Forward-Chaining RBES

Forward-Chaining RBES

- Control cycle:
  - Find rules whose antecedents are true: *triggered* rules
  - Select one: *conflict resolution*
  - *Fire* the rule to take some action
- Continue forever or until some goal is achieved
- Used for synthesis, often, or process control

Example: Winston's "Bagger" Program

- Toy forward chainer – domain = bagging groceries
- Steps in this process:
  1. Check what customer has and suggest additions
  2. Bag large items, putting large bottles in first
  3. Bag medium items, putting frozen food in freezer bags
  4. Bag small items wherever there is room
- Working memory:
  - Needs to have information about:
    - items already bagged
    - unbagged items
    - which step (context) we're in

Example: Winston's "Bagger" Program

- Representation: could be literals, could have more structure than that
- Initial state:

```
Step: check-order
Bagged: nil
Unbagged: bread, Glop brand cheese, granola,
          ice cream
```

- Also need information about the world; this might be in the form of a table for this problem:

Object	Size	Container	Frozen?
bread	M	bag	nil
Glop	S	jar	nil
granola	L	box	nil
ice cream	M	box	t
Pepsi	L	bottle	nil
potato chips	M	bag	nil

Example: Winston's "Bagger" Program

Conflict resolution strategies – possibilities:

- specificity ordering:
  - if two rules conflict and one is more specific than the other, use it
  - Rule 1 is more specific than Rule 2 if Rule 1's antecedent literals are a superset of Rule 2's (assuming conjunction)
- rule ordering – implicit in rule base (unless using a rete net)
- data ordering – look at some data first (rete does this, sort of)
- size of antecedent – prefer rules with larger antecedent, since it's likely to be more specific
- recency – least/most recently used (depending on needs of designer)
- context-limiting

Example: Winston's "Bagger" Program

- Rules in form of IF-THEN pairs
- Examples:

```
R1: if  step = check-order &
        exists bag of chips &
        not exists soft drink bottle
    then add bottle of pepsi to order

R2: if  step = check-order
    then step = bag-large-items

R3: if step = bag-large-items &
       exists large item to be bagged &
       exists large bottle to be bagged &
       exists bag with < 6 large items
    then put bottle in bag
```


Example: Winston's "Bagger" Program

bag

- Initial state:

```
Step: check-order
Bagged: nil
Unbagged: bread, Glop brand cheese, granola,
          ice cream
```

chips | Pepsi

- World info:

Object	Size	Container	Frozen?
bread	M	bag	nil
Glop	S	jar	nil
granola	L	box	nil
ice cream	M	box	t
Pepsi	L	bottle	nil
potato chips	M	bag	nil

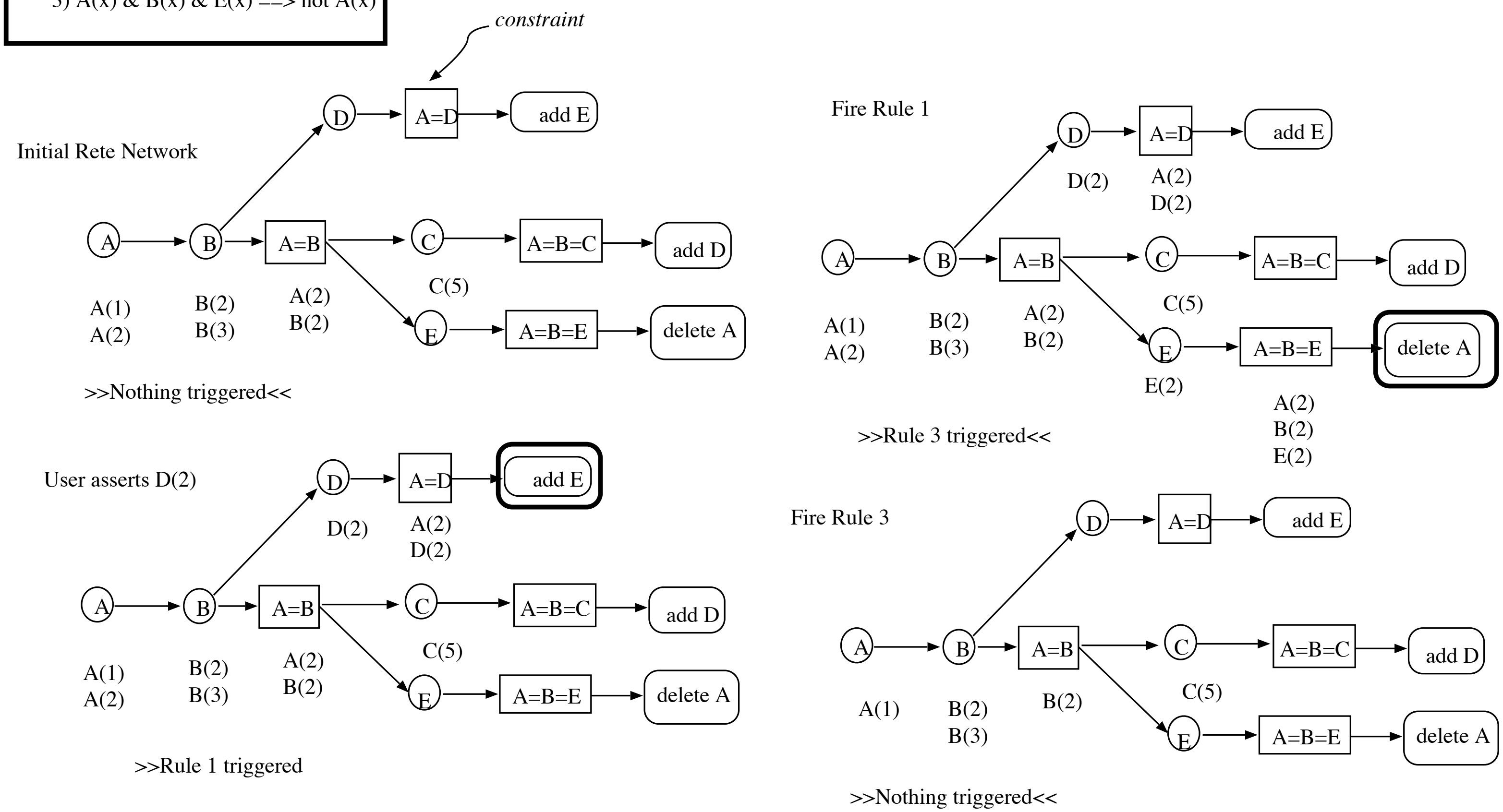
Finding Triggered Rules

- Possibly very time-consuming
- Observations:
  - Rules often share LHS elements (literals)
  - Rules don't usually change over short term
  - When WM changes: usually only a few changes per cycle
- Forgy: build a *rete network* based on the rules
- Rete records state of WM, rules in network – update on change

Rete Network

Rule base:
1) A(x) & B(x) & C(x) ==> D(x)
2) A(x) & B(y) & D(x) ==> E(x)
3) A(x) & B(x) & E(x) ==> not A(x)

Iniital WM: A(1), A(2), B(2), B(3), C(5)



Backward-Chaining RBES

Backward-Chaining RBES

- Synthesis: pick a solution
- Analysis: gather evidence, form best hypothesis – e.g., medical diagnosis
- Work backward from goal: focus question–asking on relevant facts, tests
- Need uncertainty management
- Follow all (relevant) lines of reasoning: no conflict resolution

How Does It Work?

- Sort of like a backward-chaining theorem prover
- Want to conclude something about x:
  - Is x in WM? Then conclude something from that.
  - Are there rules that conclude something about x? Then for each rule:
    - Try to conclude something about each antecedent (*backchain*).
    - If that's possible, fire the rule, giving some evidence for x.
  - Combine evidence for and against x.

Example: Zoo World

- Goal: id(Animal1,?x)
- Initial state 1:
  ```
  color(Animal1,tawny),
  eye-direction(Animal1,forward),
  teeth-shape(Animal1,pointed),
  eats(Animal1,meat),
  hair(Animal1), dark-spots(Animal1)
  ```
- Initial state 2:
  ```
  color(Animal1,tawny),
  eye-direction(Animal1,forward),
  teeth-shape(Animal1,pointed),
  eats(Animal1,meat),
  hair(Animal1)
  ```

Uncertainty Handling

- Obvious way: probability theory
- Need some way to assess belief, given some evidence

Uncertainty Handling

- Obvious way: probability theory
- Need some way to assess belief, given some evidence
- Bayes' rule:

$$P(H \mid E) = \frac{P(E \mid H) \cdot P(H)}{P(E)}$$

where $P(E) = P(E \mid H) \cdot P(H) + P(E \mid \neg H) \cdot P(\neg H)$

Uncertainty Handling

- Obvious way: probability theory
- Need some way to assess belief, given some evidence
- Bayes' rule:

$$P(H \mid E) = \frac{P(E \mid H) \cdot P(H)}{P(E)}$$

where $P(E) = P(E \mid H) \cdot P(H) + P(E \mid \neg H) \cdot P(\neg H)$

- Example:
  - H: Joey has lung cancer
  - E: Joey smokes

$$P(lung\text{–}Ca \mid smoking) = \frac{P(smoking \mid lung\text{–}Ca) \cdot P(lung\text{–}Ca)}{P(smoking)}$$

Uncertainty Handling

- General form:

$$P(H_i \mid E) = \frac{P(E \mid H_i) \cdot P(H_i)}{\sum P(E \mid H_j) \cdot P(H_j)}$$

- And with some prior evidence E and a new observation e:

$$P(H \mid e, E) = P(H \mid e) \cdot \frac{P(E \mid e, H)}{P(E \mid e)}$$

Problems with Bayesian approach

- There are problems with Bayesian probability for expert systems (in dispute recently)
- Probabilities may be difficult to obtain
  - P(E), P(H), P(E| H) may be hard to get in general – for example, where E = cough, or H = AIDS
  - empirical evidence suggests that people are not very good at estimating probabilities [Tversky & Kahneman, e.g.]
- Size of set of probabilities needed $O(2^n)$
  - Even if we could obtain them – requires too much space
  - ...and too much time to use, and compute

Problems with Bayesian approach

- In the general case, we're interested in

$$P(H \mid E_1 \wedge E_2 \wedge \ldots \wedge E_n)$$

  which is completely impractical to get
- Also assumes that $P(H_1), P(H_2), \ldots$ are disjoint probability distributions, that is, that H_i are independent and that they cover the set of all hypotheses!
- *Bayesian nets* address many of these problems in a different formalism

A Kludge: Certainty Factors

- Approximation to probability theory
- MYCIN (e.g.): $CF[H, E] = MB[H, E] - MD[H, E]$
- Since rule only supports/denies one fact: need only one number to give CF for H given E
- One CF per literal, one per rule

Combining Certainty Factors

- Formally, when two rules give evidence about same literal:

$$MB[H, s_1 \wedge s_2] = 0 \text{ if } MD = 1,$$

$$MB[H, s_1] + MB[H, s_2] \cdot (1 - MB[H, s_1])$$

- Similarly for MD
- Simple update function!

Example

- Rule A: If x then s_1
  Rule B: If y then s_2
  Rule C: If s_1 then H
  Rule D: If s_2 then H
- suppose $MB[H, s_1] = 0.3$, $MD = 0 \Rightarrow CF = 0.3$
- now rule B fires, giving $MB[H, s_2]$ as, say, 0.2:

$$MB[H, s_1 \wedge s_2] = 0.3 + 0.2 \cdot 0.7 = 0.44$$

$$MD = 0$$

$$CF = 0.44$$

Certainty Factors

- How to compute $CF(A \wedge B)$ for rule antecedents?

$$MB[H_1 \wedge H_2, E] = \min(MB[H_1, E], MB[H_2, E]$$

and for $CF(A \vee B)$:

$$MB[H_1 \wedge H_2, E] = \max(MB[H_1, E], MB[H_2, E]$$

Certainty Factors

- How to update certainty based on rule firing?
  - Two things to consider: MB/MD in antecedents (computed as above) and the CF of the rule:

$$MB[H,S] = MB'[H,S] \cdot \max(0, CF[S,E])$$

    where $MB'[H,S]$ is how much you'd believe S if E were completely believed (i.e., the rule CF), and $CF[S,E]$ is the certainty you have in S given all the evidence.
  - Essentially: you multiply the CF of the rule times the CF of the evidence

Certainty Factors

- More recently (1986), it's been found that CFs aren't in conflict with basic probability theory
- Why, then, do they work and Bayesian techniques seem not to?

Certainty Factors

- More recently (1986), it's been found that CFs aren't in conflict with basic probability theory
- Why, then, do they work and Bayesian techniques seem not to?
  - Heuristics
  - They assume rule independence – conditional probabilities are 0
  - The knowledge engineer has to ensure this
  - Leads to compound antecedents, but...
  - ...makes it tractable and modular
- Many recent expert systems are based on *Bayesian networks*

Example Expert Systems

- DENDRAL
- R1/XCON [J. McDermott] – DEC
- MYCIN, EMYCIN, ONCOCIN, PUFF, VM, CENTAUR, MDX, MDX2,...
- Blackboard systems

Topic: Structured knowledge representation

Structured Knowledge Representations

- Problem with logic and rules:
  - No real structure
  - Representation doesn't reflect patterns—structure—in world
- Need a knowledge representation that is *structured*

Ontological Commitment

- Ontological commitment for structured representations:
  - World consists of objects
  - Objects have properties
  - Relations exist between objects
- I.e., pretty much same as FOPC...

Ontological Commitment

- Ontological commitment for structured representations:
  - World consists of objects
  - Objects have properties
  - Relations exist between objects
- I.e., pretty much same as FOPC...
- ...difference is *structure* of representation

Advantages and Disadvantages

- Reflects structure of the world
- Groups knowledge together:
  - Easier access
  - Easy to establish salient features
  - Conceptually easier for many people
- But managing relationships is not easy

Kinds of Structured Representation

- Several different types: frames, semantic networks
- Functionally equivalent (and they're all formally equivalent to FOPC)

Frames

Frames

- Frames are one kind of structured representation
- Originally: used to describe visual scenes [Minsky]
- Frames are *slot-filler* representations:
  - *Slots* of frame: name attributes or relations
  - *Filler* of slot contains its value
- Since frames can fill slots $\Rightarrow$ interconnected *frame system*
- Frame rely heavily on *isa* relationships $\Rightarrow$ isa hierarchies

ISA hierarchies

- Creates an abstraction hierarchy
- Captures relationships between classes and subclasses (or types and subtypes)
- Inheritance: class $\Rightarrow$ subclass
  - If X ISA Y, then X inherits Y's characteristics - unless explicitly overwritten by more specific class
  - ISA is transitive and anti-symmetric
- Saves space
- Gives access to default information by identifying type

Caveats for Object-Oriented Programmers

- ISA hierarchies are not copied from C++, Java, Python...
- OOP: inheritance partly (mainly?) to share function, abstraction – ISA: class–subclass relationship is semantic, not for convenience
- Create classes that "make sense"
- Make sure ISA reflects a subclass/class relationship

Artificial Intelligence

Let's Create an Animal Hierarchy

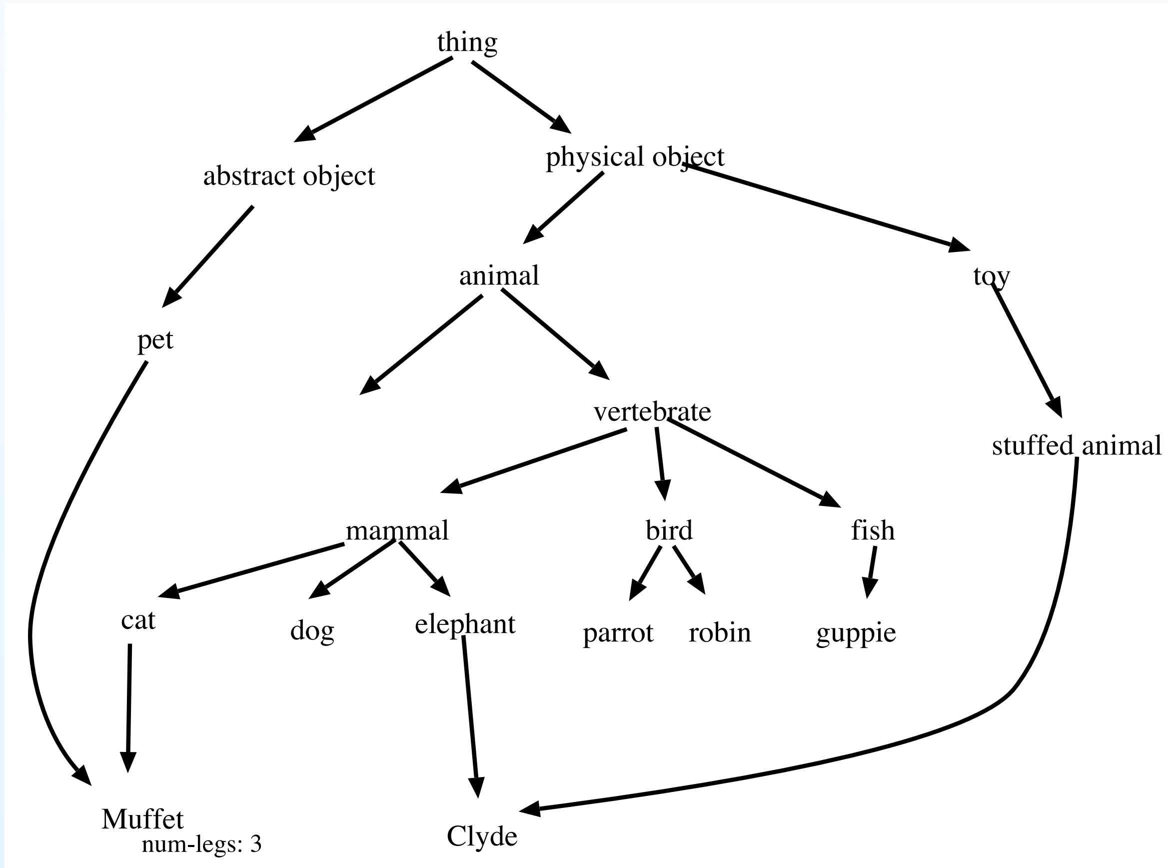
Animals to include:

dog	cat	monkey	elephant	guppie
catfish	parrot	robin	Muffet	Clyde

Let's Create an Animal Hierarchy

Animals to include:

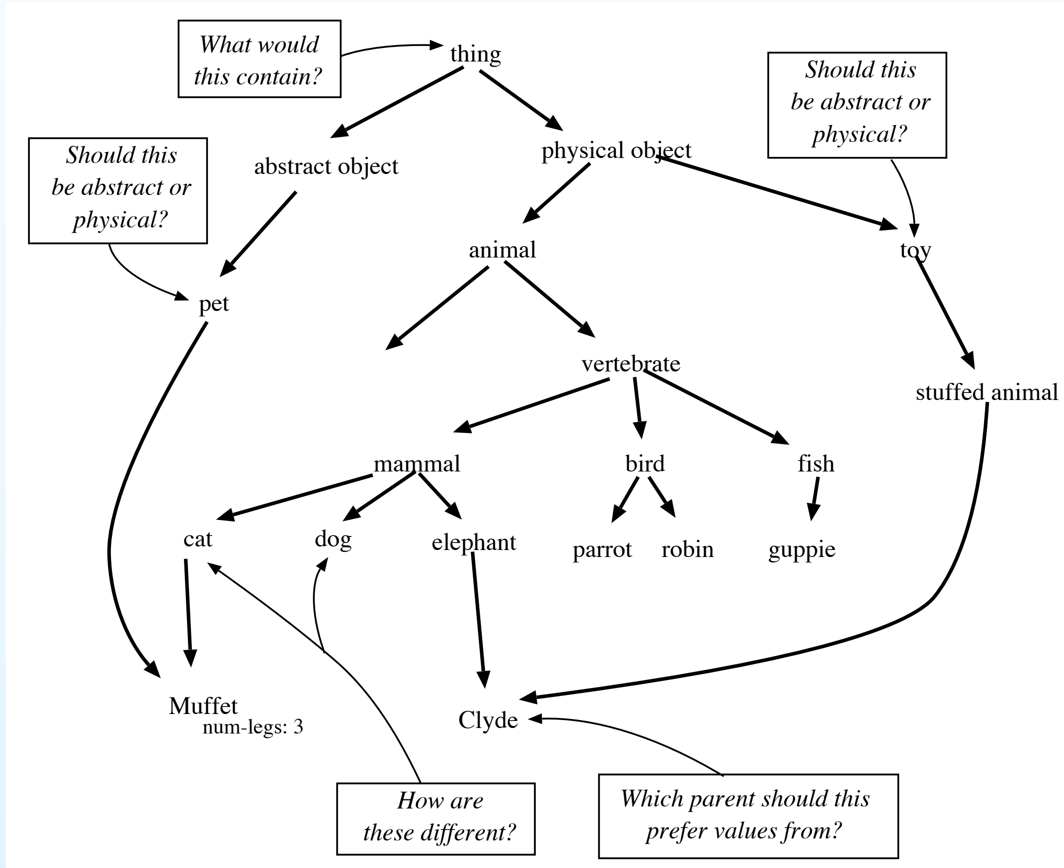
dog	cat	monkey	elephant	guppie
catfish	parrot	robin	Muffet	Clyde



Let's Create an Animal Hierarchy

Animals to include:

dog	cat	monkey	elephant	guppie
catfish	parrot	robin	Muffet	Clyde



Deciding on the Nodes

- Instances must have their own nodes
  - cannot inherit from an instance
  - no default information is stored
- Prototypes or Classes?
  - A prototype describes some typical member of a group
  - Classes partition the knowledge base – may have more than one partitioning
    - *is-covered-by*: the set of classes that form a partitioning
    - *mutually-disjoint-with*: the relationship between classes in a partitioning

Deciding on the Nodes

- The right types
  - group things by significant properties
  - properties identified with types should be unlikely to change
  - existing taxonomies, *basic level categories* can help

Artificial Intelligence

Tangled ISA Hierarchies

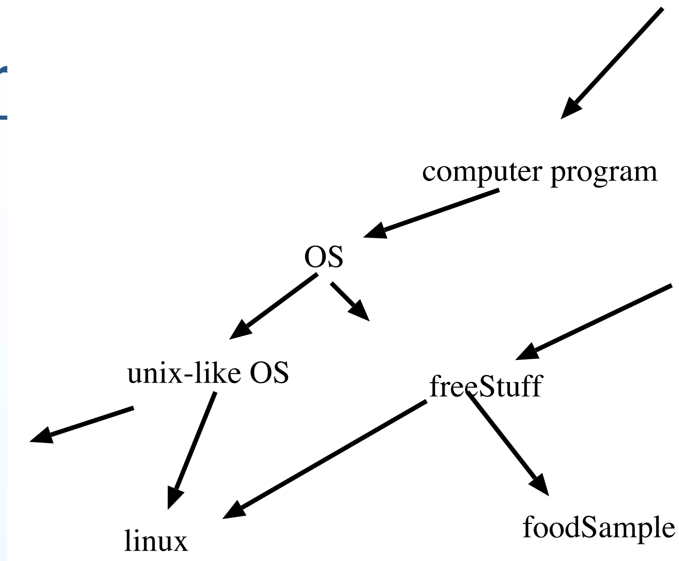
- The problem: An entity is a member of more than one type or class and need to get information about the entity from the correct parent
- Possible solutions:
  - if there are no conflicting slots, take information from wherever it resides
  - weight parents for the slots
  - inferential distance

Tangled ISA Hierarchies: Inferential distance

- NEVER want to count links

Tangled ISA Hierarchies: Inferer

- NEVER want to count links
- $class_2$ is further than $class_3$ from $class_1$ if there is a path through $class_3$ to $class_2$
- only a partial order
- conflicts are unresolved if the classes are not related



Linux is more like a computer program, or samples of sausage at Hannaford's?

Other Hierarchies

- Partonomic hierarchy
- Can make your own
  - must be transitive and anti-symmetric
  - must inherit relation
  - can inherit other features

Inheritance in Frames

- Frames make use of inheritance through the *isa* links
- Slots are inherited:
  - Helps determine which slots (attributes, relations) the frame has
  - A kind of default knowledge
- Fillers are inherited, too

Representing Knowledge with Frames

- Frames can be used to represent abstract as well as physical "objects"
- Frames as classes of objects: e.g., HUMANS
- Frames as prototypes of objects: e.g., HUMAN
- Frames as *instances* of a class/exemplar of a prototype: e.g., ROY, HUMAN001, etc.
- *isa*: sub-type or instance-of link?

Representing Knowledge with Frames

How would you represent each of the following?

- Car

Representing Knowledge with Frames

How would you represent each of the following?

- Car
- Police car

Representing Knowledge with Frames

How would you represent each of the following?

- Car
- Police car
- A particular police car, say `Car54`

Representing Knowledge with Frames

How would you represent each of the following?

- Car
- Police car
- A particular police car, say `Car54`
- Water

Representing Knowledge with Frames

How would you represent each of the following?

- Car
- Police car
- A particular police car, say `Car54`
- Water
- River or lake

Representing Knowledge with Frames

How would you represent each of the following?

- Car
- Police car
- A particular police car, say `Car54`
- Water
- River or lake
- Music

Representing Knowledge with Frames

How would you represent each of the following?

- Car
- Police car
- A particular police car, say `Car54`
- Water
- River or lake
- Music
- Numbers

Representing Knowledge with Frames

How would you represent each of the following?

- Car
- Police car
- A particular police car, say `Car54`
- Water
- River or lake
- Music
- Numbers
- Sets

Representing Knowledge with Frames

How would you represent each of the following?

- Car
- Police car
- A particular police car, say `Car54`
- Water
- River or lake
- Music
- Numbers
- Sets
- Logical relationships

Procedural Attachment

- Mechanism to allow principled way to execute procedures when events happen in the frame system
- Procedures are attached to individual slots
- Two types generally defined:
  - If-needed: executed when a value is retrieved from the slot
  - If-added: executed when a value is added to a slot or the value of the slot is changed
- Why use them?

Other Information Associated with Slots

- *constraints:* range or type of values that must fill slots
- *defining values:* all members of the type must have this value
- *special inheritance:* inherit from some other frame or hierarchy than the *isa* hierarchy

Examples

```
(defframe mobile-object (^physical-object)
  (motile t)
  medium              ;what it moves under/on/over/through
  (velocity (0 0 0))
  (orientation (0 0 0))
  (speed - (if-needed
             (lambda (filler frame slot)
               (let* ((vel (role-filler 'velocity frame))
                      (x (first vel))
                      (y (second vel))
                      (z (third vel)))
                 (sqrt (+ (* x x)
                          (* y y)
                          (* z z))))))))
```


Examples

```
(defframe living-thing (^natural-object)
  (living? t)
  (density ^moderate)
  (physical-state ^solid)
  (substance ^protoplasm)
  (status ^nominal-health))
```

Structured KRep

Frames
- Overview
- Inheritance
- Representation
- Example
- Proc. attachment
- Other info
- Examples

Semantic Networks

CD

Cyc

Description Logics

Examples

```
(defframe water-surface (^interface)
  (object1 - (isa ^air))
  (object2 - (isa ^water))
  (position (^above @object1 @object2))
  surface-traffic
  ice-status    ; nil? solid? percent?
  sea-state
  )
```

Artificial Intelligence

Examples


```
(defframe orca (^planner ^mobile-agent)
  (mission - (default (^mission)))
  (plan - (default (^intention-structure)))
  (location @(vehicle location))
  (heading @(vehicle heading))
  (depth @(vehicle depth))
  (altitude @(vehicle altitude))
  (velocity @(motion velocity))
  (acceleration @(motion acceleration))
  (motion @(vehicle motion))
  (vehicle - (default (^EAVE)))
  (equipment @(vehicle mission-package))
  (communication-system @(vehicle communication-system))
  (communication - (default (^set))))
```

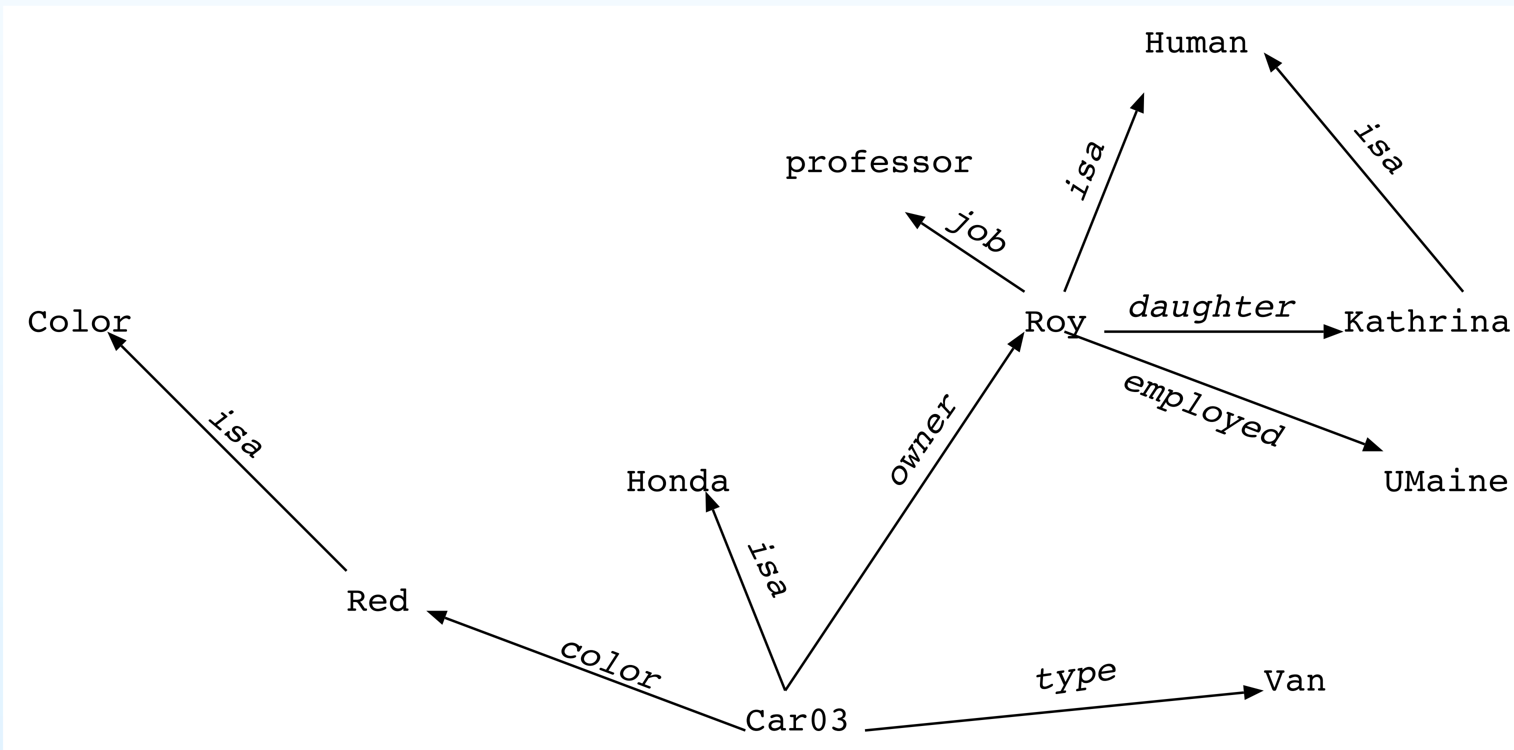
Semantic Networks

Semantic Networks

- A semantic network is a set of nodes an arcs:
  - Nodes = concepts
  - Arcs = relationships or attributes
- Example

Semantic Networks

- A semantic network is a set of nodes an arcs:
  - Nodes = concepts
  - Arcs = relationships or attributes
- Example

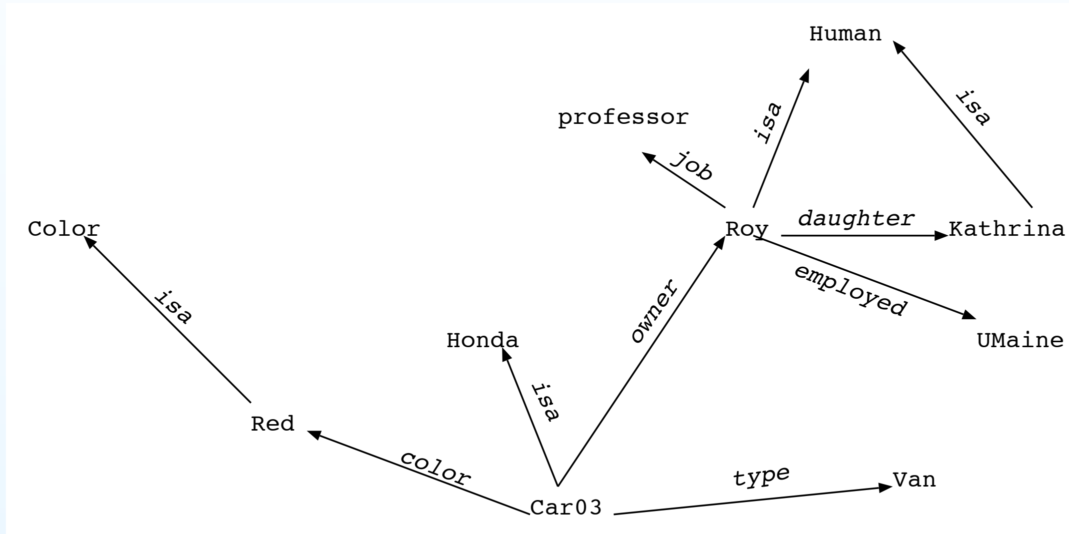


Semantic networks and frames

- Semantic nets and frames are very similar
- Can view frames as portions of semantic nets

Semantic networks and frames

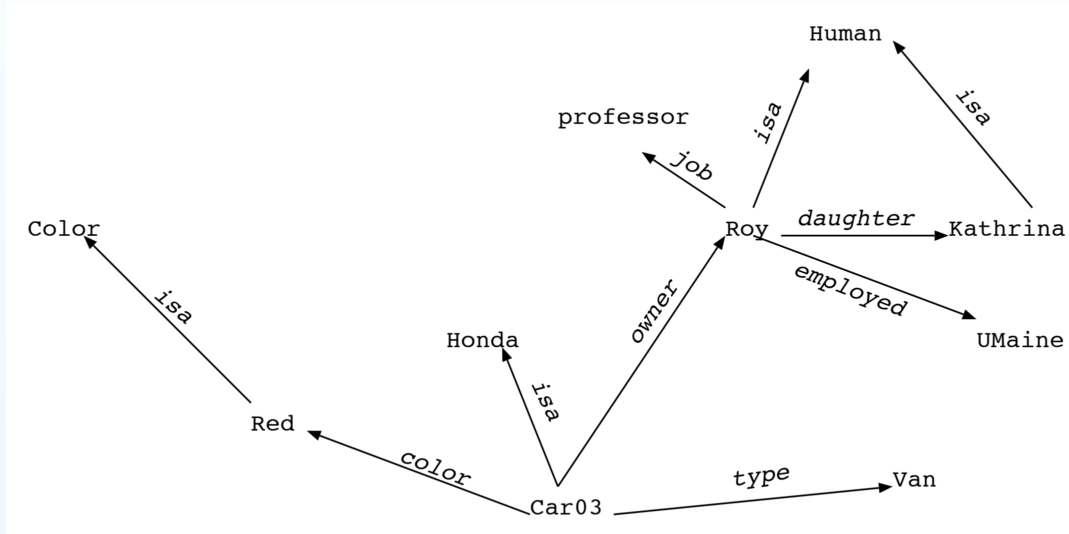
- Semantic nets and frames are very similar
- Can view frames as portions of semantic nets




```
Car03:                      Roy:
   isa: Honda                 isa: Human
   type: Van                  daughter: Kathrina
   color: Red                 job: Professor
   owner: Roy                 employed: UMaine

Red:                        Kathrina:
   isa: Color                  isa: Human
   ...                         ...
```

Semantic networks and frames

- Semantic nets and frames are very similar
- Can view frames as portions of semantic nets




```
Car03:
   isa: Honda
   type: Van
   color: Red
   owner: Roy

Red:
  isa: Color
  ...

Roy:
  isa: Human
  daughter: Kathrina
  job: Professor
  employed: UMaine

Kathrina:
   isa: Human
   ...
```

- Hard to do procedural attachment in semantic net

Cyc

Cyc

- Huge project: many, many person-years of effort
- Doug Lenat and others at MCC; now CYCorp
- Goal:
    - Initially, to have the knowledge required to read the encyclopedia
    - Then: common sense knowledge – shoot for level of three year-old

Why CYC?

- Try to overcome *brittleness* of expert systems, other AI programs
- Test many/all existing knowledge representation techniques to see if they scale up
- Provide a shared commonsense knowledge base for smaller, special-purpose programs
- Study what commonsense knowledge *is*

Representation in CYC

- Two kinds of representation used:
  - *epistemological level:* easy interface for people, simple semantics,
  - *heuristic level:* more efficient, allows logically superfluous knowledge which makes processing more efficient
- Creates an ontology that must be used as base
- Huge body of knowledge, heterogeneous, with many inference mechanisms
- Many ideas about characterizing slots came from CYC

How to Get All the Knowledge in There?

- Possibilities:
  - By hand
  - NLP
  - Machine learning
- Their approach:
  - Hand code the first "ten million or so facts that make up commonsense knowledge"
  - Then try to "bootstrap" using NLP, learning, etc.

CYCL

- CYC's representation language is CYCL
- Standard extentions to frame programs:
  - tangled hierarchies
  - slots as objects/frames
  - inheritance via other slots (*transfers-through*) slot

```
frame color
  isa: slot
  transfers-through: top-level-part-of
frame car01:
  color: red
frame car-door01:
  top-level-part-of: car01
```

  - mutually-disjoint with
  - distinction between *instance* and *isa*

CYCL-specific extensions

- Additional inheritance mechanisms
- Constraint language

```
 Me:
     likes:
     constraints (beerConstraint)

 beerConstraint:
    slotConstrained: (likes)
    slotValueSubsumes:
        (TheSetOf X (Person allInstances)
           (And (likes-to-drink X beer)
              (Not (ThereExists Y (Drinks allInstances)
                       (And (Equal Y sissyDrink)
                            (likes-to-drink X Y))))))
    propagateDirection: forward

Mike:
  likes-to-drink: (beer)
```

CYCL-specific extensions

- Additional inheritance mechanisms
- Constraint language

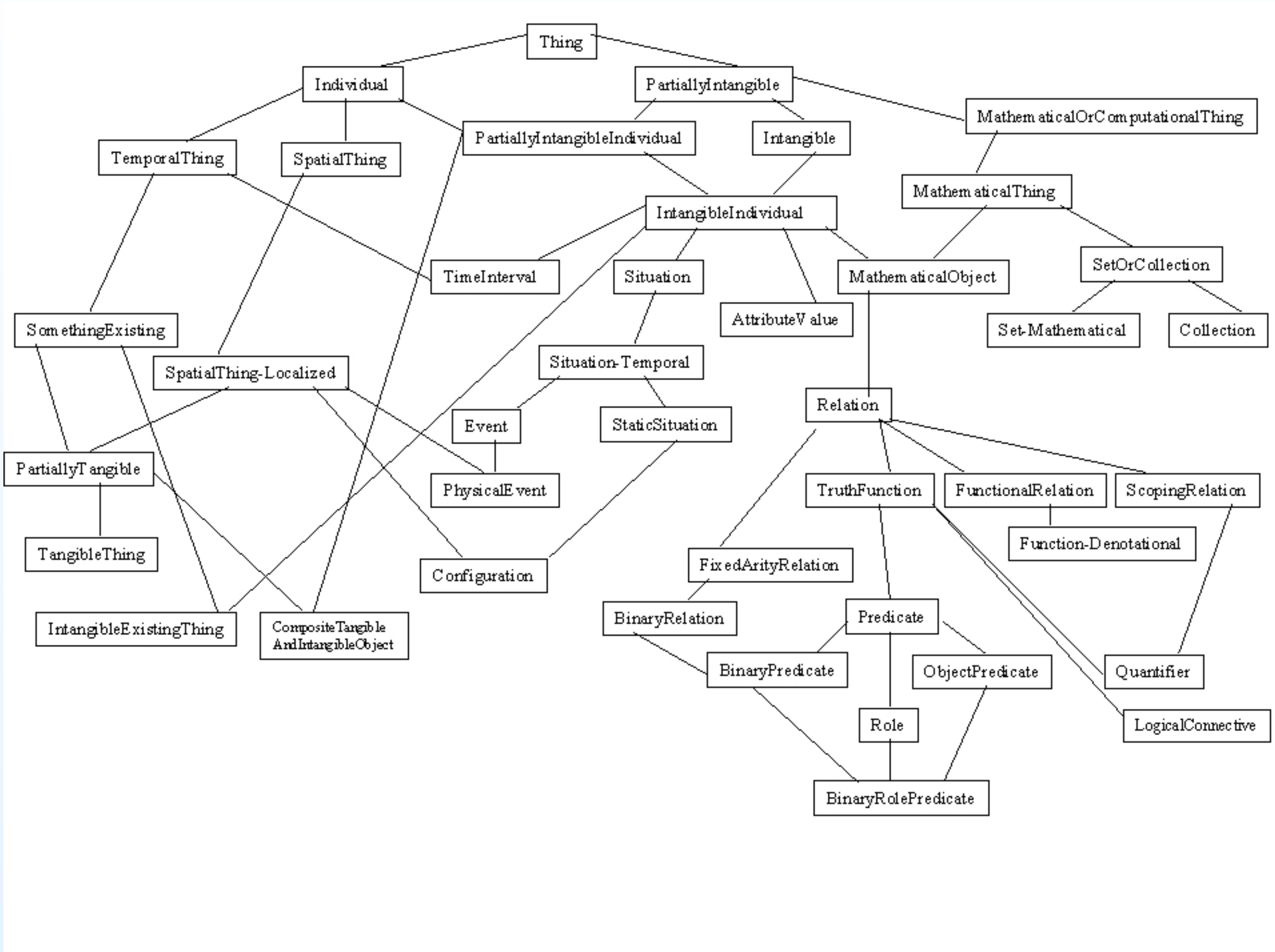
```
Me:
    likes:
    constraints (beerConstraint)

beerConstraint:
   slotConstrained: (likes)
   slotValueSubsumes:
       (TheSetOf X (Person allInstances)
          (And (likes-to-drink X beer)
             (Not (ThereExists Y (Drinks allInstances)
                      (And (Equal Y sissyDrink)
                           (likes-to-drink X Y))))))
   propagateDirection: forward

Mike:
  likes-to-drink: (beer)

Mike:
  likes-to-drink: (beer sissyDrink)
```

- Constraints enforced by constraint system, TMS
- If propagateDirection is "backward", then it fires when we want slot's value – if "forward" – takes a while!

CYC's Ontology

- What is an "ontology"?
- Thing concept
- Collection vs IndividualObject
- Intangible, TangibleObject, CompositeObject
- Substance
- Intrinsic properties and extrinsic properties
- Event, process
- Slot
- Time
- Agent

CYC's Upper Ontology



Description Logics

Description logics

- Logic:
  - very general, good semantics, but:
  - cumbersome
  - intractable, not decidable
- Frames and semantic nets ("network representations"):
  - specialized reasoning, intuitive, but:
  - semantics lacking/inconsistent
- Brachman's KL-ONE system: attempted to add rigor to network representations
- Gave rise to what is now called *description logics*

Basics

- Concerned with concepts and roles
- Concepts correspond to sets of individuals
- Primitive concepts:
  - e.g., $\texttt{Car}$, $\texttt{Human}$, etc.
  - equivalent to: $\texttt{Car}(x)$, etc., in FOL

Basics

- Concerned with concepts and roles
- Concepts correspond to sets of individuals
- Primitive concepts:
  - e.g., `Car`, `Human`, etc.
  - equivalent to: $\texttt{Car}(x)$, etc., in FOL
- Roles:
  - Like slots in frames
  - E.g., `hasChildren`

Basics

- Concerned with concepts and roles
- Concepts correspond to sets of individuals
- Primitive concepts:
  - e.g., $\mathtt{Car}$, $\mathtt{Human}$, etc.
  - equivalent to: $\mathtt{Car}(x)$, etc., in FOL
- Roles:
  - Like slots in frames
  - E.g., $\mathtt{hasChildren}$
- Complex (compound) concepts:
  - Built by composition from other concepts and roles
  - Often *intersection of concepts* ($\sqcap$) as operator
  - Different composition operators $\Rightarrow$ different logics

Tbox and Abox

- Knowledge in a DL system divided into two “boxes”
- *Tbox* (terminological box):
  - definitions – the ontology, i.e.
  - consists of concepts – e.g., `Human`
  - relatively static across problems

Tbox and Abox

- Knowledge in a DL system divided into two "boxes"
- *Tbox* (terminological box):
  - definitions – the ontology, i.e.
  - consists of concepts – e.g., `Human`
  - relatively static across problems
- *Abox* (assertion box):
  - facts about current problem
  - instances of concepts – e.g., `Human(Roy)`
  - dynamic across, even within problems

Tbox Examples

- Woman:

Tbox Examples

- Woman:

$$\texttt{Woman} \equiv \texttt{Person} \sqcap \texttt{Female}$$

Tbox Examples

- Woman:

$$\texttt{Woman} \equiv \texttt{Person} \sqcap \texttt{Female}$$

- Parent:

Tbox Examples

- Woman:

$$\mathtt{Woman} \equiv \mathtt{Person} \sqcap \mathtt{Female}$$

- Parent:

$$\mathtt{Parent} \equiv \mathtt{Person} \sqcap \exists \mathtt{hasChild.Person}$$

Tbox Examples

- Woman:

$$\texttt{Woman} \equiv \texttt{Person} \sqcap \texttt{Female}$$

- Parent:

$$\texttt{Parent} \equiv \texttt{Person} \sqcap \exists \texttt{hasChild.Person}$$

- Mother:

Tbox Examples

- Woman:

$$\texttt{Woman} \equiv \texttt{Person} \sqcap \texttt{Female}$$

- Parent:

$$\texttt{Parent} \equiv \texttt{Person} \sqcap \exists \texttt{hasChild.Person}$$

- Mother:

$$\texttt{Mother} \equiv \texttt{Parent} \sqcap \texttt{Woman}$$

Tbox Examples

- Woman:

$$\mathtt{Woman} \equiv \mathtt{Person} \sqcap \mathtt{Female}$$

- Parent:

$$\mathtt{Parent} \equiv \mathtt{Person} \sqcap \exists \mathtt{hasChild.Person}$$

- Mother:

$$\mathtt{Mother} \equiv \mathtt{Parent} \sqcap \mathtt{Woman}$$

- Students who take COS 470:

Tbox Examples

- Woman:

$$\mathtt{Woman} \equiv \mathtt{Person} \sqcap \mathtt{Female}$$

- Parent:

$$\mathtt{Parent} \equiv \mathtt{Person} \sqcap \exists \mathtt{hasChild.Person}$$

- Mother:

$$\mathtt{Mother} \equiv \mathtt{Parent} \sqcap \mathtt{Woman}$$

- Students who take COS 470:

$$\mathtt{Student} \sqcap \exists \mathtt{classSchedule.}(\exists \mathtt{contains.COS470})$$

Abox Examples

- Joe is Harry's son:

Abox Examples

- Joe is Harry's son:

$$hasSon(Harry, Joe)$$

Abox Examples

- Joe is Harry's son:

$$hasSon(Harry, Joe)$$

- Roy is a professor:

Abox Examples

- Joe is Harry's son:

$$hasSon(Harry, Joe)$$

- Roy is a professor:

$$\texttt{Professor(Roy)}$$

Abox Examples

- Joe is Harry's son:

$$hasSon(Harry, Joe)$$

- Roy is a professor:

$$\texttt{Professor(Roy)}$$

$$\texttt{Person(Roy)} \sqcap \texttt{hasRole(Roy,Professor)}$$

Abox Examples

- Joe is Harry's son:

$$hasSon(Harry, Joe)$$

- Roy is a professor:

$$\texttt{Professor(Roy)}$$

$$\texttt{Person(Roy)} \sqcap \texttt{hasRole(Roy,Professor)}$$

$$(\texttt{Person} \sqcap \exists \texttt{hasRole.Professor})(\texttt{Roy})$$

Counting

- Some logics can count, too
- E.g.: “A mother with two female and at least one male children”:

Counting

- Some logics can count, too
- E.g.: “A mother with two female and at least one male children”:

$$\texttt{Mother} \sqcap = 2(\texttt{hasChild.Female}) \sqcap \geq 1(\texttt{hasChild.Male})$$

Inference in DL

- Reasoning in DL systems occurs in context of Tbox and Abox
- Tbox reasoning: *subsumption*
  - Is concept $A \sqsubseteq$ concept B?
  - E.g.:

$$\begin{array}{lcl} \texttt{Mother} & \equiv & \texttt{Person} \sqcap \texttt{Female} \sqcap \exists \texttt{hasChild.Person} \\ \texttt{Parent} & \equiv & \texttt{Person} \sqcap \exists \texttt{hasChild.Person} \\ \texttt{Mother} & \sqsubseteq & \texttt{Parent} \end{array}$$

  - Can be much more complicated and indirect
- Abox reasoning: *classification*
  - Is A an instance of concept B?
- Often other kinds of reasoning, too

Different DLs

- DL really comprised of a family of logics
- Basic is $\mathcal{AL}$ (ascription language)
- Add other operators, get new languages – e.g., $\mathcal{ALU}$ would be $\mathcal{AL}$ plus union, etc.
- Simple DLs: decidable, (relatively) efficient inferences
- More expressive DLs: give up efficiency, even decidability

Example Implementation: CLASSIC

- The CLASSIC language is an implementation of a DL ($\mathcal{AL}$?)

Example Implementation: CLASSIC

- The CLASSIC language is an implementation of a DL ($\mathcal{AL}$?)
- Example: a bachelor

Example Implementation: CLASSIC

- The CLASSIC language is an implementation of a DL ($\mathcal{AL}$?)
- Example: a bachelor

$$\texttt{Bachelor} = \texttt{And(Unmarried, Adult, Male)}$$

Example Implementation: CLASSIC

- The CLASSIC language is an implementation of a DL ($\mathcal{AL}$?)
- Example: a bachelor

$$\texttt{Bachelor} = \texttt{And(Unmarried, Adult, Male)}$$

- (From R&N) Men with at least three sons who are all unemployed and married to doctors, and at most two daughters who are all professors in physics or math departments:

Example Implementation: CLASSIC

- The CLASSIC language is an implementation of a DL ($\mathcal{AL}$?)
- Example: a bachelor

$$\texttt{Bachelor} = \texttt{And(Unmarried, Adult, Male)}$$

- (From R&N) Men with at least three sons who are all unemployed and married to doctors, and at most two daughters who are all professors in physics or math departments:

```
And(Man,AtLeast(3,Son),AtMost(2,Daughter),
    All(Son,And(Unemployed, Married,
        All(Spouse,Doctor))),
    All(Daughter,And(Professor,
        Fills(Department,Physics,Math))))
```

Artificial Intelligence

Uses

- General-purpose knowledge representation
- Natural language processing
- Reasoning in intelligent databases: entity-relation models
- Web Ontology Language (OWL):
  - Part of *semantic Web*
  - Associate machine-understandable semantics with Web pages
  - One language is OWL-DL
  - Complete and decidable

Topic: Local DL example: Orca

Example Orca DL

```
----------------------------------------
Definition=(SOME expectsPresenceOf Salinity)
Certainty=0.401
----------------------------------------
Definition=(SOME expectsPresenceOf OceanSurface)
Certainty=0.436
----------------------------------------
Definition=(SOME expectsPresenceOf
                 (AND Thruster (SOME hasAdvisedValue ShoreBased)))
Certainty=0.769
----------------------------------------
Definition=(SOME expectsPresenceOf
                 (AND Location
                      (SOME hasNumber
                            (AND Float
                                 (D-FILLER hasNumericValue
```

```
                                    (D-LITERAL 19.115639 (D-BASE-TYPE float)))
                                   (D-FILLER hasUnitOfMeasure
                                    (D-LITERAL somerandomstring
                                     (D-BASE-TYPE string)))))
                      (SOME hasNumber
                            (AND Integer
                                 (D-FILLER hasNumericValue
                                  (D-LITERAL 31 (D-BASE-TYPE integer)))
                                 (D-FILLER hasUnitOfMeasure
                                  (D-LITERAL somerandomstring
                                   (D-BASE-TYPE string)))))))
Certainty=0.482
----------------------------------------
Definition=(SOME expectsPresenceOf
                 (AND Survey (SOME hasDegreeExpected Mine)
                      (SOME definesGoal ActiveMission)))
Certainty=0.125
----------------------------------------
Definition=(SOME expectsPresenceOf
                 (AND DetectSubmarine
                      (D-FILLER hasEventDescription
```

```
                    (D-LITERAL somerandomstring
                     (D-BASE-TYPE
                      http://www.w3.org/2001/XMLSchema#string)))))
Certainty=0.243
----------------------------------------
Definition=(SOME hasFuzzyFeature
                 (AND Danger
                      (SOME hasFuzzyMembershipFunction
                            (AND TrapezoidalFunction
                                 (SOME hasLocalMaxAt Number)
                                 (SOME hasLocalMaxAt
                                       (AND Float
                                            (D-FILLER hasNumericValue
                                             (D-LITERAL 24.848389
                                              (D-BASE-TYPE
                                               http://www.w3.org/2001/XMLSchema#flo
                                            (D-FILLER hasUnitOfMeasure
                                             (D-LITERAL somerandomstring
                                              (D-BASE-TYPE
                                               http://www.w3.org/2001/XMLSchema#str
                                 (SOME hasLocalMinAt Number)
```

```
                                    (SOME hasLocalMinAt
                                          (AND Integer
                                               (D-FILLER hasNumericValue
                                                (D-LITERAL 5
                                                 (D-BASE-TYPE
                                                  http://www.w3.org/2001/XMLSchema#inte
                                               (D-FILLER hasUnitOfMeasure
                                                (D-LITERAL somerandomstring
                                                 (D-BASE-TYPE
                                                  http://www.w3.org/2001/XMLSchema#str
Certainty=0.334
----------------------------------------
Definition=(AND (SOME hasActivePeriod EnteringContext)
                (SOME hasOperationalSetting
                      (AND SelfDepth (SOME hasAdvisedValue Medium))))
Certainty=0.943
----------------------------------------
Definition=(AND
            (SOME definesGoal
                  (AND SamplingComplete
                       (D-FILLER hasEventDescription
```

```
                    (D-LITERAL somerandomstring
                     (D-BASE-TYPE
                      http://www.w3.org/2001/XMLSchema#string)))))
            (SOME hasCost Medium) (SOME hasDegreeExpected High)
            (SOME hasImportance High)
            (SOME isAchievedBy (AND Maneuver (SOME hasActor PeerAgent))))
Certainty=0.559
----------------------------------------
Definition=(AND
            (SOME respondsWithAction
                  (AND CommunicateStatus
                       (SOME hasObject
                             (AND NavigationComputer
                                  (SOME hasCost
                                        (AND SelfBatteryLevel
                                             (SOME hasStateValue Medium)))))
                       (SOME hasActor AdversaryAgent)
                       (SOME isSampleTargetOf PeerAgent)))
            (SOME hasImportance Medium)
            (SOME handlesEvent
                  (AND SensorFailure
```

```
                        (D-FILLER hasEventDescription
                         (D-LITERAL somerandomstring
                          (D-BASE-TYPE
                           http://www.w3.org/2001/XMLSchema#string))))))
Certainty=0.124
----------------------------------------
Definition=(AND
            (SOME handlesEvent
                  (AND PowerFailure
                       (SOME hasStateValue
                             (AND ThrusterFailure
                                  (D-FILLER hasEventDescription
                                   (D-LITERAL somerandomstring
                                    (D-BASE-TYPE
                                     http://www.w3.org/2001/XMLSchema#string)))))))
            (SOME hasImportance Low)
            (SOME respondsWithAction
                  (AND MaintainPosition (SOME hasActor Agent))))
Certainty=0.904
----------------------------------------
Definition=(SOME definesAction
```

```
                (AND Thruster
                     (SOME hasObject
                           (AND PeerAgent (SOME hasNumber Targeted)))
                     (SOME hasSpeed AdversaryAgent)))
Certainty=0.655
----------------------------------------
Definition=(SOME definesAction
                (AND MaintainPosition
                     (SOME hasDirection
                           (AND Number (SOME handlesEvent Submarine)))
                     (SOME hasSpeed
                           (AND Float
                                (SOME hasObject
                                      (AND Navigate
                                           (SOME hasActor AdversaryAgent)))))
                     (SOME definesGoal Thruster)))
Certainty=0.117
```