IAI : Overview and Revision

W1 : Module Aims and Learning Outcomes

Aims:

1. Provide a general introduction to AI, its techniques and its main sub-fields.
2. Give an overview of key underlying ideas, such as knowledge representation, rule based systems, search, and learning.
3. Demonstrate the need for different approaches for different problems
4. Provide a foundation for further study of specific areas of AI.

Learning Outcomes:

1. Recognise the important features of AI systems and structure the field of AI into its main sub-fields.
2. Explain some of the most important knowledge representation formalisms and why they are needed, discussing their advantages and disadvantages. Apply these knowledge representation formalisms to simple unseen examples.
3. Describe and apply some simple search algorithms.
4. Outline the processes involved in Expert Systems and in building such systems.
5. Discuss the importance of learning in intelligent systems, and their implementation.
6. Provide examples of different types of AI systems, and explain their differences, common techniques, and limitations.

W2 : What Exactly is AI?

"Artificial Intelligence (AI) is the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit characteristics we associate with intelligence in human behaviour – understanding language, learning, reasoning, solving problems, and so on." (Barr & Feigenbaum, 1981)

Engineering Goal To solve real world problems using AI techniques such as knowledge representation, learning, rule systems, search, and so on.

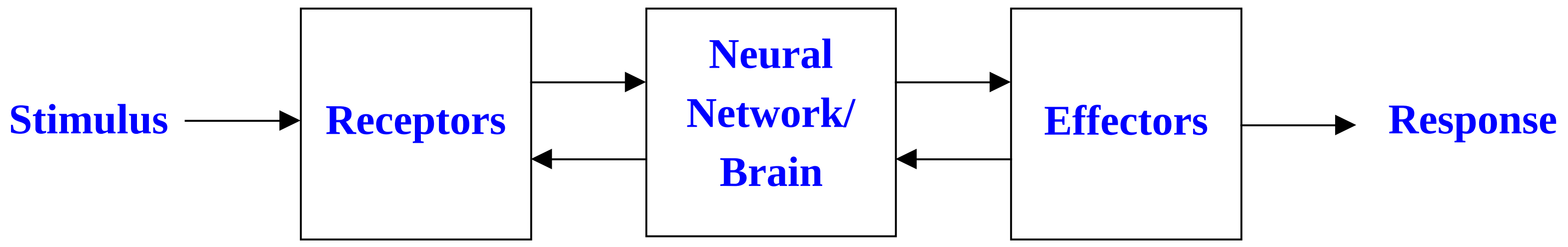
Scientific Goal To determine which ideas about knowledge representation, learning, rule systems, search, and so on, explain various sorts of real intelligence.

AI has ***roots*** in a number of older fields: Philosophy, Logic/Computation, Psychology/ Cognitive Science, Biology/Neuroscience, Evolution. Note: **Strong AI** v. **Weak AI.**

AI has many ***sub-fields*** (such as Neural Networks, Evolutionary Computation, Expert Systems, Natural Language Processing, Planning, Robotics, Vision), but they employ common ***techniques*** (such as Representation, Learning, Rules, Search).

W3 : Biological Intelligence and Neural Networks

The human nervous system that forms the basis of our natural intelligence has the form:



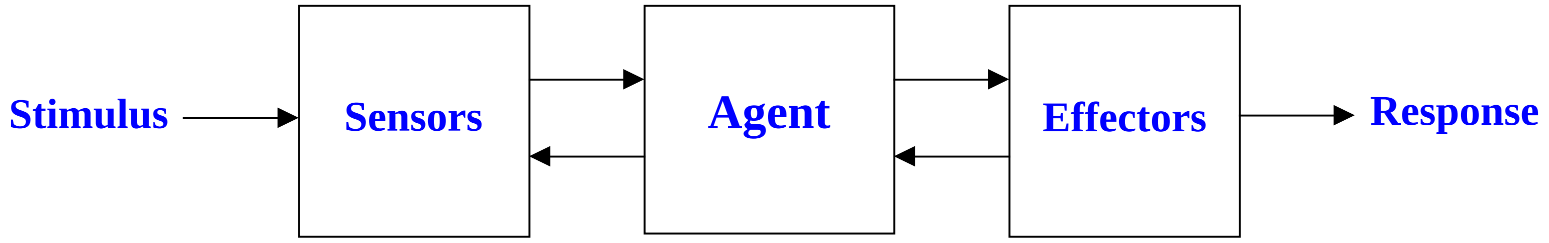
We can attempt to use ***Artificial Neural Networks*** as the basis of AI systems, because:

1. They are extremely powerful computational devices (Turing equivalent).
2. Massive parallelism makes them very efficient.
3. They can learn and generalize – so no need for enormous feats of programming.
4. They are very fault tolerant – like the “graceful degradation” of biological systems.
5. They are very noise tolerant – coping where normal symbolic systems have difficulty.

They can be used for both ***Brain Modelling*** and ***Artificial System Building***.

W4 : Rational Agents

"An ***agent*** is anything that can be viewed as perceiving its environment through ***sensors*** and ***acting*** upon that environment through ***effectors*.**" (Russell & Norvig, 1995)



A ***rational agent*** is one that acts in a manner that causes it to be as successful as it can. We need to determine appropriate ***performance measures*** to judge success in each case.

An ***ideal rational agent*** is one that takes whatever action is expected to maximise its performance measure on the basis of the evidence provided by its perceptual history and whatever built-in knowledge it has. An ***autonomous agent*** will supplement its built-in knowledge with its own acquired (or learned) knowledge in order to act appropriately.

We design agents according to their ***Percepts***, ***Actions***, ***Goals*** and ***Environment***.

Types of Intelligent Agents

We can classify four types of intelligent agents of increasing sophistication:

1. ***Simple Reflex Agents*** – use simple *condition-action rules* or *IF–THEN rules* to produce appropriate (intelligent looking) actions for given percepts.
2. ***Reflex Agents with an Internal State/Model*** – can keep track of previous states and use knowledge of how the world evolves to produce better actions.
3. ***Goal based agents*** – can determine sequences of actions to reach its goal state(s). *Search* and *planning* techniques will usually be required.
4. ***Utility based agents*** – can use utility/quality measures to choose between alternative sequences of actions/states that lead to a goal state being obtained.

We need to take into account the five principal features concerning agent environments: ***accessibility, determinism, discreteness, episodicness,*** and ***staticness***. To build the most intelligent agents possible, percepts should not only be used for generating actions, but also to improve the ability to act in the future, i.e. to ***learn***.

W5 : Knowledge Representation

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

A ***knowledge representation language*** is defined by two aspects:

1. ***Syntax*** The syntax of a language defines which configurations of 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 can make.

A good knowledge representation system for a particular domain should possess four ***important properties***: Representational Adequacy, Inferential Adequacy, Inferential Efficiency, and Acquisitional Efficiency.

Common knowledge representations are: Natural Language, Databases, First Order Logic, Rule Systems, Semantic Networks and Frames. Advantages/Disadvantages?

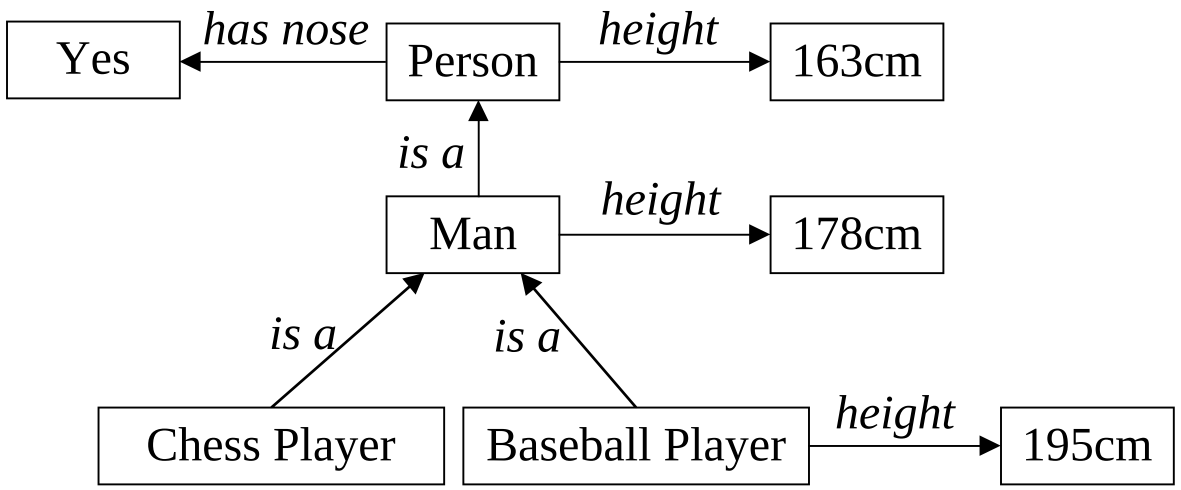
Practical Aspects of Good Representations

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.
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.
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.

W6 : Semantic Networks

A ***semantic network*** represents knowledge as a set of labelled nodes and links, e.g.



Two important features are ***inheritance*** and ***defaults*** (i.e. typical values). We can assign expected/default values of parameters (e.g. height, has nose) and inherit them from higher up the hierarchy. This is more efficient than listing all the details at each level. We can also ***over-ride*** the defaults. For example, baseball players are taller than average men, so we make their default height over-ride the default height for men.

Components of a Semantic Network

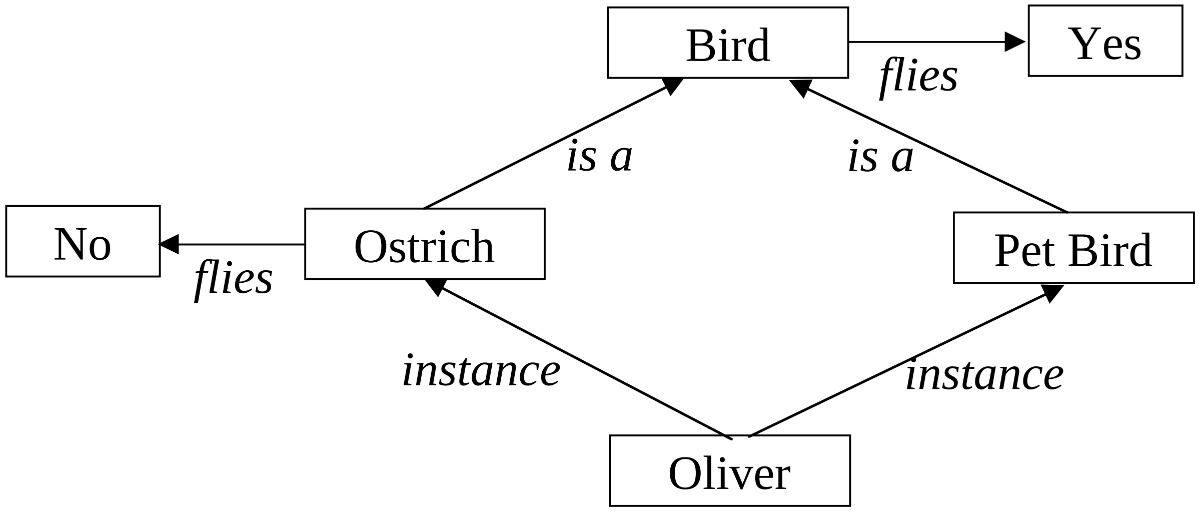
The formal components of semantic networks are quite straight-forward:

Lexical part	nodes – denoting objects links – denoting relations between objects labels – denoting particular objects and relations
Structural part	the links and nodes form directed graphs the labels are placed on the links and nodes
Semantic part	meanings are associated with the link and node labels (the details will depend on the application domain)
Procedural part	constructors allow creation of new links and nodes destructors allow the deletion of links and nodes writers allow the creation and alteration of labels readers can extract answers to questions

Some common examples include AND/OR Trees, IS-A and IS-PART Hierarchies, and Representations of Events and Natural Language Sentences.

Multiple Inheritance and Tangled Hierarchies

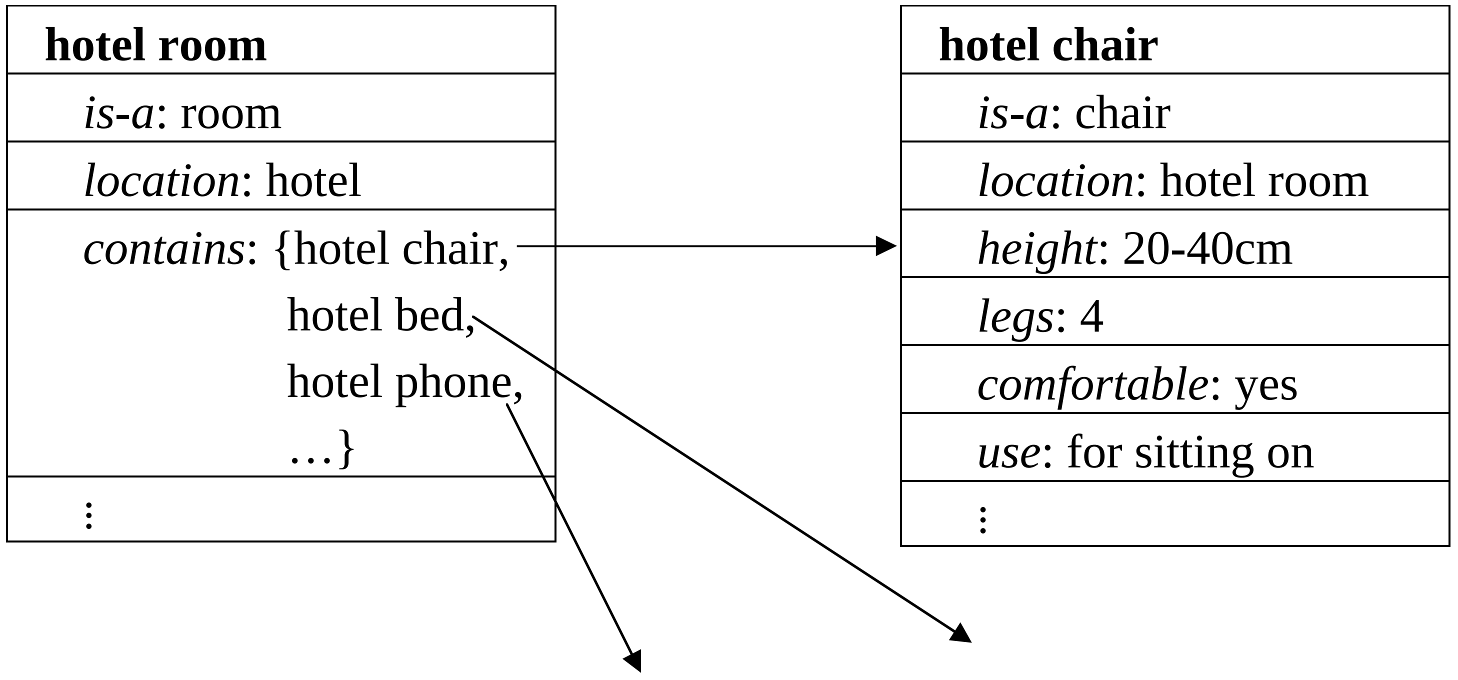
If ***multiple inheritance*** is allowed, we must avoid ***inheritance conflicts***. Hierarchies that are not simple trees are called ***tangled hierarchies***. For example, can Oliver fly in:



A better solution than having a specific “flies no” for all individual instances of an ostrich, would be to have an algorithm for traversing the hierarchy which guarantees that specific knowledge will always dominate over general knowledge. We can define the concept of ***inferential distance*, ** that provides a *partial ordering* of closeness.

Frame Based Systems

Frames are a natural extension of *Semantic Networks*. They consist of sets of slots filled by values, procedures for calculating values, or pointers to other frames. For example:



Scripts are frame based systems that describe stereotyped sequences of events and actions that enable an intelligent agent to perform appropriately in a particular context.

W7 : Production Systems

A ***production system*** consists of four basic components:

1. A ***set of rules*** of the form $C_i \rightarrow A_i$ where C_i is the condition part and A_i is the action part. The condition determines when a given rule is applied, and the action determines how it is applied.
2. One or more ***knowledge databases*** that contain whatever information is relevant for the given problem. Some parts of the database may be permanent, while others may temporary and only exist during the solution of the current problem. The information in the databases may be structured in any appropriate manner.
3. A ***control strategy*** that determines the order in which the rules are applied to the database, and provides a way of resolving any conflicts that can arise when several rules match at once.
4. A ***rule applier*** which is the computational system that implements the control strategy and applies the rules.

The whole system operates according to the deductive inference rule “modus ponens”.

Recognize-Act Cycle

Typically, our production systems will have a rule interpreter that takes the form of a ***Recognize-Act Cycle***. This cycle has four stages:

1. ***Match*** condition/premise patterns in the rules against the elements in the working memory to identify the set of applicable rules. This will usually involve ***binding*** specific values to the variables in the rules.
2. If there is more than one rule that can be 'fired' (i.e. that can be applied) at a given time, then use a ***conflict resolution*** strategy to choose which one from that ***conflict set*** to apply. If no rules are applicable, then stop.
3. ***Apply*** the chosen rule, which may involve adding a new item to the working memory or deleting an old one.
4. ***Check*** if the terminating condition is fulfilled. If it is, then stop. Otherwise, return to stage 1.

The ***termination condition*** can either be defined by a goal state, or by a cycle condition (e.g. a maximum number of cycles).

Forward and Backward Chaining

Forward chaining or ***data-driven inference*** works by repeatedly: starting from the current state, matching at the premises of the rules (the IF parts), performing the corresponding actions (the THEN parts), and possibly updating the knowledge base or working memory.

Backward chaining or ***goal-driven inference*** works towards a final state by looking at the working memory to see if the sub-goal states already exist there. If not, then look at the actions (the THEN parts) of the rules that will establish the sub-goals, and set up new sub-goals for achieving the premises of those rules (the IF parts).

There are four major factors to help us choose between forward and backward reasoning:

1. Are there more possible start states or goal states? Start at the lowest.
2. Do we require the program to justify its reasoning? If so, follow human reasoning.
3. What kind of events trigger problem solving? New facts or queries?
4. In which direction is the branching factor greatest? Go with the lower.

Conflict Resolution Strategies

Perhaps the five most common ***general conflict resolution strategies*** are:

1. Delete instantiations of rules that have already fired (to prevent repetitions).
2. Order instantiations by the generation age of all the elements. Prefer the younger ones, since new elements are more likely to describe the actual situation.
3. Compare the generation age of the elements in working memory which match the first condition of the rules. Prefer the younger ones. May be more efficient than 2.
4. Prefer the most specific rules (i.e. those with the most pre-conditions).
5. Random choice (which, if nothing else, is very easy to compute).

The easiest way to proceed in ***problem specific cases*** is to simply *add extra conditions* to the rules to avoid the conflicts. These extra conditions can be related to the inference strategies, e.g. to what is currently being searched for. However, we will end up with a mixture of heuristic and factual knowledge, and large knowledge bases will not be easily maintainable. It makes sense to separate the *object level knowledge* from the *meta-level knowledge*, and to supplement our rules with appropriate *meta-rules*.

W8 : Search

The ***state space*** is the space of all possible states, or configurations, our system may be in. Generally, we work with some convenient ***representation*** of that search space.

If the number of possible states of the system is small enough, we can represent them all, along with the transitions between them, in a ***state space graph***.

The aim of our search algorithms is to find a route, or sequence of transitions, through the state space graph from our ***initial state*** to a ***goal state***.

There are four important properties of search algorithms we need to consider:

1. ***Completeness*** – Is a solution guaranteed to be found if at least one solution exists?
2. ***Optimality*** – Is the solution found guaranteed to be best possible solution?
3. ***Time Complexity*** – The upper bound on the time to find a solution.

5. ***Space Complexity*** – The upper bound on the storage space (memory) required.

Types of Uninformed Search

It is helpful to think of the search process as building up a ***search tree*** within the state space graph. The root of the search tree is the initial state. The leaf nodes correspond to states that have not yet been expanded, or have been expanded but generated no further nodes. A good strategy for avoiding ***repeated states*** will improve the search efficiency.

For tree branching factor b, maximum depth m, solution depth d, depth limit l, we find:

Strategy	**Complete**	**Optimal**	**Time Complexity**	**Space Complexity**
BFS	Yes	Yes	$O(b^d)$	$O(b^d)$
DFS	No	No	$O(b^m)$	$O(bm)$
DLS	If $l \geq d$	No	$O(b^l)$	$O(bl)$
DFIDS	Yes	Yes	$O(b^d)$	$O(bd)$
BDS	Yes	Yes	$O(b^{d/2})$	$O(b^{d/2})$

The ***best overall*** is DFID which is complete, optimal and has low memory requirements.

Informed Search

Informed search uses some kind of ***evaluation function*** to tell us how far each expanded state is from a goal state, and/or some kind of ***heuristic function*** to help us decide which state is likely to be the best one to expand next.

The simplest idea of ***greedy best first search*** is to expand the node that is already closest to the goal, as that is most likely to lead quickly to a solution. This is like DFS, it is not complete, not optimal, and has time and complexity of O(b^m).

Suppose each node *n* in a search tree has an evaluation function *f(n)* defined as the sum of the cost *g(n)* to reach that node from the start state, plus the estimated cost *h(n)* to get from that state to a goal state. ***A* search*** repeatedly picks the node with the lowest *f(n)* to expand next. If *h(n)* is ***admissible***, then this strategy is both complete and optimal.

Hill climbing and ***gradient descent*** learning repeatedly follow transitions that result in improvements in the evaluation function. This is often how neural networks are trained.

W9 : Expert Systems

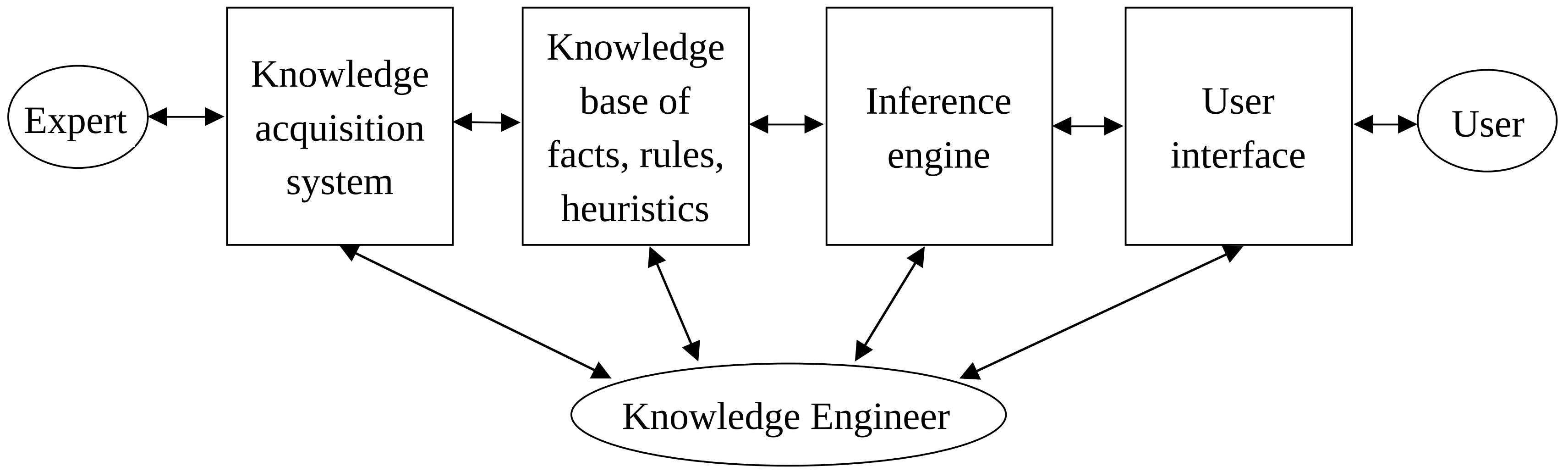
"An ***expert system*** is a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice." (Jackson, 1999)

There are no natural limits on what problem domains expert systems can be built to deal with. Expert systems can be distinguished from conventional computer systems in that:

1. They ***simulate human reasoning*** about the problem domain, rather than simulating the domain itself.
2. They perform ***reasoning over representations of human knowledge***, in addition to doing numerical calculations or data retrieval. They have corresponding distinct modules referred to as the ***inference engine*** and the ***knowledge base***.
3. Problems tend to be solved using ***heuristics*** (rules of thumb) or ***approximate methods*** or ***probabilistic methods*** which, unlike algorithmic solutions, are not guaranteed to result in a correct solution.
4. They usually have to provide ***explanations*** and ***justifications*** of their solutions or recommendations in order to convince the user that the reasoning is in fact correct.

Building Expert Systems

Expert systems are typically built around a powerful production system:



Expert
Knowledge acquisition system
Knowledge base of facts, rules, heuristics
Inference engine
User interface
User
Knowledge Engineer

The ***knowledge acquisition*** component allows the expert to enter knowledge or expertise into the system, and refine it later when required. The principal stages are: knowledge elicitation, intermediate representation, and compilation into executable form.

The major ***technical problems*** that need to be overcome are: the knowledge acquisition bottleneck, brittleness, lack of meta-knowledge, and the difficulty of reliable validation.

W10 : Treatment of Uncertainty

Potential ***sources*** of uncertainty for AI systems, such as expert systems, fall into two types: ***imperfect domain knowledge*** and ***imperfect case data.***

Then there are three ***types*** of uncertainty: ***randomness*** (e.g. 3% of sensors fail in first year), ***vagueness*** (e.g. tall, little, etc.), and ***inadequacy*** (e.g. reliability of experts' rules).

The obvious way to treat uncertainty is to use the laws of probability (***Bayes Rule***, ***frequentist probabilities***, etc.) to perform ***probabilistic reasoning***. But this is generally intractable because of the enormous number of joint/conditional probabilities involved.

Bayesian Networks avoid this problem by using ***conditional independencies*** among the variables to ignore many interactions. If $p(A|\{x_i\},\{y_i\}) = p(A|\{x_i\})$ then A is conditionally independent of $\{y_i\}$ and, as far as A is concerned, if we know $\{x_i\}$ we can ignore $\{y_i\}$.

Alternative treatments of uncertainty include ***Dempster-Shafer Theory*** based on ***belief functions,*** and ***Fuzzy Logic*** based on ***fuzzy set theory***.

W11 : Machine Learning

Strategies for learning can be classified according to the amount of inference the system has to perform on its training data. In increasing order we have:

1. ***Rote learning*** – new knowledge is implanted directly with no inference at all.
2. ***Supervised learning*** – the system is given a set of training examples consisting of inputs and outputs and is required to discover the relation or mapping between them.
3. ***Reinforcement learning*** – the system is given a set of training examples consisting of inputs and is required to improve its outputs using feedback on how good they are.
4. ***Unsupervised learning*** – the system is given a set of training examples consisting only of inputs and is required to discover all by itself what the outputs should be.

A general learning agent has four basic components: a ***performance element***, a ***critic***, a ***learning element***, and a ***problem generator***.

Traditional machine learning procedures include: ***rule induction systems***, the ***version space*** approach to ***concept learning***, and ***decision tree algorithms***. Nature inspired machine learning techniques include ***neural networks*** and ***evolutionary computation***.

Overview and Reading

1. The module appears to have achieved its aims and learning outcomes.
2. We began with an overview of the roots, goals and sub-fields of AI, and a discussion of biological intelligence and its basis in neural network systems.
3. The general ideas of rational and intelligent agents were then covered.
4. This led to the need for a range of powerful knowledge representations such as semantic networks and frames, and production systems.
5. Procedures for performing search were then discussed.
6. The above ideas could then be used in building expert systems, and for handling uncertainty in them.
7. We ended looking at how to build machine learning systems.

Reading

Your lecture notes!