Knowledge Representation

Lecture 1: Knowledge Acquisition

Instructor: Hazim N. Abed

CONCEPTS OF KNOWLEDGE ENGINEERING

- The process of acquiring knowledge from experts and building a knowledge base is called **knowledge engineering**.

- **Knowledge engineering** involves the cooperation of human experts in the domain working with the knowledge engineer to codify and make explicit the rules (or other procedures) that a human expert uses to solve real problems. The field is related to **software engineering**.

- Knowledge engineering can be viewed from two perspectives: **narrow** and **broad**.

CONCEPTS OF KNOWLEDGE ENGINEERING

- According to the **narrow perspective**, knowledge engineering deals with **knowledge acquisition**, **representation**, **validation**, **inference**, **explanation**, and **maintenance**.

- Alternatively; according to the **broad perspective**, the term describes the entire process of developing and maintaining intelligent systems.

- The knowledge possessed by human experts is often unstructured and not explicitly expressed. A **major goal of knowledge engineering** is to help experts articulate what they know and document the knowledge in a reusable form.

CONCEPTS OF KNOWLEDGE ENGINEERING

- The knowledge-engineering process includes five major activities:

1. **Knowledge acquisition**

   - Knowledge acquisition involves the acquisition of knowledge from human experts, books, documents, sensors, or computer files.

   - The knowledge may be specific to the problem domain or to the problem-solving procedures, it may be general knowledge (e.g., knowledge about business), or it may be metaknowledge (knowledge about knowledge).

CONCEPTS OF KNOWLEDGE ENGINEERING

- (By metaknowledge, we mean information about how experts use their knowledge to solve problems and about problem-solving procedures in general.)

2. **Knowledge representation.**

- Acquired knowledge is organized so that it will be ready for use, in an activity called knowledge representation. This activity involves preparation of a knowledge map and encoding of the knowledge in the knowledge base.

CONCEPTS OF KNOWLEDGE ENGINEERING

3. **Knowledge validation.**

- Knowledge validation (or verification) involves validating and verifying the knowledge (e.g., by using test cases) until its quality is acceptable. Testing results are usually shown to a domain expert(s) to verify the accuracy of the ES.

4. **Inference.**

- This activity involves the design of software to enable the computer to make inferences based on the stored knowledge and the specifics of a problem. The system can then provide advice to non-expert users.

CONCEPTS OF KNOWLEDGE ENGINEERING

3. **Knowledge validation.**

- Knowledge validation (or verification) involves validating and verifying the knowledge (e.g., by using test cases) until its quality is acceptable. Testing results are usually shown to a domain expert(s) to verify the accuracy of the ES.

4. **Inference.**

- This activity involves the design of software to enable the computer to make inferences based on the stored knowledge and the specifics of a problem. The system can then provide advice to non-expert users.

CONCEPTS OF KNOWLEDGE ENGINEERING

5. **Explanation and justification.**

- This step involves the design and programming of an explanation capability (e.g., programming the ability to answer questions such as why a specific piece of information is needed by the computer or how a certain conclusion was derived by the computer).

- Following figure shows the process of knowledge engineering and the relationships among the knowledge engineering activities. Knowledge engineers interact with human experts or collect documented knowledge from other sources in the knowledge acquisition stage

CONCEPTS OF KNOWLEDGE ENGINEERING

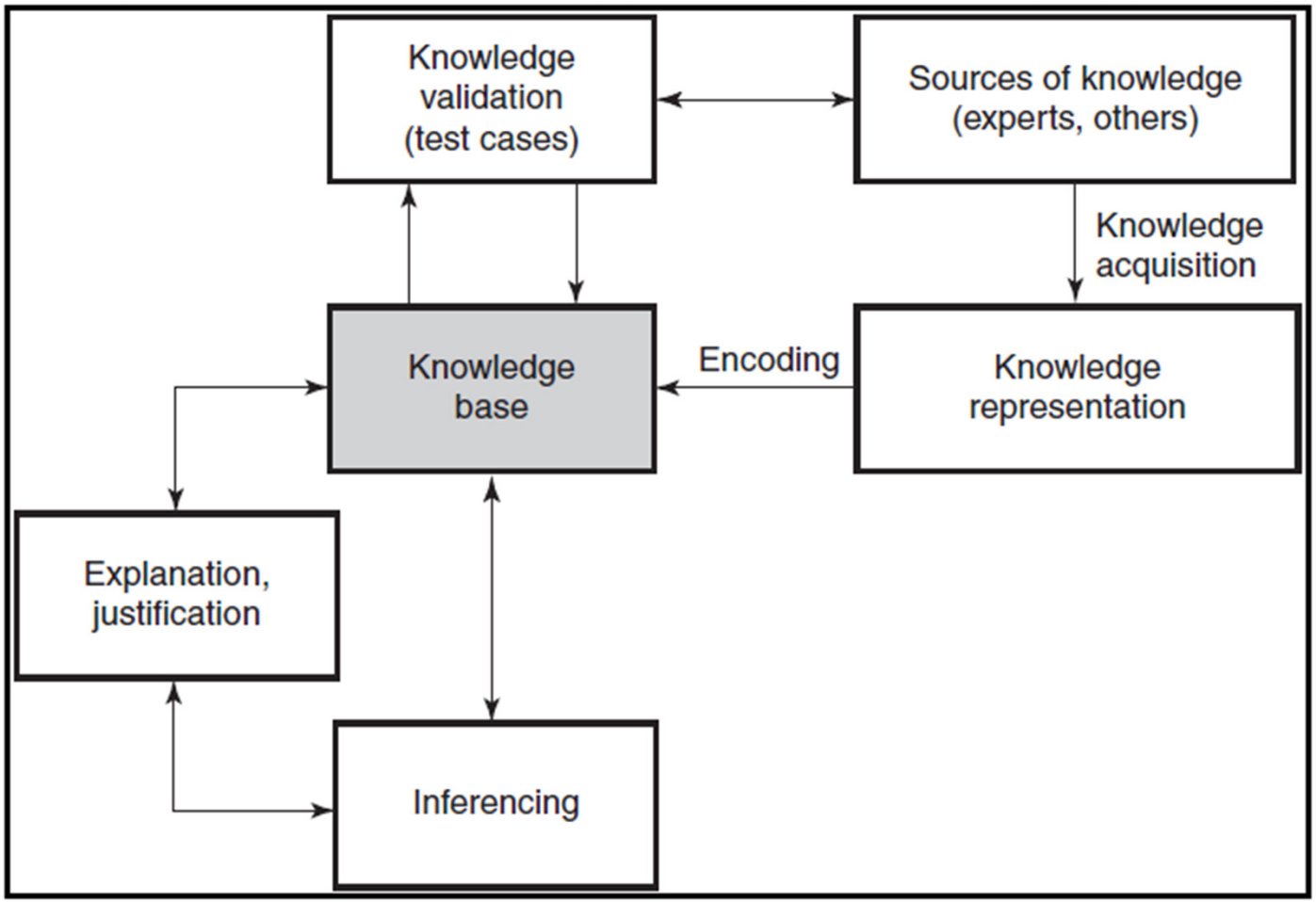


Figure 1.1: Process of Knowledge Engineering

CONCEPTS OF KNOWLEDGE ENGINEERING

- The acquired knowledge is then coded into a representation scheme to create a knowledge base. The knowledge engineer can collaborate with human experts or use test cases to verify and validate the knowledge base.

- The validated knowledge can be used in a knowledge-based system to solve new problems via machine inference and to explain the generated recommendation.

THE SCOPE AND TYPES OF KNOWLEDGE

- The **most important factor** in knowledge acquisition is the extraction of knowledge from sources of expertise and its transfer to the knowledge base and then to the inference engine. Acquisition is actually done throughout the entire development process.

- **Knowledge** is a collection of specialized facts, procedures, and judgment usually expressed as rules. Some types of knowledge used in artificial intelligence are shown in Figure 1.2.These types of knowledge may come from one or from several sources.

THE SCOPE AND TYPES OF KNOWLEDGE

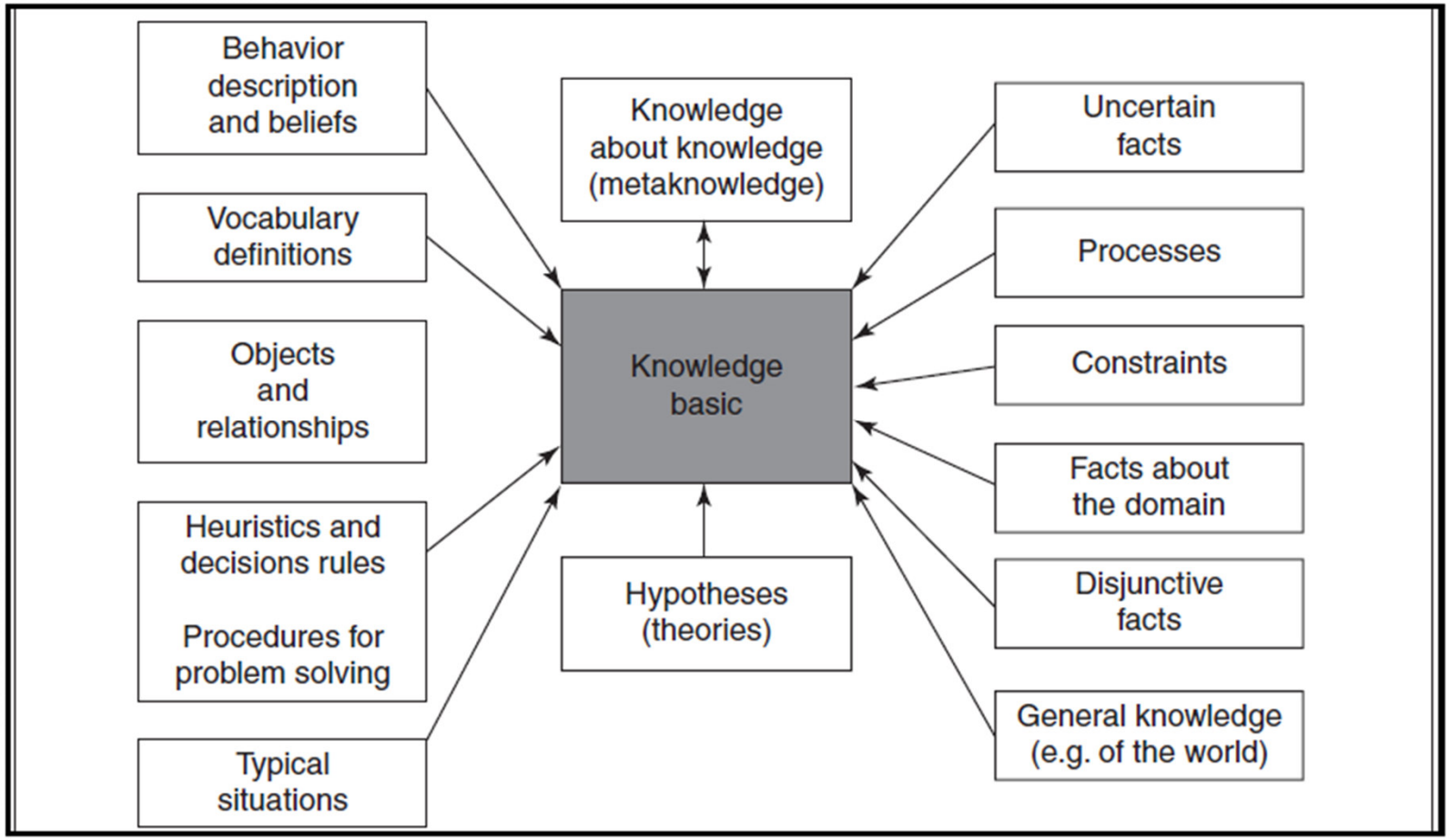


Figure 1.2: Types of Knowledge to be represented in a Knowledge Base

THE SCOPE AND TYPES OF KNOWLEDGE

Types of Knowledge

- **Knowledge** can be collected from many sources, such as books, films, computer databases, pictures, maps, flow diagrams, stories, sensors, radio frequency identification (RFID), songs, or even observed behavior. Knowledge are of two types: **documented knowledge(Explicit Knowledge)** and **undocumented knowledge (tacit knowledge).**

Documented knowledge:

- For ES, stored knowledge sources not based directly on human expertise

THE SCOPE AND TYPES OF KNOWLEDGE

Undocumented knowledge:

- Knowledge that comes from sources that are not documented, such as human experts

Sources of Knowledge

1. **Knowledge Acquisition from Databases**

- Many ES are constructed from knowledge extracted in whole or in part from databases. With the increased amount of knowledge stored in databases, the acquisition of such knowledge becomes more difficult (Data Mining).

THE SCOPE AND TYPES OF KNOWLEDGE

2. Knowledge Acquisition via the Internet

- With the increased use of the Internet, it is possible to access vast amounts of knowledge online.

- The acquisition, availability, and management of knowledge via the Internet are becoming critical success issues for the construction and maintenance of knowledge-based systems, particularly because they allow the acquisition and dissemination of large quantities of knowledge in a short time across organizational and physical boundaries.

THE SCOPE AND TYPES OF KNOWLEDGE

Levels Of Knowledge

- Knowledge can be represented at different levels. The two extremes are **shallow knowledge** (i.e., **surface knowledge**) and **deep knowledge**.

1. **Shallow knowledge:** is the representation of surface-level information that can be used to deal with very specific situations. For example, if you don't have gasoline in your car, the car won't start. This knowledge can be shown as a rule:

If gasoline tank is empty, then car will not start.

THE SCOPE AND TYPES OF KNOWLEDGE

- The shallow version basically represents the input/output relationship of a system. As such, it can be ideally presented in terms **of IF-THEN** rules. Shallow representation is limited. A set of rules by itself may have little meaning for the user. It may have little to do with the manner in which experts view the domain and solve problems.

- This may limit the capability of the system to provide appropriate explanations to the user. Shallow knowledge may also be insufficient in describing complex situations. Therefore, a deeper presentation is often required.

THE SCOPE AND TYPES OF KNOWLEDGE

2. Deep Knowledge

- Human problem solving is based on deep knowledge of a situation.

- **Deep knowledge** is the internal and causal structure of a system and involves the interactions between the system's components. Deep knowledge can be applied to different tasks and different situations.

- It is based on a completely integrated, cohesive body of human consciousness that includes emotions, commonsense, intuition, and so on. This type of knowledge is difficult to computerize

THE SCOPE AND TYPES OF KNOWLEDGE

- The system builder must have a perfect understanding of the basic elements and their interactions, as produced by nature. To date, such a task has been found to be impossible. However, it is possible to implement a computerized representation that is deeper than shallow knowledge.

- To explain how this is done, let us return to the gasoline example. If we want to investigate at a deeper level the relationship between lack of gasoline and a car that won't start, we need to know the various components of the gas system (e.g., pipes, pump, filters, starter). Such a system is shown schematically in Figure 1.3.

THE SCOPE AND TYPES OF KNOWLEDGE

- To represent this system and knowledge of its operation, we use special knowledge representation methods such as semantic networks and frames. These allow the implementation of deeper-level reasoning such as abstraction and analogy, which are important expert activities. We can also represent the objects and processes of the domain of expertise at this level; the relationships between objects are important.

THE SCOPE AND TYPES OF KNOWLEDGE

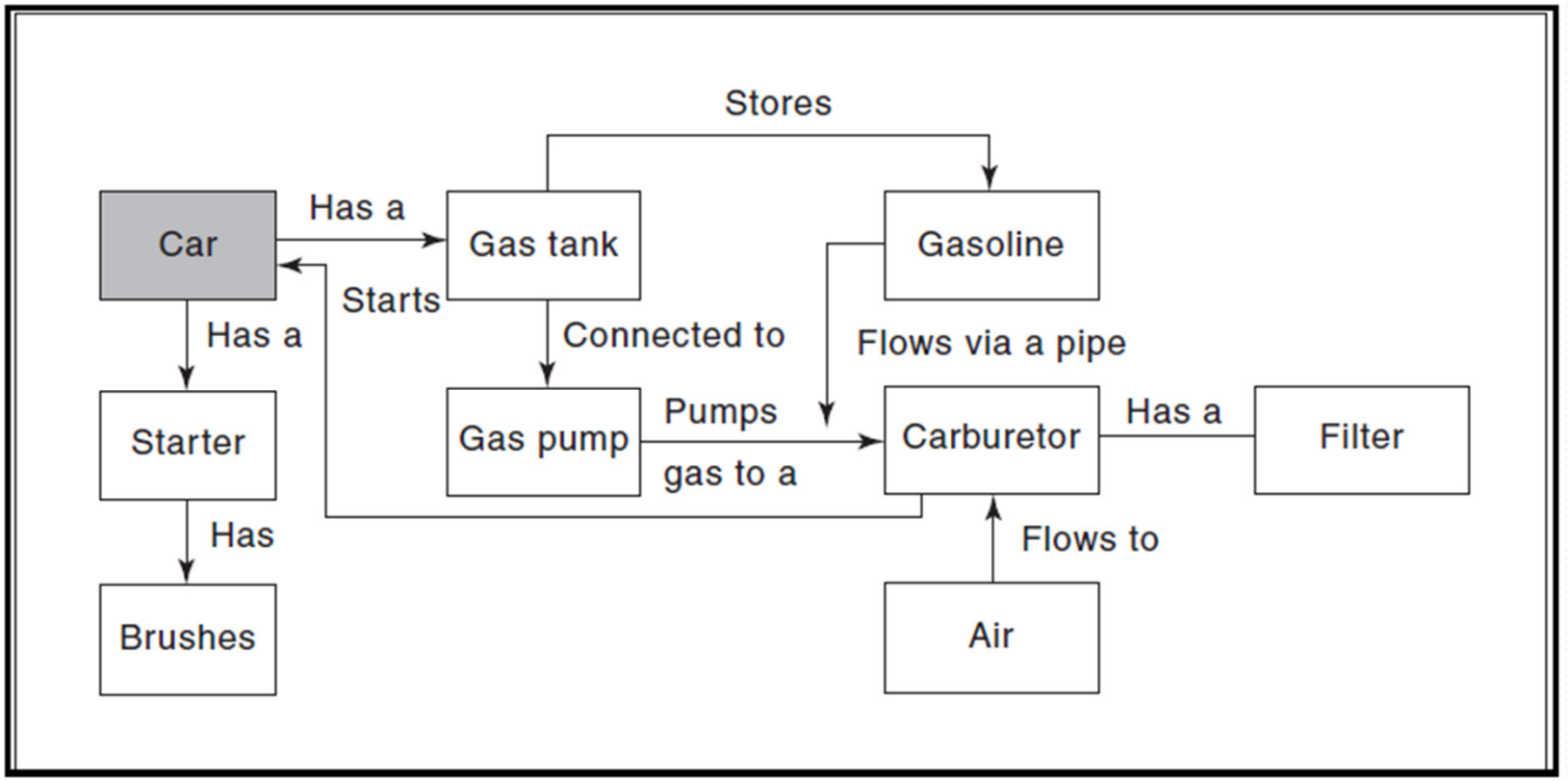


Figure 1.3: Schematic Representation of Deep Knowledge: Automobile Gas System

THE SCOPE AND TYPES OF KNOWLEDGE

Major Categories of Knowledge

- Knowledge can be categorized as declarative knowledge, procedural knowledge, or metaknowledge.

1. **Declarative knowledge:** is a descriptive representation of knowledge. It tells us facts: what things are. It is expressed in a factual statement, such as "**There is a positive association between smoking and cancer**." Domain experts tell us about truths and associations.

THE SCOPE AND TYPES OF KNOWLEDGE

- This type of knowledge is considered shallow, or surface-level, information that experts can verbalize. Declarative knowledge is especially important in the initial stage of knowledge acquisition.

2. **Procedural knowledge:** considers the manner in which things work under different sets of circumstances. The following is an example:

THE SCOPE AND TYPES OF KNOWLEDGE

- **"Compute the ratio between the price of a share and the earnings per share. If the ratio is larger than 12, stop your investigation. Your investment is too risky. If the ratio is less than 12, check the balance sheet."**

- Thus, procedural knowledge includes step-by-step sequences and how-to types of instructions; it may also include explanations. Procedural knowledge involves automatic responses to stimuli. It may also tell us how to use declarative knowledge and how to make inferences.

THE SCOPE AND TYPES OF KNOWLEDGE

- **Declarative knowledge** relates to a specific object. It includes information about the meaning, roles, environment, resources, activities, associations, and outcomes of the object. **Procedural knowledge** relates to the procedures used in the problem-solving process (e.g., information about problem definition, data gathering, the solution process, evaluation criteria).

3. **Metaknowledge**: is knowledge about knowledge. In ES, metaknowledge is knowledge about the operation of knowledge-based systems (i.e., about their reasoning capabilities).

The End

Knowledge Representation

Lecture2: Methods of Acquiring Knowledge

Instructor: Hazim N. Abed

Part I

Methods of Acquiring Knowledge From Experts

Methods of Acquiring Knowledge From Experts

- **Knowledge acquisition** is not an easy task. It includes **identifying the knowledge**, **representing the knowledge in a proper format**, **structuring the knowledge**, and **transferring the knowledge to a machine.**

- The process of knowledge acquisition can be greatly influenced by the roles of the three major participants: the knowledge engineer, the expert, and the end user.

Methods of Acquiring Knowledge From Experts

- The following are some factors that add to the complexity of knowledge acquisition from experts and its transfer to a computer (**Difficulties in Knowledge Acquisition**):

1. Experts may not know how to articulate their knowledge or may be unable to do so.
2. Experts may lack time or may be unwilling to cooperate.
3. Testing and refining knowledge is complicated.
4. Methods for knowledge elicitation may be poorly defined.

Methods of Acquiring Knowledge From Experts

5. System builders tend to collect knowledge from one source, but the relevant knowledge may be scattered across several sources.
6. Builders may attempt to collect documented knowledge rather than use experts. The knowledge collected may be incomplete.
7. It is difficult to recognize specific knowledge when it is mixed up with irrelevant data.
8. Experts may change their behavior when they are observed or interviewed.
9. Problematic interpersonal communication factors may affect the knowledge engineer and the expert.

Methods of Acquiring Knowledge From Experts

Required Skills of Knowledge Engineers.

- Knowledge engineers are human professionals who are able to communicate with experts and consolidate knowledge from various sources to build a valid knowledge base.

- They can use computers and special methods to overcome difficulties in knowledge engineering. The following are some of the skills and characteristics that are desirable in knowledge engineers:

Methods of Acquiring Knowledge From Experts

1. Computer skills (e.g., hardware, programming, software)
2. Tolerance and ambivalence
3. Effective communication abilities (e.g., sensitivity, tact, diplomacy)
4. Broad educational background
5. Advanced, socially-sophisticated verbal skills.
6. Fast-learning capabilities (of different domains)
7. Understanding of organizations and individuals

Methods of Acquiring Knowledge From Experts

8. Wide experience in knowledge engineering
9. Empathy and patience
10. Persistence
11. Logical thinking
12. Versatility and inventiveness
13. Self-confidence

- Because of these requirements, knowledge engineers are in short supply (and they are costly as well, demanding high salaries).

Roles Of Knowledge Engineers

- The ability and personality of the knowledge engineer directly influence the expert. Part of successful knowledge acquisition involves developing a positive relationship with the expert.

- The knowledge engineer is responsible for creating the right impression, positively communicating information about the project, understanding the expert's style, preparing the sessions, and so on.

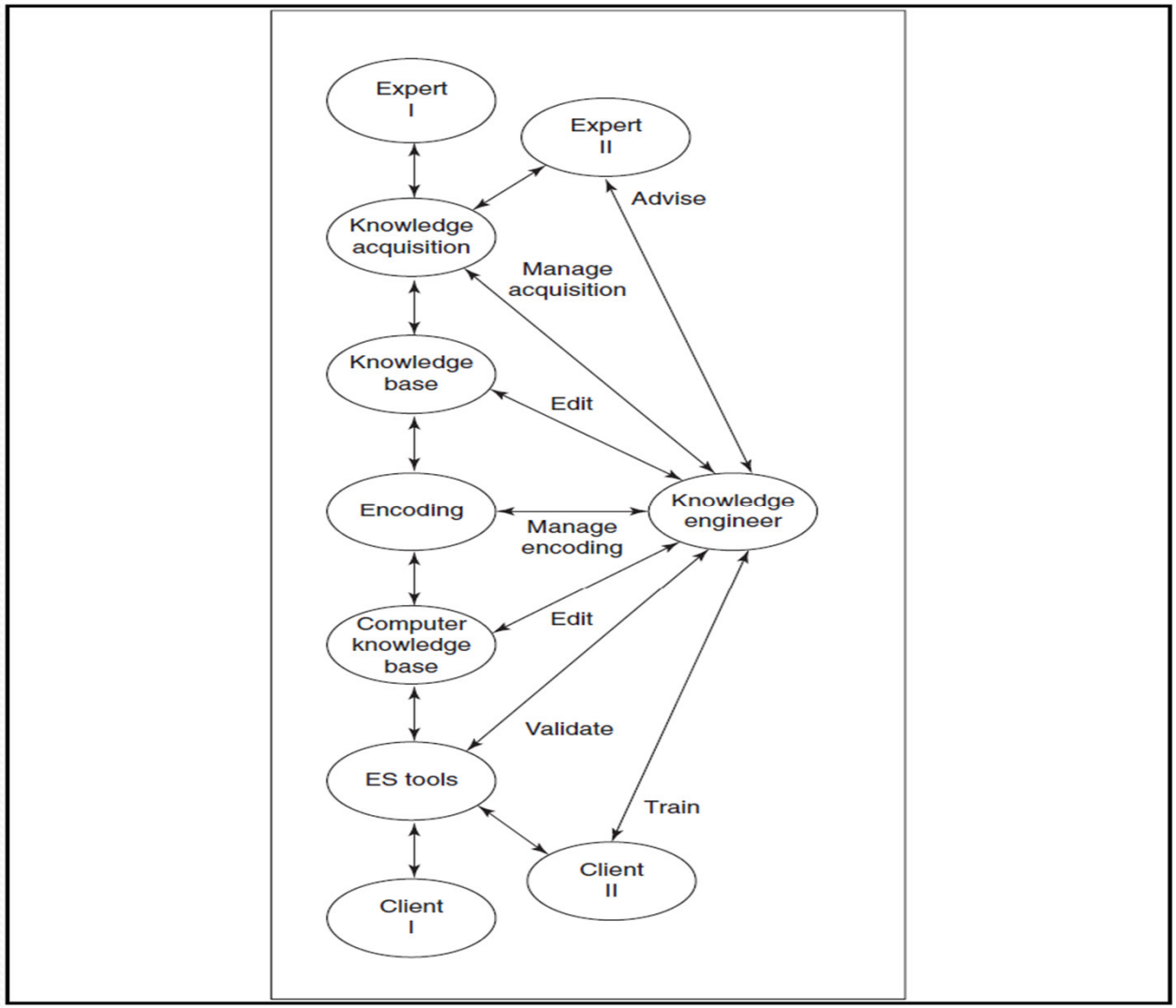
- The basic model of knowledge engineering portrays teamwork in which a knowledge engineer mediates between the expert and the knowledge base.

Roles Of Knowledge Engineers

- Figure 2.1 shows the following tasks performed by knowledge engineers at different stages of knowledge acquisition:

1. Advise the expert on the process of interactive knowledge elicitation.
2. Set up and appropriately manage the interactive knowledge acquisition tools.
3. Edit the unencoded and coded knowledge base in collaboration with the expert.
4. Set up and appropriately manage the knowledge-encoding tools.
5. Validate application of the knowledge base in collaboration with the expert.
6. Train clients in effective use of the knowledge base in collaboration with the expert by developing operational and training procedures.

Roles Of Knowledge Engineers



Knowledge Modeling Methods

- The knowledge modeling methods can be classified into three categories: manual, semiautomatic, and automatic.

1. **Manual methods** are basically structured around an interview of some kind. The knowledge engineer elicits knowledge from the expert or other sources and then codes it in the knowledge base.

- Manual methods are **slow**, **expensive**, and sometimes **inaccurate**. Therefore, there is a trend toward automating the process as much as possible.

Knowledge Modeling Methods

A. **Interviews** The most commonly used form of knowledge acquisition is face-to-face interview analysis. This is an explicit technique that appears in several variations. It involves a direct dialog between the expert and the knowledge engineer.

- Information is collected with the aid of conventional instruments (e.g., **tape recorders**, **questionnaires**) and is subsequently transcribed, analyzed, and coded.

Knowledge Modeling Methods

The three major manual methods are interviewing (i.e., **structured**, **semistructured**, **unstructured**).

1. **Unstructured interview**

- An interview in which the knowledge engineer has no pre-defined questions. Basically a chat to find out broad aspects of the expert's knowledge.

- Expert and knowledge engineer are free to explore the domain. This type of interview can prove useful as an initial interview when known of the domain is little

Knowledge Modeling Methods

- By interviewing the expert, the knowledge engineer slowly learns what is going on. Then he or she builds a representation of the knowledge in the expert's terms.

- The process of knowledge acquisition involves uncovering the attributes of a problem and making explicit the thought process (usually expressed as rules) that the expert uses to interpret them.

Knowledge Modeling Methods

2. **Semi-structured interview:**

- An interview in which pre-prepared questions are used to focus and scope what is covered Also involves unprepared supplementary questions for clarification and probing.

- The questions for a semi-structured interview are ideally constructed some time before the interview and are sent to the expert so he/she can start to prepare responses.

Knowledge Modeling Methods

3. **Structured interview:**

- An interview in which the knowledge engineer follows a pre-defined set of structured questions but can ask no supplementary questions .

- A significant benefit of the structured interview is that it provides structured transcripts that are easier to analyze than unstructured conversation. Often involves filling-in a matrix or generic headings.

Knowledge Modeling Methods

B. Observations :

- Sometimes it is possible to observe an expert at work. In many ways, this is the most obvious and straightforward approach to knowledge acquisition. However, the difficulties involved should not be underestimated.

- For example, most experts advise several people and may work in several domains simultaneously. In this case, the knowledge engineer's observations will cover all the other activities as well.

Knowledge Modeling Methods

- Therefore, large quantities of knowledge are being collected, of which only a little is useful. In particular, if recordings or videotapes are made, the cost of transcribing large amounts of knowledge should be carefully considered.

C. Case analysis.

- Experts are asked how they handled specific cases in the past. Usually, this method involves analyzing documentation. In addition to the experts, other people (e.g., managers, users) may be questioned.

Knowledge Modeling Methods

D. Critical incident analysis.

- In this approach, only selected cases are investigated, usually those that are memorable, difficult, or of special interest. Both experts and non-experts may be questioned.

E. Discussions with users.

- Even though users are not experts, they can be quite knowledgeable about some aspects of the problem. They can also indicate areas where they need help. The expert may be unaware of some of the users' needs.

Knowledge Modeling Methods

F. Commentaries.

- With this method, the knowledge engineer asks experts to give a running commentary on what he or she is doing. This method can be supported by videotaping the experts in action or by asking an observer to do the commentary.

G. Conceptual graphs and models.

- Diagrams and other graphical methods can be instrumental in supporting other acquisition methods. A conceptual model can be used to describe how and when the expert's knowledge will come into play as the ES performs its task.

Knowledge Modeling Methods

2. Semiautomatic Methods

- Knowledge acquisition can be supported by computer-based tools. These tools provide an environment in which knowledge engineers or experts can identify knowledge through an interactive process. **Repertory grid analysis (RGA)** is one typical method.

- **Repertory grid analysis (RGA):** An approach in which each person is viewed as a "personal scientist" who seeks to predict and control events by forming theories, testing hypotheses, and analyzing results of experiments

Knowledge Modeling Methods

How RGA works

- The expert identifies the important objects in the domain of expertise. **For example, computer languages (e.g., LISP, C11, COBOL)**

- The expert identifies the important attributes considered in making decisions in the domain. **For example, the availability of commercial packages and ease of programming are important factors in selecting a computer language.**

Knowledge Modeling Methods

- For each attribute, the expert is asked to establish a bipolar scale with distinguishable characteristics **(i.e., traits)** and their opposites**. For example, in selecting a computer language, the information shown in Table 2.1 can be included.**

Table 2.1: RGA Input for Selecting a Computer Language

Attribute	*Trait*	*Opposite*
Availability	Widely available	Not available
Ease of programming	High	Low
Training time	Low	High
Orientation	Symbolic	Numeric

Knowledge Modeling Methods

- The interviewer picks any three of the objects and asks, "What attributes and traits distinguish any two of these objects from the third?" **For example, if a set includes LISP, PROLOG, and COBOL, the expert may point to orientation.**

- **Then the expert will say that LISP and PROLOG are symbolic in nature, whereas COBOL is numeric.**

- **These answers are translated to points on a scale of 1 to 3 (or 1 to 5).This step continues for several triplets of objects.**

Knowledge Modeling Methods

- **The answers are recorded in a grid, as shown in Table 2.2. The numbers in the grid designate the points assigned to each attribute for each object..**

Table 2.2: Example of a Grid

Attribute	*Orientation*	*Ease of Programming*	*Training Time*	*Availability*
Trait	*Symbolic (3)*	*High (3)*	*High (1)*	*High (3)*
Opposite	*Numeric (1)*	*Low (1)*	*Low (3)*	*Low (1)*
LISP	3	3	1	1
PROLOG	3	2	2	2
C++	3	2	2	3
COBOL	1	2	1	3

Knowledge Modeling Methods

- The grid can be used afterward to make recommendations in situations in which the importance of the attributes is known.

Use of RGA in ES.

- A number of knowledge acquisition tools have been developed based on RGA. These tools are aimed at helping in the conceptualization of the domain.

- Three representative tools are ETS,AQUINAS,

Knowledge Modeling Methods

- **An expert transfer system (ETS)** is a computer program that interviews experts and helps them build ES.

- **AQUINAS** is a very complex tool that extends the problem-solving and knowledge representation of ETS by allowing experts to structure knowledge in hierarchies.

- Other RGA Tools: **PCGRID (PC-based)**, **WebGrid** , **Circumgrids**

Knowledge Modeling Methods

- **Integrated Knowledge Acquisition Aids tools:**
  - PROTÉGÉ-II
  - KSM
  - ACQUIRE
  - KADS (Knowledge Acquisition and Documentation System)

Knowledge Modeling Methods

3. **Automatic methods**

- The process of using computers to extract knowledge from data is called **knowledge discovery**

- **Two reasons for the use of automated knowledge acquisition:**

1. Good knowledge engineers are highly paid and difficult to find
2. Domain experts are usually busy and sometimes uncooperative

Part II

Methods of Acquiring Knowledge From Multiple Experts

Methods of Acquiring Knowledge From Multiple Experts

- An important element in the development of an ES is the identification of experts.
- The usual approach to this problem is to build ES for a very narrow domain in which expertise is clearly defined; then it is easy to find one expert.
- However, even though many ES have been constructed with one expert an approach advocated as a good strategy for ES construction there could be a need for multiple experts, especially when more serious systems are being constructed or when expertise is not particularly well defined

Methods of Acquiring Knowledge From Multiple Experts

- **The following are the major purposes of using multiple experts:**

1. To better understand the knowledge domain
2. To improve knowledge-base validity, consistency, completeness, accuracy, and relevancy
3. To provide better productivity
4. To identify incorrect results more easily
5. To address broader domains
6. To be able to handle more complex problems and combine the strengths of different reasoning approaches

Methods of Acquiring Knowledge From Multiple Experts

- **The benefits of multiple experts:**

1. On average, fewer mistakes than with a single expert
2. Elimination of the need for using a world-class expert (who is difficult to get and expensive)
3. Wider domain than a single expert's Synthesis of expertise
4. Enhanced quality due to synergy among experts

Methods of Acquiring Knowledge From Multiple Experts

- **The Limitations of multiple experts:**

1. Groupthink phenomenon
2. Fear on the part of some domain experts of senior experts or a supervisor (i.e., lack of confidentiality)
3. Compromising solutions generated by a group with conflicting opinions
4. Wasted time in group meetings
5. Difficulties in scheduling the experts
6. Dominating experts (i.e., controlling, not letting others speak)

Methods of Acquiring Knowledge From Multiple Experts

- **There are four possible scenarios, or configurations, for using multiple experts.**

1. **Individual Experts:** In this case, several experts contribute knowledge individually. Using multiple experts in this manner relieves the knowledge engineer of the stress associated with multiple expert teams.

Methods of Acquiring Knowledge From Multiple Experts

2. Primary and Secondary Experts

- A primary expert may be responsible for validating information retrieved from other domain experts. Knowledge engineers may initially consult the primary expert for guidance in domain familiarization, refinement of knowledge acquisition plans, and identification of potential secondary experts.

- These scenarios determine in part the method to be used for handling multiple experts.

Methods of Acquiring Knowledge From Multiple Experts

3. Small Groups

- Several experts may be consulted together and asked to provide agreed-upon information. Working with small groups of experts allows the knowledge engineer to observe alternative approaches to the solution of a problem and the key points made in solution-oriented discussions among experts.

Methods of Acquiring Knowledge From Multiple Experts

4. Panels

- To meet goals for verification and validation of ongoing development efforts, a program may establish a council of experts. The members of the council typically meet together at times scheduled by the developer for the purpose of reviewing knowledge base development efforts, content, and plans. In many cases, the functionality of the ES is tested against the expertise of such a panel.

Automatic Knowledge Acquisition from Documented

- Knowledge can often be acquired from other sources in addition to, or instead of, human experts. This approach has the major advantage of eliminating the need to use an expert.

- It is used in knowledge-based systems in which handling a large amount of or complex information rather than world-class expertise is the main concern.

Automatic Knowledge Acquisition from Documented

- **The following are the objectives of using automated knowledge acquisition:**
  - To increase the productivity of knowledge engineering (reduce the cost)
  - To reduce the skill level required from the knowledge engineer
  - To eliminate (or drastically reduce) the need for an expert
  - To eliminate (or drastically reduce) the need for a knowledge engineer
  - To increase the quality of the acquired knowledge

Knowledge Verification and Validation

- Knowledge acquired from experts needs to be evaluated for quality, including **evaluation**, **validation**, and **verification**. These terms are often used interchangeably.

- **Evaluation** is a broad concept. Its objective is to assess an ES's overall value. In addition to assessing acceptable performance levels, it analyzes whether the system would be usable, efficient, and cost-effective.

Knowledge Verification and Validation

- **Validation** is the part of evaluation that deals with the performance of the system (e.g., as it compares to the expert's). Simply stated, validation is building the right system (i.e., substantiating that a system performs with an acceptable level of accuracy).

- **Verification** is building the system right or substantiating that the system is correctly implemented to its specifications.

Knowledge Verification and Validation

- **Validation method:** This approach tests the extent to which the system and the expert decisions agree, the inputs and processes used by an expert compared to the machine, and the difference between expert and novice decisions.

- **Automated verification:** Verification is conducted by measuring the system's performance and is limited to classification cases with probabilities. It works as follows: When an ES is presented with a new case to classify, it assigns a confidence factor to each selection

Knowledge Verification and Validation

- By comparing these confidence factors with those provided by an expert, it is possible to measure the accuracy of the ES for each case. By performing comparisons on many cases, it is possible to derive an overall measure of ES performance

The End

Knowledge Representation

Lecture 3:Introduction to KR

Instructor: Hazim N. Abed

Knowledge Representation

What is Knowledge?

- The Chambers 20th Century Dictionary provides as good a definition as any: “knowledge, nol¢ij, n. assured belief; that which is known; information; …”

- In order to solve the complex problems encountered in AI, one generally needs a large amount of knowledge, and suitable mechanisms for representing and manipulating all that knowledge.

- Knowledge can take many forms. Some simple examples are:

Knowledge Representation

- *John has an umbrella*
- *It is raining*
- *An umbrella stops you getting wet when it's raining*
- *An umbrella will only stop you getting wet if it is used properly*
- *Umbrellas are not so useful when it is very windy*

- So, how should an AI agent store and manipulate knowledge like this?

Knowledge Representation

- **Knowledge representation** (KR) is the study of
  - how knowledge and facts about the world can be represented, and
  - what kinds of reasoning can be done with that knowledge.

- **Knowledge representation** is the study of how to put knowledge into a form that a computer can reason with

- The object of **a knowledge representation** is to express knowledge in a computer tractable form, so that it can be used to enable our AI agents to perform well.

Knowledge Representation

- A knowledge representation language is defined by two aspects:

- 1. **Syntax**: The syntax of a language defines which configurations of the components of the language constitute valid sentences.

- 2. **Semantics**: The semantics defines which facts in the world the sentences refer to, and hence the statement about the world that each sentence makes.

Knowledge Representation

- This is a very general idea, and not restricted to natural language.

- **For Example:** suppose the language is arithmetic, then ‘x’, ‘≥’ and ‘y’ are components (or symbols or words) of the language the **syntax** says that ‘x ≥ y’ is a valid sentence in the language, but ‘≥ ≥ x y’ is not the semantics say that ‘x ≥y’ is false if y is bigger than x, and true otherwise

Requirements of a Knowledge Representation

- A good knowledge representation system for any particular domain should possess the following properties:

1. **Representational Adequacy**:- the ability to represent all the different kinds of knowledge that might be needed in that domain.

2. **Inferential Adequacy:-** the ability to manipulate the representational structures to derive new structures (corresponding to new knowledge) from existing structures.

Knowledge Representation

3. Inferential Efficiency:- the ability to incorporate additional information into the knowledge structure which can be used to focus the attention of the inference mechanisms in the most promising directions.

4. Acquisitional Efficiency:- the ability to acquire new information easily. Ideally the agent should be able to control its own knowledge acquisition, but direct insertion of information by a 'knowledge engineer' would be acceptable.

- Finding a system that optimizes these for all possible domains is not going to be feasible.

Practical Aspects of Good Representations

- In practice, the theoretical requirements for good knowledge representations can usually be achieved by dealing appropriately with a number of practical requirements:

1. The representations need to be ***complete*** – so that everything that could possibly need to be represented, can easily be represented.

2. They must be ***computable*** – implementable with standard computing procedures.

Practical Aspects of Good Representations

3. They should make the important ***objects*** and ***relations*** explicit and accessible – so that it is easy to see what is going on, and how the various components interact.

4. They should ***suppress irrelevant detail*** – so that rarely used details don't introduce unnecessary complications, but are still available when needed.

5. They should expose any natural ***constraints*** – so that it is easy to express how one object or relation influences another.

Practical Aspects of Good Representations

6. They should be ***transparent*** – so you can easily understand what is being said.

7. The implementation needs to be ***concise*** and ***fast*** – so that information can be stored, retrieved and manipulated rapidly.

Components of a Good Representation

- For analysis purposes it is useful to be able to break any knowledge representation down into their four fundamental components:

1. The **lexical** part:- that determines which symbols or words are used in the representation's ***vocabulary***.

2. The **structural** or **syntactic** part :- that describes the ***constraints*** on how the symbols can be arranged, i.e. a grammar.

Components of a Good Representation

3. The **semantic** part:- that establishes a way of associating ***real world meanings*** with the representations.

4. The **procedural** part:- that specifies the access procedures that enables ways of ***creating*** and ***modifying*** representations and ***answering questions*** using them, i.e. how we generate and compute things with the representation.

Knowledge Representation Applications

- The industry has applied Knowledge Representation techniques mainly to extend database properties and capabilities. Then, as research matured, several sophisticated applications such as:

  - Knowledge-Based Systems (KBS),
  - Expert Systems (ES),
  - Intelligent Decision Support Systems (DSC).

Knowledge Representation Methods

- There are many methods can be used for knowledge representation and they can be described as follows:-

1- Semantic net.

2- Conceptual graph.

3- Frames

4- Prepositional and Predicates calculus.

5- Resolution.

Knowledge Representation Methods

1- Semantic net:

- Semantic networks are knowledge representation schemes involving nodes and links (arcs or arrows) between nodes. The nodes represent objects or concepts and the links represent relations between nodes. A semantic net has a binary relation.

Drawbacks:

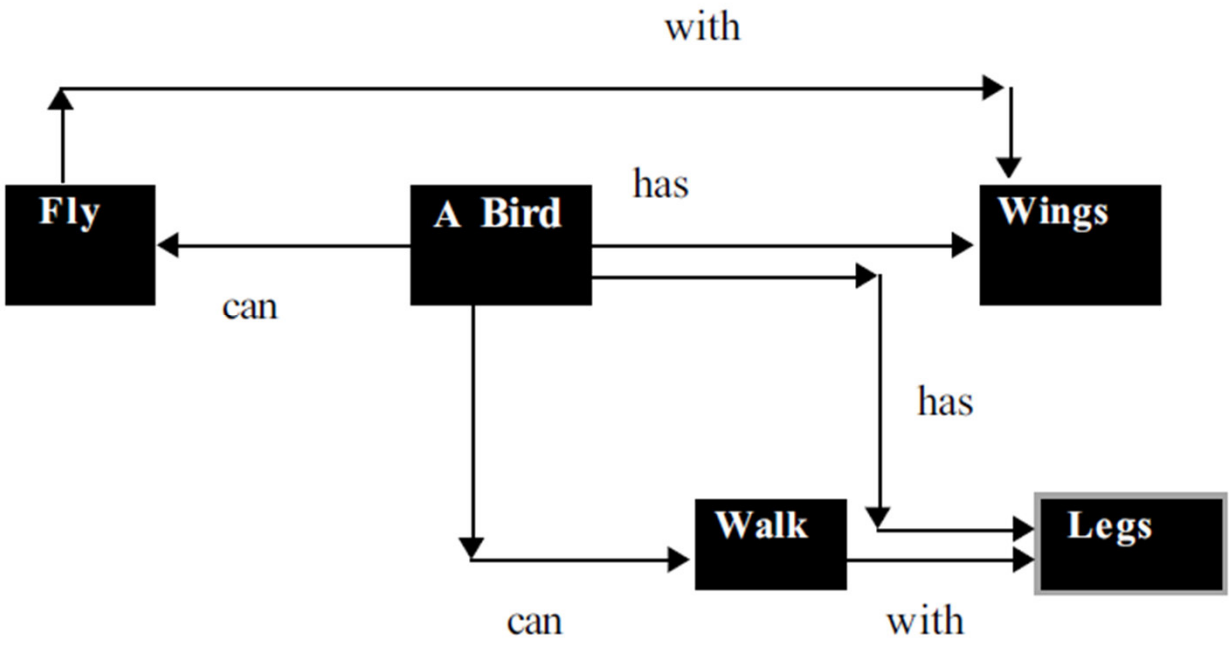
- Disjunctive and conjunctive information cannot be included into semantic nets
  - E.g. apple can be either green or red
  - E.g. panda has color black and white

Knowledge Representation Methods

- Examples of relationship labeled on arcs (notice that there is an underscore)
  - is_a
  - has_a
  - has_part
- Examples of concepts (nodes)
  - bird
  - person
  - book
  - famous
  - intelligent

Knowledge Representation Methods

- **Example**: Create the semantic network for the following facts A bird can fly with wings. A bird has wings. A bird has legs. A bird can walk with legs.



A semantic net representation of "birds"

Knowledge Representation Methods

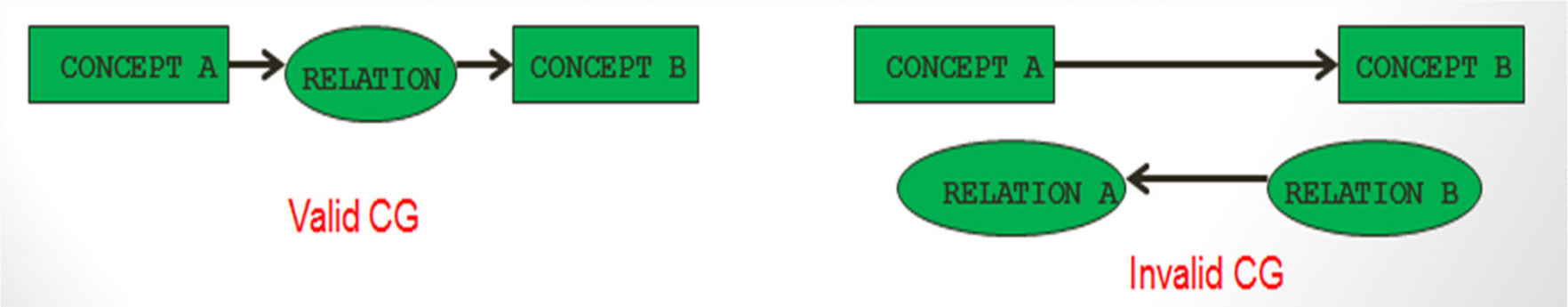
2- Conceptual Graph

- The main feature is standardized graphical representation that like in the case of semantic networks allows human to get quick overview of what the graph means. Conceptual graph is a bipartite orientated graph where instances of concepts are displayed as ***rectangle*** and conceptual relations are displayed as ***ellipse***.

- Developed in 1984, conceptual graphs (networks) overcome the restriction to binary relation.

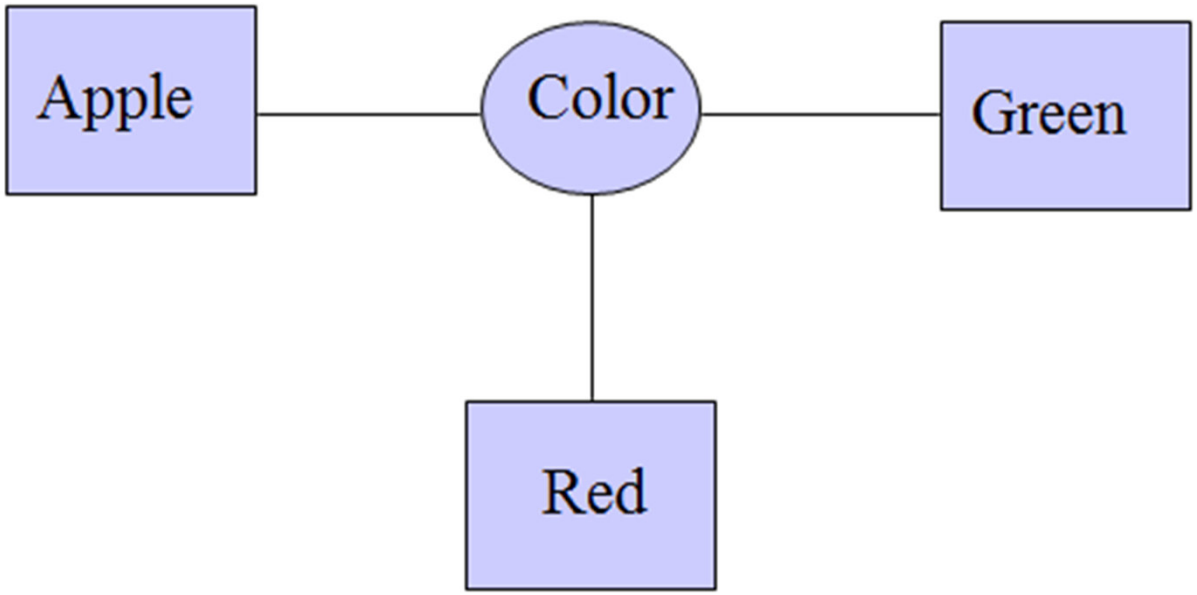
Knowledge Representation Methods

- There 2 types of node in the CG:
  - **Concept**: the knowledge / fact / action
  - **Relation**: the type of relationship between 2 concepts.
- Rule in CG:
  - There are NO arcs between a concept and another concept, and no arcs between a relation and another relation. All arcs either go from a concept to a relation or from a relation to a concept.



Knowledge Representation Methods

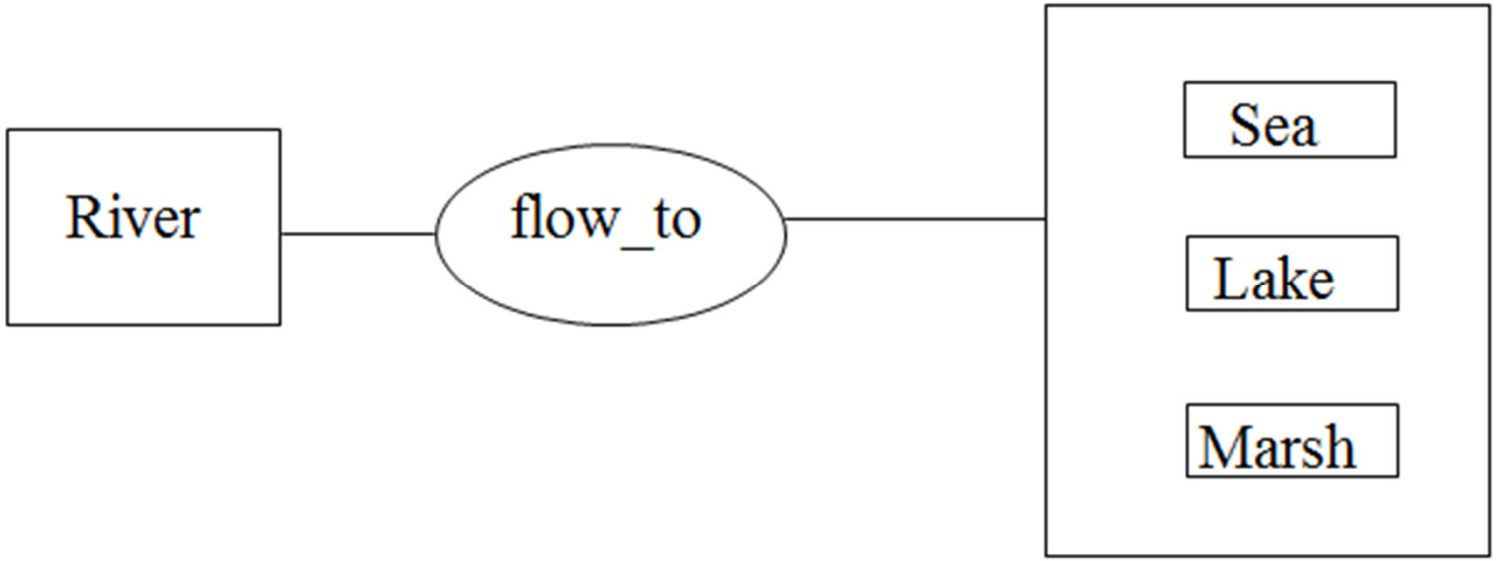
- A Disjunctive Net for Red or Green Apple.



A Conceptual Net that represents "OR"

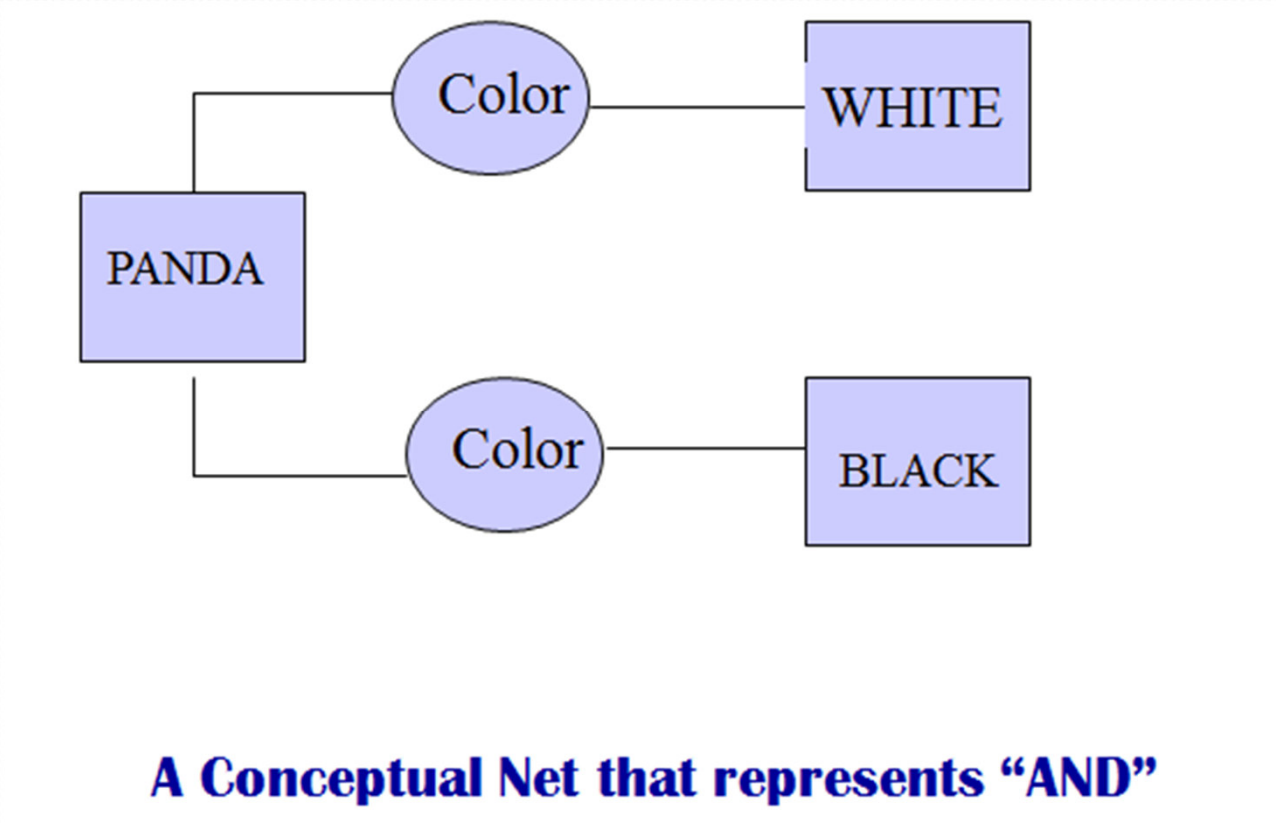
Knowledge Representation Methods

- Conceptual Nets For ‘Where do Rivers Flow to’?



Knowledge Representation Methods

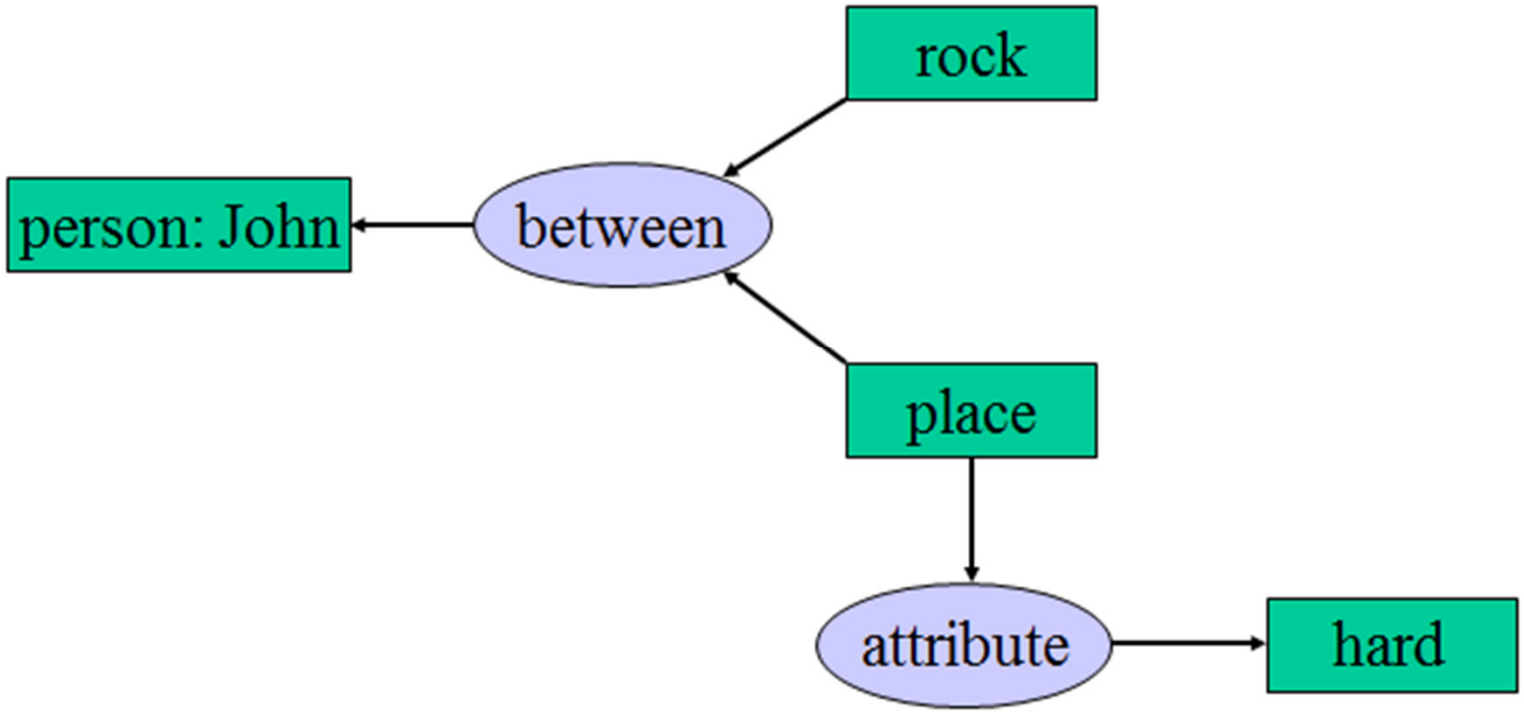
- A Conjunctive Net for black and white panda



A Conceptual Net that represents "AND"

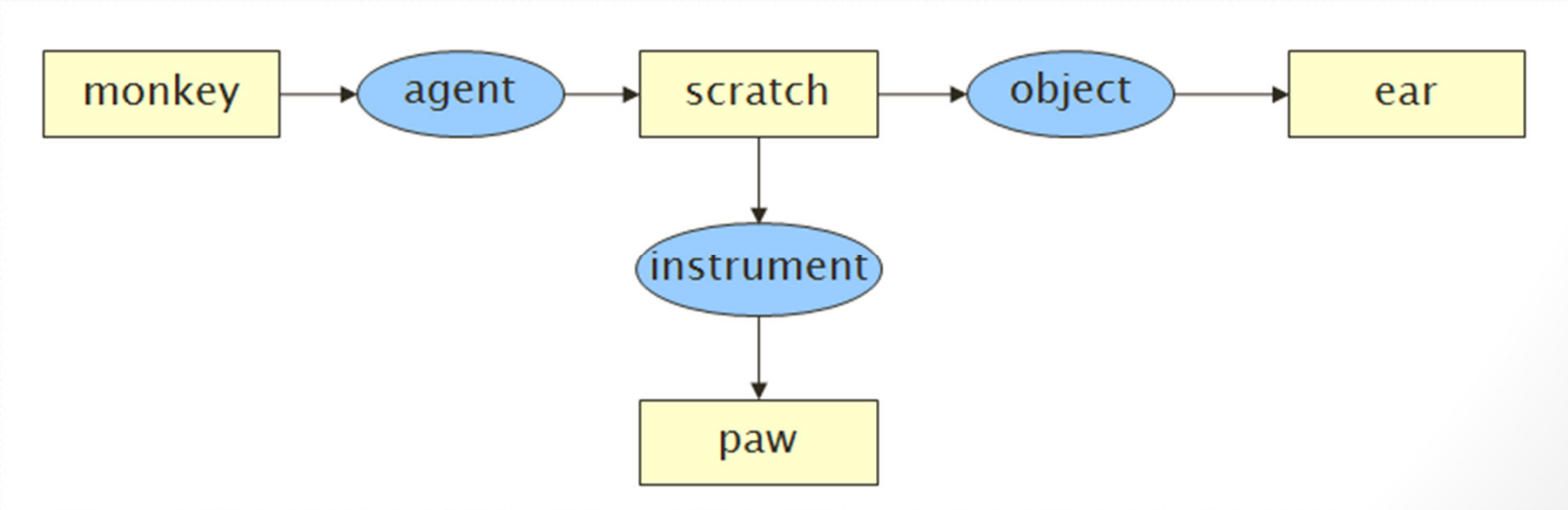
Knowledge Representation Methods

- "John is between a rock and a hard place"



Knowledge Representation Methods

- For Example: A monkey scratch its ear with a paw



Knowledge Representation Methods

3- Frames

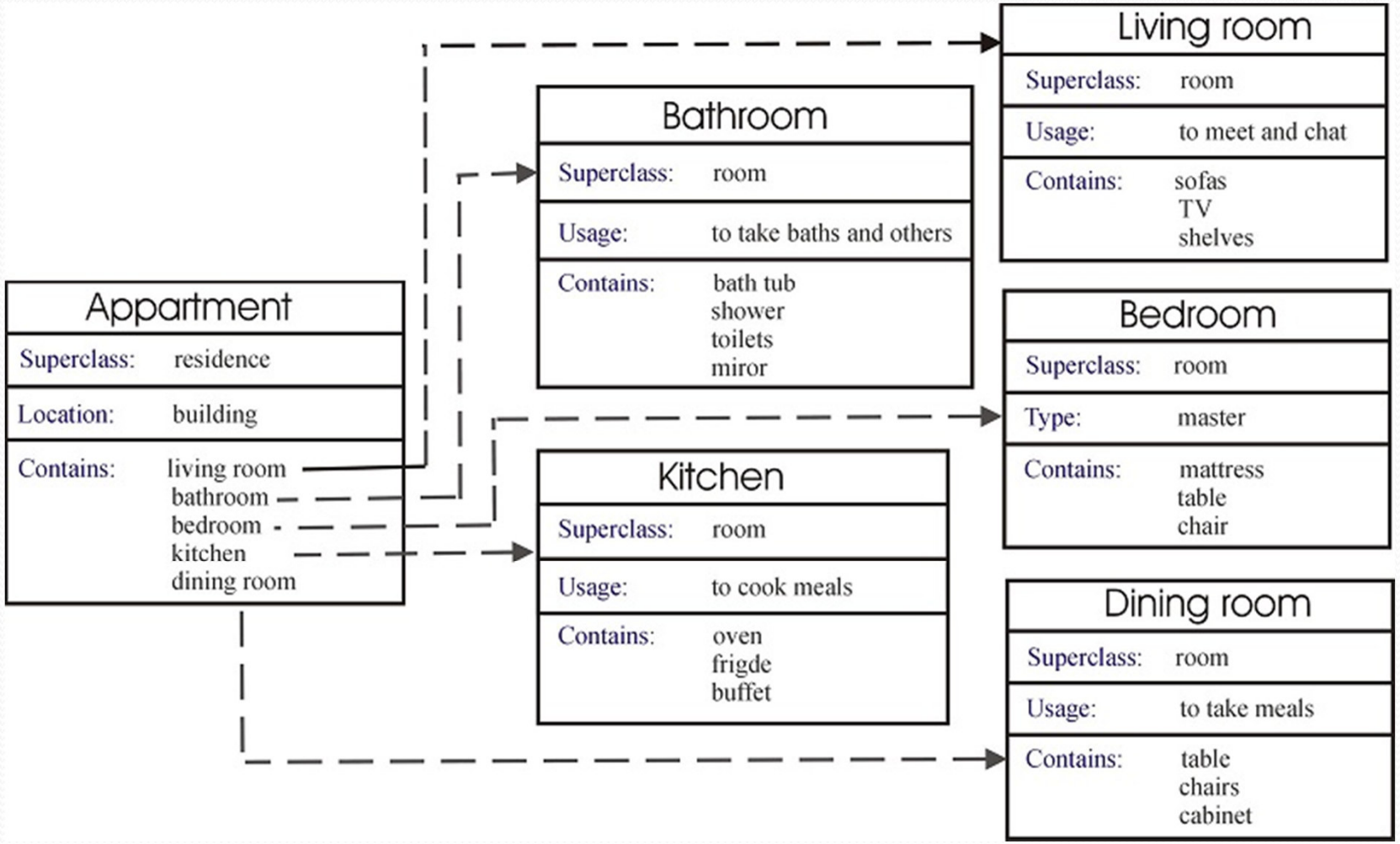
- Introduced by Marvin Minsky in 1974.
- It extends semantic network to provide a more structured way of representing a knowledge base.
- It stores properties, values, methods and relevant information of object.
- Frame supports class hierarchies applied in object oriented concept.
- Each frame has:
  - A name
  - A slot which stores information like specific value, default value, inherited value, a pointer to another frame (superclass or subclass)

Knowledge Representation Methods

Example 1: In fig. 2, we are attempting to represent in a frame format [5] knowledge contained in the following sentence: “An apartment (name) is a kind of residence (super class) which is typically composed of five rooms (attributes/slots):

- a bathroom which contains a bath tub, a shower, toilets and a mirror;
- a kitchen where we cook meals contains an oven, a fridge and a buffet;
- a living room where people meet and chat, contains sofas, a TV set and shelves;
- a bedroom and
- a dining room. ” (more details in the following picture)

Knowledge Representation Methods

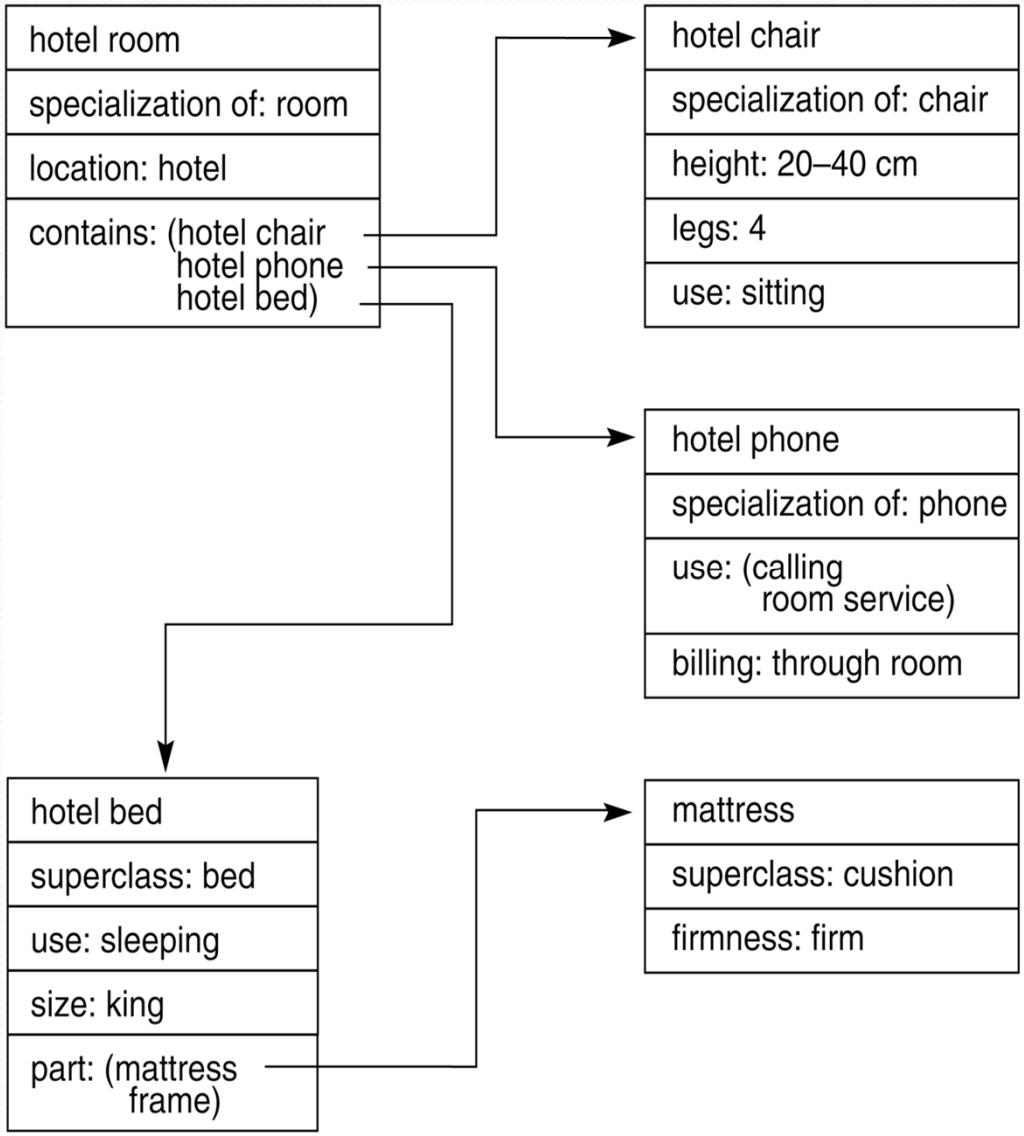


Knowledge Representation Methods

- It is a brief description which fits perfectly the expectations of a person who enters an apartment. Expectations are exactly what a frame representation is all about. The same example can be described in terms of semantic networks, decision trees, or with ontologies.

Knowledge Representation Methods

Example 2:



Chapter 2: The Predicate Calculus

Instructor: Dr. Ebaa Fayyoumi
IT 229
email: enfayyoumi@hu.edu.jo

1 Propositional Calculus

In this chapter we will introduce the propositional calculus and its presentation language for AI. We will explain the syntax and the semantic of the Propositional and predicate calculus language plus the inference rule.

- Propositional calculus uses *words*, *phrases* and *sentences* to represent and reason about properties and relationships in the world.
- Proposition: True (1) or False (0)
- Examples about propositional calculus:
  - The moon is made of green cheese.
  - Open the door $\longrightarrow$ X imperative.
  - "What time is it?" $\longrightarrow$ X interrogative
- Propositional calculus consists of
  - Syntax {Symbols and Sentences}.
  - semantic.

1.1 Propositional Syntax

DEFINITION

PROPOSITIONAL CALCULUS SYMBOLS

The *symbols* of propositional calculus are the propositional symbols:

P, Q, R, S, …

truth symbols:

true, false

and connectives:

$\wedge, \vee, \neg, \rightarrow, \equiv$

DEFINITION

PROPOSITIONAL CALCULUS SENTENCES

Every propositional symbol and truth symbol is a sentence.

For example: true, P, Q, and R are sentences.

The *negation* of a sentence is a sentence.

For example: $\neg$ P and $\neg$ false are sentences.

The *conjunction*, or *and*, of two sentences is a sentence.

For example: P $\wedge \neg$ P is a sentence.

The *disjunction*, or *or*, of two sentences is a sentence.

For example: P $\vee \neg$ P is a sentence.

The *implication* of one sentence from another is a sentence.

For example: P $\rightarrow$ Q is a sentence.

The *equivalence* of two sentences is a sentence.

For example: P $\vee$ Q $\equiv$ R is a sentence.

Legal sentences are also called *well-formed formulas* or *WFFs*.

Notes!!!

- The expression of the form $(P \bigwedge Q)$ (P and Q) is called "Conjuncts".
- The expression of the form $(P \bigvee Q)$ (P or Q) is called "Disjuncts".
- The expression of the form $(P \Rightarrow Q)$ (P implies Q),
  - P is the "premise" or "antecedent"
  - Q is a "conclusion" or "consequent"

1.2 Propositional Semantic

- An interpretation of a set of propositions is the assignments of a truth value , either T or F, to each propositional symbols.
- The symbol True is always assigned T, and the symbol False is assigned F.

DEFINITION

PROPOSITIONAL CALCULUS SEMANTICS

An *interpretation* of a set of propositions is the assignment of a truth value, either T or F, to each propositional symbol.

The symbol true is always assigned T, and the symbol false is assigned F.

The interpretation or truth value for sentences is determined by:

The truth assignment of *negation*, $\neg$ P, where P is any propositional symbol, is F if the assignment to P is T, and T if the assignment to P is F.

The truth assignment of *conjunction*, $\wedge$, is T only when both conjuncts have truth value T; otherwise it is F.

The truth assignment of *disjunction*, $\vee$, is F only when both disjuncts have truth value F; otherwise it is T.

The truth assignment of *implication*, $\rightarrow$, is F only when the premise or symbol before the implication is T and the truth value of the consequent or symbol after the implication is F; otherwise it is T.

The truth assignment of *equivalence*, $\equiv$, is T only when both expressions have the same truth assignment for all possible interpretations; otherwise it is F.

- Negation "Not" Symbol $\neg$.

  P : I am going to town.

  $\neg P$: I am not going to town;

  It is not the case that I am going to town;

p	$\neg p$
T	F
F	T

- Conjunction "and" Symbol $\bigwedge$

  P : I am going to town

  Q : It is going to rain

  $P \wedge Q$: I am going to town and it is going to rain.

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

- Disjunction "or" Symbol $\bigvee$

  P : I am going to town

  Q : It is going to rain

  $P \vee Q$: I am going to town or it is going to rain.

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

- Implication "If...then..." Symbol $\rightarrow$

  P : I am going to town

  Q : It is going to rain

  $P \rightarrow Q$: If I am going to town then it is going to rain.

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

- Biconditional "if and only if" Symbol $\leftrightarrow$

  P : I am going to town

  Q : It is going to rain

  $P \leftrightarrow Q$: I am going to town if and only if it is going to rain.

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

For propositional expressions **P**, **Q** and **R**:

$\neg (\neg \mathbf{P}) \equiv \mathbf{P}$

$(\mathbf{P} \vee \mathbf{Q}) \equiv (\neg \mathbf{P} \rightarrow \mathbf{Q})$

the contrapositive law: $(\mathbf{P} \rightarrow \mathbf{Q}) \equiv (\neg \mathbf{Q} \rightarrow \neg \mathbf{P})$

de Morgan's law: $\neg (\mathbf{P} \vee \mathbf{Q}) \equiv (\neg \mathbf{P} \wedge \neg \mathbf{Q})$ and $\neg (\mathbf{P} \wedge \mathbf{Q}) \equiv (\neg \mathbf{P} \vee \neg \mathbf{Q})$

the commutative laws: $(\mathbf{P} \wedge \mathbf{Q}) \equiv (\mathbf{Q} \wedge \mathbf{P})$ and $(\mathbf{P} \vee \mathbf{Q}) \equiv (\mathbf{Q} \vee \mathbf{P})$

the associative law: $((\mathbf{P} \wedge \mathbf{Q}) \wedge \mathbf{R}) \equiv (\mathbf{P} \wedge (\mathbf{Q} \wedge \mathbf{R}))$

the associative law: $((\mathbf{P} \vee \mathbf{Q}) \vee \mathbf{R}) \equiv (\mathbf{P} \vee (\mathbf{Q} \vee \mathbf{R}))$

the distributive law: $\mathbf{P} \vee (\mathbf{Q} \wedge \mathbf{R}) \equiv (\mathbf{P} \vee \mathbf{Q}) \wedge (\mathbf{P} \vee \mathbf{R})$

the distributive law: $\mathbf{P} \wedge (\mathbf{Q} \vee \mathbf{R}) \equiv (\mathbf{P} \wedge \mathbf{Q}) \vee (\mathbf{P} \wedge \mathbf{R})$

P	Q	¬P	¬P∨Q	P⇒Q	(¬P∨Q)=(P⇒Q)
T	F	F	F	F	T
F	T	T	T	T	T
F	F	T	T	T	T

Constructing New Logical Equivalence

$\neg(p \to q) \equiv p \wedge \neg q$

$\neg(\neg p \vee q)$ using Implication
$\neg(\neg p) \wedge \neg q$ using Demorgan's Law
$p \wedge \neg q$ using Double Negation

$\neg(p \vee (\neg p \wedge q)) \equiv \neg p \wedge \neg q$

$\neg p \wedge \neg(\neg p \wedge q)$ using Demorgan's Law
$\neg p \wedge (p \vee \neg q)$ using Demorgan's Law and Double Negation
$(\neg p \wedge p) \vee (\neg p \wedge \neg q)$ using Distribution Law
$F \vee (\neg p \wedge \neg q)$ using Negation Law
$\neg p \wedge \neg q$ using Identity Law

$p \wedge q \rightarrow p \vee q \equiv T$

$\neg(p \wedge q) \vee (p \vee q)$	using Implication
$(\neg p \vee \neg q) \vee (p \vee q)$	using Demorgan's Law
$(\neg p \vee p) \vee (\neg q \vee q)$	using Commutative Law
$T \vee T$	using Domination Law
T	using Tatulogy.

2 The Predicate Calculus

In propositional calculus

- Each atomic symbol P, Q or R represents a single proposition.
- There is no way to access the component of an individual assertions.

Predicate calculus exists for the following reasons:

- Accessing the component of an individual assertion.
- Manipulating predicate calculus expressions.
- Inferring new predicate sentences.

Examples:

Instead of saying ***P = "It rains on Tuesday".***
We could create predicate **weather** in order to describe the relationship between the week day and the weather.
weather(tuesday, rainy)
We could generalize this sentence to describe the weather for any day in the week such as:

- ***weather(sunday,rainy)***
- ***weather(monday,rainy)***
- ***weather(tuesday,rainy)***
  ⋮

- ***weather(X,rainy)*** X is a variable represents any day in the week.
- ***weather(X,sunny)***
- ***weather(X,windy)***
  ⋮

- ***weather(X, Y)*** Y is a variable represents the weather at certain day.

2.1 The Syntax of Predicates and Sentences

DEFINITION

PREDICATE CALCULUS SYMBOLS

The alphabet that makes up the symbols of the predicate calculus consists of:

1. The set of letters, both upper- and lowercase, of the English alphabet.
2. The set of digits, 0, 1, ..., 9.
3. The underscore, _.

Symbols in the predicate calculus begin with a letter and are followed by any sequence of these legal characters.

Legitimate characters in the alphabet of predicate calculus symbols include

a R 6 9 p _ z

Examples of characters not in the alphabet include

% @ / & " "

Legitimate predicate calculus symbols include

George fire3 tom_and_jerry bill XXXX friends_of

Examples of strings that are not legal symbols are

3jack "no blanks allowed" ab%cd ***71 duck!!!

Important Note

- l(g,k) ≡ like(john,kate).
  These two expressions are equivalent, but the second expression can be of great help in indicating what relationship the expression presents.
  These descriptive names are intended to improve the readability of expression.

- Improper Symbols are Parentheses "()", Comma "," and Period ".", which are used to construct a WFF.

- Predicate calculus may represent
  - Constant: Specifying certain object or property in the world. It must start with lowercase letter.
    * Examples: blue, tree and tall.
    * "true" and "false" are reserved truth symbols.

  - Variables: Specifying general class of objects or properties in the world. It must start with uppercase letter.
    * Kate and George are legal variables.
    * bill and george are not legal variables.
  - Functions: Mapping one or more elements in the domain to a unique elements in the range. It must start with lowercase letter.

- Each function has an associated ***arity***, which indicates the number of elements in the domain mapped onto each element of the range.
  - father(sam) $\rightarrow$ arity = 1.
  - plus(x1,x2)$\rightarrow$ arity = 2.
- Function expression is function symbol followed by its arguments.
  - Arguments are elements from the domain of the function.
  - Number of the arguments is equal to the arity of the function.
  - Arguments are enclosed in the parentheses and separated by commas.
- Value of the function: It is a single object in the range that the arguments are mapped to.
  Examples:
  - The value of the function "father(sam)" is george.
  - The value of the function "plus(one,two)" is three.
- Evaluation is replacing the function with its value.

DEFINITION

SYMBOLS and TERMS

Predicate calculus symbols include:

1. *Truth symbols* true and false (these are reserved symbols).
2. *Constant symbols* are symbol expressions having the first character lowercase.
3. *Variable symbols* are symbol expressions beginning with an uppercase character.
4. *Function symbols* are symbol expressions having the first character lowercase. Functions have an attached arity indicating the number of elements of the domain mapped onto each element of the range.

A *function expression* consists of a function constant of arity n, followed by n terms, $t_1, t_2, ..., t_n$, enclosed in parentheses and separated by commas.

A predicate calculus *term* is either a constant, variable, or function expression.

- Atomic sentence is a primitive unit of the predicate language.
- Atomic sentence is a predicate with arity followed by n terms enclosed between parentheses and separated by commas.
- Examples:
  - friends(george, kate)
  - friends(george, X)
  - friends(george, sara, sam)
  - friends(father(david), father(sara)) → friends(mike,john)

DEFINITION

PREDICATES and ATOMIC SENTENCES

Predicate symbols are symbols beginning with a lowercase letter.

Predicates have an associated positive integer referred to as the *arity* or "argument number" for the predicate. Predicates with the same name but different arities are considered distinct.

An atomic sentence is a predicate constant of arity n, followed by n terms, $t_1, t_2, ..., t_n$, enclosed in parentheses and separated by commas.

The truth values, **true** and **false**, are also atomic sentences.

- Atomic sentences might be combined by using logical operations such as:
  - not $\neg$
  - and $\wedge$
  - or $\vee$
  - if $\rightarrow$
  - equivalence $\equiv$
  - universal quantifier $\forall$
  - existential quantifiers $\exists$
- $\forall X likes(X, cake)$ This sentence will be true when all of the values of the variable (that belong to the disclosure domain) are satisfying this predicate calculus.
- $\exists X friends(X, sara)$ This sentence will be true when there is at least one value of the variable (that belongs to the disclosure domain) is satisfying this predicate calculus.

DEFINITION

PREDICATE CALCULUS SENTENCES

Every atomic sentence is a sentence.

1. If s is a sentence, then so is its negation, $\neg$ s.
2. If s_1 and s_2 are sentences, then so is their conjunction, $s_1 \wedge s_2$.
3. If s_1 and s_2 are sentences, then so is their disjunction, $s_1 \vee s_2$.
4. If s_1 and s_2 are sentences, then so is their implication, $s_1 \rightarrow s_2$.
5. If s_1 and s_2 are sentences, then so is their equivalence, $s_1 \equiv s_2$.
6. If X is a variable and s a sentence, then $\forall$ X s is a sentence.
7. If X is a variable and s a sentence, then $\exists$ X s is a sentence.

verify_sentence algorithm

```
function verify_sentence(expression);
begin
  case
    expression is an atomic sentence: return SUCCESS;
    expression is of the form Q X s, where Q is either ∀ or ∃, X is a variable,
        and s is an expression;
      if verify_sentence(s) returns SUCCESS
      then return SUCCESS
      else return FAIL;
    expression is of the form ¬ s:
      if verify_sentence(s) returns SUCCESS
      then return SUCCESS
      else return FAIL;
    expression is of the form s1 op s2, where op is a binary logical operator:
      if verify_sentence(s1) returns SUCCESS and
        verify_sentence(s2) returns SUCCESS
      then return SUCCESS
      else return FAIL;
    otherwise: return FAIL
  end
end.
```

Examples: Determine whether the following is a sentence or not.

- plus(two,three) No
- equal(plus(two,three),five) yes
- equal(plus(two,three),seven) yes
- $\exists X$ likes(X, ice-cream)$\wedge$ equal(plus(two, three),five) yes

Write a predicate calculus expression in order to capture the following relationships by using the provided facts.

- parent(mike,sam)
- parent(sara,sam)
- parent(mike,john)
- parent(sara,john)
- female(sara)
- male(mike)
- male(john)
- male(sam)
- ***Mother relationship:*** $\forall X \forall Y$ parent(X,Y) $\wedge$ female(X)
- ***Son relationship.*** $\forall X \forall Y$ parent(X,Y) $\wedge$ male(Y)
- ***Sibling relationship.*** $\forall X \forall Y \forall Z$ parent(X,Y) $\wedge$ parent(X,Z)
- ***Brother relationship.*** $\forall X \forall Y \forall Z$ parent(X,Y) $\wedge$ parent(X,Z) $\wedge$ male(Y)

A Semantic for Predicate Calculus

DEFINITION

INTERPRETATION

Let the domain D be a nonempty set.

An *interpretation* over D is an assignment of the entities of D to each of the constant, variable, predicate, and function symbols of a predicate calculus expression, such that:

1. Each constant is assigned an element of D.
2. Each variable is assigned to a nonempty subset of D; these are the allowable substitutions for that variable.
3. Each function f of arity m is defined on m arguments of D and defines a mapping from D^m into D.
4. Each predicate p of arity n is defined on n arguments from D and defines a mapping from D^n into {T, F}.

DEFINITION

TRUTH VALUE OF PREDICATE CALCULUS EXPRESSIONS

Assume an expression E and an interpretation I for E over a nonempty domain D. The truth value for E is determined by:

1. The value of a constant is the element of D it is assigned to by I.
2. The value of a variable is the set of elements of D it is assigned to by I.
3. The value of a function expression is that element of D obtained by evaluating the function for the parameter values assigned by the interpretation.
4. The value of truth symbol "true" is T and "false" is F.
5. The value of an atomic sentence is either T or F, as determined by the
6. The value of the negation of a sentence is T if the value of the sentence is F and is F if the value of the sentence is T.
7. The value of the conjunction of two sentences is T if the value of both sentences is T and is F otherwise.

8.–10. The truth value of expressions using $\vee$, $\rightarrow$, and $\equiv$ is determined from the value of their operands as defined in Section 2.1.2.

Finally, for a variable X and a sentence S containing X:

11. The value of $\forall$ X S is T if S is T for all assignments to X under I, and it is F otherwise.
12. The value of $\exists$ X S is T if there is an assignment to X in the interpretation under which S is T; otherwise it is F.

Definitions

- friends(sara,X) $\quad X \varepsilon D = \{sam, george, lura, mike, louis, nel\}$
- X can be replaced by any other dummy variable such as Y.
- Any variable has to be quantified in their **universal** or **existential** quantifier.
- $\forall X \exists Y (parent(X, Y) \wedge femal(Y)) \wedge like(X, Z)$
- X variable in parent(X,Y) is bounded by the universal quantifier.
- Y variable in parent(X,Y) and female (Y) is bounded by the existential quantifier.
- X and Z variables in like(X,Z) are free variables.
- **Closed expression:** An expression is closed if all of its variables are quantified. *i.e.,* $\forall X \exists Y parent(X, Y)$.
- **Ground expression:** An expression is ground if it does not contain any variable. *i.e.,* $P \wedge (Q \vee (R \rightarrow S))$.
- Parentheses are used to indicate the scope of quantification.
- $\forall X \exists Y (parent(X, Y)$
  - X and Y belong to certain domain
  - If the domain of an interpretation is infinite, exhaustive testing of all substitutions to a universally and existential quantified variables are computationally impossible : the algorithm may never halt.

1. $\neg \exists X p(X) = \forall \neg p(X)$
2. $\neg \forall X p(X) = \exists \neg p(X)$
3. $\exists X p(X) = \exists Y P(Y)$
4. $\forall X p(X) = \forall Y P(Y)$
5. $\forall X (p(X) \wedge q(X)) = \forall X p(X) \wedge \forall Y q(Y)$

DEFINITION

FIRST-ORDER PREDICATE CALCULUS

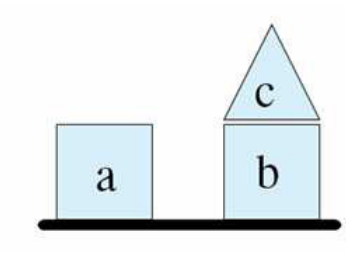
First-order predicate calculus allows quantified variables to refer to objects in the domain of discourse and not to predicates or functions.

6. $\exists X(p(X) \vee q(X)) = \exists X p(X) \vee \exists Y q(Y)$

Translate the following English sentence to predicate calculus:

- If it does not rain on Thursday, Sara will go to the beach.
  $\neg weather(thursday, rain) \rightarrow goes(sara, beach)$.
- All basketball players are tall.
  $\forall X basketball\_player(X) \rightarrow tall(X)$.
- Nobody likes taxes.
  $\neg \exists X likes(X, taxes)$

Example: Block World Relationship



clear(c)
clear(a)
ontable(a)
ontable(b)
on(c, b)
cube(b)
cube(a)
pyramid(c)

General rule

$\forall X \neg \exists Y$ on(Y,X) $\Rightarrow$ clear(X)

3 Using Inference Rule to Produce Predicate Calculus Expressions

- An expression **X** *logically follows* from a set of expressions **S** if every interpretation that **satisfies S** also satisfies **X**.
- An interpretation that makes the sentence true is said to satisfy that sentence.
- **Question: Why do we need Inference Rule and what is the meaning of Inference Rule**
- Predicate system can have an infinite number of possible interpretations, which is hard to be traced or tested.
- Therefore, Inference Rules are used to determine when an expression is logically followed.
- Inference Rule is a mechanical means of producing new predicate sentence from other sentences.
- Inference Rule is divided into ***Sound*** and ***Complete***.
- Sound Inference Rule: If the expression produced by the inference rule logically follows from **S**.
- Complete Inference Rule: It has the ability to produce every expression that logically follows from **S**.

DEFINITION

SATISFY, MODEL, VALID, INCONSISTENT

For a predicate calculus expression X and an interpretation I:

If X has a value of T under I and a particular variable assignment, then I is said to *satisfy* X.

If I satisfies X for all variable assignments, then I is a *model* of X.

X is *satisfiable* if and only if there exist an interpretation and variable assignment that satisfy it; otherwise, it is *unsatisfiable*.

A set of expressions is *satisfiable* if and only if there exist an interpretation and variable assignment that satisfy every element.

If a set of expressions is not satisfiable, it is said to be *inconsistent*.

If X has a value T for all possible interpretations, X is said to be *valid*.

- $\exists X p(X) \wedge \neg p(X) \Rightarrow$ "Inconsistence".
- $\forall X p(X) \vee \neg p(X) \Rightarrow$ "Valid".

DEFINITION

LOGICALLY FOLLOWS, SOUND, and COMPLETE

A predicate calculus expression X *logically follows* from a set S of predicate calculus expressions if every interpretation and variable assignment that satisfies S also satisfies X.

An inference rule is *sound* if every predicate calculus expression produced by the rule from a set S of predicate calculus expressions also logically follows from S.

An inference rule is *complete* if, given a set S of predicate calculus expressions, the rule can infer every expression that logically follows from S.

Modus Ponens

$P \rightarrow Q$

P

$- - - - -$

Q

Modus Tollens

$P \rightarrow Q$

$\neg Q$

$- - - - -$

$\neg P$

DEFINITION

MODUS PONENS, MODUS TOLLENS, AND ELIMINATION, AND INTRODUCTION, and UNIVERSAL INSTANTIATION

If the sentences P and P → Q are known to be true, then *modus ponens* lets us infer Q.

Under the inference rule *modus tollens*, if P → Q is known to be true and Q is known to be false, we can infer ¬ P.

And elimination allows us to infer the truth of either of the conjuncts from the truth of a conjunctive sentence. For instance, P ∧ Q lets us conclude P and Q are true.

And introduction lets us infer the truth of a conjunction from the truth of its conjuncts. For instance, if P and Q are true, then P ∧ Q is true.

Universal instantiation states that if any universally quantified variable in a true sentence is replaced by any appropriate term from the domain, the result is a true sentence. Thus, if a is from the domain of X, ∀ X p(X) lets us infer p(a).

Elimination

$P \wedge Q$

$-----$

P

Q

Introduction

P

Q

$-----$

$P \wedge Q$

Universal Instantiation

$\forall X\ tall(X)$

$-----$

$tall(sara)$

Determine whether the following arguments are valid or invalid, Please justify your answer.

1. **All men are mortal, Socrates is man. Therefore, Socrates is mortal**

   Solution

   It is valid using Universal Instantiation and Modes ponens inference rule.

$\forall X\ man(X) \rightarrow mortal(X)$.
$man(socrates)$.

$man(socrates) \rightarrow mortal(socrates)$
$man(socrates)$

$\therefore mortal(socrates)$

2. **Sara is student in class COMP1805. Sara does not read the book. Everyone in class COMP1805 passed the midterm exam. Therefore, Sara passed the midterm and she did not read the book.**

   Solution

   It is valid.

   $S(x)$: x is a student in class COMP1805.

   $R(x)$: x is a student who does not read the book.

   $P(x)$: x is a student who pass the midterm exam.

   $S(Sara)$
   $\neg R(Sara)$
   $\forall x S(x) \rightarrow P(x)$

   $S(Sara) \rightarrow P(Sara)$ UI
   $S(Sara)$

   $P(Sara)$ Modus ponens
   $\neg R(Sara)$

   $\therefore P(Sara) \wedge \neg R(Sara)$ Conjunction

4 Unification

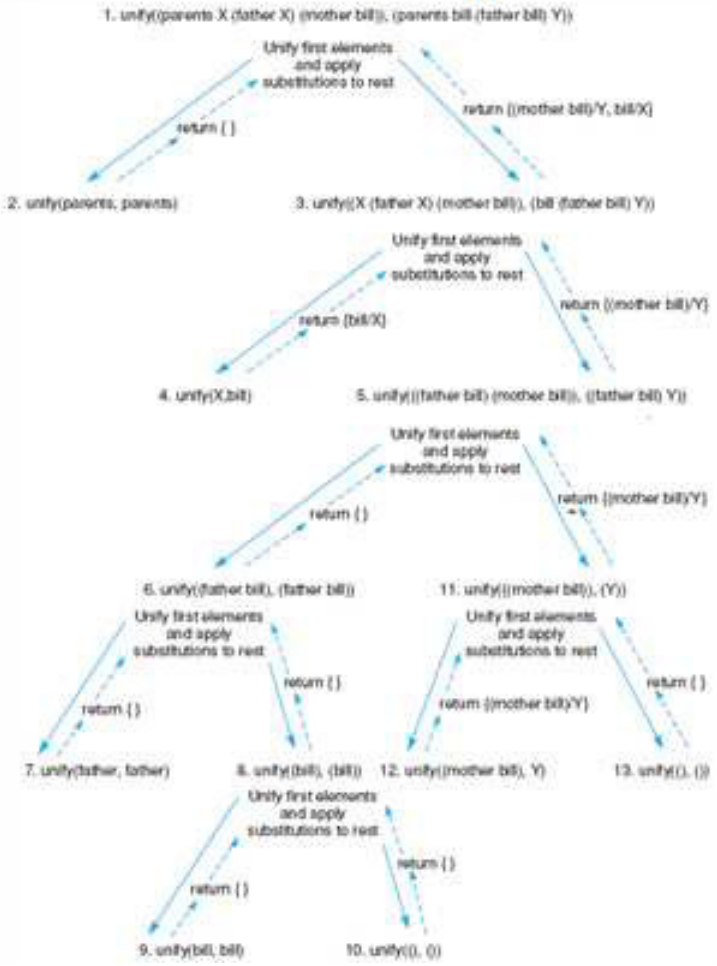
- Unification: Determine whether the following two sentences are similar or not.
- Propositional calculus → very trivial.
- Predicate calculus → very difficult (variables).
- X/Y means X is substituted for the variable Y.
- Substitution → Binding.
- **Examples:** What is the final value of Y and Z?
  $\{X/Y, W/Z\},\ \{V/X\},\ \{a/V, f(b)/W\}$
- Predicate calculus expression should be transformed to list syntax.
- p(a,b) → (p a b).
- p(f(a),g(X,Y)) → (p (f(a)) (g(X,Y))).
- equal(eva, mother(sara)) → (equal eva (mother(sara))).

Skolemization

- Skolemization is a process of replacing each existentially quantified variable with a function that returns the appropriate constant.
- Eliminating existentially quantified variable is complicated due to the dependency of this variable on another variables.
- $\forall X \exists Y mother(X, Y)$ It is equivalent to $\forall X mother(X, f(X))$
- $\forall X\ \forall Y\ \exists Z\ m(X, Y, Z) \Rightarrow \forall X\ \forall Y\ m(X, Y, f(X, Y))$

```
function unify(E1, E2);
  begin
    case
      both E1 and E2 are constants or the empty list:          %recursion stops
        if E1 = E2 then return {}
          else return FAIL;
      E1 is a variable:
        if E1 occurs in E2 then return FAIL
          else return {E2/E1};
      E2 is a variable:
        if E2 occurs in E1 then return FAIL
          else return {E1/E2}
      either E1 or E2 are empty then return FAIL     %the lists are of different sizes
      otherwise:                                          %both E1 and E2 are lists
        begin
            HE1 := first element of E1;
            HE2 := first element of E2;
            SUBS1 := unify(HE1,HE2);
            if SUBS1 : = FAIL then return FAIL;
            TE1 := apply(SUBS1, rest of E1);
            TE2 : = apply (SUBS1, rest of E2);
            SUBS2 : = unify(TE1, TE2);
            if SUBS2 = FAIL then return FAIL;
                else return composition(SUBS1,SUBS2)
        end
  end                                                                    %end case
end
```

Figure 2.6: Final trace of the unification of (parents X (father X) (mother bill)) and (parents bill (father bill) Y).



Question:- A logical_Based Financial Advisor?

1. savings_account(inadequate) → investment(savings).
2. savings_account(adequate) ∧ income(adequate) → investment(stocks).
3. savings_account(adequate) ∧ income(inadequate)
   → investment(combination).
4. ∀ amount_saved(X) ∧ ∃ Y (dependents(Y) ∧
   greater(X, minsavings(Y))) → savings_account(adequate).
5. ∀ X amount_saved(X) ∧ ∃ Y (dependents(Y) ∧
   ¬ greater(X, minsavings(Y))) → savings_account(inadequate).
6. ∀ X earnings(X, steady) ∧ ∃ Y (dependents (Y) ∧
   greater(X, minincome(Y))) → income(adequate).
7. ∀ X earnings(X, steady) ∧ ∃ Y (dependents(Y) ∧
   ¬ greater(X, minincome(Y))) → income(inadequate).
8. ∀ X earnings(X, unsteady) → income(inadequate).
9. amount_saved(22000).
10. earnings(25000, steady).
11. dependents(3).

where $minincome(X) = 15000 + (4000 * X)$
and $minisaving(X) = 5000 * X$

5 Clause From

Clause from represents the logical databases as a set of disjunctions of literals.
$\forall X([a(X) \wedge b(X)] \rightarrow [c(X, I) \wedge \exists Y(\exists Z c(Y, Z) \rightarrow d(X, Y))]) \vee \forall X e(X)$

1. First we eliminate the $\rightarrow$ by using $a \rightarrow b \equiv \neg a \vee b$.
   $\forall X(\neg[a(X) \wedge b(X)] \vee [c(X, I) \wedge \exists Y(\exists Z \neg c(Y, Z) \vee d(X, Y))]) \vee \forall X e(X)$
2. Reduce the scop of the negation by using
   (a) $\neg(\neg a) \equiv a$
   (b) $\neg\exists X\ a(X) \equiv \forall X\ \neg a(X)$
   (c) $\neg\forall X\ a(X) \equiv \exists X\ \neg a(X)$
   (d) $\neg(a \wedge b) \equiv \neg a\ \vee\ \neg b$
   (e) $\neg(a \vee b) \equiv \neg a\ \wedge\ \neg b$

   $\forall X([\neg a(X) \vee \neg b(X)] \vee [c(X, I) \wedge \exists Y(\exists Z \neg c(Y, Z) \vee d(X, Y))]) \vee \forall X e(X)$
3. Standardize all variables by renaming them so that variables bound by different quantifiers have unique names.
   $\forall X([\neg a(X) \vee \neg b(X)] \vee [c(X, I) \wedge \exists Y(\exists Z \neg c(Y, Z) \vee d(X, Y))]) \vee \forall W e(W)$
4. Move all quantifiers to the left without changing their order.
   $\forall X \exists Y \exists Z \forall W([\neg a(X) \vee \neg b(X)] \vee [c(X, I) \wedge (\neg c(Y, Z) \vee d(X, Y))] \vee e(W))$
5. All existential quantifiers are eliminated by a process called skolemization.
   $\forall X \forall W([\neg a(X) \vee \neg b(X)] \vee [c(X, I) \wedge (\neg c(f(X), g(X)) \vee d(X, f(X)))] \vee e(W))$
6. Drop all universal quantifiers.
   $[\neg a(X) \vee \neg b(X)] \vee [c(X, I) \wedge (\neg c(f(X), g(X)) \vee d(X, f(X)))] \vee e(W)$
7. Convert the expression to the conjunct of disjuncts form.
   $\neg a(X) \vee \neg b(X) \vee c(X, I) \vee e(W) \wedge$
   $\neg a(X) \vee \neg b(X) \vee \neg c(f(X), g(X)) \vee d(X, f(X)) \vee e(W)$
8. Call each conjunct a separate clause.
   $\neg a(X) \vee \neg b(X) \vee c(X, I) \vee e(W)$
   $\neg a(X) \vee \neg b(X) \vee \neg c(f(X), g(X)) \vee d(X, f(X)) \vee e(W)$

9. Standardize each variable again.
   $\neg a(X) \vee \neg b(X) \vee c(X, I) \vee e(W)$
   $\neg a(U) \vee \neg b(U) \vee \neg c(f(U), g(U)) \vee d(U, f(U)) \vee e(V)$

6 Resolution Theorem Proving

Resolution

- It is a technique for proving theorems in propositional and predicate calculus.
- It describes a way of finding contradictions in the databases with a minimum use of substitutions.
- Method:
  1. It proves a theorem by negating the statement to be proved and adding this negated goal to the set of axioms that are known to be true.
  2. It uses the inference rules to show that this leads to a contradiction.
  3. If the negated goal is inconsistent with the given set of axioms, it follows that the original goal is consistent.

Resolution refutation proofs involve the following steps:

1. Put the premises or axioms into *clause form* (13.2.2).
2. Add the negation of what is to be proved, in clause form, to the set of axioms.
3. *Resolve* these clauses together, producing new clauses that logically follow from them (13.2.3).
4. Produce a contradiction by generating the empty clause.
5. The substitutions used to produce the empty clause are those under which the opposite of the negated goal is true (13.2.4).

- Resolution requires the axioms and the negation of the goal to be in a normal form called ***clause from***.
- ***Clause From*** represents the logical databases as a set of disjunctions of ***literals***.
- ***Literal*** is an atomic expression or the negation of an atomic expression.

- In general, binary resolution is the common form of resolution.
  - It is applied to two clauses: the first one contains the literal and the second one contains its negation.
  - The literals must be unified to make them equivalent.
  - The new clause that is produced consisting of the disjuncts of all the predicates in the two clauses minus the literal and its negative instance.

Example 1

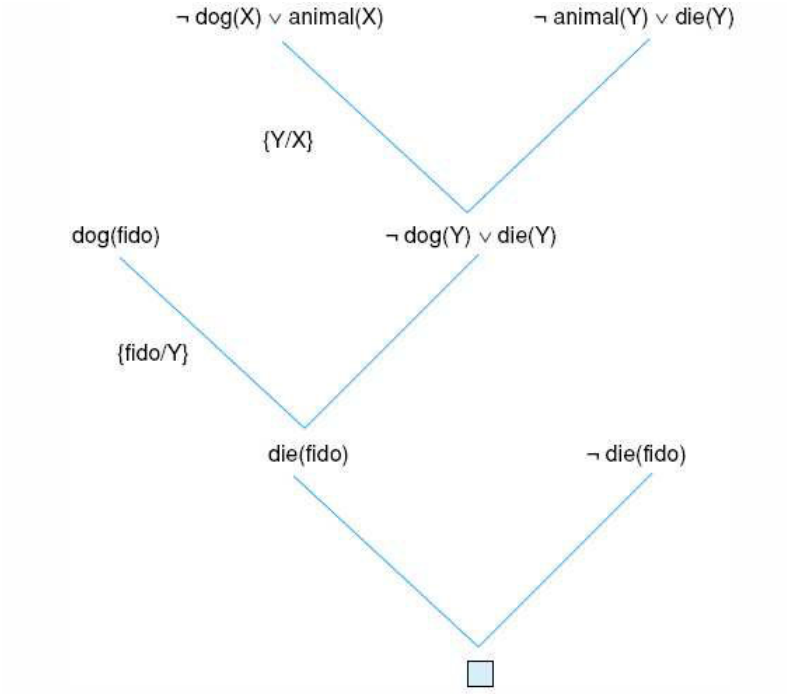
1. All dogs are animals
2. Fido is a dog
3. All animals will die
4. We want to prove that "Fido will die"

Predicate Calculus

- $\forall X\ dog(X) \rightarrow animal(X)$.
- $dog(fido)$.
- $\forall Y\ animal(Y) \rightarrow die(Y)$.
- Negate the conclusion $\neg die(fido)$.

Clause Form

- $\neg dog(X) \vee animal(X)$.
- $dog(fido)$.
- $\neg animal(Y) \vee die(Y)$.
- Negate the conclusion $\neg die(fido)$.



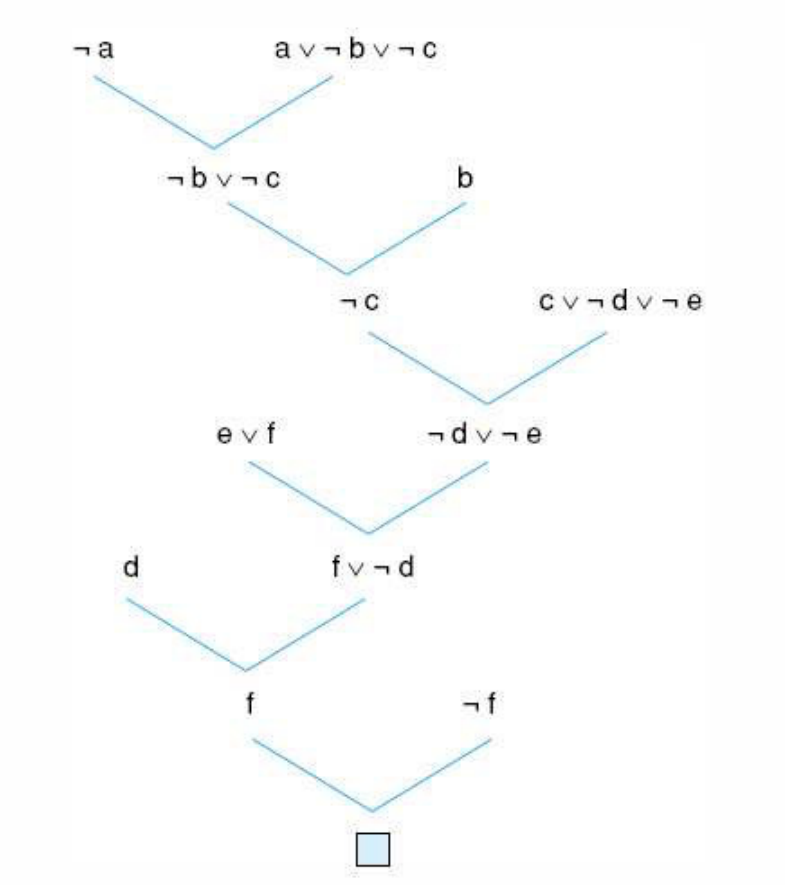
Example 2

1. $a \leftarrow b \wedge c$
2. b
3. $c \leftarrow d \wedge e$
4. $e \vee f$
5. $d \wedge \neg f$
6. We want to prove that "a"

Clause Form

- $a \vee \neg b \vee \neg c$
- b
- $c \vee \neg d \vee \neg e$
- $e \vee f$
- d

- $\neg f$
- Negate the conclusion $\neg a$.



Example 3

Anyone passing his history exams and winning the lottery is happy.

$\forall$ **X (pass (X,history)** $\wedge$ **win (X,lottery)** $\rightarrow$ **happy (X))**

Anyone who studies or is lucky can pass all his exams.

$\forall$ **X** $\forall$ **Y (study (X)** $\vee$ **lucky (X)** $\rightarrow$ **pass (X,Y))**

John did not study but he is lucky.

$\neg$ **study (john)** $\wedge$ **lucky (john)**

Anyone who is lucky wins the lottery.

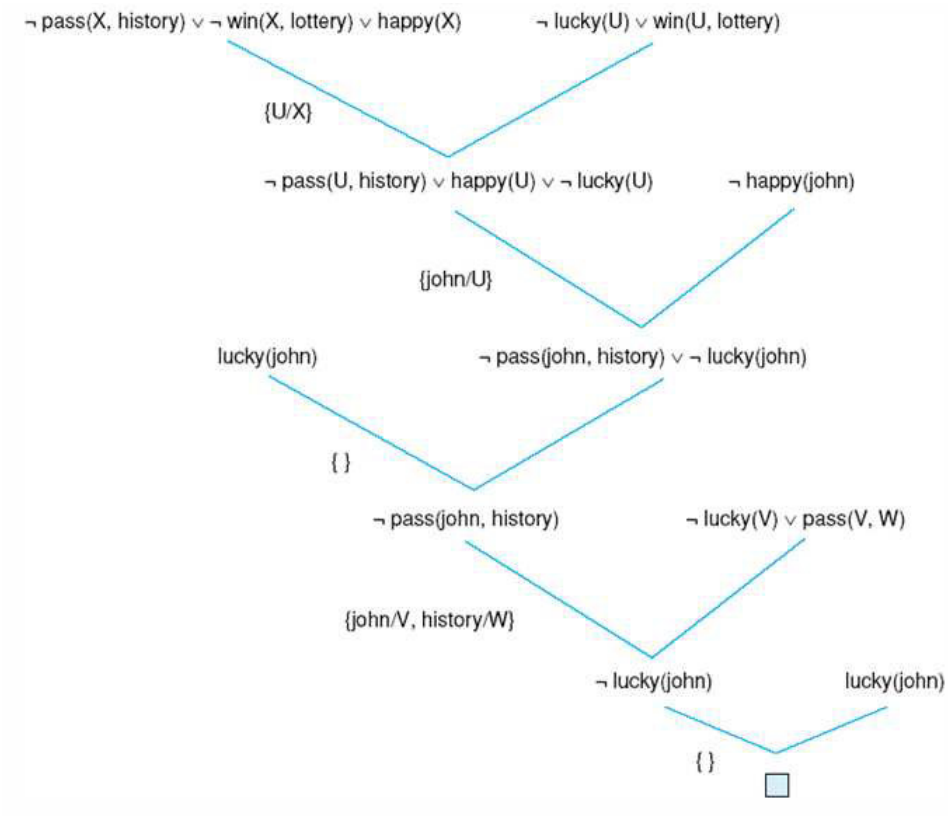
$\forall$ **X (lucky (X)** $\rightarrow$ **win (X,lottery))**

These four predicate statements are now changed to clause form (Section 12.2.2):

1. $\neg$ **pass (X, history)** $\vee$ $\neg$ **win (X, lottery)** $\vee$ **happy (X)**
2. $\neg$ **study (Y)** $\vee$ **pass (Y, Z)**
3. $\neg$ **lucky (W)** $\vee$ **pass (W, V)**
4. $\neg$ **study (john)**
5. **lucky (john)**
6. $\neg$ **lucky (U)** $\vee$ **win (U, lottery)**

Into these clauses is entered, in clause form, the negation of the conclusion:

7. $\neg$ **happy (john)**


¬ pass(X, history) ∨ ¬ win(X, lottery) ∨ happy(X)
¬ lucky(U) ∨ win(U, lottery)
{U/X}
¬ pass(U, history) ∨ happy(U) ∨ ¬ lucky(U)
¬ happy(john)
{john/U}
lucky(john)
¬ pass(john, history) ∨ ¬ lucky(john)
{ }
¬ pass(john, history)
¬ lucky(V) ∨ pass(V, W)
{john/V, history/W}
¬ lucky(john)
lucky(john)
{ }

Example 4

All people who are not poor and are smart are happy. Those people who read are not stupid. John can read and is wealthy. Happy people have exciting lives. Can anyone be found with an exciting life?

We assume $\forall$**X (smart (X)** $\equiv \neg$ **stupid (X)) and** $\forall$**Y (wealthy (Y)** $\equiv \neg$ **poor (Y)), and get**:

$\forall$**X (**$\neg$ **poor (X)** $\wedge$ **smart (X)** $\rightarrow$ **happy (X))**

$\forall$**Y (read (Y)** $\rightarrow$ **smart (Y))**

read (john) $\wedge \neg$ **poor (john)**

$\forall$**Z (happy (Z)** $\rightarrow$ **exciting (Z))**

The negation of the conclusion is:

$\neg \exists$ **W (exciting (W))**

These predicate calculus expressions for the "exciting life" problem are transformed into the following clauses:

poor (X) $\vee \neg$ **smart (X)** $\vee$ **happy (X)**

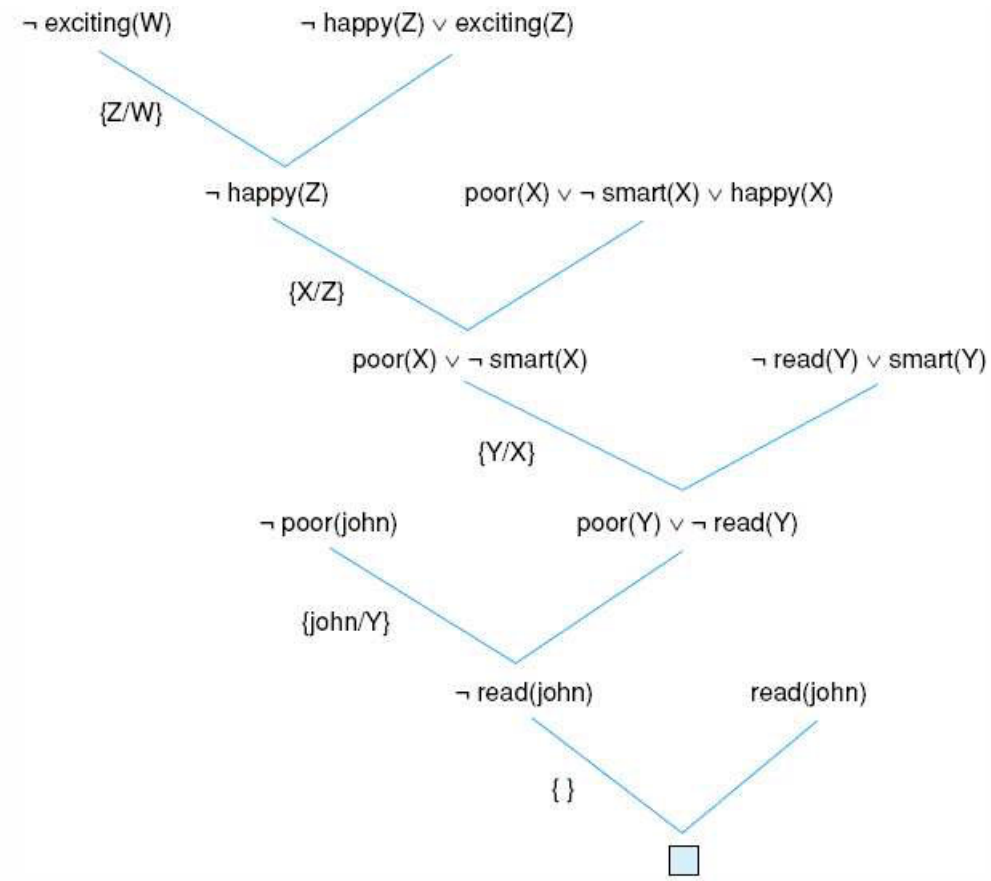
$\neg$ **read (Y)** $\vee$ **smart (Y)**

read (john)

$\neg$ **poor (john)**

$\neg$ **happy (Z)** $\vee$ **exciting (Z)**

$\neg$ **exciting (W)**


¬ exciting(W)
¬ happy(Z) ∨ exciting(Z)
{Z/W}
¬ happy(Z)
poor(X) ∨ ¬ smart(X) ∨ happy(X)
{X/Z}
poor(X) ∨ ¬ smart(X)
¬ read(Y) ∨ smart(Y)
{Y/X}
¬ poor(john)
poor(Y) ∨ ¬ read(Y)
{john/Y}
¬ read(john)
read(john)
{ }

Knowledge Representation

Lecture 5:Ontology

Instructor: Hazim N. Abed

Ontologies

- The history of artificial intelligence shows that knowledge is critical for intelligent systems. In many cases, better knowledge can be more important for solving a task than better algorithms.

- To have truly intelligent systems, knowledge needs to be captured, processed, reused, and communicated. Ontologies support all these tasks.

Ontologies

- The term "ontology" can be defined as an *explicit specification of conceptualization*. Ontologies capture the structure of the domain, i.e. conceptualization.

- This includes the model of the domain with possible restrictions. The conceptualization describes knowledge about the domain, not about the particular state of affairs in the domain.

Ontologies

- Ontology is a formal representation of a set of concepts within a domain and relationships between those concepts .

- Gruber (1993) gave a different definition of ontology. According to Gruber (1993) ontology is a specification of a conceptualization. Conceptualization means structured interpretation of a part of the world in which people used to think and communicate in the world.

CONCEPTS OF KNOWLEDGE ENGINEERING

- The knowledge-engineering process includes five major activities:

1. **Knowledge acquisition**

   - Knowledge acquisition involves the acquisition of knowledge from human experts, books, documents, sensors, or computer files.

   - The knowledge may be specific to the problem domain or to the problem-solving procedures, it may be general knowledge (e.g., knowledge about business), or it may be metaknowledge (knowledge about knowledge).

Ontologies

Why Do we Develop the Ontology?

- To share common understanding of the structure of information among people or software agents

- To enable of the reusing of domain knowledge.

  - to avoid “re-inventing the wheel”

Ontologies

- To make domain assumptions explicit. Making explicit domain assumptions underlying an implementation makes it possible to change these assumptions easily if our knowledge about the domain changes .

- To separate domain knowledge from the operational knowledge. We can describe a task of configuring a product from its components according to a required specification and implement a program that does this configuration independent of the products and components themselves.

Ontologies

Ontology Representation

- Ontology is comprised of four main components:

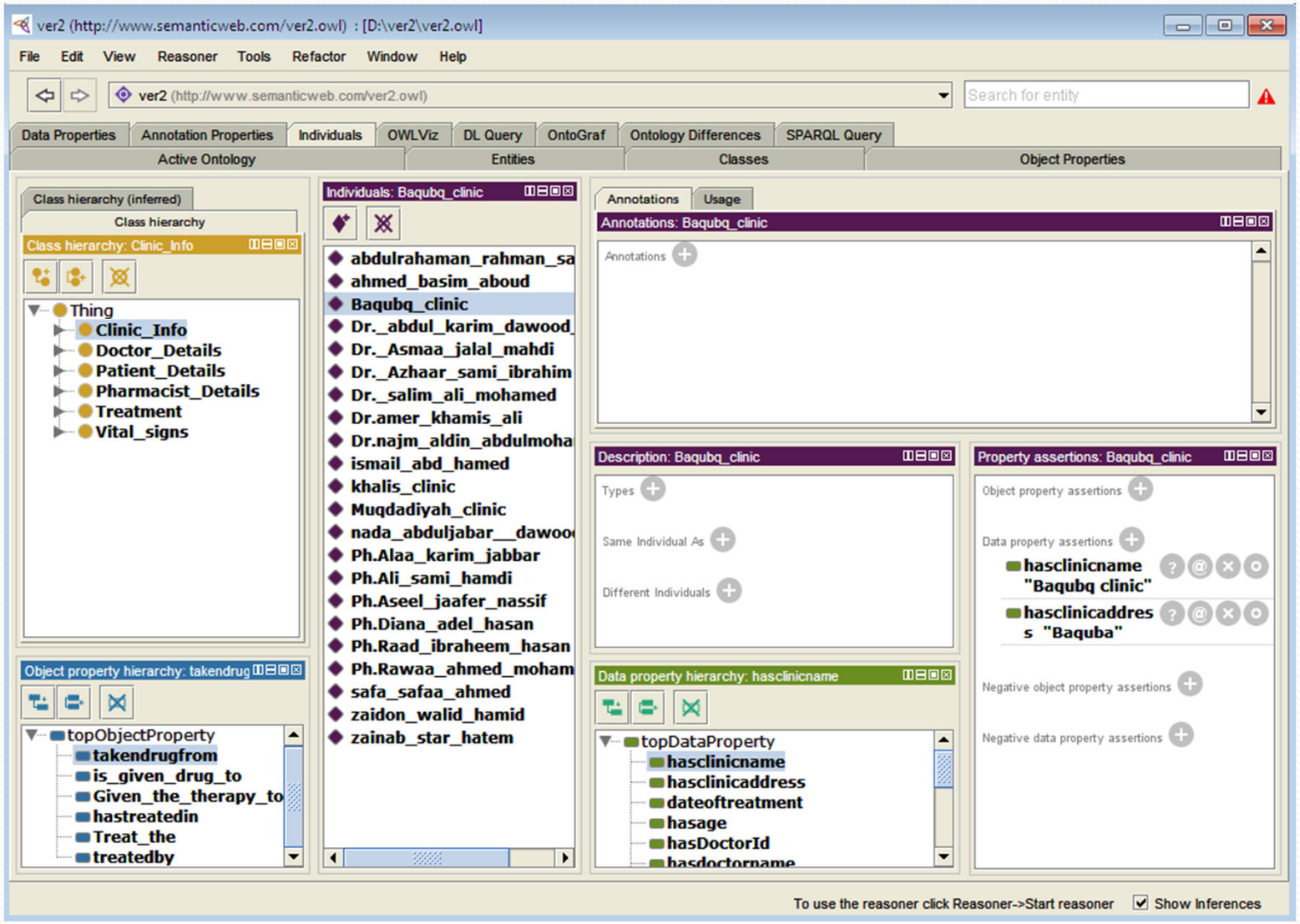
1. **A Concept**: (also known as a class or a term) is an abstract group, set or collection of objects. It is the fundamental element of the domain and usually represents a group or class whose members share common properties.
2. **An Instance**: (also known as an individual) is the "ground-level" component of an ontology which represents a specific object or element of a concept or class. For example, "Jordan" could be an instance of the class "Arab countries" or simply "countries".

Ontologies

3. **A Relation:** (also known as a slot) is used to express relationships between two concepts in a given domain.

4. **An Axiom:** is used to impose constraints on the values of classes or instances, so axioms are generally expressed using logic-based languages such as first-order logic; they are used to verify the consistency of the ontology.

Ontologies


ver2 (http://www.semanticweb.com/ver2.owl) : [D:\ver2\ver2.owl]
File Edit View Reasoner Tools Refactor Window Help
ver2 (http://www.semanticweb.com/ver2.owl)
Search for entity
Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query
Active Ontology Entities Classes Object Properties
Class hierarchy (inferred)
Class hierarchy
Class hierarchy: Clinic_Info
Thing
Clinic_Info
Doctor_Details
Patient_Details
Pharmacist_Details
Treatment
Vital_signs
Individuals: Baqubq_clinic
abdulrahaman_rahman_sa
ahmed_basim_aboud
Baqubq_clinic
Dr._abdul_karim_dawood
Dr._Asmaa_jalal_mahdi
Dr._Azhaar_sami_ibrahim
Dr._salim_ali_mohamed
Dr.amer_khamis_ali
Dr.najm_aldin_abdulmoha
ismail_abd_hamed
khalis_clinic
Muqdadiyah_clinic
nada_abduljabar__dawoo
Ph.Alaa_karim_jabbar
Ph.Ali_sami_hamdi
Ph.Aseel_jaafer_nassif
Ph.Diana_adel_hasan
Ph.Raad_ibraheem_hasan
Ph.Rawaa_ahmed_moham
safa_safaa_ahmed
zaidon_walid_hamid
zainab_star_hatem
Annotations Usage
Annotations: Baqubq_clinic
Annotations
Description: Baqubq_clinic
Types
Same Individual As
Different Individuals
Property assertions: Baqubq_clinic
Object property assertions
Data property assertions
hasclinicname "Baqubq clinic"
hasclinicaddress "Baquba"
Negative object property assertions
Negative data property assertions
Object property hierarchy: takendrug
topObjectProperty
takendrugfrom
is_given_drug_to
Given_the_therapy_to
hastreatedin
Treat_the
treatedby
Data property hierarchy: hasclinicname
topDataProperty
hasclinicname
hasclinicaddress
dateoftreatment
hasage
hasDoctorId
hasdoctorname
To use the reasoner click Reasoner->Start reasoner
Show Inferences

Ontologies

- **Types of Ontologies**: There are two types of ontologies

1. **General-purpose Ontologies**: this type aims to provide conceptualizations of general notions.

2. **Domain-specific or material Ontologies**: this type forms most of the Ontologies. They are intended for sharing concepts and relations in any given interest.

Ontologies

- **Features of ontologies:** There are two features of ontology:

1. **Vocabulary** : Ontology contains a vocabulary for presenting the terms of a particular domain in the way these terms are understood and applied by a community.
2. **Taxonomy**: Taxonomy is a hierarchical categorization or classification of entities within a domain.

- The ontology's vocabulary and taxonomy with each other provide a conceptual framework for discussion, analysis, and information retrieval in a particular domain

Ontologies


owl:Thing
DomainEntity
SelfStandingEntity
PizzaBase
Pizza
PizzaTopping
ValuePartition
FatContent
Spiciness
ThinBase
HeavyBase
VegetarianPizza
SpicyBeefPizza
ProteinLoversPizza
CheesePizza
MargheritaPizza
JamAndCheesePizza
Uno
Dos
LowFat
HighFat
Bland
Medium
Hot
FishTopping
MeatTopping
CheeseTopping
VegetableTopping
FruitTopping
Tuna
Anchovy
Jam
Pepperoni
SpicyBeef
Parmesan
Mozzarella
Onion
HotPeper
Mushroom
Tomato
Pineapple
is-a

Ontologies

- Ontology building method: many methods are conducted for building ontology. One of the useful method to build ontology from scratch is Uschold and king methodology:

- **This methodology provides guidelines for developing ontologies, which are:**

1. **Identify purpose.** It is important to be clear why the ontology is being built and what its intended uses are.

Ontologies

2. **Building the ontology,** which is broken down into three steps:

   A. **Ontology capture,** which means:

- Identification of the key concepts and relationships in the domain of interest, that is, scoping.

- Production of precise unambiguous text definitions for such concepts and relationships.

- Identification of terms to refer to such concepts and relationships.

Ontologies

B. **Coding.** Involves explicitly representing the knowledge acquired in previse step in a formal language.

C. **Integrating existing ontologies.** During either or both of the capture and coding processes, there is the question of how and whether to use ontologies that already exist

Ontologies

3. **Evaluation**, evaluation defined is “to make a technical judgment of the ontologies, their associated software environment, and documentation with respect to a frame of reference ... The frame of reference may be requirements specifications, competency questions, and/or the real world”.

4. **Documentation,** recommends that guidelines be established for documenting ontologies, possibly differing according to the type and purpose of the ontology.

Ontologies

Ontology Applications

- Knowledge Management
- Enterprise Application Integration
- e-Commerce