Artificial Intelligence
CS 6364

Professor Dan Moldovan

Section 10
Knowledge Representation
Outline

- Types of Knowledge
- Knowledge representation principles
- Methodology for constructing KB
- Ontologies
- Composite objects
- Measures
- Situation calculus
- Events
Types of Knowledge

- Some principles for organizing a KB
- A methodology for constructing a KB
- An example: Electronic circuits domain

Here are some types of K we will be referring to:
- Declarative Knowledge—Knowledge in the form of assertions.
- Procedural Knowledge—Knowledge that specifies what to do and when. Production rule is a good example.
- Inferential Knowledge—Knowledge that is not explicitly stated in a knowledge base, but that could be derived (inferred) by applying inferential (or reasoning) procedure.
- Inherited Knowledge—Knowledge that is inferred via inheritance reasoning mechanism (more specific concepts inherit properties of more general concepts).
Type of knowledge

- Temporal Knowledge—Knowledge that deals with time (not that changes over time).
  
  Example: Tom had a drink before the game.

- Acquired Knowledge—Knowledge that is gathered either automatically by a K acquisition system or by a K engineer. Acquired K must be merged with existing KB
Knowledge Representation Principles

A good KR system should possess the following properties:

1. Representational Adequacy—is the ability to represent all kinds of knowledge that are needed in a domain. Representation must capture semantic meaning.

2. Inferential Adequacy—is the ability to manipulate old knowledge to derive new knowledge.

3. Inferential Efficiency—is the ability to derive new knowledge fast and accurate.

4. Acquisitional Efficiency—is the ability to acquire new information easily—and merge it with existing information.
Knowledge Representation Principles

These are some broad principles.

When building a KB, a knowledge engineer should code the K in general terms, so that assertions can be used across domains. In other words, the KR should respect the generality principle. The fewer predicates, the better.

It is wrong to code

Bear of Very Small Brain (Pooh)

This predicate will probably not be used in another domain.

Here is how to build a good KB for this sentence. It is long, but general.

1. Pooh is a bear; bears are animals; animals are physical things.

   Bear (Pooh)
   ∀b Bear(b) ⇒ Animal (b)
   ∀a Animal(a) ⇒ Physical Thing (a)
Knowledge Representation Principles

2. Pooh has a very small brain.
   \[
   \text{Relative size (Brain of (Pooh), Brain of (TypicalBear))} = \text{Very(Small)}.
   \]

3. All animals and only animals have a brain, which is a part of the animal.
   \[
   \forall a \text{ Animal}(a) \Leftrightarrow \text{Brain (Brain of (a))} \quad \forall a \text{ Part of (Brain of (a), a)}
   \]

4. If something is part of a physical thing, then is also a physical thing.
   \[
   \forall x,y \text{ Partof}(x,y) \land \text{PhysicalThing}(y) \Rightarrow \text{PhysicalThing}(x)
   \]

5. Animals with brains that are small relative to a normal brain size for their species are silly.
   \[
   \forall a \text{ Relative Size(Brain of (a), Brain of (Typical Member(} \quad \text{(Species of(a)))))} \leq \text{small} \Rightarrow \text{silly(a)}
   \]
Knowledge Representation Principles

6. Every physical thing has a size. Sizes are arranged on a scale from Tiny to Huge. A relative size is a ratio of two sizes.

   ∀x PhysicalThing(x) ⇒ ∃s Size(x) = s

   Tiny < Small < Medium < Large < Huge

   ∀a,b Relative Size (a,b) = size(a)/size(b)

7. The function Very maps a point on a scale to a more extreme value. Medium is the neutral value for a scale.

   Medium = 1

   ∀x x > Medium ⇒ Very(x) > x

   ∀x x < Medium ⇒ Very(x) < x
A Methodology for Constructing A KB

Step 1. Decide what to talk about. In this step, the K. engineer has to identify which type of information needs to be represented.

Step 2. Decide on a vocabulary of predicates, functions and constants. In this step, the K engineer maps the information that needs to be represented (as found on step 1) into a small set of predicates, functions and constants. This is done by respecting the KR. principles outlined above. The ontology of a domain is this vocabulary (or set of predicates, functions, constants) that is structured-or classified into a hierarchy.

Step 3. Encode general K. about domain. This step refers to generation axioms and procedural knowledge that refer to a specific domain. Often, common sense K. is needed to be able to reason.
A Methodology for Constructing A KB

Step 4. Encode a description of the specific problem instance.
   In this step, the description of the problem that needs to be solved is mapped into a K.R.

Step 5. Pose queries to the inference procedure and get answers.
   In this step, the system performs reasoning and inferential K. is discovered.
An Example: Verification of an Electronic Circuit

Step 1. Key elements of the domain: We need to talk about circuits, terminals, signals at the terminals, gate types, and individual gates.

Step 2. Vocabulary of predicates, functions, and constants consists of:

| Terminals | $\text{In}(1, X_1)$ ; $\text{Out}(1, X_1)$ |
| Signals   | $\text{Signal} (t)$ |
| Gate Types| AND, OR, NOT, XOR are constants |
|           | Type $(X_1, \text{XOR})$ is a function |
| Circuit   | Connected $(\text{Out}(1, X_1), \text{In}(1, X_2))$ is a function |

Figure 8.1 A digital circuit $C_1$, with three inputs and two outputs, containing two XOR gates, two AND gated and one OR gate. The inputs are bit values to be added, and the outputs are the sum bit and the carry bit.
An Example: Verification of an Electronic Circuit

**Step 3.** Encode general knowledge.

1. If two terminals are connected, then they have the same signal:
   \[ \forall t_1, t_2 \text{ Connected}(t_1, t_2) \Rightarrow \text{Signal}(t_1) = \text{signal}(t_2) \]

2. The signal at every terminal is either on or off, but not both:
   \[ \forall t \text{ Signal}(t) = \text{On} \lor \text{signal}(t) = \text{off} \]
   \[ \text{on} \neq \text{Off} \]

3. Connected is a commutative predicate:
   \[ \forall t_1, t_2 \text{ Connected}(t_1, t_2) \Leftrightarrow \text{Connected}(t_2, t_1) \]

4. An OR gate’s output is on if and only if any of its inputs are on.
   \[ \forall g \text{ Type}(g) = \text{OR} \Rightarrow \]
   \[ \text{Signal}(\text{Out}(1, g)) = \text{On} \Leftrightarrow \exists n \text{ Signal}(\text{In}(n, g)) = \text{On} \]

5. An AND gate’s output is off if and only if its inputs are off.
   \[ \forall g \text{ Type}(g) = \text{AND} \Rightarrow \]
   \[ \text{Signal}(\text{Out}(1, g)) = \text{off} \Leftrightarrow \exists n \text{ Signal}(\text{In}(n, g)) = \text{Off} \]
An Example: Verification of an Electronic Circuit

6. An XOR gate’s output is on if its inputs are different:
   \[ \forall g \text{ Type}(g) = \text{XOR} \Rightarrow \]
   \[ \text{Signal}(\text{Out}(1,g)) = \text{On} \Leftrightarrow \text{Signal}(\text{In}(1,g)) \neq \text{Signal}(\text{In}(2,g)) \]

7. A NOT gate’s output is different from it input:
   \[ \forall g \ (\text{Type}(g) = \text{NOT}) \Rightarrow \text{Signal}(\text{Out}(1,g)) \neq \text{Signal}(\text{In}(1,g)) \]

**Step 4.** Encode specific instance.

- Type \((X_1) = \text{XOR}\)
- Type \((X_2) = \text{XOR}\)
- Type \((A_1) = \text{AND}\)
- Type \((A_2) = \text{AND}\)
- Type \((O_1) = \text{OR}\)

Connections:
- Connected \((\text{Out}(1,X_1), \text{In}(1, X_2))\)
- Connected \((\text{In}(1,C_1), \text{In}(1, X_1))\)
An Example: Verification of an Electronic Circuit

**Step 5.** Pose queries and reason

Ex: What combination of inputs would cause the first output of $C_1$ (sum bit) to be off and the second output of $C_1$ (carry) to be on?

$$\exists i_1, i_2, i_3 \text{ Signal(In}(1,C_1)) = i_1 \land$$

$$\text{Signal(In}(2,C_1)) = i_2 \land$$

$$\text{Signal(In}(3,C_1)) = i_3 \land$$

$$\text{Signal(Out}(1,C_1)) = \text{off} \land$$

$$\text{Signal(Out}(2,C_1)) = \text{On}$$

For circuit verification, that is to verify that system provides correct outputs for given inputs, we have to write the complete circuit equation (stage sentence) followed by applications of resolution inference procedure.
Ontologies

- Large KB can be constructed using layers of ontologies
- An ontology is a classification of concepts
  - Upper ontology — most general concepts (general framework of concepts)
  - Middle ontology — general purpose concepts, such as those in WordNet
  - Lower ontology — domain specific concepts
Upper Ontology

Anything

AbstractObjects
- Sets
- Categories
- Sentences
- Measurements
- Times
- Weights

RepresentationalObjects
- Places
- Intervals
- Moments
- Animals
- Agents

GeneralizedEvents
- Numbers
- PhysicalObjects
- Processes
- Stuff
- Liquid
- Solid
- Gas

PhysicalObjects
- Moments
- Things
- Humans

Categories

Processes

Numbers
High Level Ontology of Sorts

Entity (ent)

<table>
<thead>
<tr>
<th>Situation (si)</th>
<th>Quantity (qn)</th>
<th>Object (o)</th>
<th>Quality (ql)</th>
<th>Descriptor (des)</th>
</tr>
</thead>
<tbody>
<tr>
<td>State (st)</td>
<td>Event (ev)</td>
<td>Concrete (co)</td>
<td>Abstract (ao)</td>
<td>Temporal (tmp)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Local (loc)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Animated (aco)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inanimated (aco)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Temporal (tao)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Non temporal (ntao)</td>
</tr>
</tbody>
</table>
High Level Ontology of Sorts

- **Entities**: all things about which something can be said.
  - **Objects**
    - Concrete: occupy space, are touchable, tangible
      - Animated: have life, vigor, spirit
        - *John, the girl*
      - Inanimated: dull, without life
        - *Table, computer*
    - Abstract: intangible, they are somehow product of human reasoning
      - Temporal: *last week, July*
      - Non-temporal: *justice, pain, odor*
  - **Situations**: anything that happens at a time and place
    - Events: imply a change in the status of other entities
      - *Mix, grow, conference, hurricane*
    - States: do not imply a change
      - *Standing next to the door, account for 10% of the sales*
High Level Ontology of Sorts

- Descriptors:
  - They complement Entities by stating properties about their spatial or temporal context
    - Local: *on the roof, near the Stadium*
    - Temporal: *by the end of the day, on July 15th*
- Quantities:
  - *a few, two, 22*
- Qualities:
  - *heavy, difficult*
Categories and Objects

- Categories are classes or sets of objects that have some common properties
  - Example: basketballs, children, dogs
- Objects are members of a category
  \[ BB_9 \in \text{Basketballs} \]
- Two ways of representing categories
  1. using unary predicates: \( \text{basketball}(b) \)
  2. reify the category as an object: \( \text{basketballs} \)
     \[ \text{Member}(b,\text{Basketballs}) \]

Reification is the process of regarding something abstract as a material thing, ie in AI transform a category into an object
Categories and Objects

- An object is a member of a category
  \[ BB_9 \in \text{Basketballs} \]
- A category can be a subclass of another category (Subset relation)
  \[ \text{Basketballs} \subset \text{Balls} \]
- All member of a category have the same properties
  \[ x \in \text{Basketballs} \Rightarrow \text{Round}(x) \]
- Members of a category can be recognized by some properties
  \[ \text{Orange}(x) \land \text{Round}(x) \land \text{Diameter}(x) = 9.5'' \land x \in \text{Balls} \Rightarrow x \in \text{Basketballs} \]
- A category as a whole has some properties
  \[ \text{Dogs} \in \text{DomesticatedAnimals} \]
Categories and Objects

- **Inheritance** is a property through which more specific concepts adopt the properties of more general concepts

  \[ \text{Apple} \subset \text{Fruit} \subset \text{Food} \]

  Edibility is a property of Food inherited by Fruit and Apple

  More specific concepts add new properties to differentiate from their subsumers
Composite Objects

- Objects may belong to categories by virtue of their constituent architecture

- Example: cars have wheels, car engine, windshield

How do we represent structure?
Composite Objects

- The PartOf relation → transitive
  → reflexive

  • There are PartOf hierarchies similar to Subset hierarchies

Example:

PartOf(Bucharest, Romania)
PartOf(Romania, EasternEurope)
PartOf(EasternEurope, Europe)

PartOf(Bucharest, Europe)
Structure and Composite Objects

• An object that has parts is a composite object
Structure and Composite Objects

⇒ Composite objects:
Europe, Eastern-Europe, Western-Europe, Romania, Hungary, France

• Composite objects are the nodes of the hierarchy.
• Not the leaves of the PartOf hierarchy
• Structure of composite objects shows the parts and how they are related
Structure Example

- Biped has 2 legs that are attached to its body

∀a Biped(a) ⇒

∃l₁, l₂, b  Leg(l₁) ∧ Leg(l₂) ∧ Body(b) ∧

PartOf(l₁,a) ∧ PartOf(l₂,a) ∧ PartOf(b,a) ∧

Attached(l₁,b) ∧ Attached(l₂,b) ∧ l₁≠l₂ ∧

[∀l₃ Leg(l₃) ∧ PartOf(l₃,a) ⇒

(l₃=l₁ ∨ l₃=l₂)]
It is also helpful to define composite objects with definite parts but *no particular structure*

- *The apples in this bag weigh two pounds*

Temptation: ascribe the weight to the set of apples in the bag \(\rightarrow\) mistake, because the set is an abstract mathematical concept that has elements but does not have weight

\[\Rightarrow\] We need a new concept \(\rightarrow\) called *bunch*

Example: If the apples are Apple\(_1\), Apple\(_2\), Apple\(_3\) then

\[\text{BunchOf}\{\text{Apple}_1, \text{Apple}_2, \text{Apple}_3\}\]

denotes the composite object with three apples as parts (not elements)
We can define **BunchOf** in terms of the **PartOf** relation.

Each element of $S$ is part of $\text{BunchOf}(S)$

$$\forall x \ x \in S \Rightarrow \text{PartOf}(x, \text{BunchOf}(S))$$

$\text{BunchOf}(S)$ is the smallest object satisfying this condition. $\text{BunchOf}(S)$ must be part of any object that has all the elements of $S$ as parts:

$$\forall y \ [\forall x \ x \in S \Rightarrow \text{PartOf}(x, y)] \Rightarrow \text{PartOf}(\text{BunchOf}(S), y)$$
Measures

Useful properties to represent
- mass
- age
- cost

relate objects to quantities of particular types

represent them in logic
relate to units of measure
Quantitative Measures

• The universe includes abstract “measure objects”

Example: length

\[ \text{Length}(L1) = \text{Inches}(1.5) = \text{Centimeters}(3.81) \]

• Units function
Conversion between Units

∀l Centimeters(2.54 * l) = Inches(l)
∀t Centigrade(t) = Fahrenheit(32 + 1.8 * t)

Similar conversions for other measures

Mass(Tomato_{12}) = Kilograms(0.16)
Price(Tomato_{12}) = $(0.32)
∀d ∈ Days ⇒ Duration(d) = Hours(24)
Important Conversion Hints

- It is important to distinguish between monetary amounts and monetary instruments
  \[ \forall b \in \text{DollarBills} \implies \text{CashValue}(b) = \$(1.00) \]

- It is more difficult to represent measures that do not have an agreed scale of values
  
  Example

  - difficult homework?
  - desert that have deliciousness?
  - poems that have beauty?

  Cannot assign numbers
Solutions for Non-Quantitative Measures

1) Dismiss properties that cannot be measured on a scale

2) Attempt to impose numerical scales on beauty, difficulty,...

3) Order/compare the measures ← this is the solution

Use $<$ as an ordering symbol. How do we formalize the belief that

Norvig’s exercises are tougher than Russell’s?
Solutions for measures

∀e1, e2 ∈ Exercises ∧ e1 ∈ Exercises ∧
  Wrote(Norvig,e1) ∧ Wrote(Russell,e2)
⇒ Difficulty(e1) > Difficulty(e2)

∀e1, e2 ∈ Exercises ∧ e2 ∈ Exercises ∧
  Difficulty(e1) > Difficulty(e2)
⇒ ExpectedScore(e1) < ExpectedScore(e2)
Substances

• Objects
  ➢ tomatoes

• Substances – stuff
  ➢ tomato juice ← not an object !?

  ➔ Is it a category?
  ➔ Is it a physical object?
  ➔ What are its constituents?
Ways of Dealing with Change

- By changing the KB to erase one sentence and replace it with one describing the current fact

  ⇒ drawback: if we erase the past, we cannot speculate about the future

- Search through the space of past and possible future states

  ⇒ does not allow the agent to reason about more than one situation simultaneously

- Situation calculus

  ⇒ conceives the world as a sequence of situations, each of which is a snapshot of the state of the world
Situation Calculus

Spring 2016  Dan I. Moldovan, Human Language Technology Research Institute, The University of Texas at Dallas
Situation

- In situation calculus, each situation (except $S_0$) is the result of an action
Conventions

• Every relation or property that can change over time is given an additional argument, which is always last

  ⇒ instead of At(Agent, location), we have

  At(Agent, [1,1], S_0) ∧ At(Agent, [1,2], S_1)

• Situation calculus uses the function

  Result(action,situation)

  to denote the situation resulting when performing an action in a given situation
Situations and Fluents

- **Situations** are logical terms consisting of the initial situation (usually called $S_0$) and all situations that are generated by applying an action to a situation.

- The function $Result(a,s)$ names the situation that results when $a$ is executed in situation $s$.

- **Fluents** are functions and predicates that vary from one situation to the next, such as is something that flows, like a liquid.

- By convention, the situation is always the last argument of a fluent. For example, $\neg Holding(G_1, S_0)$ says that the agent is not holding the gold $G_1$ in the initial situation $S_0$.

- Atemporal or eternal predicates and functions are also allowed. For example: $Gold(G_1)$.
Describing Actions in Situation Calculus

- Each action is described by two axioms:
  1) A possibility axiom – says when it is possible to execute the action.
  2) An effect axiom – says what happens when a possible action is executed.
- \( Poss(a,s) \) predicate to describe that it is possible to execute action \( a \) in situation \( s \)
- Formally, the axioms are described as:
  - **Possibility Axiom**: Preconditions \( \Rightarrow Poss(a,s) \)
  - **Effect Axiom**: \( Poss(a,s) \Rightarrow \) Changes that result from taking action \( a \) in situation \( s \)
Examples in Wumpus World

- Possibility axioms

\[ P_1: \text{At}(\text{Agent},x,s) \land \text{Adjacent}(x,y) \Rightarrow \text{Poss}(\text{Go}(x,y),s) \]

\[ P_2: \text{Gold}(g) \land \text{At}(\text{Agent},x,s) \land \text{At}(g,x,s) \Rightarrow \text{Poss}(\text{Grab}(g),s) \]

\[ P_3: \text{Holding}(g,s) \Rightarrow \text{Poss}(\text{Release}(g),s) \]

- Effect axioms (if an action is possible, certain properties (fluents) will hold in the situation that results from executing the action)

\[ E_1: \text{Poss}(\text{Go}(x,y),s) \Rightarrow \text{At}(\text{Agent},y,\text{Result}(\text{Go}(x,y),s)) \]

\[ E_2: \text{Poss}(\text{Grab}(g),s) \Rightarrow \text{Holding}(g,\text{Result}(\text{Grab}(g),s)) \]

\[ E_3: \text{Poss}(\text{Release}(g),s) \Rightarrow \neg \text{Holding}(g,\text{Result}(\text{Release}(g),s)) \]
How can Situation Calculus be Used?

- Can the agent grab the gold?
  
  \[\text{Go}([1,1],[1,2])?\]

- Since \textit{Agent} starts at \([1,1]\) and \([1,2]\) is adjacent
  
  \[P_1: \text{At}(\text{Agent},x,s) \land \text{Adjacent}(x,y) \Rightarrow \text{Poss}(\text{Go}(x,y),s)\]

- We conclude the \textit{Agent} reaches \([1,2]\) because
  
  \[E_1: \text{Poss}(\text{Go}(x,y),s) \Rightarrow \text{At}(\text{Agent},y,\text{Result}(\text{Go}(x,y),s))\]

- Therefore
  
  \[\text{At}(\text{Agent},[1,2],\text{Result}(\text{Go}([1,1],[1,2]),S_0))\]
How can Situation Calculus be Used?

- Suppose there is gold at $y = [1,2] \Rightarrow \text{Gold}(G_1)$
- We know that

\[
\text{At}(\text{Agent},[1,2], \text{Result}(\text{Go}(x,y),S_0))
\]

\[
\text{At}(\text{Agent},[1,2], \text{Result}(\text{Go}(x,y),S_0))
\]

- From $P_2$: $\text{Gold}(g) \land \text{At}(\text{Agent},x,s) \land \text{At}(g,x,s) \Rightarrow \text{Poss}(\text{Grab}(g),s)$
  
  \[
g = G_1, \ x = [1,2], \ s = S_1
\]

- We need to prove $\text{At}(g,x,s) = \text{At}(G_1,[1,2],S_1)$ or $\text{At}(G_1,[1,2],\text{Result}(\text{Go}(1,2)[1,2],S_0))$
  
  From what we represented in the KB it cannot be proven

- Why? - the agent’s Go action should have no effects on the gold’s location

- Effect axioms describe what changes, but do not describe what stays the same
The Frame Problem

- Representing things that stay the same is called the **frame problem**
- **Solution**: write **frame axioms**
- **Example**: describe the fact that agent's movements leave other objects stationary unless they are held

\[ F_1: \text{At}(o,x,s) \land (o \neq \text{Agent}) \land \neg \text{Holding}(o,s) \Rightarrow \text{At}(o,x,\text{Result}(\text{Go}(y,z),s)) \]

- Suppose there are F fluent predicates and A actions \( \Rightarrow \) we need \( O(AF) \) frame axioms
- The **representational frame problem**: how to represent everything in a KB of size \( O(AE) \) (if all actions have at most E effects, where \( E < F \))
- The **inferential frame problem**: project the results of a \( t \)-step sequence of axioms in time \( O(Et) \) rather than \( O(Ft) \) or \( O(AEt) \)
Effect Axioms + Frame Axioms

- Together, they provide a complete description of how the world evolves in response to the agent’s actions

- Representational frame problem
  - Evolution of each fluent predicate over time (successor-state axioms) to replace the effect axioms of actions
  - Action is possible $\Rightarrow$ (Fluent is true in result state $\iff$ [action’s effect made it true $\lor$ it was true before and action left it alone]

Agent is at location y after executing an action
  - $\text{Poss}(a,s) \Rightarrow \text{At(Agent},y,\text{Result}(a,s)) \iff a = \text{Go}(x,y) \lor (\text{At(Agent},y,s) \land a \neq \text{Go}(y,z))$
Successor-State Axiom

- One such axiom is needed for each predicate that may change its value over time.
- A successor-state axiom must list all the ways in which the predicate can become true and all the ways in which it can become false.

Axiom for Holding

\[ \text{Poss}(a,s) \Rightarrow (\text{Holding}(g,\text{Result}(a,s)) \Leftrightarrow a = \text{Grab}(g) \lor (\text{Holding}(g,s) \land a \neq \text{Release}(g))) \]

- F different axioms, each of the E effects of the A actions is mentioned exactly once \( \Rightarrow O(AE) \) literals = total size of axioms in KB.
Time, Space and Change

• To allow for actions and events that have different duration and can occur simultaneously
  Include time in the ontology

• The universe is continuous in both temporal and spatial dimensions

• Times, places and objects are parts of this universe
Representing Change with Events

Situation calculus was used to represent change
1) Situations are instantaneous points in time
   Not useful when describing gradual processes
   Example: kitten → cat

2) Situation calculus works best when only one action happens at a time
   Multiple agents, the world changes spontaneously
Event Calculus

- Continuous version of situation calculus
  \[\Rightarrow\] A world with both spatial and temporal dimensions
- An event: a chunk of the universe with both spatial and temporal extent
Subevents

Subevent(BattleOfBritain, WorldWarII)
Subevent(WorldWarII, TwentiethCentury)

Interval

An event that includes as subevents all events occurring in a given time period
Event Structure

• Events can be grouped into categories
  Example: *WorldWarII ∈ Wars*

• *A war occurred in the Middle East in 1967*
  \[∃w, w ∈ \text{Wars} ∧ \text{SubEvent}(w, \text{AD1967}) ∧ \]
  \[\text{PartOf}(\text{Location}(w), \text{MiddleEast})\]

• *Shankar travelled from New York to New Delhi yesterday*
  use the category Journeys
  \[∃j, j ∈ \text{Journeys} ∧ \text{Origin}(\text{NewYork}, j) ∧ \text{Destination}(\text{New Delhi}, j) ∧ \]
  \[\text{Traveler}(\text{Shankar}, j) ∧ \text{SubEvent}(j, \text{Yesterday})\]
Event Properties

Go(Shankar, NewYork, NewDelhi)

with Go defined as

\( \forall e, x, o, d \ e \in Go(x, o, d) \iff e \in Journeys \land Traveler(x, e) \land Origin(o, e) \land Destination(d, e) \)
Subevent Notation

- \( E(c,i) \) – an event of category \( c \) is a subevent of the event or interval \( i \)

\[ \forall c, i \; E(c,i) \iff \exists e, e \in C \land \text{SubEvent}(e,i) \]

\[ \rightarrow E(\text{Go(Shankar,NewYork,NewDelhi)}, \text{Yesterday}) \]
Places

- Places, like intervals, are special kinds of space-time chunks
  A place can be thought of as a constant piece of space extended through time
- Predicate in denotes a special kind of subevent relation that holds between places
  \texttt{In(New York, USA)}
- Places come in different varieties
  New York is an Area
  Solar System is a Volume
Places

The Location() function maps an object to the smallest place that contains it

\[ \forall x, l \quad \text{Location}(x) = l \iff \text{At}(x, l) \land \forall l_2 \text{At}(x, l_2) \implies \text{In}(l, l_2) \]

Minimization
Processes

• Discrete events ⇒ they have a definite structure
  Example: Shankar’s trip has beginning, middle, end

• If interrupted halfway, the event is different
  ➢ it would not be a trip from New York to New Delhi, but a trip from New York to Ankara
  ➢ the category of events denoted as Flying(Shankar) maintains all events with Intervals < InterruptionTime

Processes(liquid events)
Features of Processes

• Any subinterval of a process is a member of the same process category
  ➢ denote them like discrete events
    \[ E(\text{Flying(Shankar),Yesterday}) \]
  ➢ predicate \( T \Rightarrow throughout \)
    \[ T(c,i) \Rightarrow \text{event of type c occurred over interval i exactly} \]

    \[ T(\text{Working(Stuart),TodayLunchHour}) \]
States

- Liquid events describe processes of continuous change
- Liquid events describe processes of continuous non-change
- Example: Mary was in the local supermarket all afternoon
  \[ T(\text{In(Mary,SuperMarket}_1), \text{ThisAfternoon}) \]
- An interval is also a discontinuous sequence of times
  \[ \Rightarrow \text{Example: the supermarket is closed every Sunday} \]
  \[ T(\text{Closed(SuperMarket}_1), \text{BunchOf(Sundays)}) \]
Mental Objects and Beliefs

- An agent will often need to reason about its own beliefs

⇒ It also needs to reason about the beliefs of others

⇒ In an ontology, sentences are explicitly represented and are believed by agents