ADVANCED KNOWLEDGE ENGINEERING

Instructor:
KHUAT Thanh Tung
University of Technology Sydney

Subject's Outline

- **Topic 1:** An Overview of Knowledge Engineering
- **Topic 2:** An Overview of Knowledge-based Systems
- **Topic 3:** Knowledge Acquisition
- **Topic 4:** Knowledge Representation and Reasoning

Mid-term assessment

- **Topic 5:** Ontology
- **Topic 6:** Knowledge Graphs
- **Topic 7:** Expert Systems
- **Topic 8:** Uncertain Reasoning
- **Topic 9:** Hybrid Knowledge-based Systems
- **Topic 10:** Automated AI Planning

Group projects for the advanced topics

KNOWLEDGE REPRESENTATION AND REASONING

TOPIC 4

Objectives of this topic

By the end of this topic, you will be able to:

- ✓ explain how knowledge can be represented in declarative programs
- ✓ describe and analyze the inference process
- ✓ explain the principles of backward and forward chaining
- ✓ analyze the type of chaining used by a specific expert system
- ✓ explain how semantic networks represent data
- ✓ identify the advantages and disadvantages of semantic networks
- ✓ explain how frames can be used to represent knowledge

Agenda

- Introduction to Knowledge Representation and Reasoning
- Procedural vs. Declarative Programming
- Knowledge Representation Methods
- First-Order Logic
- Reasoning
- Developing Rule-based Systems
- Semantic Networks
- Frames

Introduction to Knowledge Representation and Reasoning

- Knowledge representation and reasoning is the field of study concerned with how to use a symbol system to represent a domain of knowledge with functions that allow inference (formalized reasoning) about the objects within the domain.
- We defined before knowledge as a justified belief that increases an entity's capacity for effective action.
  - Propositions
  - Formal symbols
  - Reasoning

Procedural vs. Declarative Programming

- Procedural programming
  - A program written in procedural language (e.g., C++ or Java) consists of a set of procedures that must be performed in a strict sequence to accomplish a purpose
    - Implies automatic response to stimuli – little or no thinking about the response involved
- Declarative programming
  - A program consists of a set of rules and facts that can be used by an inference engine to reach other true conclusions

Procedural vs. Declarative Programming

Procedural programming

```
If (x.equals("snow"))
    system.out.print("It is white.");
Else if (x.equals("grass"))
    system.out.print("It is green.");
Else if (x.equals("sky"))
    system.out.print("It is yellow.");
Else
    system.out.print("Beats me.");
```

Declarative programming

```
printColor(X) :- color(X,Y), !, write("It
    is "), write(Y), write(".").
printColor(X) :- write("Beats me.").
color(snow, white).
color(sky, yellow).
color(X,Y) :- madeof(X,Z), color(Z,Y).
madeof(grass, vegetation).
color(vegetation, green).
```

Why Knowledge Representation and Reasoning

- Why knowledge representation?
  - We can add new tasks and easily make them depend on previous knowledge
  - We can extend the existing behavior by adding new beliefs.
  - We can debug faulty behavior by locating the erroneous beliefs of the system
  - We can concisely explain and justify the behavior of the system.
- Why reasoning?
  - We would like action to depend on what the system believes about the world, as opposed to just the system has explicitly represented.

Requirements for Knowledge Representation Facility

- It should be able to represent the given knowledge to a sufficient depth.
- It should preserve the fundamental characteristics of knowledge, such as completeness, accessibility, transparency, naturalness, and so on.
- It should be able to infer new knowledge.
- It should be able to provide reasoning and explanation.
- It should be adaptive enough to store updates and support incremental development.

Common Knowledge Representation Methods

- Logic
  - First-order rules
- Rules
  - Production rules
- Frames
- Semantic networks

Factual Knowledge

- Constants
- Variables
- Functions
- Predicates (Mối quan hệ, vị từ)
  - Special functions that return only Boolean values (true or false)
- (Well Formed) Formulas (biểu thức)
  - String of symbols that is generated by a formal language

Introduction to First-Order Logic

- A formal logic generated by combining predicate logic and propositional logic.
  - Propositional logic is used to assert propositions, which are statements that are either true or false. It deals only with the truth value of complete statements and does not consider relationships or dependencies between objects.
  - Predicate logic is an extension and generalization of propositional logic. Its formulas contain variables which can be quantified. Two common quantifiers are the existential $\exists$ and universal $\forall$ quantifiers. The variables could be elements in the universe, or perhaps relations or functions over the universe.

First-Order Logic Syntax

- **Symbols**
  - Variable symbols: x, y, z, A, B, C…
  - Function symbols: f(), g(), h(), bestFriend(),…
  - Predicate symbols: P(), Q(), R(), OlderThan(),…
  - Logic symbols: "$\neg$", "$\wedge$", "$\vee$", "$\exists$", "$\forall$", "$=$", "$\rightarrow$", "$\leftrightarrow$"
  - Punctuation symbols: "(", ")", and "."

First-Order Logic Syntax

- **Terms** (hạng từ)
  - A term is used to refer to something in the world
  - Variables are terms and f(T) is a term, where f is a function and T is a sequence of n terms.
- **Formulas** (biểu thức)
  - A formula is used to express a proposition (mệnh đề)
  - Atomic formula - P(T) is an atomic formula, where P is a predicate and T is a sequence of n terms.
  - Literals (*thể hiện*) - atomic formulas (e.g., P(x)) and negated atomic formulas (e.g., ¬P(x))
  - Well-formed formulas (wffs, *biểu thức có cấu trúc chuẩn*) – literals are wffs and wffs connected or quantified are also wffs.
- **Sentence** (câu mệnh đề)
  - A sentence is any formula in which all variables are within the scope of corresponding quantifiers.
- **Clause** (mệnh đề hợp)
  - A wff consisting of a literal or a disjunction of literals (literals connected by ORs).

Predicates in First-Order Logic

- Predicates express properties, relations, or conditions that hold between objects
- They describe the state of the world or assert facts about entities within the domain
- Special functions that return only Boolean values (true or false)
- Examples:
  - IsHuman(x)
  - IsParent(x, y)

Quantifiers in First-Order Logic

- Quantifiers allow for the specification of statements about the entirety or existence of objects within the domain.
- Universal quantifiers (∀) (*lượng từ phổ quát*)
  - Denotes that a statement holds for all objects in the domain
  - **Example:** ∀x P(x) means "for all x, P(x) is true", indicating that property P holds for all objects x in the domain.
- Existential quantifiers (∃) (*lượng từ tồn tại)*
  - Denotes that a statement holds for at least one object in the domain
  - **Example:** ∃x P(x) means "there exists an x such that P(x) is true", indicating that there is at least one object x in the domain for which property P holds

Connectives in First-Order Logic

- Conjunction (Λ) (phép toán hội)
  - Represents logical "AND" between two propositions. The conjunction of two propositions is **true** *only if* **both propositions** are **true**
  - **Example:** If P(x) represents "x is red" and Q(x) represents "x is round", then P(x) Λ Q(x) represents "x is red and round".
- Disjunction (V) (phép toán tuyển)
  - Represents logical "OR" between two propositions. The disjunction of two propositions is **true** if **at least one** of the propositions is **true**.
  - **Example:** If P(x) represents "x is a cat" and Q(x) represents "x is a dog", then P(x) V Q(x) represents "x is either a cat or a dog".

Connectives in First-Order Logic

- Implication (→) (phép toán kéo theo)
  - Represents logical "**if-then**" relationship between two propositions. The implication P→Q is **false** if and only if P is **true**, but Q is **false**. Otherwise, it is always **true**.
  - **Example:** If P(x) represents "x is a mammal" and Q(x) represents "x produces milk", then P(x)→Q(x) represents "if x is a mammal, then x produces milk".
- Negation (¬) (phép toán phủ định)
  - Represents logical "NOT" or negation of a proposition. It reverses the truth value of the proposition.
  - **Example:** If P(x) represents "x is intelligent", then ¬P(x) represents "x is not intelligent"

First-Order Logic Example

- Examples
  - C = Bob drinks coffee
  - D = Bob eats cake
  - C ∧ D: Bob drinks coffee AND Bob eats cake
  - C ∨ D: Bob drinks coffee OR Bob eats cake
  - ¬C: Bob does NOT drink coffee
  - C → D: If Bob drinks coffee, then Bob eats cake
  - C ↔ D: If Bob drinks coffee, then Bob eats cake and *vice versa*
- Statements can be joined together using the symbols above to provide more complicated logical statements
  - Likes(Bob, carrots) ∧ Likes(Bob, cabbage)
- Semantic statements can be expanded to include more general ideas
  - ∀X (Likes(Bob, X) → Eats (Bob, X))
  - implies that Bob eats everything that he likes

Representing Procedural/Relational Knowledge

- Production Rules
  - If <premise>, then <conclusion>
  - If <condition>, then <action>
  - Rules permit the generation of new knowledge in the form of facts that are not initially available but that can be deduced from other knowledge parts. These facts are generated as the conclusions of the rules are applied.
- Semantic Networks
  - Graphical descriptions of knowledge composed of nodes and links that carry semantic information about the relationships between the nodes.
- Frames
  - Organizes knowledge typically according to **cause-and-effect relationships**. The slots of a frame contains items like rules, facts, references, and so on.

Reasoning: Types of Logic

- **Deduction (Diễn giải)**
  - The process of reasoning in which a conclusion follows necessarily from the stated premises; reasoning from the general to the specific.
  - If X is true and if X being true implies Y is true, then Y is true.
- **Induction (Quy nạp)**
  - The process of reasoning in which a conclusion about all members of a class from examination of only a few members of the class; reasoning from the particular to the general.
  - For a set of objects, X={a,b,c…}, if property P is true for a, b, and c, then P is true for all X.
- **Abduction (lý luận suy đoán)**
  - A form of deductive logic which provides only a "plausible inference." Using statistics and probability theory, abduction may yield the most probable inference among many possible inferences.
  - If Y is true and X implies Y, then X is true.

Reasoning: Forward Chaining

- In order to prove X, where X has the form A → C, find an axiom or theorem of the form A → B and transform the problem to the problem of proving B → C.
- Starts with some facts and applies rules to find all possible conclusions (data-driven)
- Data is normally entered prior to the system commencing the inference process.
- Rules are normally checked individually.
- Relevant rules are grouped together to make the system easier to write and validate
- Rules only fire when all the information concerning that rule is available.
- The inference engine is not programmed to ask questions and obtain new information while the program is running.
- Multiple conclusions can be reached

Reasoning: Forward Chaining

- Steps:
  1. Consider the initial facts and store them in working memory of the knowledge base.
  2. Check the antecedent part of the rules.
  3. If all the conditions are matched, fire the rule.
  4. If there is only one rule, do the following:
     a) Perform necessary actions
     b) Modify working memory and update facts
     c) Check for new conditions
  5. If more than one rule is selected, use the conflict resolution strategy to select the most appropriate rule and go to Step 4
  6. Continue until an appropriate rule is found and executed
  7. State the solution, or if there is no solution, then state that the rule base is insufficient

Reasoning: Forward Chaining

- **Example**: Consider a system with three rules
  1. If someone is a third-year student, then they need a job
  2. If someone is a third-year student, then they live on campus.
  3. If someone needs a job, they will look at job adverts.

Suppose we put the following data into memory:

John is a third-year student.

What will happen?

Note: In forward chaining, because the system is constantly alert for new data, the system would have searched all the rules for any whose conditions weren't true before but are now. It then adds their conclusions into memory.

Reasoning: Forward Chaining

- Example: **Solution**
  - Rules 1 and 2 have conditions, which match this new fact (*John is a third-year student.*)
  - So, the system will immediately create and add the two facts:
    - *John needs a job.*
    - *John lives on campus.*
  - These facts in turn can trigger rules. As each arrives, the system would look for yet more rules that are made true.
  - In this case, the fact John needs a job would trigger Rule 3, resulting in the addition of another fact into memory:
    - *John will look at job adverts.*
  - The fact that John lives on campus would not trigger anything else.

Reasoning: Backward Chaining

- In order to prove X, where X has the form A → C, find an axiom or theorem of the form B → C and transform the problem to the problem of proving A → B.
- Starts with the desired conclusion(s) and works backward to find supporting facts (goal-driven).
- the system is driven from the goals back to the data
- In backward chaining, the system does no work until required, i.e., goal is specified.

Reasoning: Backward Chaining

- Steps:
  1. Start with a possible hypothesis, H.
  2. Store the hypothesis H in working memory, along with the available facts.
  3. If H is in the initial facts, the hypothesis is proven. Go to Step 7.
  4. If H is not in the initial facts, find a rule R that has a descendent (action) part mentioning the hypothesis.
  5. Store R in the working memory.
  6. Check conditions of R and match with the existing facts.
  7. If matched, then fire the rule R and stop. Otherwise, continue to Step 4.
  8. Provide a result—which is that the goal can or cannot be achieved.

Reasoning: Backward Chaining

- **Example**: In the same three-rule knowledge base as we used in the previous activity,
  1. If someone is a third-year student, then they need a job
  2. If someone is a third-year student, then they live on campus.
  3. If someone needs a job, they will look at job adverts.

- we add the data (a fact):

  John is a third-year student.

What does the system do immediately?

What does the system do when we ask the following question:

Is there anyone who will look at job adverts?

Reasoning: Backward Chaining

Example: **Solution:**

- In backward chaining, because the system is goal driven, the system would do nothing at all until it was asked a question, i.e., provided with a goal to seek or a hypothesis to test.
- When asked the question

  Is there anyone who will look at job adverts?

the system would try to answer it.

- The first step would be to search either for a **fact** that gives the answer directly, or for a **rule** by which the answer could be inferred
  - it searches the entire knowledge base for rules whose conclusions, if made true, will answer the question
- In this example, there are NO facts directly giving the answer.

Reasoning: Backward Chaining

- Example: **Solution:**
- there's one rule whose **conclusion,** if true, would supply an answer: **Rule 3**
- The system next checks the Rule 3's **conditions**
  - Is there anyone who needs a job?
- The system will look either for a **fact** that **answers directly,** or for a **rule.**
  - There are no facts
  - **Rule 1** is relevant
- Then, the system will check the conditions of **Rule 1.**
  - Is there a third-year student?
  - There is a **fact** that answers this: *John is a third-year student*
  - we've **proved Rule 1,** and by doing so also **proved Rule 3,** and that answers the original question.

Reasoning: Backward Chaining

Example: **Solution:**

- What if we did not know that John is a third-year student
  - If no rule provides this as a conclusion and this is not currently known, then backward chaining systems will ask the user for an answer
- Backward chaining systems will engage in a dialogue with the user.
- Backward chaining resolved the specific goal only
  - it did not determine that John lives on campus as this was not relevant.
- Forward chaining would find every possible conclusion

Reasoning: Backward Chaining

- Use of backward chaining

Reason for backward chaining	Examples
There is a clear set of statements, which must be confirmed or denied.	Is machine one causing the quality control problem?
A large number of questions could be asked of the user, but typically only a few are necessary to resolve a situation.	When processing of a motor claim for vandalism; it is not necessary to know about personal injuries.
It is desirable to have interactive dialogue with the user.	Asking machine operator detailed questions about suspect machinery.
Rule execution depends on data gathering which may be expensive or difficult.	Real-time observations by the user.

Comparison of Forward and Backward Chaining

- Useful factors can be used to consider the choice between a forward or backward chaining in Expert Systems

Factor	Reason
The logical reasoning process.	Some processes naturally use forward chaining logic, e.g. using errors in computer systems to determine the cause of the error.
What are the inputs and where do they come from?	Where there are few inputs but many outputs, then forward chaining will be more efficient.
What are the outputs and where to they go?	Where there are few outputs, then backward chaining is more appropriate.
Hypothesis driven.	Backward chaining is relatively efficient where hypotheses are involved.

Comparison of Forward and Backward Chaining

- Examples of forward and backward chaining

Use forward chaining	Use backward chaining
Sensor indicates machine failure; need to find out what happens next.	Defect observed in product; need to locate faulty machine.
User types erroneous input for insurance claim; need to alert user.	Suspect an overpayment on an insurance claim; need to check form for erroneous input.
Stock value suddenly drops; need to predict market responses.	FTSE industrials drop; need to know if a particular stock will be affected.

FTSE: Financial Times Stock Exchange Index

Developing Rule-based Systems

- Main Problems in Building a Knowledge-based system (KBS)
  - lack of explanation facilities
  - brittleness (tính dễ đổ vỡ của hệ thống)
    - not be able to solve the problem outside the scopes of the provided knowledge

Explanation Facilities

- Explain to the user of the system how decisions have been made by the system
  - why particular rules have been applied
- The output from expert systems (ESs) must have the ability to provide a similar level of explanation as human experts
  - Users like to understand why they are being given advice
  - Users like to see why certain courses of action have been recommended
  - Users like to see problems associated with other alternative actions
- Explanations provided by an ES may not be as detailed as those provided by a human expert
  - ES only has knowledge in a very specific subject domain
  - Answers cannot be related to any wider context
  - the knowledge base may not provide sufficient details in the specific subject domain

Explanation Facilities

- Rule tracing
  - A 'how' trace enables the user to find out how an ES has arrived at a conclusion
  - A 'why' trace helps the user understand why a particular question is being asked
  - Provide the user with some feedback on how the ES is working
  - simply providing a chain of reasoning
  - link a problem and a solution
    - the ES can explain why a particular conclusion was reached.
    - does not necessarily know why the appropriate rules were in the knowledge base
    - The system can state **IF something THEN something else happens,** but not why those events are linked

Explanation Facilities

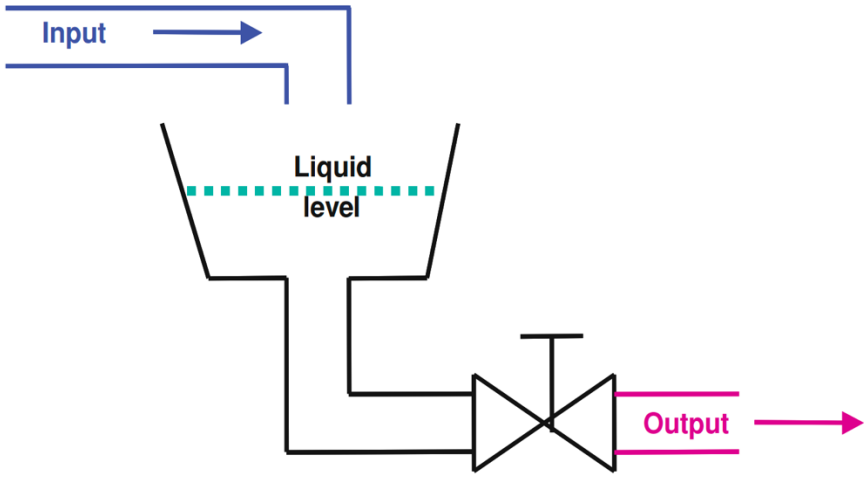
- Building Explanation Text into an Expert System (ES)
  - Explanation text can be built into the ES to help the user understand the outputs provided
  - *Example*:

Q: What happens to the water level when the valve is opened a little more?

A: It will go down.

Q: Why?

A: Because the output flow will be larger than the input flow



Explanation Facilities

- Building Explanation Text into an ES
  - *Example*:

Q: Why?

A: Because more water will be released through the output pipe.

Q: Why will more water be released through the output pipe?

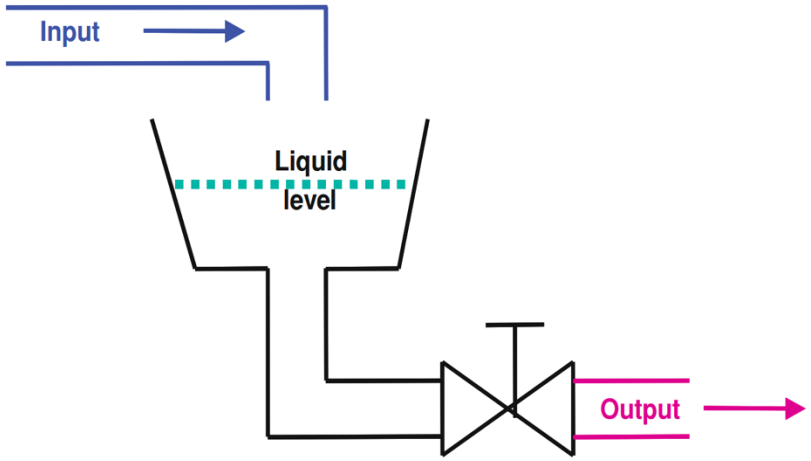
A: Because opening the valve widens the hole, letting more water through.

Q: Why does this happen. . .?

- Having placed all above explanations into the system, questions can be asked such as:

How did you arrive at that answer?

- The ES can then provide the rule explanations to show how the answer was derived



Explanation Facilities

- Building Explanation Text into an ES
  - *Example*:

User: What happens to the water level when the valve is opened?
System: The water level will go down.
User: How do you know this?
System: Because I applied Rule 1 which states when the valve opens, more water will be released. I then applied Rule 2 which states when more water is released then the water level will fall.

Explanation Facilities

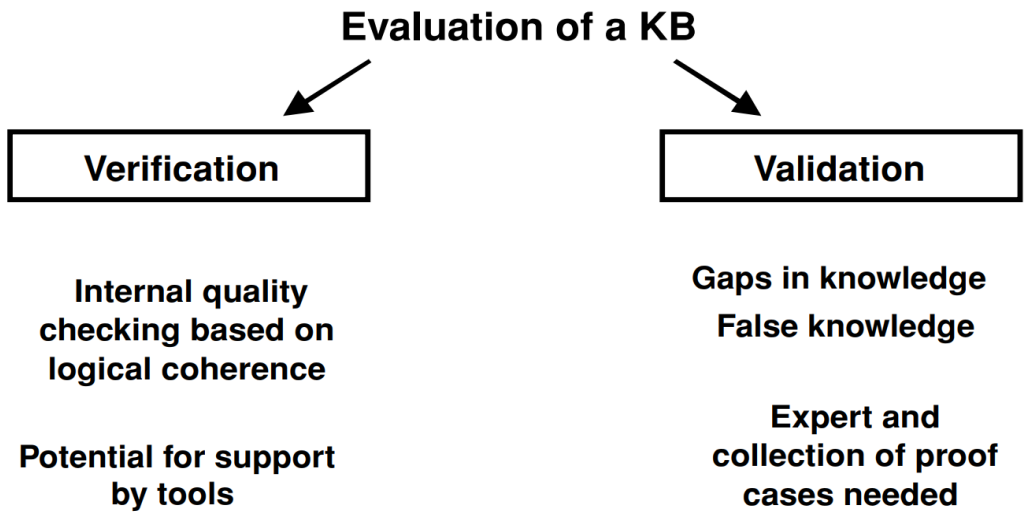
- Dangers of Unintelligent Explanations
  - there is no mechanism to ensure that this recommendation is appropriate to your individual circumstances.
  - As systems grow in apparent intelligence, they are given more responsibility
    - not to place too much trust in the system
    - The user is still responsible for checking any answer for reasonableness
  - Adding poor quality or simplistic explanation facilities can inspire undue confidence in a system that does not warrant it
    - This may mislead the user into taking incorrect decisions
  - The apparent intelligence may vastly exceed the true level of understanding
    - the ES is only as good as its rule base, and if this is incorrect, then solutions from the system will also be wrong
    - Poor quality explanation facilities may encourage the user to accept erroneous recommendations.

Brittleness

- The apparent level of intelligence exceeds the true level of intelligence.
- When the new problem requires knowledge not contained within the system, it will not be able to solve the problem.
  - The system proposes a faulty solution
- The inclusion of an unintelligent explanation facility which will encourage the user to accept the faulty output.
- Recent work on **defining ontologies** is helping to **overcome** the problem of **brittleness.**
  - defining an ontology of the limits of knowledge contained within a system
  - an ES could also recognize the limitations of its knowledge base

Evaluation of Knowledge-based Systems

- An attempt to assess the overall value of the KBS
- Checking not only that the KBS has acceptable performance levels, but also that the system is useable, efficient and cost-effective.
- Involving two more terms: **validation** and **verification**
  - Validation (xác thực/nhận) measures the performance of the KBS
  - Verification (xác minh) checks that the KBS has been built correctly



Evaluation of Knowledge-based Systems

- **Verification:** involves checks for the following
  - **Syntactic coherence:** check that all objects in the KB are correctly defined with respect to the inference engine
  - **Logical coherence:** detect logical contradictions
  - **Contextual coherence:** check that the KB is consistent with the model of the problem.
- Type of errors can be detected by verification
  - **Subsumed Rules:** two rules have the same conclusion but one rule has additional conditions:
    - Rule 1. IF A AND B **AND C** THEN X
    - Rule 2. IF A AND B THEN X
  - **Unnecessary IF Conditions:** the conclusions of two rules are the same and the conditions of the rules except for one are the same.
    - Rule 1. IF the patient has pink spots AND **has a fever** THEN measles.
    - Rule 2. IF the patient has pink spots AND **does not have a fever** THEN measles.

Evaluation of Knowledge-based Systems

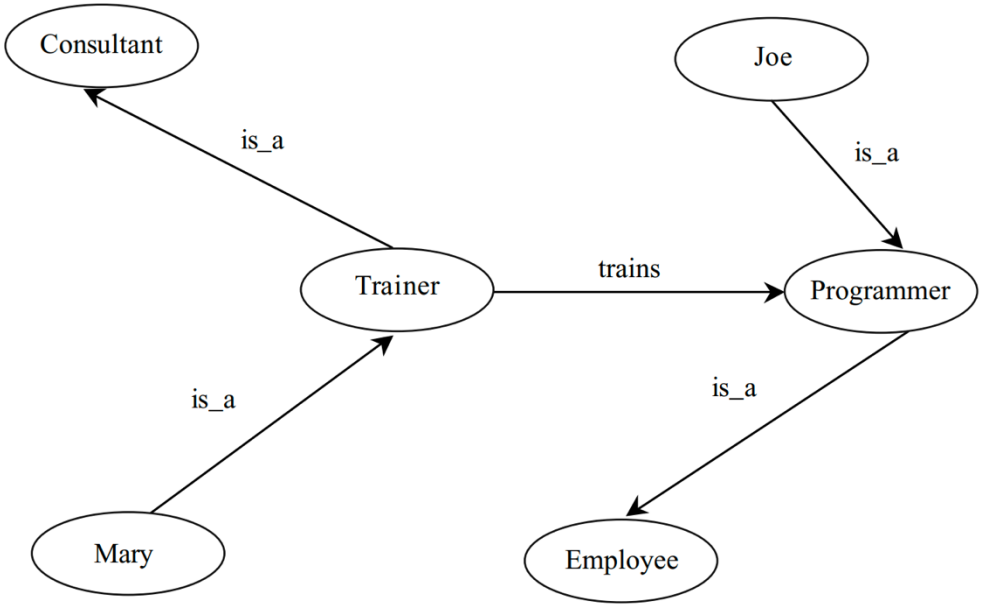
- **Validation:** ensuring that the work domain is correctly linked to and reflected in the knowledge domain
  - defining the work domain
  - defining the proof cases to use
  - deciding how many proof cases to use
- Proof cases test the KBS by ensuring that the results from the KBS conform to the results already predicted by the human expert.
- The KBS will be validated where the proof cases match those of the human expert.
- The number of proof cases required depends on variables such as the number of rules in the KBS and the accuracy required from the outputs.

Evaluation of Knowledge-based Systems

- **Validation**: some pre-defined measures to check the outputs of the system:
  - Accuracy: how well the system reflects reality
  - Adequacy: how much of the required knowledge is included within the knowledge base.
  - Realism: whether the KBS provides realistic solutions
  - Sensitivity: how changes in the knowledge base affect the quality of outputs.
  - Usefulness: how useful the outputs are for solving problems.
  - Validity (tính hợp lệ): whether the outputs can be used to make accurate predictions.

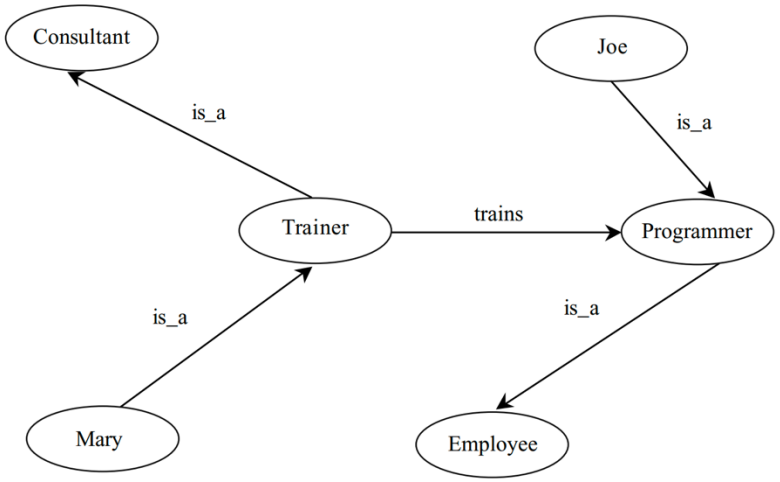
Semantic Networks

- One of the oldest and easiest to understand knowledge representation schemes
  - a graphical representation of knowledge that shows objects and their relationships
- Objects are shown by nodes
- Links between the nodes describe the relationship between two objects



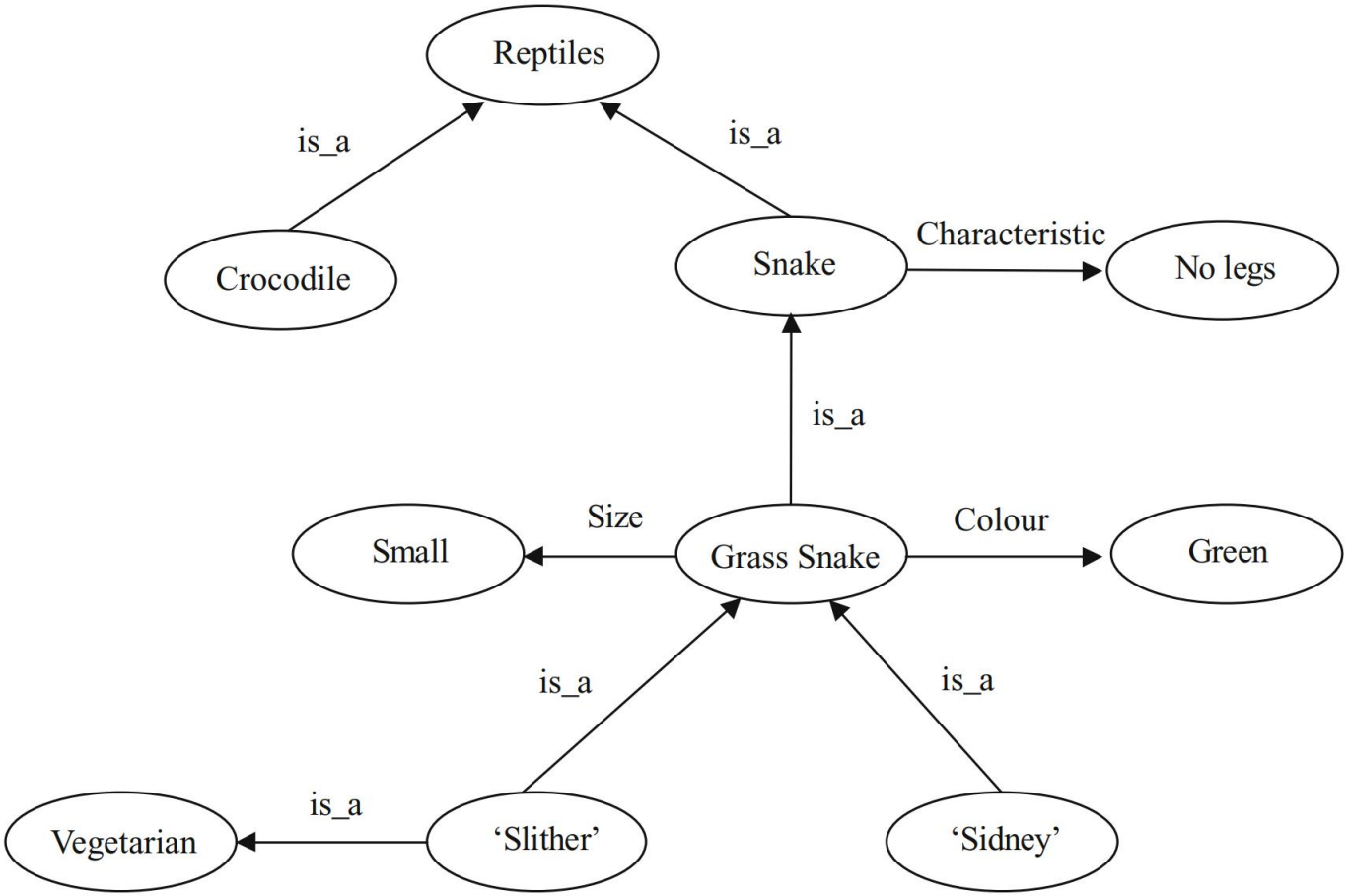
Semantic Networks

- **Inheritance**
  - how one object inherits the properties of another object.
  - The hierarchical nature of the diagram helps explain the elements of the network
- Example:
  - Mary, in being a trainer, inherits the properties of the consultant class
  - Joe, in being a programmer, inherits the properties of the employee class



Semantic Networks

- **Inheritance**



Semantic Networks

- **Reasoning with Semantic Networks**
  - Be very difficult to reason from a semantic network
    - Because an inference engine must understand what type of link exists between nodes
    - there are no constraints on allowable links
  - If an inference engine were to reason with the semantic network, then it must have some understanding of the words representing the relationships between nodes (e.g., is_a, eats, colour,…)
  - Much of the reasoning with semantic networks is in the form of **deducing indirect links** between concepts based on **direct links** explicitly stated
    - difficult for an inference engine to achieve in practice
  - **Semantic networks are often used only as a communication tool between the knowledge engineer and the domain expert**

Semantic Networks

- **Advantages**
  - They tend to be a powerful and adaptable method of representing knowledge
    - many different types of object can be included in the network
  - The network is graphical and therefore relatively easy to understand.
  - Can be used as a common communication tool between the knowledge engineer and the human expert during the knowledge acquisition phase of designing an expert system

Semantic Networks

- **Disadvantages**
  - Difficult to show all the different inference situations using a network
  - Less reliable than other knowledge representation techniques
    - Because inferring becomes a process of searching across the diagram
  - Diagrams can become very complex
  - The wide range of possible kinds of links and the ways they might combine to form indirect linkages
    - make the network susceptible to a combinatorial explosion
  - Have difficulty associating procedural knowledge with the facts represented by the network
    - they lack any means of bundling-related facts into associated clusters
    - require extensive search operations to reach conclusions

Semantic Networks

- **Limitations**
  - In semantic network representations, there is no formal semantics, no agreed-upon notion of what a given representational structure means, as there is in logic, for instance.
  - Semantic networks do tend to rely upon the procedures that manipulate them
  - The system is limited by the user's understanding of the meanings of the links in a semantic network
  - Links between nodes are not all alike in function or form.
    - need to differentiate between links that assert some relationship and links that are structural in nature

Frames

- Frames are a simplified version of a semantic network where **only 'is a'** relationships apply.
- Provide a method of storing knowledge, collecting specific information about one object in an ES.

Coffee mug FRAME	
IS_A	Mug
COLOUR	
CAN_HOLD_LIQUID	True
NUMBER_OF-HANDLES	Default = 1
SIZE	Range: Small, Medium, Large
PURPOSE	Value : drinking coffee
COST	Demon (£ needed)
MATERIAL	Default = pottery

Frames

- Within a frame structure, slots (i.e., rows) can:
  - store details of each data object
  - provide links to other frames
  - contain procedural code, linking to other applications to obtain data
  - indicate if certain properties of each object are needed within that frame

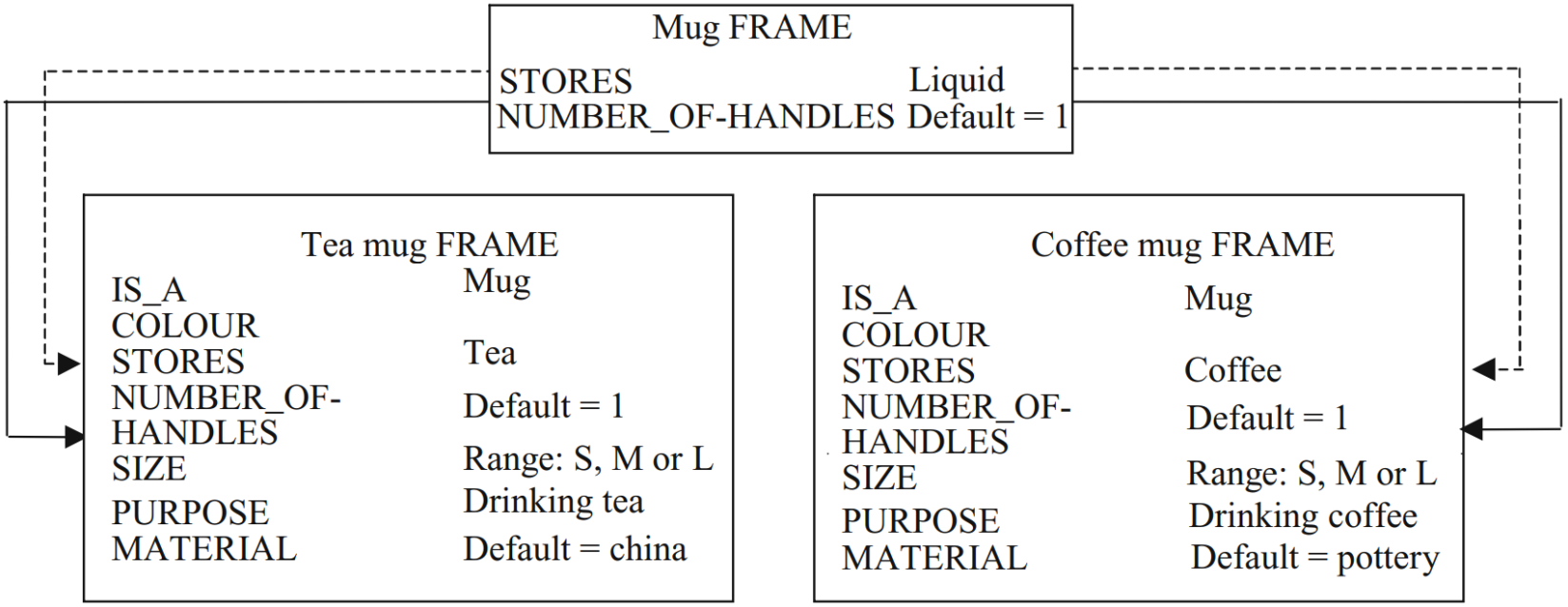
Coffee mug FRAME	
IS_A	Mug
COLOUR	
CAN_HOLD_LIQUID	True
NUMBER_OF-HANDLES	Default = 1
SIZE	Range: Small, Medium, Large
PURPOSE	Value : drinking coffee
COST	Demon (£ needed)
MATERIAL	Default = pottery

Frames

- Levels within a frame:
  - The highest level in a frame is literally FRAME: stores the name of the specific frame
  - Below FRAME are SLOTS, with each slot providing information on one of the attributes of that frame
  - Within the slot, the FACET provides detail on each attribute including a value, ranges, default values, calculated values
  - the DATA provides specific information about each *attribute*
    - the NUMBER_OF_WHEELS being 4 in a frame describing a motor vehicle

Frames

- **Inheritance:**
  - frames can inherit the attributes of other frames, in a hierarchical structure
  - the objects lower in the hierarchy automatically inherit the contents of the corresponding slots, unless this data is overwritten



Frames

- **Advantages:**
  - be represented in the form of a table, making the information easy to read
  - store default values
  - use default values in the reasoning process
    - If later, the default value is found to be incorrect, then the system can overwrite the default value and then run through its reasoning again
  - be structured hierarchically and thus allow easy classification of knowledge
  - reduce complexity by allowing a hierarchy of frames to be built up
  - clearly document information using common formats and syntax
  - combine procedural and declarative knowledge
  - constrain allowed values, or allow values to be entered within a specific range

Frames

- **Disadvantages:**
  - can be inefficient at runtime
    - because they do not provide the most efficient method to store data for a computer
  - can lead to 'procedural fever'
    - the apparent requirement to focus on making appropriate procedures rather than checking the overall structure and content of the frames
  - require care in the design stage to ensure that suitable taxonomies