Translating English to FOL…

- No purple mushroom is poisonous.

Translating English to FOL…

- No purple mushroom is poisonous.

$\neg(\exists x)$ purple(x) $\wedge$ mushroom(x) $\wedge$ poisonous(x)

or, equivalently,

$(\forall x)$ (mushroom(x) $\wedge$ purple(x)) => $\neg$poisonous(x)

Translating English to FOL…

- There are exactly two purple mushrooms.

```
(∃ x)(∃ y) mushroom(x) ∧ purple(x) ∧
  mushroom(y) ∧ purple(y) ∧ ¬(x=y) ∧
    [(∀ z) (mushroom(z) ∧ purple(z)) =>
              ((x=z) ∨ (y=z))]
```

- Deb is not tall.

```
¬tall(Deb)
```

Translating English to FOL…

- X is above Y if (and only if) X is on directly on top of Y or else there is a pile of one or more other objects directly on top of one another starting with X and ending with Y.

$$(\forall\ x)(\forall\ y)\ \mathbf{above}(x,y) \Leftrightarrow (\mathbf{on}(x,y) \vee (\exists\ z)\ (\mathbf{on}(x,z) \wedge \mathbf{above}(z,y)))$$

Equality

$term_1 = term_2$ is true under a given interpretation
if and only if $term_1$ and $term_2$ refer to the same object

E.g., $1 = 2$ and $\forall x \ \times(Sqrt(x), Sqrt(x)) = x$ are satisfiable
$2 = 2$ is valid

E.g., definition of (full) $Sibling$ in terms of $Parent$:

$$\forall x, y \ \ Sibling(x, y) \ \Leftrightarrow \ [\neg(x = y) \land \exists m, f \ \ \neg(m = f) \land Parent(m, x) \land Parent(f, x) \land Parent(m, y) \land Parent(f, y)]$$

How to Use First Order Logic

- Assertions are “facts” we TELL the Knowledge-based Agent
- AXIOMS: are the “built-in” rules of the domain
- THEOREMS: are the additional sentences that are entailed by the AXIOMS
- AXIOMS should be “minimal” in the sense that they capture ALL the necessary properties of the domain, but do not include any extra assertions
- AXIOMS could be used to describe both finite and infinite domains

Natural Numbers

- E.g., Peano's axioms for natural numbers:

Natural(0)

$\forall$ x Natural (x) => Natural(Successor(x))

RESTRICTING THE Successor function:

$\forall$ n $\neg$ (0 = Successor(n))

$\forall$ m, n $\neg$ (m = n) => $\neg$ (Successor(m) = Successor(n))

DEFINING + in terms of Successor

$\forall$ x Natural (x) => +(0, x)

$\forall$ x,y Natural (x) $\wedge$ Natural(y) =>

+(Successor(x), y) = Successor(+(x, y))

notice we use PREFIX notation "+(x, y)" rather than infix "x+y"

Sets

The set domain – an important one in the history of Logic/Math: defined in terms of membership and adjoinment

- $\forall s \quad Set(s) \Leftrightarrow (s = \{\}) \vee (\exists x,s_2\ Set(s_2) \wedge s = \{x|s_2\})$
- $\neg\exists x,s \quad \{x|s\} = \{\}$
- $\forall x,s \quad x \in s \Leftrightarrow s = \{x|s\}$
- $\forall x,s \quad x \in s \Leftrightarrow [\ \exists y,s_2\ (s = \{y|s_2\} \wedge (x = y \vee x \in s_2))]$
- $\forall s_1,s_2 \quad s_1 \subseteq s_2 \Leftrightarrow (\forall x\ x \in s_1 \Rightarrow x \in s_2)$
- $\forall s_1,s_2 \quad (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$
- $\forall x,s_1,s_2 \quad x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2)$
- $\forall x,s_1,s_2 \quad x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$

Kinship Domain

- Male and Female are disjoint

  $\forall x\ \mathit{Male}(x) \Rightarrow \mathit{not}\ \mathit{Female}(x)$

- Parent and Child are inverse relationships

  $\forall p, c\ \ \mathit{Parent}(p, c) \Leftrightarrow \mathit{Child}(c, p)$

- Definition of Sibling (siblings share a parent)

  $\forall x,y\ \mathit{Sibling}(x,y) \Leftrightarrow \mathrm{not}(x = y) \wedge\ (\exists\, p\ \ \mathit{Parent}(p, x) \wedge \mathit{Parent}(p, y)\ )$

- Full definition of Sibling (both mother and father are the same)

  $\forall x,y\ \mathit{Sibling}(x,y) \Leftrightarrow \mathrm{not}(x = y) \wedge$

  $(\exists\, m,f\ \ \mathit{not}(m = f) \wedge \mathit{parent}(m, x) \wedge \mathit{parent}(m, y) \wedge \mathit{parent}(f, x) \wedge \mathit{parent}(f, y)\ )$

One's mother is one's female parent

$\forall m,c\ \mathit{Mother}(c) = m \Leftrightarrow (\mathit{Female}(m) \wedge \mathit{Parent}(m,c))$

AXIOMS vs THEOREMS

- Take the Kinship Domain:

- "Sibling" is symmetric

  $\forall x,y \; \mathit{Sibling}(x,y) \Leftrightarrow \mathit{Sibling}(y,x)$

- Is this an axiom or a theorem?

Can be derived from the axioms of Kinship from prior page:

$$\forall x,y \; \mathit{Sibling}(x,y) \Leftrightarrow \text{not}(x = y) \wedge \; (\exists\, p \;\; \mathit{Parent}(p, x) \wedge \mathit{Parent}(p, y)\,)$$

So, the symmetry assertion is not a axiom. But, it is a "theorem" because the Axioms entail the symmetry assertion

Logical agents for the Wumpus world

Remember: generic knowledge-based agent:

```
function KB-AGENT(percept) returns an action
   static: KB, a knowledge base
           t, a counter, initially 0, indicating time

   TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
   action ← ASK(KB, MAKE-ACTION-QUERY(t))
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

1. TELL KB what was perceived
   Uses a KRL to insert new sentences, representations of facts, into KB

2. ASK KB what to do.
   Uses logical reasoning to examine actions and select best.

Using the FOL Knowledge Base

Suppose a wumpus-world agent is using an FOL KB
and perceives a smell and a breeze (but no glitter) at $t = 5$:

$\textsc{Tell}(KB, Percept([Smell, Breeze, None], 5))$
$\textsc{Ask}(KB, \exists\, a\ \ Action(a, 5))$

I.e., does the KB entail any particular actions at $t = 5$?

Answer: $Yes,\ \{a/Shoot\}$ $\leftarrow$ substitution (binding list)

Set of solutions

Given a sentence S and a substitution σ,
$S\sigma$ denotes the result of plugging σ into S; e.g.,
$S = Smarter(x, y)$
$\sigma = \{x/Hillary, y/Bill\}$
$S\sigma = Smarter(Hillary, Bill)$

$\textsc{Ask}(\mathrm{KB}, \mathrm{S})$ returns some/all σ such that $KB \models S\sigma$

Wumpus world, FOL Knowledge Base

"Perception"

$\forall\, b, g, t \;\; Percept([Smell, b, g], t) \;\Rightarrow\; Smelt(t)$

$\forall\, s, b, t \;\; Percept([s, b, Glitter], t) \;\Rightarrow\; AtGold(t)$

Reflex: $\forall\, t \;\; AtGold(t) \;\Rightarrow\; Action(Grab, t)$

Reflex with internal state: do we have the gold already?

$\forall\, t \;\; AtGold(t) \land \neg Holding(Gold, t) \;\Rightarrow\; Action(Grab, t)$

$Holding(Gold, t)$ cannot be observed

$\Rightarrow$ keeping track of change is essential

Describing actions

"Effect" axiom—describe changes due to action

$\forall s \;\; AtGold(s) \Rightarrow Holding(Gold, Result(Grab, s))$

"Frame" axiom—describe non-changes due to action

$\forall s \;\; HaveArrow(s) \Rightarrow HaveArrow(Result(Grab, s))$

May result in too many frame axioms

Frame problem: find an elegant way to handle non-change

(a) representation—avoid frame axioms

(b) inference—avoid repeated "copy-overs" to keep track of state

Qualification problem: true descriptions of real actions require endless caveats—what if gold is slippery or nailed down or ...

Ramification problem: real actions have many secondary consequences—what about the dust on the gold, wear and tear on gloves, ...

Describing actions (cont'd)

<u>Successor-state axioms</u> solve the representational frame problem

Each axiom is "about" a <u>predicate</u> (not an action per se):

P true afterwards $\Leftrightarrow$ [an action made P true

$\lor$ P true already and no action made P false]

For holding the gold:

$$\forall a, s \;\; Holding(Gold, Result(a, s)) \;\Leftrightarrow$$
$$[(a = Grab \land AtGold(s))$$
$$\lor\; (Holding(Gold, s) \land a \neq Release)]$$

Diagnostic Rules

- Lead from observed effects to POTENTIAL or HIDDEN causes

In the Wumpus world for example:

$$\forall s \text{ Breezy}(s) \Rightarrow \exists r \text{ Adjacent}(r, s) \wedge \text{Pit}(r)$$

$$\forall s \neg\text{Breezy}(s) \Rightarrow \neg \exists r \text{ Adjacent}(r, s) \wedge \text{Pit}(r)$$

Leads to the following bidirectional sentence:

$$\forall s \text{ Breezy}(s) \Leftrightarrow \exists r \text{ Adjacent}(r, s) \wedge \text{Pit}(r)$$

- Diagnostic rules provide possible explanations for what one observes or know to be the case

Causal Rules

- Causal rules reflect the EFFECT or IMPACT of observations
- Certain conditions "cause" or will result in the other conditions becoming true
- For example, KNOWING there is a pit in a location, will cause the adjacent locations to become breezy:

  $\forall x \ \ Pit(x) \Rightarrow \forall\, y\ Adjacent(x, y) \Rightarrow Breezy(y)$

- If the adjacent cells to a cell are all pitless, then there is NO breeze in that cell:

  $\forall x \ \ [\forall\, y\ Adjacent(x, y) \Rightarrow \neg Pit(y)\ ] \Rightarrow \neg Breezy(x)$

One can work with Causal or Diagnostic styles of rules

Higher-order logic?

- First-order logic allows us to quantify over objects (= the first-order entities that exist in the world).

- Higher-order logic also allows **quantification over relations and functions.**

  e.g., "two objects are equal iff all properties applied to them are equivalent":

  $\forall\ x,y \quad (x=y) \Leftrightarrow (\forall\ \mathbf{p}, \mathbf{p}(x) \Leftrightarrow \mathbf{p}(y))$

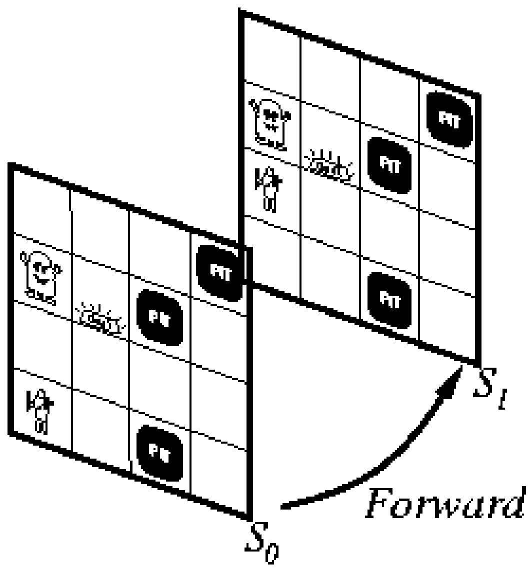
- Higher-order logics are more expressive than first-order; however, so far we have little understanding on how to effectively and automatically reason with sentences in higher-order logic.

Situation calculus

Facts hold in <u>situations</u>, rather than eternally
E.g., $Holding(Gold, Now)$ rather than just $Holding(Gold)$

<u>Situation calculus</u> is one way to represent change in FOL:
Adds a situation argument to each non-eternal predicate
E.g., Now in $Holding(Gold, Now)$ denotes a situation

Situations are connected by the $Result$ function
$Result(a, s)$ is the situation that results from doing a is s



Planning

Initial condition in KB:

$$At(Agent, [1,1], S_0)$$
$$At(Gold, [1,2], S_0)$$

Query: $\text{ASK}(KB, \exists\, s \;\; Holding(Gold, s))$

i.e., in what situation will I be holding the gold?

Answer: $\{s/Result(Grab, Result(Forward, S_0))\}$

i.e., go forward and then grab the gold

This assumes that the agent is interested in plans starting at S_0 and that S_0 is the only situation described in the KB

Generating action sequences

Represent plans as action sequences $[a_1, a_2, \ldots, a_n]$

$PlanResult(p, s)$ is the result of executing p in s

Then the query $\text{Ask}(KB, \exists p \;\; Holding(Gold, PlanResult(p, S_0)))$ has the solution $\{p/[Forward, Grab]\}$

Definition of $PlanResult$ in terms of $Result$:

$\forall s \;\; PlanResult([\,], s) = s$ [ ] = empty plan

$\forall a, p, s \;\; PlanResult([a|p], s) = PlanResult(p, Result(a, s))$

Recursively continue until it gets to empty plan [ ]

Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner

Summary of FOL

First-order logic:
- objects and relations are semantic primitives
- syntax: constants, functions, predicates, equality, quantifiers

Increased expressive power: sufficient to define wumpus world

Situation calculus:
- conventions for describing actions and change in FOL
- can formulate planning as inference on a situation calculus KB

Knowledge Representation

- Knowledge engineering: principles and pitfalls
- Ontologies
- Examples

Knowledge Engineer

- Populates KB with facts and relations
- Must study and understand domain to pick important objects and relationships
- Main steps:
  - Decide what to talk about
  - Decide on vocabulary of predicates, functions & constants
  - Encode general knowledge about domain
  - Encode description of specific problem instance
  - Pose queries to inference procedure and get answers

Knowledge engineering vs. programming

	Knowledge Engineering	Programming
1.	Choosing a logic	Choosing programming language
2.	Building knowledge base	Writing program
3.	Implementing proof theory	Choosing/writing compiler
4.	Inferring new facts	Running program

Why knowledge engineering rather than programming?

Less work: just specify objects and relationships known to be true, but leave it to the inference engine to figure out how to solve a problem using the known facts.

Properties of good knowledge bases

- Expressive
- Concise
- Unambiguous
- Context-insensitive
- Effective
- Clear
- Correct
- …

Trade-offs: e.g., sacrifice some correctness if it enhances brevity.

Efficiency

- Ideally: Not the knowledge engineer's problem

  The inference procedure should obtain same answers no matter how knowledge is implemented.

  Focus on human understanding of the KB – not unlike programming style

- In practice:
  - use automated optimization
  - knowledge engineer should have some understanding of how inference is done

Pitfall: design KB for human readers

- KB should be designed primarily for inference procedure!
- e.g., *VeryLongName* predicates:

BearOfVerySmallBrain(Pooh) does not allow inference procedure to infer that Pooh is a bear, an animal, or that he has a very small brain, …

In other words:
BearOfVerySmallBrain(pooh) = x(pooh)

Rather, use:

Bear(Pooh)
$\forall$ b, Bear(b) $\Rightarrow$ Animal(b)
$\forall$ a, Animal(a) $\Rightarrow$PhysicalThing(a)
…
[See AIMA book for full treatment and example]

Debugging

- In principle, easier than debugging a program,

  because we can look at each logic sentence ***in isolation*** *and tell whether it is correct.*

Example:

$\forall$ x, Animal(x) $\Rightarrow$ $\exists$ b, BrainOf(x) = b

means

"there is some object that is the value of the BrainOf function applied to an animal"

and can be corrected to mean

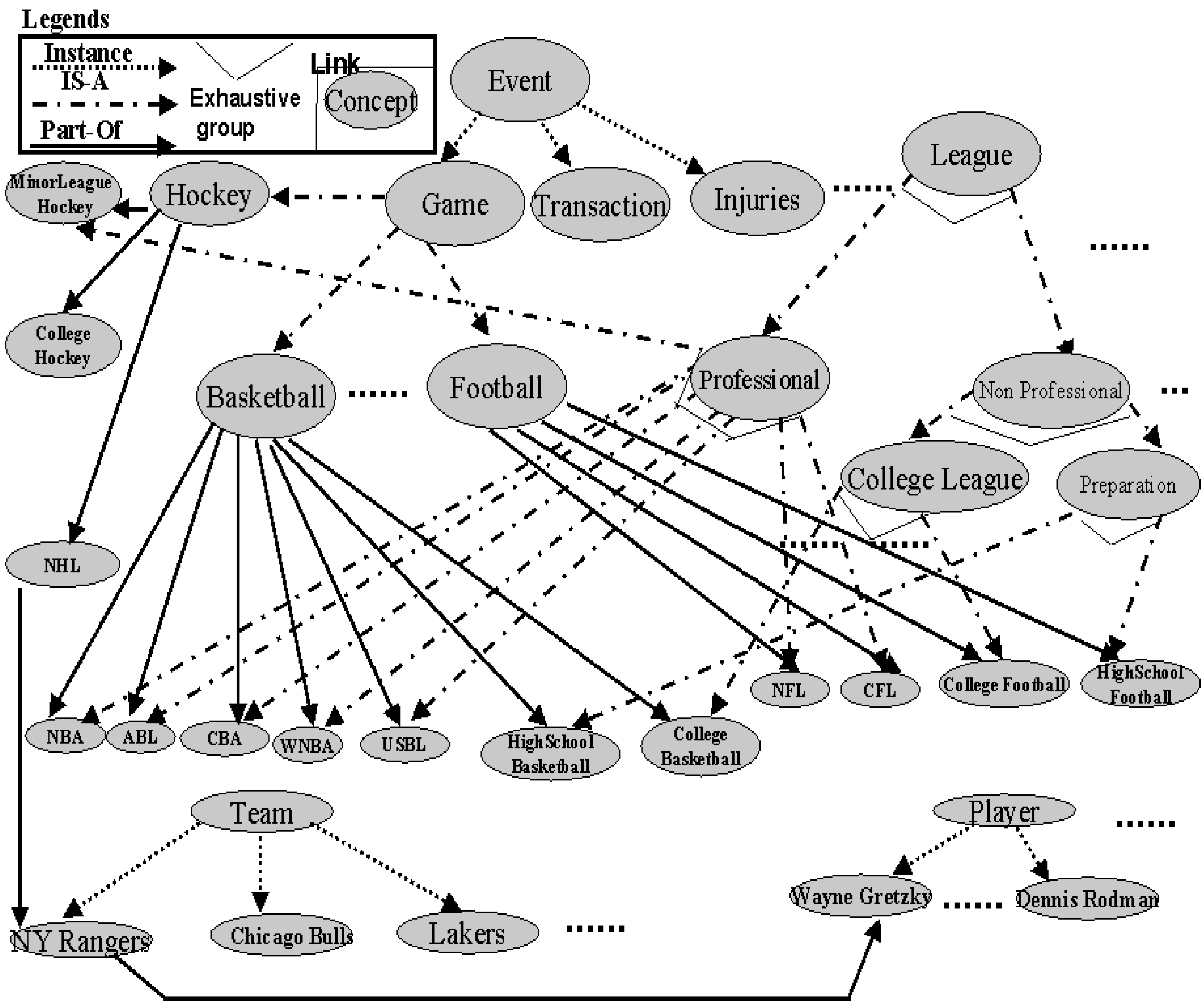
"every animal has a brain"

without looking at other sentences.

Ontology

- Collection of concepts and inter-relationships
- Widely used in the database community to “translate” queries and concepts from one database to another, so that multiple databases can be used conjointly (database federation)

Ontology Example



Khan & McLeod, 2000

Towards a general ontology

Develop good representations for:

- categories
- measures
- composite objects
- time, space and change
- events and processes
- physical objects
- substances
- mental objects and beliefs
- …

Representing Categories

- We interact with individual objects, but…
  much of reasoning takes place at the level of categories.

- **Representing categories in FOL:**
  - use unary predicates
    e.g., *Tomato(x)*

  -in a table form (small set of objects)
  -based on its properties

  - reification: turn a predicate or function into an object
    e.g., use constant symbol Tomatoes to refer to **set** of all tomatoes
    "*x* is a tomato" expressed as "$x \in$ *Tomatoes*"

- Strong property of reification: can make assertions about reified category itself rather than its members
  e.g., *Population(Humans)* = 5e9

Categories: inheritance

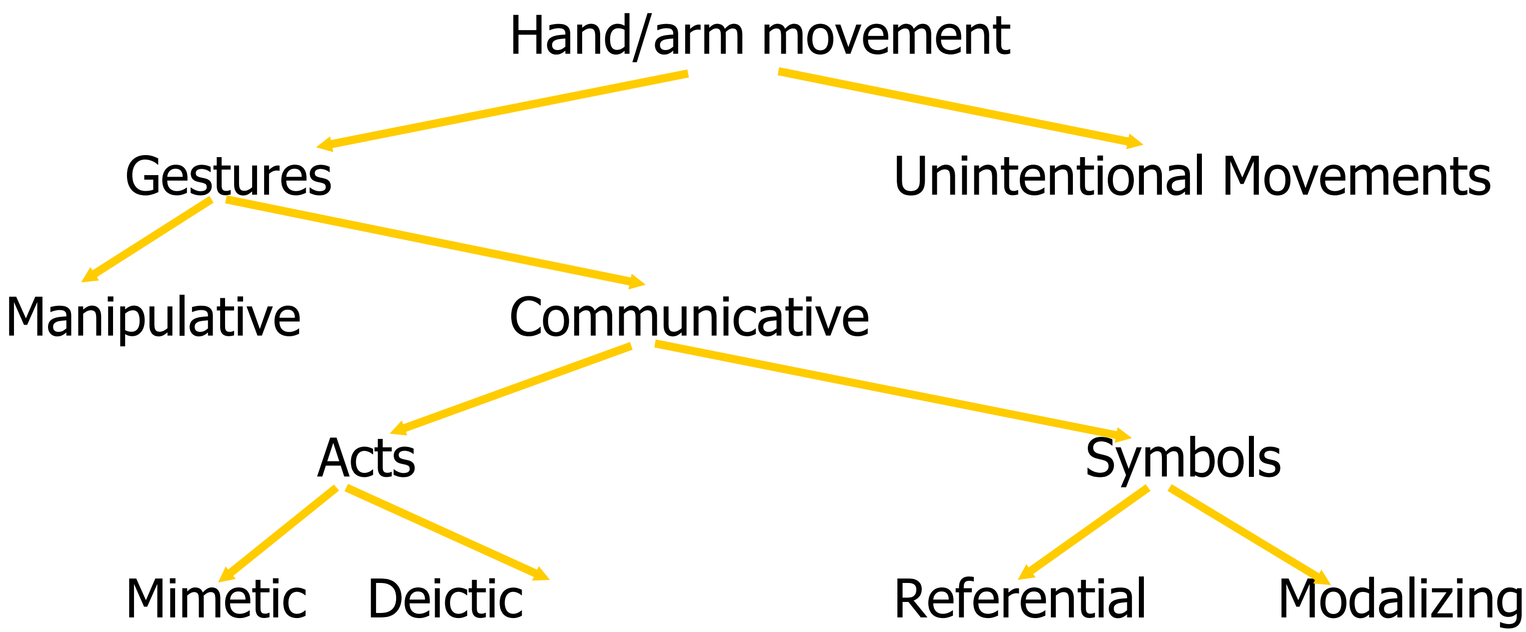
- Allow to organize and simplify knowledge base

  e.g., if all members of category *Food* are edible
  and *Fruits* is a subclass of *Food*
  and *Apples* is a subclass of *Fruits*
  then we know (through inheritance) that apples are edible.

- Taxonomy: hierarchy of subclasses

- Because categories are sets, we handle them as such.
  e.g., two categories are disjoint if they have no member in common
  a disjoint exhaustive decomposition is called a partition
  etc…

Example: Taxonomy of hand/arm movements



Quek,1994, 1995.

Measures

- Can be represented using units functions
  e.g., $Length(L_1) = Inches(1.5) = Centimeters(3.81)$

- Measures can be used to describe objects
  e.g., $Mass(Tomato_{12}) = Kilograms(0.16)$

- Caution: be careful to distinguish between measures and objects
  e.g., $\forall b,\ b \in DollarBills \Rightarrow CashValue(b) = \$(1.00)$

Composite Objects – Also Called AGGREGATION

- One object can be part of another.

- PartOf relation is transitive and reflexive:
  e.g., PartOf(Bucharest, Romania)
  PartOf(Romania, EasternEurope)
  PartOf(EasternEurope, Europe)

Then we can infer Part Of(Bucharest, Europe)

- Composite object: any object that has parts

Composite Objects (cont.)

- Categories of composite objects often characterized by their structure, i.e., what the parts are and how they relate.

e.g., $\forall a \; Biped(a) \Rightarrow$

$$
\begin{aligned}
&\exists\, ll, lr, b \\
&Leg(ll) \wedge Leg(lr) \wedge Body(b) \wedge \\
&PartOf(ll, a) \wedge PartOf(lr, a) \wedge PartOf(b, a) \wedge \\
&Attached(ll, b) \wedge Attached(lr, b) \wedge \\
&ll \neq lr \wedge \\
&\forall x \; Leg(x) \wedge PartOf(x, a) \Rightarrow (x = ll \vee x = lr)
\end{aligned}
$$

- Such description can be used to describe any objects, including events. We then talk about schemas and scripts.

Events

- Chunks of spatio-temporal universe

e.g., consider the event WorldWarII

it has parts or sub-events: SubEvent(BattleOfBritain, WorldWarII)

it can be a sub-event: SubEvent(WorldWarII, TwentiethCentury)

- Intervals: events that include as sub-events all events occurring in a given time period (thus they are temporal sections of the entire spatial universe).

- Cf. situation calculus: fact true in particular situation
  event calculus: event occurs during particular interval

Events (cont.)

- Places: spatial sections of the spatio-temporal universe that extend through time
- Use In(x) to denote sub-event relation between places; e.g. In(NewYork, USA)

- Location function: maps an object to the smallest place that contains it:

$\forall x,l\ Location(x) = l \Leftrightarrow At(x, l) \wedge \forall ll\ At(x, ll) \Rightarrow In(l, ll)$

Times, Intervals and Actions

- Time intervals can be partitioned between moments (=zero duration) and extended intervals:
- Absolute times can then be derived from defining a time scale (e.g., seconds since midnight GMT on Jan 1, 1900) and associating points on that scale with events.
- The functions Start and End then pick the earliest and latest moments in an interval. The function Duration gives the difference between end and start times.

$\forall i\ Interval(i) \Rightarrow Duration(i) = (Time(End(i)) - Time(Start(i)))$

$Time(Start(AD1900)) = Seconds(0)$

$Time(Start(AD1991)) = Seconds(2871694800)$

$Time(End(AD1991)) = Seconds(2903230800)$

$Duration(AD1991) = Seconds(31536000)$

Times, Intervals and Actions (cont.)

- Then we can define predicates on intervals such as:

$\forall i, j\ Meet(i, j) \Leftrightarrow Time(End(i)) = Time(Start(j))$

$\forall i, j\ Before(i, j) \Leftrightarrow Time(End(i)) < Time(Start(j))$

$\forall i, j\ After(j, i) \Leftrightarrow Before(i ,j)$

$\forall i, j\ During(i, j) \Leftrightarrow Time(Start(j)) \leq Time(Start(i)) \wedge$
$Time(End(j)) \geq Time(End(i))$

$\forall i, j\ Overlap(i, j) \Leftrightarrow \exists k\ During(k, i) \wedge During(k, j)$

Objects Revisited

- It is legitimate to describe many objects as events
- We can then use temporal and spatial sub-events to capture changing properties of the objects

e.g.,

Poland event

19thCenturyPoland temporal sub-event

CentralPoland spatial sub-event

We call **fluents** objects that can change across situations.

Substances and Objects

- Some objects cannot be divided into distinct parts –

  e.g., butter: one butter? no, some butter!

$\Rightarrow$ butter substance (and similarly for temporal substances)

(simple rule for deciding what is a substance: if you cut it in half, you should get the same).

How can we represent substances?

- Start with a category

  e.g., $\forall x,y \quad x \in \text{Butter} \wedge \text{PartOf}(y, x) \Rightarrow y \in \text{Butter}$

- Then we can state properties

  e.g., $\forall x \ \text{Butter}(x) \Rightarrow \text{MeltingPoint}(x, \text{Centigrade}(30))$