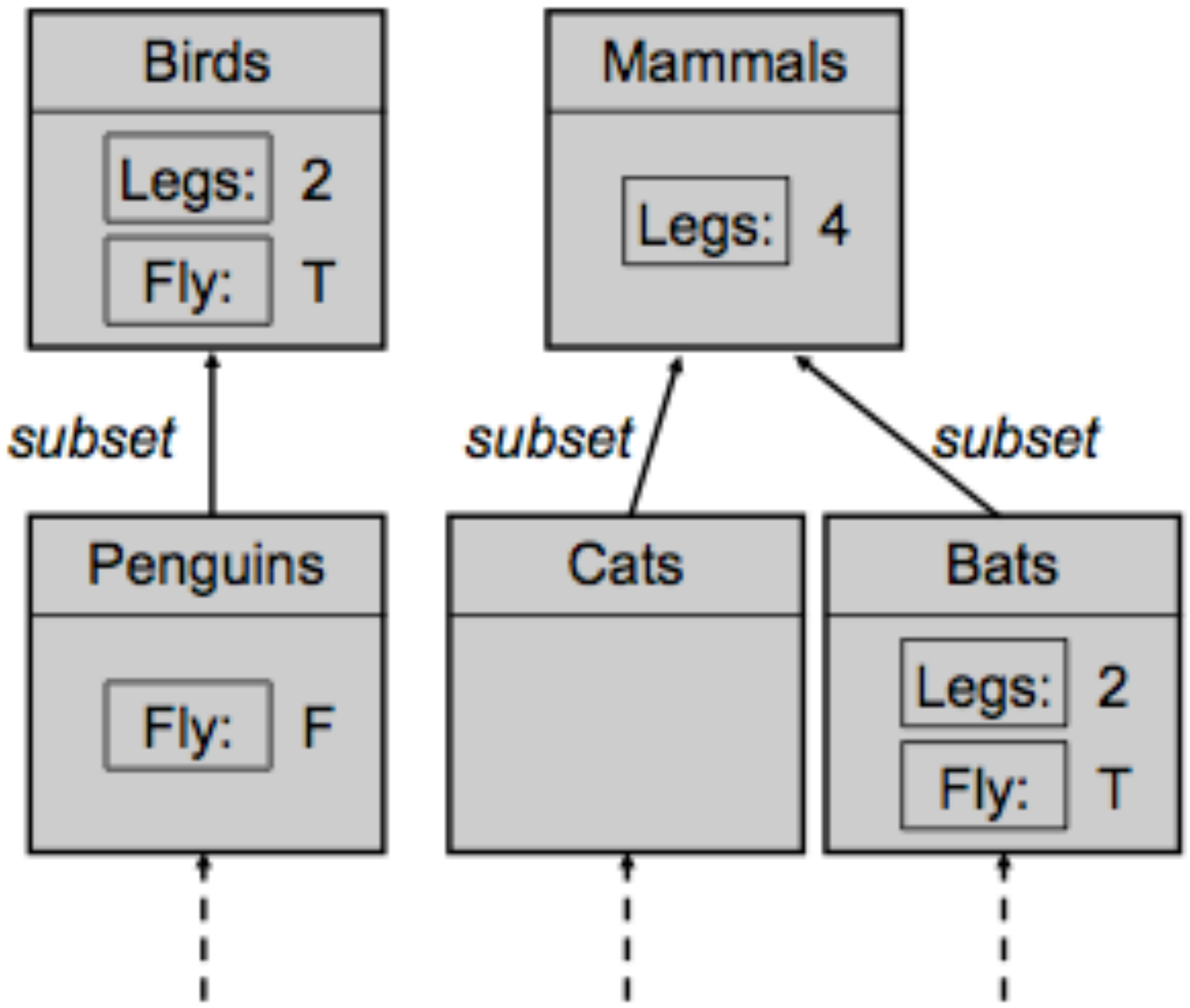
Details of the Knowledge Base

Semantic network (equivalent) representation



Details of the Knowledge Base

Frame-based representation **First-order logic equivalent**

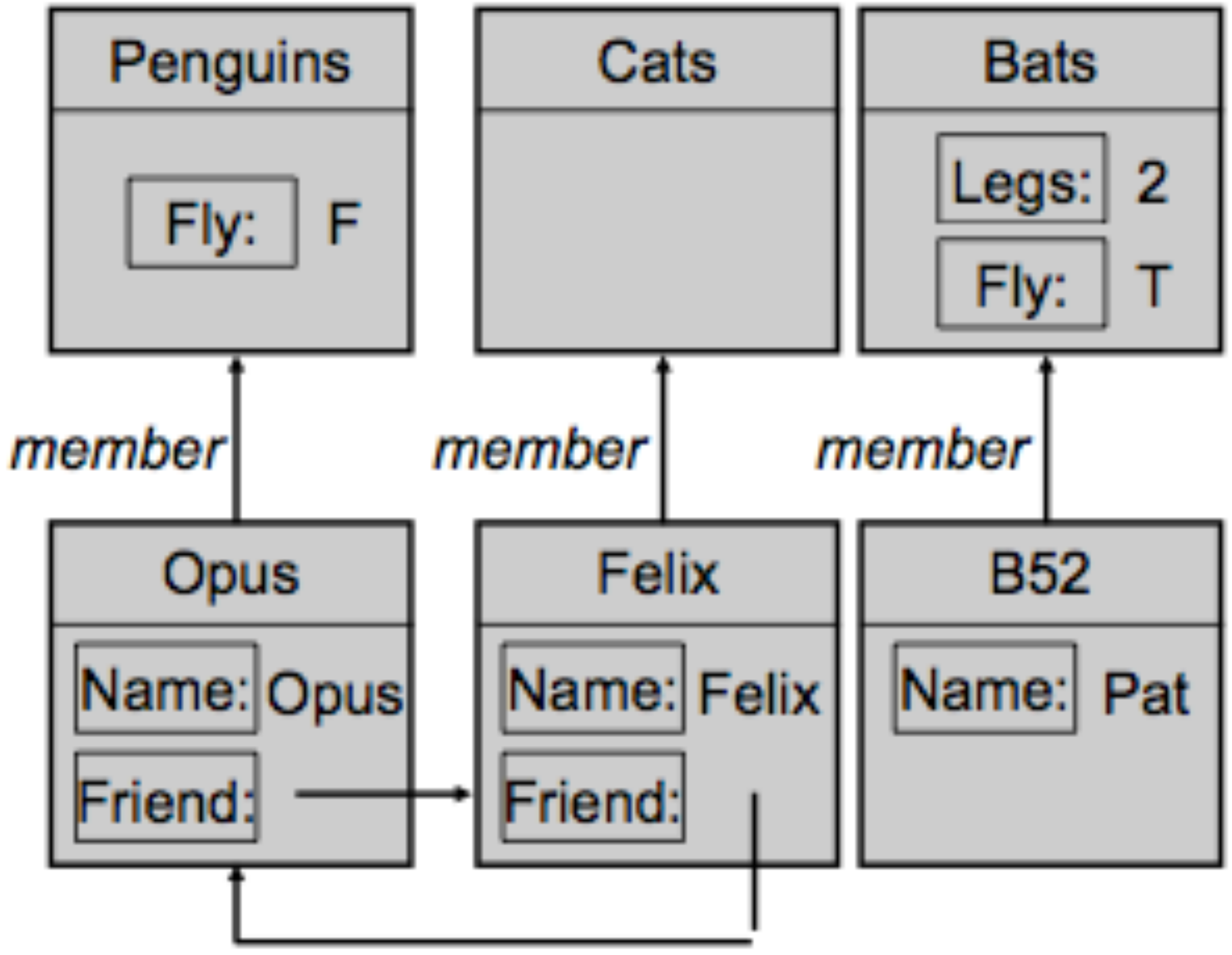


Subset(Penguins,Birds)
Subset(Cats,Mammals)
Subset(Bats,Mammals)

Rel(Fly,Penguins,F)
Rel(Legs,Bats,2)
Rel(Fly,Bats,T)

Details of the Knowledge Base

Frame-based representation **First-order logic equivalent**



Member(Opus,Penguins)
Member(Felix,Cats)
Member(Pat,Bats)

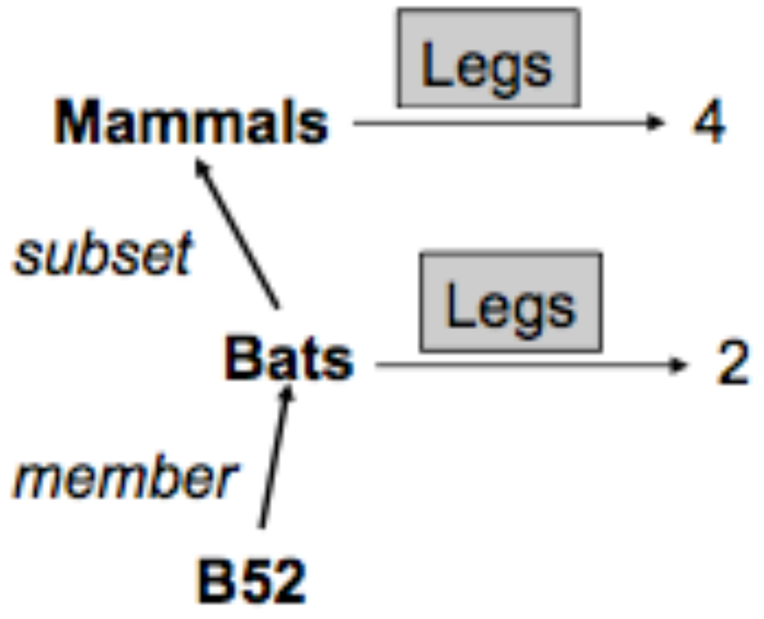
Name(Opus,Opus)
Friend(Opus,Felix)

Name(Felix,Felix)
Friend(Felix,Opus)

Name(B52,Pat)

Features of Semantic Networks as Knowledge Representations

- **Advantages**
  - Easier to understand (graphical representation)
  - Less strict (less formal), e.g. allows exceptions
  - Faster, special purpose inference ("following links")
- **Disadvantages**
  - Semantics not always clear
    implementation dependent
    - e.g. how many legs for B52?



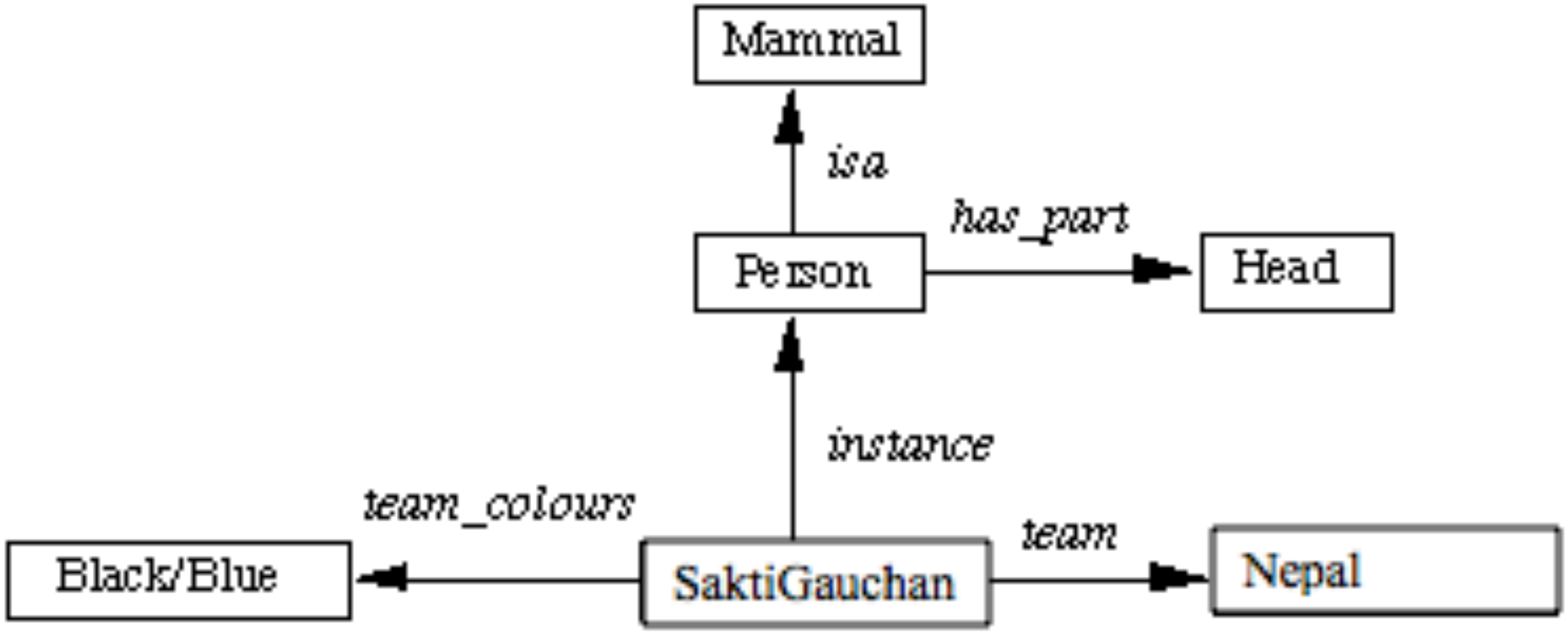
Link Types in Semantic Network

Link Type	Semantics	Example
$A \xrightarrow{Subset} B$	$A \subset B$	$\text{Cats} \subset \text{Mammals}$
$A \xrightarrow{Member} B$	$A \in B$	$\text{Felix} \in \text{Cats}$
$A \xrightarrow{R} B$	$R(A,B)$	$\text{Felix} \xrightarrow{Friend} \text{Opus}$
$A \xrightarrow{\boxed{R}} B$	$\forall x \; x \in A \Rightarrow R(x,B)$	$\text{Birds} \xrightarrow{\boxed{Legs}} 2$
$A \xrightarrow{\boxed{\boxed{R}}} B$	$\forall x \; x \in A \Rightarrow \exists y \; y \in B \wedge R(x,y)$	$\text{Bird} \xrightarrow{\boxed{\boxed{Parent}}} \text{Birds}$

Problem:

- Convert the given fopl in semantic Net:
- MAMMAL (person)
- INSTANCE (person, SAKTIGAUCHAN)
- HASPART (perosn, Nose)
- TEAM (NEPAL, SAKTIGAUCHAN)
- UNIFORMCOLOR (WHITE, SAKTI GAUCHAN)

Solution



Mammal
isa
has_part
Person
Head
instance
team_colours
team
Black/Blue
SaktiGauchan
Nepal

Conceptual Dependency (CD)

- CD theory was developed by Schank in 1973 to 1975 to represent the meaning of NL sentences.
  - It helps in drawing inferences
  - It is independent of the language
- CD representation of a sentence is not built using words in the sentence rather built using conceptual primitives which give the intended meanings of words.
- CD provides **structures** and specific **set of primitives** from which representation can be built.

Primitive Acts of CD theory

ATRANS	Transfer of an abstract relationship (i.e. give)
PTRANS	Transfer of the physical location of an object (e.g., go)
PROPEL	Application of physical force to an object (e.g. push)
MOVE	Movement of a body part by its owner (e.g. kick)
GRASP	Grasping of an object by an action (e.g. throw)
INGEST	Ingesting of an object by an animal (e.g. eat)
EXPEL	Expulsion of something from the body of an animal (e.g. cry)
MTRANS	Transfer of mental information (e.g. tell)
MBUILD	Building new information out of old (e.g decide)
SPEAK	Producing of sounds (e.g. say)
ATTEND	Focusing of a sense organ toward a stimulus (e.g. listen)

Conceptual category

- There are four conceptual categories

– ACT	Actions {one of the CD primitives}
– PP	Objects {picture producers}
– AA	Modifiers of actions {action aiders}
– PA	Modifiers of PP’s {picture aiders}

Example

- I gave a book to the man. CD representation is as follows:

$$\begin{tikzcd}
& & & \text{man (to)} \\
I \arrow[r, Leftrightarrow, "P"] & \text{ATRANS} & \text{book} \arrow[l, "O"'] & \arrow[l, "R"'] \arrow[u] \\
& & & \text{I (from)} \arrow[u]
\end{tikzcd}$$

- It should be noted that this representation is same for different saying with same meaning. For example
  - I gave the man a book,
  - The man got book from me,
  - The book was given to man by me etc.

Few conventions

- Arrows indicate directions of dependency
- Double arrow indicates two way link between actor and action.

  O – for the object case relation

  R – for the recipient case relation

  P – for past tense

  D - destination

Some of Conceptualizations of CD

- Dependency structures are themselves conceptualization and can serve as components of larger dependency structures.
- The dependencies among conceptualization correspond to semantic relations among the underlying concepts.
- We will list the most important ones allowed by CD.
- Remaining can be seen from the book.

Problems with CD Representation

- It is difficult to
  - construct original sentence from its corresponding CD representation.
  - CD representation can be used as a general model for knowledge representation, because this theory is based on representation of events as well as all the information related to events.
- Rules are to be carefully designed for each primitive action in order to obtain semantically correct interpretation.

Contd…

- Many verbs may fall under different primitive ACTs, and it becomes difficult to find correct primitive in the given context.
- The CD representation becomes complex requiring lot of storage for many simple actions.
- For example, the sentence “John bet Mike that Indian cricket team will win incoming world cup” will require huge CD structure.

Script Structure

- Scripts were introduced by Schank and Abelson introduced in 1977 that used CD framework.
- The scripts are useful in describing certain stereotyped situations such as going to theater
- It consists of set of slots containing default values along with some information about the type of values similar to frames.
- It differs from FS as the values of the slots in scripts must be ordered and have more specialized roles.
- In real world situations, we see that event tends to occur in known patterns because of clausal relationship to the occurrence of events

Script Components

- Each script contains the following main components.
  - *Entry Conditions:* Must be satisfied before events in the script can occur.
  - ***Results:*** Conditions that will be true after events in script occur.
  - *Props:* Slots representing objects involved in the events.
  - ***Roles:*** Persons involved in the events.
  - *Track:* Specific variation on more general pattern in the script. Different tracks may share many components of the same script but not all.
  - *Scenes:* The sequence of *events* that occur. Events are represented in conceptual dependency form.

Script : Play in theater	Various Scenes
Track: Play in Theater **Props:** • Tickets • Seat • Play **Roles:** • Person (who wants to see a play) – P • Ticket distributor – TD • Ticket checker – TC **Entry Conditions:** • P wants to see a play • P has a money **Results:** • P saw a play • P has less money • P is happy (optional if he liked the play)	***Scene 1: Going to theater*** • P PTRANS P into theater • P ATTEND eyes to ticket counter ***Scene 2: Buying ticket*** • P PTRANS P to ticket counter • P MTRANS (need a ticket) to TD • TD ATRANS ticket to P ***Scene 3: Going inside hall of theater and sitting on a seat*** • P PTRANS P into Hall of theater • TC ATTEND eyes on ticket POSS_by P • TC MTRANS (showed seat) to P • P PTRANS P to seat • P MOVES P to sitting position ***Scene 4: Watching a play*** • P ATTEND eyes on play • P MBUILD (good moments) from play ***Scene5: Exiting*** • P PTRANS P out of Hall and theater

Script Invocation

- It must be activated based on its significance.
- If the topic is important, then the script should be opened.
- If a topic is just mentioned, then a pointer to that script could be held.
- For example, given "John enjoyed the play in theater", a script "Play in Theater" suggested above is invoked.
- All implicit questions can be answered correctly.
- Here the significance of this script is high.
  - Did john go to theater?
  - Did he buy ticket?
  - Did he have money?
- If we have a sentence like "John went to theater to pick his daughter", then invoking this script will lead to many wrong answers.
  - Here significance of the script theater is less.
- Getting significance from the story is not straightforward. However, some heuristics can be applied to get the value.

Advantages / Disadvantages of Script

- Advantages
  - Capable of predicting implicit events
  - Single coherent interpretation may be build up from a collection of observations.
- Disadvantage
  - More specific (inflexible) and less general than frames.
  - Not suitable to represent all kinds of knowledge.
- To deal with inflexibility, smaller modules called memory organization packets (MOP) can be combined in a way that is appropriate for the situation.