Logical Reasoning Systems

Chapter 10

Some material adopted from notes
by Tim Finin,
Andreas Geyer-Schulz
and Chuck Dyer

Introduction

- Real knowledge representation and reasoning systems come in several major varieties.
- They all based on FOL but departing from it in different ways
- These differ in their intended use, degree of formal semantics, expressive power, practical considerations, features, limitations, etc.
- Some major families of reasoning systems are
  - Theorem provers
  - Logic programming languages
  - Rule-based or production systems
  - Semantic networks
  - Frame-based representation languages
  - Databases (deductive, relational, object-oriented, etc.)
  - Constraint reasoning systems
  - Truth maintenance systems
  - Description logics

Production Systems (forward-chaining)

- The notion of a “production system” was invented in 1943 by Post to describe re-write rules for symbol strings
- Used as the basis for many rule-based expert systems
- Most widely used KB formulation in practice
- A production is a rule of the form:

C1, C2, … Cn => A1 A2 …Am

Left hand side (LHS)
Conditions/antecedents
Condition which must hold before the rule can be applied

Right hand side (RHS)
Conclusion/consequence
Actions to be performed or conclusions to be drawn when the rule is applied

Three Basic Components of PS

- **Rule Base**
  - Unordered set of user-defined "if-then" rules.
  - Form of rules: *if P1* ^ ... ^ *Pm then A1*, ..., *An*
  - the *Pis* are conditions (often facts) that determine when rule is applicable.
  - Actions can add or delete facts from the Working Memory.
  - Example rule (in CLIPS format)

```
(defrule determine-gas-level
    (working-state engine does-not-start)
    (rotation-state engine rotates)
    (maintenance-state engine recent)
     => (assert (repair "Add gas.")))
```

- **Working Memory (WM) --** A set of "facts“, represented as literals, defining what's known to be true about the world
  - Often in the form of “flat tuples” (similar to predicates), e.g., (age Fred 45)
  - WM initially contains case specific data (not those facts that are always true in the world)
  - Inference may add/delete fact from WM
  - WM will be cleared when a case is finished
- **Inference Engine --** Procedure for inferring changes (additions and deletions) to Working Memory.
  - Can be both forward and backward chaining
  - Usually a cycle of three phases: match, conflict resolution, and action, (in that order)

Basic Inference Procedure

While changes are made to Working Memory do:

- **Match** the current WM with the rule-base
  - Construct the Conflict Set **--** the set of all possible (*rule*, *facts*) pairs where rule is from the rule-base, facts from WM that unify with the conditional part (i.e., LHS) of the rule.
- **Conflict Resolution**: Instead of trying all applicable rules in the Conflict set, select one from the Conflict Set for execution. (**depth-first**)
- **Act/fire:** Execute the actions associated with the conclusion part of the selected rule, after making variable substitutions determined by unification during match phase
- **Stop** when conflict resolution fails to returns any (rule, facts) pair

Conflict Resolution Strategies

- **Refraction**
  - A rule can only be used once with the same set of facts in WM. This strategy prevents firing a single rule with the same facts over and over again (avoiding loops)
- **Recency**
  - Use rules that match the facts that were added most recently to WM, providing a kind of "focus of attention" strategy.
- **Specificity**
  - Use the most specific rule,
  - If one rule's LHS is a superset of the LHS of a second rule, then the first one is more specific
  - If one rule's LHS implies the LHS of a second rule, then the first one is more specific
- **Explicit priorities**
  - E.g., select rules by their pre-defined order/priority
- **Precedence of strategies**

- **Example**
  - R1: P(x) => Q(x); R2: Q(y) => S(y); WM = {P(a), P(b)}

    conflict set: {(R1, P(a)), (R1, P(b))}

    by rule order: apply R1 on P(a); WM = {*Q(a)*, P(a), P(b)}

    conflict set: {(R2, Q(a)), (R1, P(a)), (R1, P(b))}

    by recency: apply R2 on Q(a) WM = {*S(a)*, Q(a), P(a), P(b)}

    conflict set: {(R2, Q(a)), (R1, P(a)), (R1, P(b))}

    by refraction, apply R1 on P(b): WM = {*Q(b)*, S(a), Q(a), P(a), P(b)}

    conflict set: {(R2, Q(b)), (R2, Q(a)), (R1, P(a)), (R1, P(b))}

    by recency, apply R2 on P(b): WM = {*S(b)*, Q(b), S(a), Q(a), P(a), P(b)}
  - Specificity

    R1: bird(x) => fly(x) WM={bird(tweedy), penguin(tweedy)}

    R2: penguin(z) => bird(z)

    R3: penguin(y) => ~fly(y)

    R3 is more specific than R1 because according to R2, penguin(x) implies bird(x)

Default Reasoning

- Reasoning that draws a plausible inference on the basis of *less than conclusive* evidence in the *absence* of information to the contrary
  - If WM = {bird(tweedy)}, then by default, we can conclude that fly(tweedy)
  - When also know that penguin(tweedy), then we should change the conclusion to ~fly(tweedy)
  - Bird(x) => fly(x) is a default rule (true in general, in most cases, almost)
  - Default reasoning is thus non-monotonic
  - Formal study of default reasons: *default logic* (Reiter), *nonmonotonic logic* (McDermott), *circumscription* (McCarthy)

    one conclusion: default reasoning is totally undecidable
  - Production system can handle simple default reasoning
    - By specificity: default rules are less specific
    - By rule priority: put default rules at the bottom of the rule base
    - Retract default conclusion (e.g., fly(tweedy)) is complicated

Other Issues

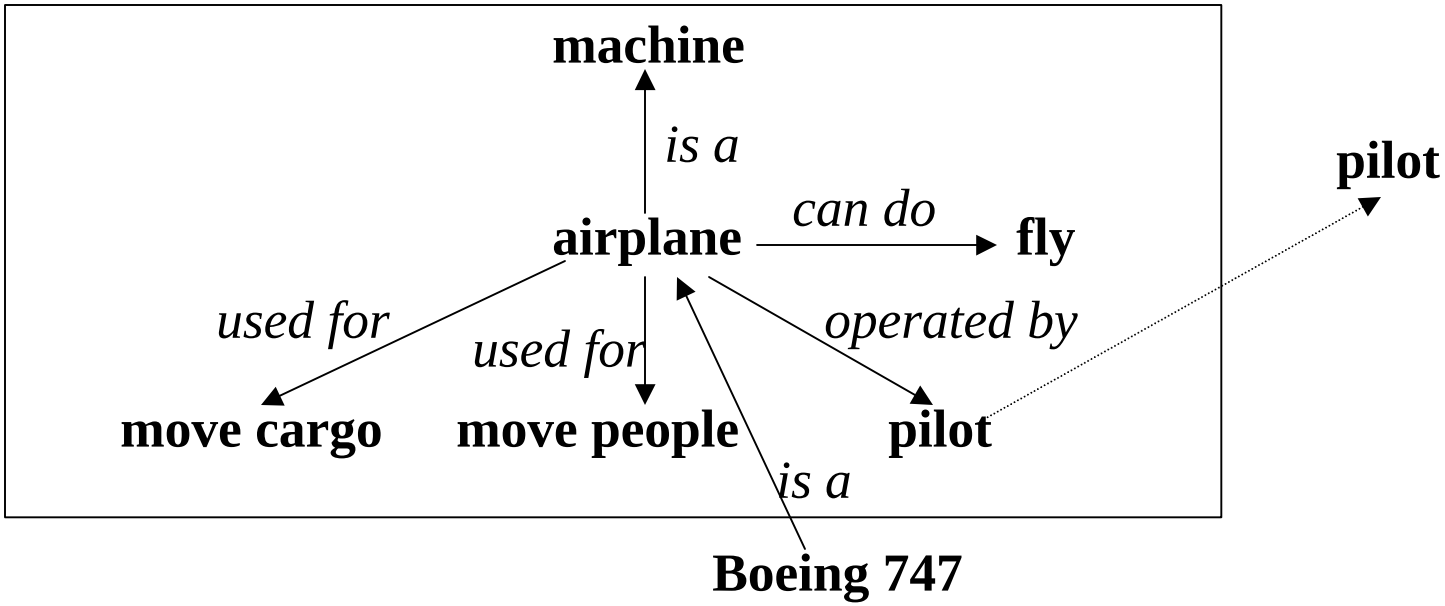
- PS can work in backward chaining mode
  - Match RHS with the goal statement to generate subgoals
  - Mycin: an expert system for diagnosing blood infectious diseases
- Expert system sell
  - A rule-based system with empty rule base
  - Contains data structure, inference procedures, AND user interface to help encode domain knowledge
  - Emycin (backward chaining) from Stanford U
  - OPP5 (forward chaining) from CMU and its descendents CLIPS, Jess.
- Metarules
  - Rules about rules
  - Specify under what conditions a set of rules can or cannot apply
  - For large, complex PS
- Consistency check of the rule-base is crucial (as in FOL)
- Uncertainty in PS (to be discussed later)

Comparing PS and FOL

- Advantages
  - Simplicity (both KR language and inference),
  - Inference more efficient
  - Modularity of knowledge (rules are considered, to a degree, independent of each other), easy to maintain and update
  - Similar to the way humans express their knowledge in many domains
  - Can handle simple default reasoning
- Disadvantages
  - No clearly defined semantics (may derive incorrect conclusions)
  - Inference is not complete (mainly due to the depth-first procedure)
  - Inference is sensitive to rule order, which may have unpredictable side effects
  - Less expressive (may not be suitable to some applications)
- ***No explicit structure among pieces of knowledge in BOTH FOL*** *(a un-ordered set of clauses)* ***and PS*** *(a list of rules)*

Semantic Networks

- Structured representations (semantic networks and frame systems)
  - Put structures into KB (capture the interrelations between pieces of knowledge
  - Center around object/classes
  - More for what it is than what to do
- History of semantics networks (Quillian, 1968)
  - To represent semantics of natural language words by dictionary-like definitions in a graphic form
  - Defining the meaning of a word in terms of its relations with other words
  - Semantic networks were very popular in the 60's and 70's and enjoy a much more limited use today.
  - The **graphical depiction** associated with a semantic network is a big reason for their popularity.



- Nodes for words
- Directed links for relations/associations between words
- Each link has its own meaning
- You know the meaning (semantics) of a word if you know the meaning of all nodes that are used to define the word and the meaning of the associated links
- Otherwise, follow the links to the definitions of related words

Semantic Networks

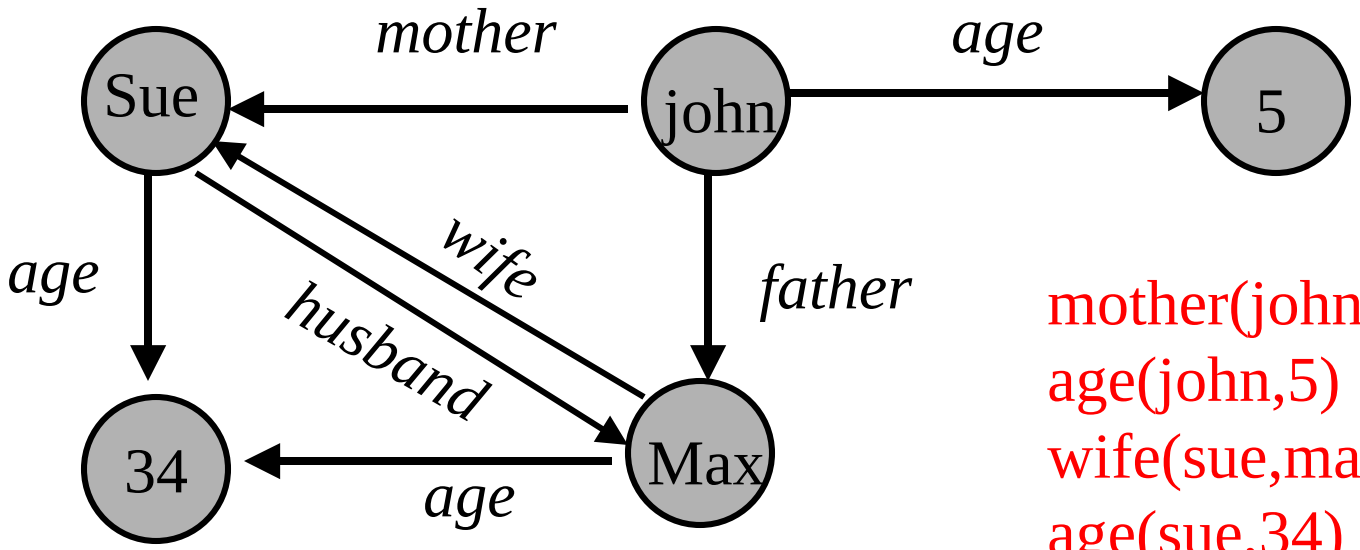
- A semantic (or associative) network is a simple representation scheme which uses a graph of *labeled* nodes and *labeled*, *directed* arcs to encode knowledge.
  - Labeled nodes: objects/classes/concepts.
  - Labeled links: relations/associations between nodes
  - Labels define the semantics of nodes and links
  - Large # of node labels (there are many distinct objects/classes)

    Small # of link labels (types of associations can be merged into a few)

    buy, sale, give, steal, confiscation, etc., can all be represented as a single relation of "*transfer ownership*" between recipient and donor
  - Usually used to represent static, taxonomic, concept dictionaries
- Semantic networks are typically used with a special set of accessing procedures which perform "reasoning"
  - e.g., inheritance of values and relationships
- often much less expressive than other KR formalisms

Nodes and Arcs

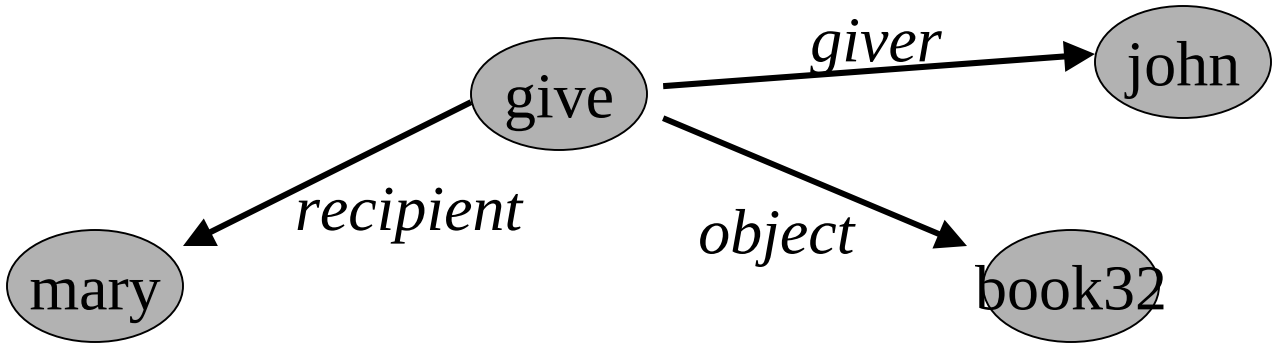
- Nodes denote objects/classes
- arcs define binary relationships between objects.



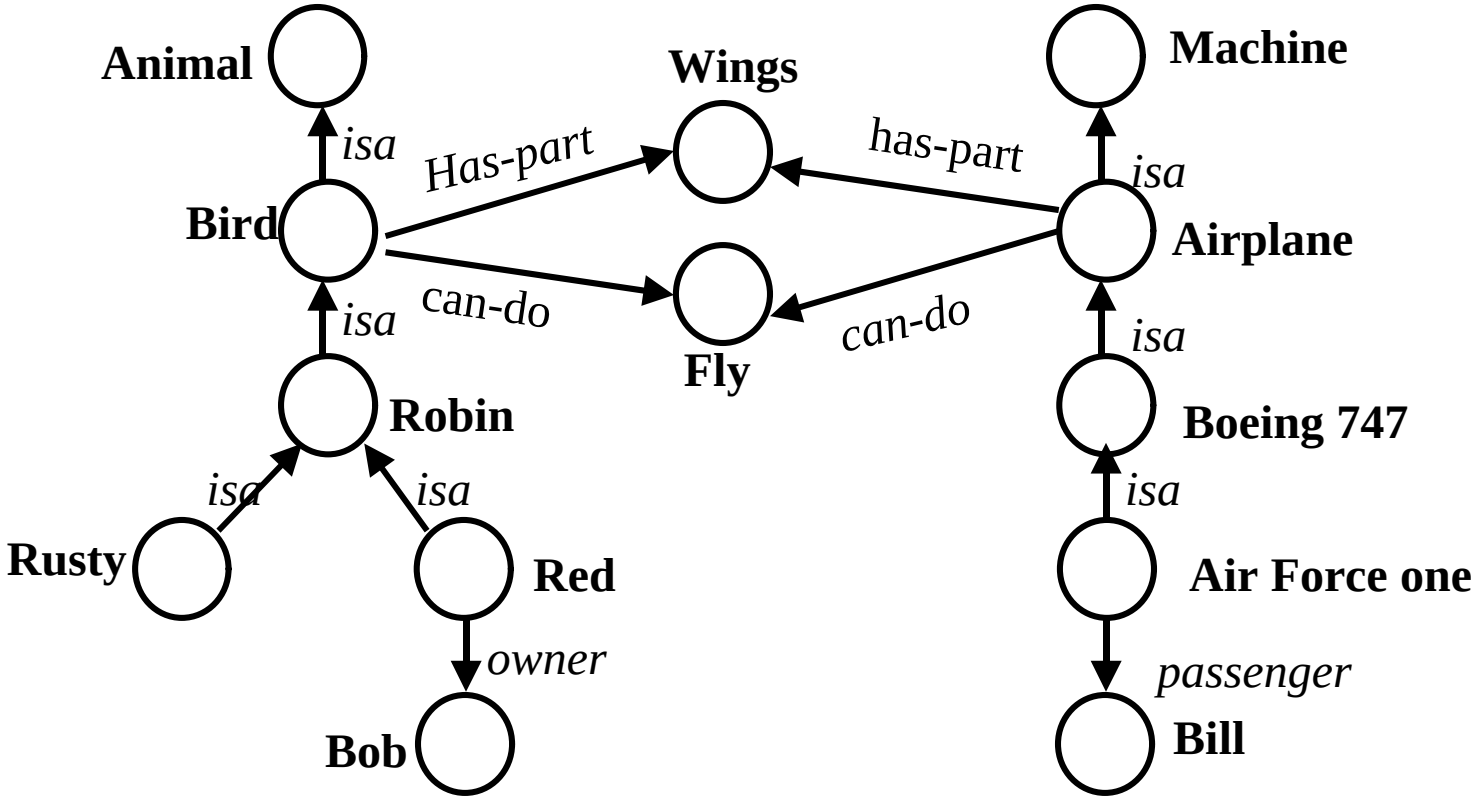
mother(john,sue)
age(john,5)
wife(sue,max)
age(sue,34)
...

Reification

- Non-binary relationships can be represented by “turning the relationship into an object”
- This is an example of what logicians call “reification”
  - reify v : consider an abstract concept to be real
- We might want to represent the generic “give” event as a relation involving three things: a giver, a recipient and an object, give(john, mary, book32)



Inference by association



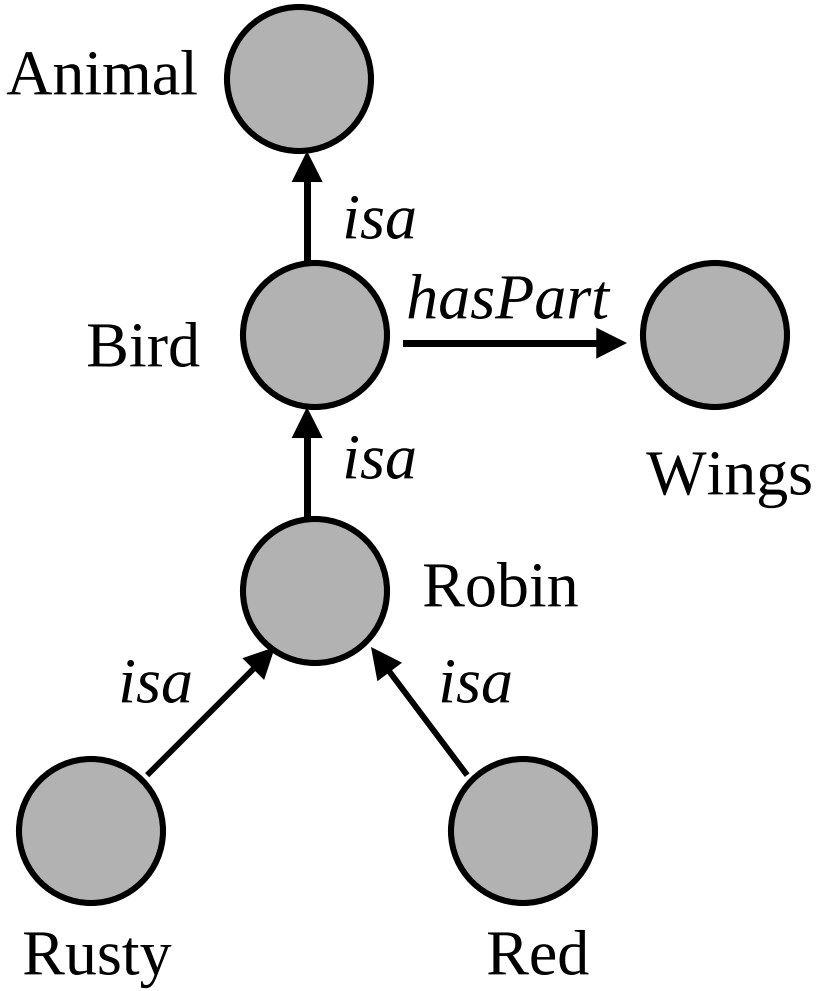
- Red (a robin) is related to Air Force One by association (as directed path originated from these two nodes join at nodes Wings and Fly)
- Bob and Bill are not related (no paths originated from them join in this network

Inferring Associations

- Marker passing
  - Each node has an unique marker
  - When a node is activated (from outside), it sends copies of its marker to all of its neighbors (following its outgoing links)
  - Any nodes receiving a marker sends copies of that marker to its neighbors
  - If two different markers arrive at the same node, then it is concluded that the owners of the two markers are associated
- Spreading activation
  - Instead of passing labeled markers, a node sends labeled activations (a numerical value), divided among its neighbors by some weighting scheme
  - A node usually consumes some amount of activation it receives before passing it to others
  - The amount of activation received by a node is a measure of the strength of its association with the originator of that activation
  - The spreading activation process will die out after certain radius

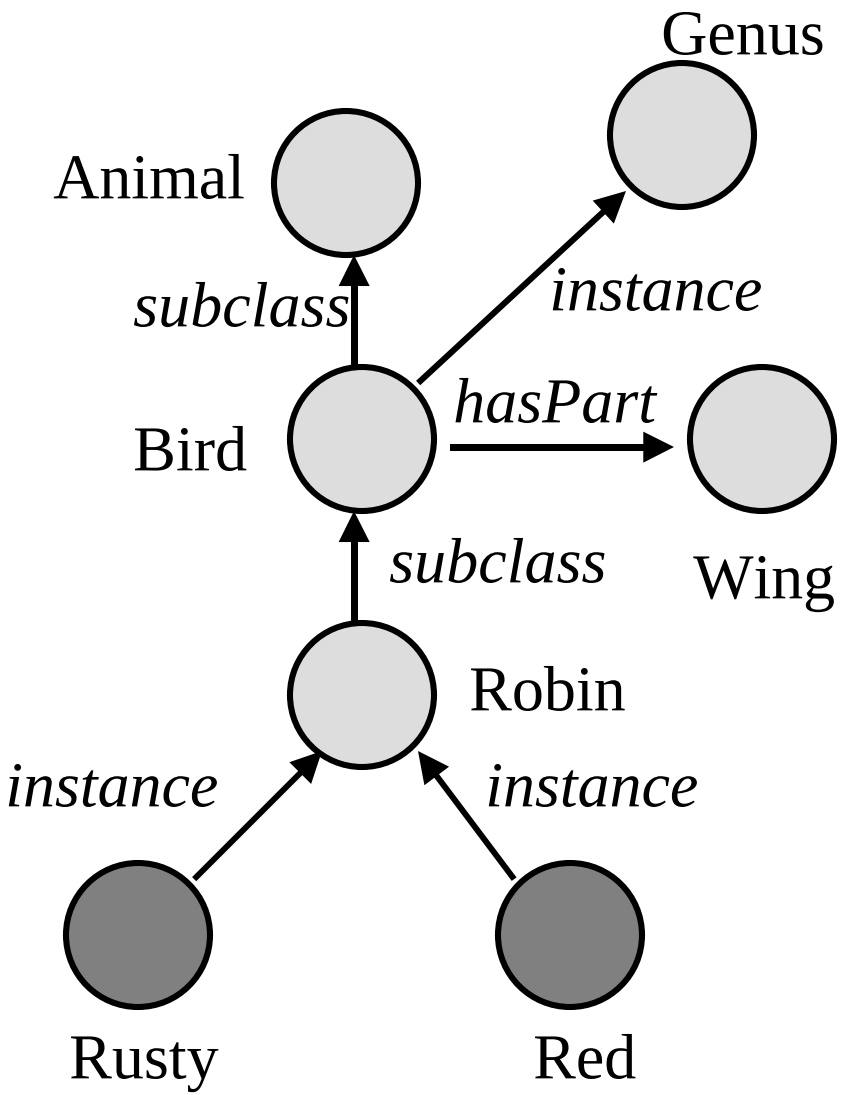
ISA hierarchy

- The ISA (is a) or AKO (a kind of) relation is often used to link a class and its superclass.
- And sometimes an instance and it's class.
- Some links (e.g. has-part) are inherited along ISA paths.
- The semantics of a semantic net can be relatively informal or very formal
  - often defined at the implementation level



Individuals and Classes

- Many semantic networks distinguish
  - nodes representing individuals and those representing classes
  - the “subclass” relation from the “instance-of” relation



Inference by Inheritance

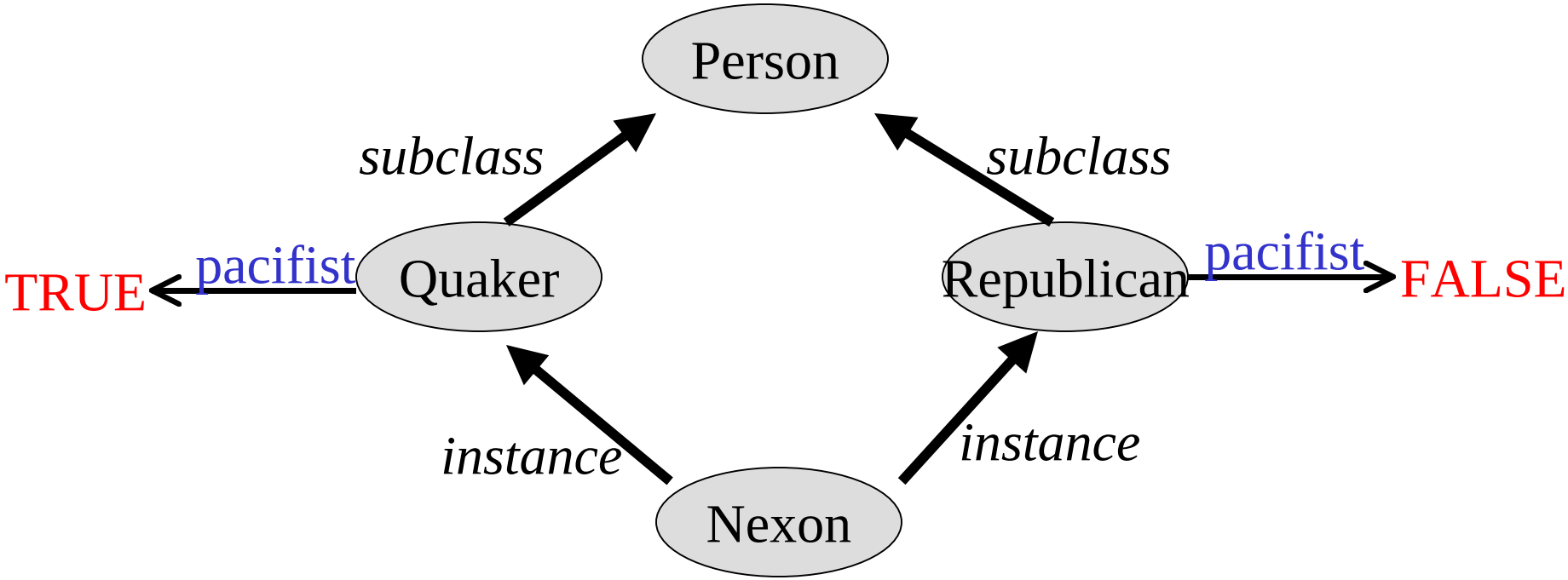
- One of the main types of reasoning done in a semantic net is the inheritance of values (properties) along the subclass and instance links.
- Semantic Networks differ in how they handle the case of inheriting multiple different values.
  - All possible properties are inherited
  - Only the “lowest” value or values are inherited

Multiple inheritance

- A node can have any number of superclasses that contain it, enabling a node to inherit properties from multiple "parent" nodes and their ancestors in the network.
- Conflict or inconsistent properties can be inherited from different ancestors
- Rules are used to determine inheritance in such "tangled" networks where multiple inheritance is allowed:
  - if $X \subseteq A \subseteq B$ and both A and B have property P (possibly with different variable instantiations), then X inherits A's property P instance (closer ancestors override far away ones).
  - If $X \subseteq A$ and $X \subseteq B$ but neither $A \subseteq B$ nor $B \subseteq A$ and both A and B have property P with different and inconsistent values, then X will not inherit property P at all; or X will present both instances of P (from A and B) to the user

Nixon Diamond

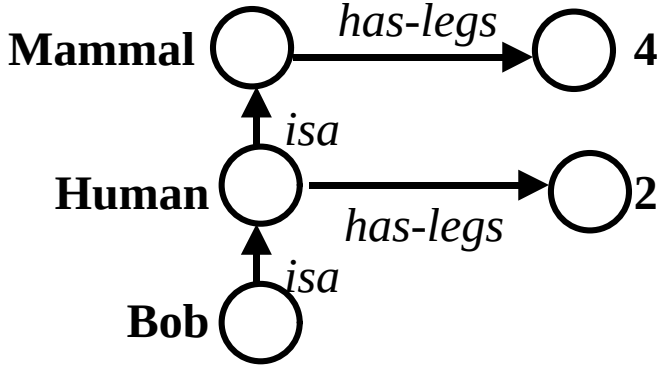
- This was the classic example circa 1980.



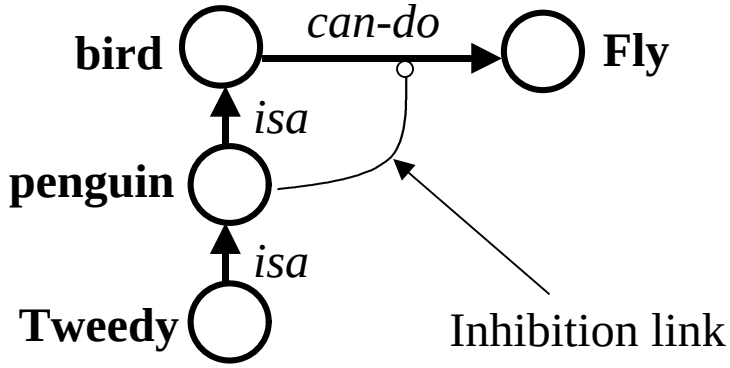
Exceptions in ISA hierarchy

- Properties of a class are often default in nature (there are exceptions to these associations for some subclasses/instances)

- Closer ancestors (more specific) overriding far way ones (more general)



- Use explicit inhibition links to prevent inheriting some properties



From Semantic Nets to Frames

- Semantic networks morphed into Frame Representation Languages in the 70's and 80's.
- A Frame is a lot like the notion of an object in OOP, but has more meta-data.
- A frame represents a **stereotypical/expected/default** view of an object
- Frame system can be viewed as adding additional structure into semantic network, a frame includes the object node and all other nodes which directly related to that object, organized in a **record like** structure
- A frame has a set of **slots,** each represents a relation to another frame (or value).
- A slot has one or more **facets,** each represents some aspect of the relation

Facets

- A slot in a frame holds more than a value.
- Other facets might include:
  - current fillers (e.g., values)
  - default fillers
  - minimum and maximum number of fillers
  - type restriction on fillers (usually expressed as another frame object)
  - attached procedures (if-needed, if-added, if-removed)
  - salience measure
  - attached constraints or axioms
  - pointer or name of another frame

Other issues

- **Procedural attachment**
  - In early time, AI community was against procedural approach and stress declarative KR
  - Procedures came back to KB systems when frame systems were developed, and later also adopted by some production systems (action can be a call to a procedure)
  - It is not called by a central control, but triggered by activities in the frame system
  - When an attached procedure can be triggered

    if-added: when a new value is added to one of the slot in the frame

    if-needed: when the value of this slot is needed

    if-updated: when value(s) that are parameters of this procedure is changed

- **Example:** a real estate frame system
  - Slots in a real estate property frame

    location

    area

    price
  - A facet in “price” slot is a procedure that finds the unit price (by location) and computes the prince value as the product of the unit price and the area
  - If the procedure is the type of *if-needed*, it then will be triggered by a request for the price from other frame (i.e., transaction frame)
  - If it is the type of *if-updated*, it then will be triggered by any change in either location or area
  - If it is the type of *if-added*, it then will be triggered by the first time when both location and area values are added into this frame

- **Description logic**
  - There is a family of Frame-like KR systems with a formal semantics.
    - E.g., KL-ONE, LOOM, Classic, …
  - An additional kind of inference done by these systems is automatic **classification**
    - finding the right place in a hierarchy of objects for a new description
  - Current systems take care to keep the language simple, so that all inference can be done in polynomial time (in the number of objects)
    - ensuring tractability of inference

- **Objects with multiple perspectives**
  - An object or a class may be associated with different sets of properties when viewed from different perspectives.
  - A passenger in an airline reservation system can be viewed as
    - a *traveler*, whose frame should include slots such as the
      date of the travel,
      departure/arrive airport;
      departure/arrive time, ect.
    - A *customer*, whose frame should include slots such as
      fare amount
      credit card number and expiration date
      frequent flier's id, etc.
  - Both traveler frame and customer frame should be children of the *passenger* frame, which has slots for properties not specific to each perspective. They may include name, age, address, phone number, etc. of that person