Knowledge Representation

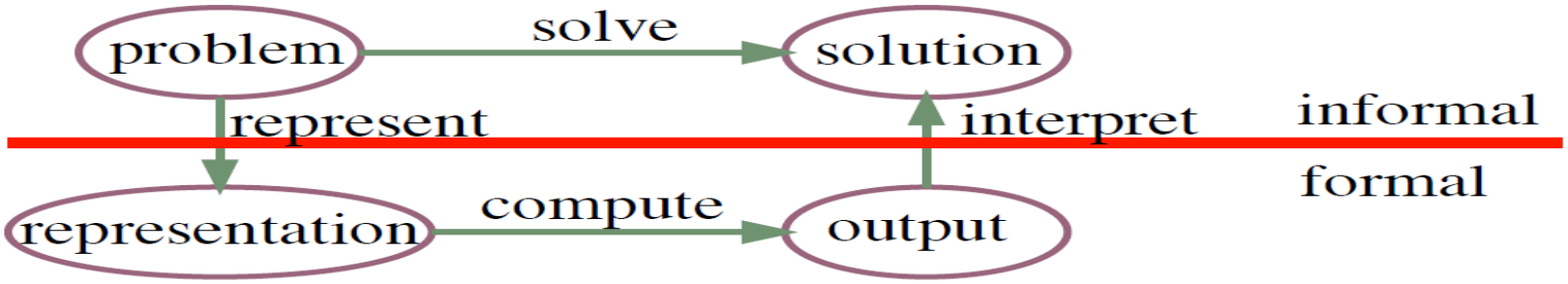
- Poole, chap. 1, 5
- Negnevitsky, chap. 5

Today

- Knowledge Representation
- Production Systems
  - Inference Engine
  - Reasoning
- Semantic Networks
- Frames

Knowledge Representation

- If we have specific knowledge about a problem it is easier to solve it
- We can combine this domain knowledge with the general knowledge that we have about problem solving
- This knowledge allows to the different AI algorithms to guide the search and reduce the cost to obtain a solution
- Problems
  - How to choose a representation formalism that allows an easy translation from the real world to the represented world?
  - How is this representation used efficiently?



Differences between information and knowledge (1)

- **Information** is defined as a set of basic facts, without interpretation, obtained as the definition of the problem
  - For example:
    - The numerical data from a blood test
    - The data from the sensors of a chemical process
- **Knowledge** is defined as a set of high level facts that model structuredly the experience from a domain or that are obtained from the interpretation of the basic facts
  - For example:
    - The interpretation of the values of a blood test or of the sensors from a chemical process, saying if they are normal, high, low, problematic, dangerous, ...
    - The data structures and methods to diagnose patients from the interpretation of a blood test or to aid to the decision process in the management of a chemical plant

Differences between information and knowledge (2)

- AI programs need different kinds of knowledge that are not available from databases and other information sources:
  - Knowledge about the objects of a domain and their relationships
  - Knowledge about the processes that involve these objects or that are useful to manipulate them
  - Knowledge that is difficult to represent as basic facts as intensionality, causality, goals, temporal information, "common sense" knowledge, ...
- Intuitively

$$Knowledge = Information + Interpretation$$

Knowledge Representation

- To represent knowledge we need
  - Its structure
  - What for the intelligent agents will use it
  - How it will be used by intelligent agents
  - How it will be acquired
  - How it will be stored and manipulated
- Unfortunately there are not complete answers to these questions from a biological or neurophysiological point of view
  - We will see formalisms that simulate the acquisition, structure and manipulation of knowledge that will allow us to build intelligent agents

Knowledge Representation formalisms

- A **Representation formalism** is an mechanism to represent the real world in a computer
- It is important to keep in mind the difference between
  - The real world (what we want to represent) $\rightarrow$ **Domain**
  - Its representation $\rightarrow$ **many representation formalism**
- A representation formalism can be seen as a combination of:
  - Data structures to code the problem that an intelligent agent is solving$\rightarrow$ **Static part**
  - Data structures that store the domain knowledge of the problem and actions to manipulate knowledge consistently with its interpretation $\rightarrow$ **Dynamic part**

Knowledge Representation formalisms: Static part

- The static structures include
  - The data structure that represents the knowledge of the problem
  - The actions that allow to create, modify or delete elements from the data structure
  - Predicates that allow to query the data structure
  - Semantics of the data structure: a semantic correspondence between the real world and the representation is chosen

R(Elements from representation, Real world)

Knowledge Representation formalisms: Dynamic part

- The dynamic part is composed of:
  - Data structures that store the domain knowledge
  - Actions that allow
    - To interpret the data from the problem (the static part) using the domain knowledge (the dynamic part)
    - Control the use of the data: Control strategies
    - Acquire new knowledge

Incompleteness of knowledge representation

- Any representation will be always incomplete because:
  - **Change:** The real world is dynamic and changes every moment, but our representation only captures an instant
  - **Volume:** lots (too much) of knowledge to represent $\rightarrow$ partial view
  - **Complexity:** Reality has lots of details
- The problem of change is related to the processes of acquiring and maintaining the knowledge in the representation (Frame Problem)
- The problems of volume and complexity are related with the granularity of the representation

Properties of Knowledge Representation formalisms

A Knowledge Representation formalism has the following properties

- Of the formalism itself
  - **Representational Adequacy:** It must support representation of all kinds of knowledge required by the problem
  - **Inferential Adequacy:** It must allow to manipulate the knowledge represented to obtain new knowledge from the current one

Properties of Knowledge Representation formalisms

- Related to the use of the representation
  - **Inferential Efficiency:** It needs to be able to infer new things from the knowledge easily. The ability to incorporate additional information that can be used to focus the attention of the inference mechanisms to promising areas (metaknowledge)
  - **Acquisitional Efficiency:** It must allow to incorporate new knowledge to the representation easily. Ideally an intelligent agent should be able to obtain new information autonomously and incorporate it to the representation

Typology of knowledge (1/2)

How to represent knowledge?

- **"Knowing how"**: procedure that can be called

  The human information processor is a stored program device, with its knowledge of the world *embedded* in the programs. What a person (or a robot) knows about the English language, the game of chess, or the physical properties of his world is coextensive with his set of programs for operating with it.

- **"Knowing that"**: declarative representation which requires an interpreter. Advantages:

  Understandability, learnability, accessibility, communicability, flexibility, economy of storage.

Typology of knowledge (2/2)

- Declarative knowledge
  - The knowledge is represented independently from the mechanisms that will use it
  - The control of the adequate use of the knowledge is performed
    - by means of general purpose heuristics that determine the best way to use the knowledge
    - by means of the use of knowledge about the control of the use of the knowledge that guides the solving mechanism
  - Kinds of declarative knowledge
    - Relational Knowledge
    - Inheritable Knowledge
    - Inferential Knowledge
- Procedural Knowledge
  - The knowledge explicitly defines how to use it

Simple relational knowledge

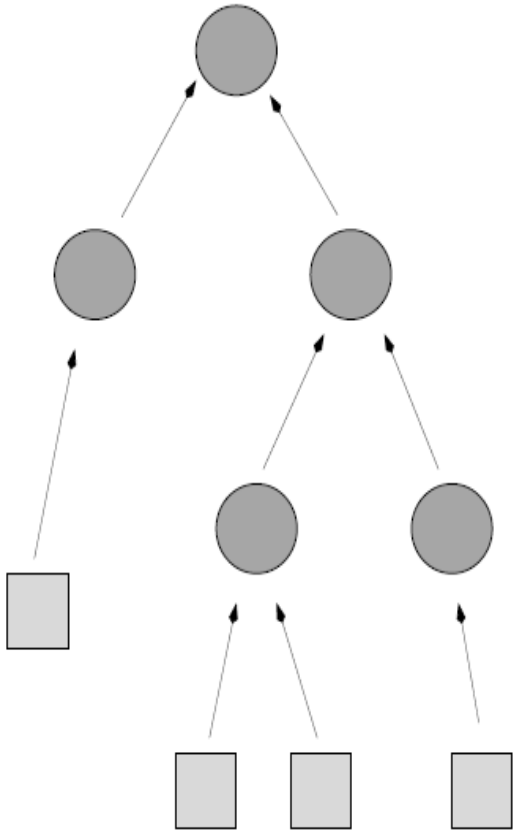
- The easier way to represent declarative facts is to use relations that can be represented as tables (as in databases)
  - Ex: Information about the clients of a company

Client	Address	Sales	...
A. Perez	Av. Diagonal	5643832	
J. Lopez	c/ Industria	430955	
...			

- Problem: There is not much knowledge represented
- We need some mechanisms to improve the representation $\rightarrow$ Inference engine: obtains new knowledge from the representation
  - Ex: mean sales of a product, best clients, client typology
- Databases can be useful for Knowledge Based Systems.

Inheritable Knowledge

- It is useful to structure the knowledge hierarchically (taxonomical hierarchy)
- The goal is to represent the knowledge as a graph or a tree and to use generalization and specialization as representation goal
  - The nodes are concepts/classes, the edges are relations
    - is-a: class-class relation
    - Instance-of: class-instance relation
- The reasoning mechanism is the inheritance of properties and values
  - Simple/multiple inheritance
  - Default values



Inferential knowledge

See Slides KR_AI

- The knowledge is described using logic
- The semantics of the logic connectives and inference mechanisms (for example Modus Ponens rule) can be used to obtain new knowledge

$$\forall x, y : person(x) \wedge \neg underage(x) \wedge \neg job(x, y) \rightarrow unemployed(x)$$

- The inference mechanisms for first order logic are the different algorithms from automatic theorem proving

Procedural knowledge

- This knowledge includes control information in its representation and/or specific procedures needed to use it
  - Programs: The knowledge is defined as algorithms that allow to infer new knowledge from facts
    - Ex: Date_of_birth= DD-MM-YYYY; `function Age (Date_of_birth: date)`
  - Production rules: if some condition holds some actions/inferences are performed
    - Ex: IF conditions THEN actions
- This kind of knowledge is more computationally efficient, but is more difficult to obtain new knowledge from it and it is complex to acquire/modify

Today

- Knowledge Representation
- Production Systems
  - Inference Engine
  - Reasoning
- Semantic Networks
- Frames

Production Systems

- Knowledge representation using logic can be seen as a procedural representation
- The steps to solve a problem are described as a chain of deductions
- This formalism is based on two elements:
  - **Facts:** Propositions or predicates
  - **Rules:** Conditional expressions where the consequent is usually an atomic predicate or an action
- Analogy with space state search
  - Facts = state of the problem
  - Rules = Search operators

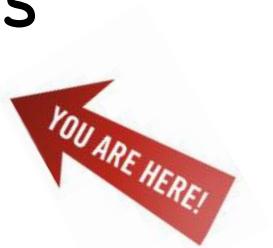
Production systems

A production system is composed by:

- **Fact base:** Predicates that describe a problem
- **Knowledge base (or rule base):** Rules that describe the reasoning mechanisms that can be used to solve problems in a domain
- **Inference engine:** Applies the rules and obtain the chain of deductions that solve the problem

Today

- Knowledge Representation
- Production Systems
  - Inference engine
  - Reasoning
- Semantic Networks
- Frames



Inference Engine

- Role
  - Deduce new facts, apply actions to solve the problem (goal) from a set of initial facts and eventually by interaction with the user or the environment
- Components
  - Rule interpreter + control strategy
- Phases
  - Detection (filter): **Activable Rules**
    - Obtain a set of rule instantiations (**rule conflict set**)
  - Selection: **Rule to execute**
    - Conflict resolution: Select the rule instantiation to execute
  - Execution:
    - Execution of the rule

Detection

- Compute the set of possible instantiations of the rules (**Conflict set**)
- The rule interpreter computes all the possible instantiations with the facts in the current state of the problem (matching)
- A rule could be instantiated more than one time (rules with variables)

Selection

- This phase selects the *best* instantiation
- The rules are selected using a control strategy (*Conflict resolution strategy*)
  - Fixed strategy
  - Fixed dynamic strategy
  - Strategy guided by metarules
- Possible criteria:
  - First rule in order
  - The rule more/less times executed
  - The more specific/general rule
  - The rule with higher certainty degree
  - The instantiation that uses the facts:
    - with more priority,
    - that are older in the fact base (the oldest instantiation),
    - that are newer in the fact base (the more recent instantiation),
    - ...
- It is possible to use a combination of strategies

Execution

- Rule execution $\Rightarrow$
  - Update the fact base (if forward chaining reasoning is used)
  - New computations, new actions applied, query the user
  - New subgoals (if backward chaining is used)
- Instantiation propagation to the consequent
- Propagation of certainty.
- The deduction process ends when:
  - The conclusion (goal) is found $\Rightarrow$ success
  - There is no more rules to apply $\Rightarrow$ success? / fail?

Today

- Knowledge Representation
- Production Systems
  - Inference engine
  - Reasoning
- Semantic Networks
- Frames

Types of reasoning

- Deductive, forward chaining (FC), data driven reasoning
  - Evidences, symptoms, facts $\Rightarrow$ conclusions, hypothesis
- Inductive, backward chaining (BC), goal driven reasoning
  - Conclusions, hypothesis $\Rightarrow$ evidences, symptoms, facts
- Hybrid chaining

Data driven reasoning

- Based on Modus Ponens rule: $A, A \rightarrow B \vdash B$
- The base of facts (BF) is initialized with the facts that describe the problem

---

Procedure: Forward chaining

Input: Facts base, Rules base, Goals

Alternative $\leftarrow$ true

while $\exists g(g \in \mathit{Goals} \wedge g \notin \mathit{Facts\_Base}) \wedge \mathit{alternative}$ **do**

    Conflict_Set $\leftarrow$ Interpreter.Satisfactible_Conditions(Facts_Base, Rules_Base)

    if $\mathit{Conflict\_Set} \neq \emptyset$ **then**

        Rule $\leftarrow$ Control_Strategy.Conflict_Resolution(Conflict_Set)

        Interpreter.Apply(Facts_Base, Rule)

    else

        Alternative $\leftarrow$ false

Data driven reasoning

- Problems
  - No goal oriented
  - Combinatorial explosion, facts not related to the goal
- Advantages
  - Intuitive to understand
  - It is easier to formalize the knowledge this way, it is the more natural representation for many problems
  - Can be used in an exploratory way

Goal driven reasoning

- Induction based. The goal is used as starting point and the chain of deductions that leads to it is obtained backwards
- Each iteration leads to new subgoals: hypothesis to validate

---

Procedure: Backward chaining

Input: Facts base, Rules base, Goals

```
No_alternative ← false
while Goals ≠ ∅ ∧ ¬No_alternative do
    Goal ← Control_Strategy.Select_Goal(Goals)
    Goals.Delete(Goal)
    Conflict_Set ← Interpreter.Satisfactible_goal(goal, Rules_Base)
    if Conflict_Set ≠ ∅ then
        Rule ← Control_Strategy.Conflict_Resolution(Conflicto_Set)
        Goals.Add(Rule.Extract_antecedent_as_goals())
    else
        No_alternative ← true
```

Goal driven reasoning

- Problems are solved by problem decomposition
- The resolution process is a search of an and/or tree
- Problems
  - The solution has to be known beforehand
- Advantages
  - Only the subgoals needed to solve the problem are considered

Hybrid reasoning

- Some parts of the chain of reasoning from the facts to the goal are constructed either deductively or inductively
- Bidirectional search
- The change from one strategy to another is done by using meta-rules
- The main advantage is to reduce the combinatorial explosion of forward chaining
- This strategy increases the efficiency of backward chaining when there is not a clear goal

Rules are used in many places

The use of rules as programming formalism is extended

- As a mechanism of transformation, compilation, translation, ...
  - Programming language compilers (LEX, YACC)
  - In the Web: XLTS (Extensible Stylesheet Language Transformations)
  - Task automation: Make, ANT, ...
- As a formalism to represent **business rules** in programs
  - Inference engines as a part of software development: Interpreted rules instead of code
  - Many comercial software development environments include rules: SAP, IBM, Oracle, Microsoft, ...
  - Are used in modern software development paradigms (Service Oriented Architectures)

Today

- Knowledge Representation
- Production Systems
- Semantic Networks
- Frames

1- Semantic Networks

- Due to Ross Quinlan 1960's
- Developed from cognitive work…
  - Humans perceive and reason about concepts
  - *main idea:* the meaning of a concept comes from the way it is connected to other concepts
  - Concepts are connected through various relationships to other concepts
  - These relationships form our understanding

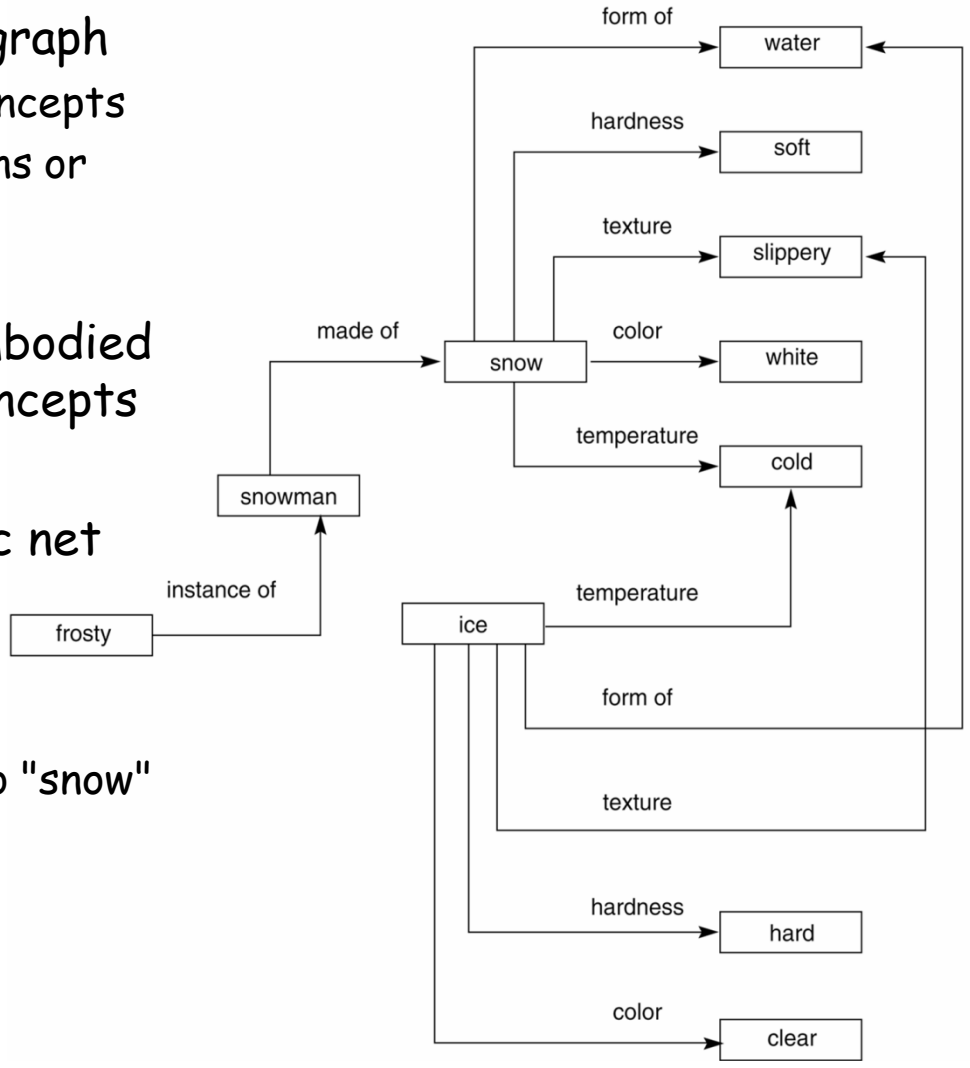
- E.g.: concept: *snow*

  After the snow storm, the backyard was all white, so I put on my boots and went out.

  After the snow storm, the backyard was all blue, so I put on my scandals and went out.

Example of a Semantic Network

- can represent knowledge as a graph
  - nodes represent objects or concepts
  - labeled arcs represent relations or associations
- the meaning of a concept is embodied by its associations to other concepts
- retrieving info from a semantic net can be seen as a graph search problem
  - to find the texture of snow
  - find the node corresponding to "snow"
  - find the arc labeled "texture"
  - follow the arc to the concept "slippery"



Semantic Networks

- Using logic as representation formalism has some practical problems
- Some more intuitive and easy to use formalisms are needed
- Cognitive psychology affirms that:
  - The representation and query of knowledge are done based on its relations
- Semantic networks is a KR formalism that allows to represent knowledge as a graph where:
  - The nodes represent concepts
  - The edges (directed) represent relations among concepts

Semantic Networks

- They are a constrained version of predicate calculus language
- The goal of this formalism is to represent declaratively the elements of a domain
- Some specific mechanisms of reasoning can be defined to make some queries about the knowledge that are represented
  - Are two concepts related?
  - What relations link two concepts?
  - What is the nearest concept that are related to a pair of concepts?
- A richer semantics can be defined to allow more complex queries
  - Concept taxonomies (class/subclass/instance)
  - Generalization/Specialization

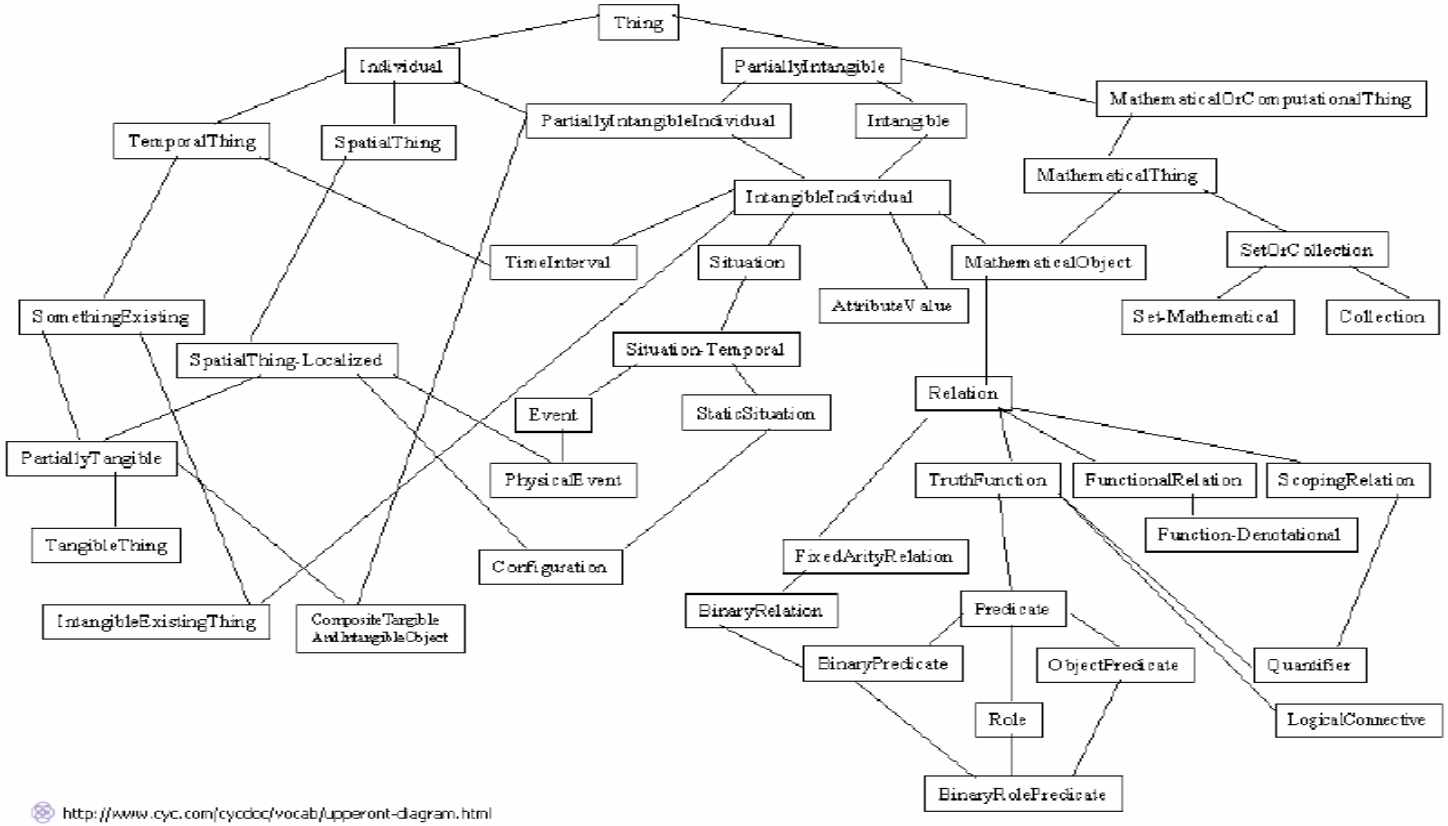
Limitations of semantic networks

- There are many formalisms under the name semantic networks with different expressive capabilities but always with a formal reasoning model
- Different levels of abstraction are mixed in the representation
  - Concepts/instances/values
  - Relations/properties
- A more structured formalism is necessary
- A formal semantic model for reasoning is necessary

CYC

OpenCyc Selected Vocabulary and Upper Ontology

E-Mail Comments to: doc@cyc.com
Last update: March 27, 2002



- Purpose: to make commonsense knowledge processable by computers

Today

- Knowledge Representation
- Production Systems
- Semantic Networks
- Frames

Frames

- Formalism that gives a structure to semantic networks
- A Frame is a collection of *attributes* and the description of their characteristics
- The *relations* connect frames
- There is an explicit separation between relations and attributes
- Relations and attributes have also properties to describe their semantics
- Some Frame languages you may know: Entity-Relationship Diagrams, UML

A frame is a data structure with typical knowledge about a particular object or concept

Example: Boarding pass frames

QANTAS BOARDING PASS	
Carrier:	*QANTAS AIRWAYS*
Name:	*MR N BLACK*
Flight:	*QF 612*
Date:	*29DEC*
Seat:	*23A*
From:	*HOBART*
To:	*MELBOURNE*
Boarding:	*0620*
Gate:	*2*

AIR NEW ZEALAND BOARDING PASS	
Carrier:	*AIR NEW ZEALAND*
Name:	*MRS J WHITE*
Flight:	*NZ 0198*
Date:	*23NOV*
Seat:	*27K*
From:	*MELBOURNE*
To:	*CHRISTCHURCH*
Boarding:	*1815*
Gate:	*4*

Frames

- The reasoning formalism that supports the declarative part is *description logic* (this is also the logic object oriented languages are based on)
  - Concept inclusion (specialization/generalization)
  - Value and attribute inheritance
  - Set relations (union, intersection, membership, transitivity)
- Frames usually add also a procedural formalism to complement the declarative formalism
  - Functions and methods that reduce the computational cost of the inference

Elements of the formalism

- A frame represents a concept
- It has a declarative part (attributes) and a procedural part (methods)
- The declarative part allows to describe the semantics of the concept (characteristics)
- The procedural part allows to define how to obtain information or to perform computations related to its attributes or to its relations with other frames
- A frame is described by a name, a list of attributes and a list of methods

Elements of the formalism - Frames

Description of a frame:

Frame <name>
    slot <slot-name>[1]
    slot <slot-name>[1]
    ...
    slot <slot-name>[1]
    methods [2]
        procedure <method-name> (parameters) [I/noI]
        ...
        function <method-name> (parameters) return <datatype> [I/noI]

[1] Each slot can have modifiers that can override the global definition of the slot

[2] The procedures/actions that describe the methods use the variable F as a self reference to the frame or instance of frame that calls the method

Elements of the formalism - Relations

- A relation connects frames
- A relation has characteristics that describe its semantics, properties and behaviour
- Relations are the basis of the inference on frames: inheritance of slots
- We will distinguish two kinds:
  - Taxonomical: is-a (class/subclass), instance-of (instance/class)
  - User defined: The rest of relations

Elements of the formalism - Slots

- The slots describe the characteristics of a frame
- They have a set of characteristics (facets) that allow to describe their semantics
  - Domain, range, cardinality, default value, ...
- The allow to define procedures that can perform computations triggered by events (demons)
- There are four kinds of demons:
  - `If-needed` (when the value of the slot is accessed)
  - `if-added` (when a slot is given a value),
  - `if-removed` (when the value of a slot is deleted)
  - `if-modified` (when the value of a slot is modified)
- Demons have no parameters
- We can declare how slots are affected by inheritance

Elements of the formalism - Slots

Description of a slot:

Slot <name>
++ domain (list of frames)
++ range <primitive-datatype>
++ cardinality (1 or N)
value (value or list of values)
demons <demon-type>
procedure <procedure-name> / **function**<function-name> returns <datatype>*
inheritance (by taxonomic relations: YES/NO; by user defined relations: YES/NO)

- The facets marked ++ are compulsory for the description of a slot
- The procedures/functions associated to demons do not have parameters. The variable `F` is used as self reference
- The demons `if-needed` only can be defined as functions
- By default the inheritance by taxonomical relations is YES and by user defined relations is NO
- To consult the value of a slot the syntax <frame-name>.<slot-name> is used. The result could be a single value or a list of values depending on the cardinality of the slot

Elements of the formalism - Methods

The methods are procedures or functions that allow to obtain information from a frame

- A method could be called from an abstract frame (classes) or from the instance of a frame
- They can be
  - Inheritable (the subclasses/instances can call the method)
  - Non inheritable (only the frame that define the method can call it)
- Methods can have parameters

Elements of the formalism - Relations

Relations allow to connect frames to represent the relationships among concepts

- Their semantic is defined from a set of properties: Domain, range, cardinality, inverse, transitivity, composition, ...
- Procedural mechanisms can be defined (demons) that are triggered by certain events:
  - `If-added`: if the relation between two instances is created
  - `If-removed`: if the relation between two instances is deleted
- The behaviour of a relation to inheritance of properties can also be defined (what slots can be inherited). Since relations are defined in both directions, the slots are inherited in the adequate direction (from the frame where the slot is defined to the frame that has to inherit the slot)

Elements of the formalism - Relations

Relation <name>
++ domain (list of frames)
++ range (list of frames)
++ cardinality (1 or N)
++ inverse <name> (cardinality: 1 or N)
transitive YES/NO [NO by default]
composed NO/<description of the composition> [NO by default]
demons (<demon-type> procedure <procedure-name>
inheritance (list of slots) [empty list by default]

- The properties marked ++ are compulsory to describe a relation
- The procedures associated to demons have no parameters. They use the variables D and R as a reference to the source and destination frames respectively of the relation that has been added or removed between two frames.

Elements of the formalism- Programming

- The syntax `<frame-name>.<relation-name>` will return the frame (if cardinality is 1) or the list of frames (if cardinality is N) that are connected to the current frame through the relation `<relation-name>`
- To access to the actual cardinality of a relation the function `card(<frame-name>.<relation-name>)` can be used
- Predefined relations:
  - relation `is-a` (inverse: `has-as-subclass`) transitive YES
  - relation `instance-of` (inverse: `has-as-instances`) composition: `instance-of` $\otimes$ `is-a`

Elements of the formalism- Programming

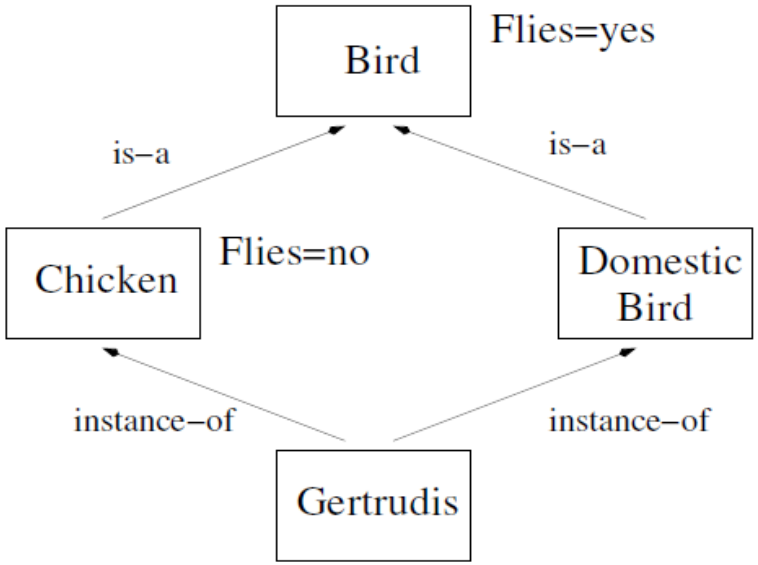
- Predefined boolean functions
  - `slot?(<frame>)` returns if `<frame>` has this slot or not (activating inheritance if needed)
  - `relation?(<frame>)`
    returns if `<frame>` is related with any other through the specified relation
  - `relation?(<frame-o>,<frame-d>)`
    returns if the specified relation links `<frame-o>` and `<frame-d>`

Inheritance

- Inheritance is the main deduction mechanism for frames
- It allows to obtain the value or values of a slot or its definition from another frame through the relations that connect them
- Taxonomic relations has inheritance activated by default (the definition of the slots is inherited)
- On the rest of relation inheritance has to be activated explicitly (the value of the slot is inherited)
- Given a frame it is possible that the representation allows to inherit the value of a slot from multiple relations or frames (Multiple inheritance)

Multiple inheritance

- Multiple inheritance is valid depending on the semantics of the inherited slot



- The **inferential distance algorithm** allows to determine which frame is the correct to inherit from

Inferential distance algorithm

1. Search for the set of frames that allow to inherit the slot value $\longrightarrow$ **Candidates**
2. Delete from Candidates all frames that are ascendant of another frame in the list
3. If the number of candidates is:
   1. 0 $\longrightarrow$ the slot can not be inherited
   2. 1 $\longrightarrow$ this is the correct value
   3. $> 1 \longrightarrow$ Multiple inheritance problem if slot cardinality is not N
4. Sometimes a problem of multiple inheritance can be solved procedurally (demons)

Today

- Knowledge Representation
- Production Systems
- Semantic Networks
- Frames