In artificial intelligence, an expert system is a computer system that emulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning about knowledge, represented primarily as \textit{if-then} rules rather than through conventional procedural code. 

- Wikipedia

But conventional procedural code is often full of \textit{if-then} statements. What's the distinction?

### Conditional logic

- The statements look similar, but they're different:
  - **Procedural** programming: 
    - \text{if condition then take-some-action}
  - **Rule based** programming
    - \text{if condition then assert-a-fact}
    - That fact may affect conditions in other rules.

- The condition is often called the \textit{premise} of the rule, and the assertion is called the \textit{conclusion}.

### Basic forward-chaining search problems

- Given:
  - a set of facts (data base)
  - and a set of rules (knowledge base)
  - and a desired or possible conclusion*

- Determine whether the conclusion is:
  - implied by (a consequence of) the facts,
  - inconsistent with the facts (contradictory),
  - neither

- * If no conclusion is specified, determine whatever logically follows from the facts.
Representing facts and forward chaining inference

- A **knowledge base** consists of a collection of predicates:
  - assertions assumed to be true for all cases,
  - rules, often as implications \( P \rightarrow Q \)

- An **inference engine** obtains (often through a user dialog) more assertions about a specific case.

- Then what happens?

Examples

- A **knowledge base** consists of a collection of predicates:
  - assertions assumed to be true for all cases,
  - rules, often as implications \( P \rightarrow Q \)
    
    - *California is part of the United States*
    
    - *If X was born in the United States then X is an American citizen*

- An **inference engine** obtains (e.g. through a user dialog) more assertions about a specific case.
  - *Mary was born in California*
  
  and then draws conclusions that follow.

Representing facts and forward chaining inference

- Whenever the inference engine evaluates a rule with **true** condition, we say that the rule **fires**.

- That may then make it possible for other rules in the data base to fire, a **chain reaction**.
  
  - Obviously, this is interesting for A.I. only with huge numbers of predicates.

  *Why?*

Should a given search . . .

- be deterministic or non-deterministic?

- yield:
  - one useful result?
  - all possible results?
  - the **optimal** result?

- show
  - just the result (end state)
  - or the path that led to the result?
**Experimental application**

- Medical diagnosis
  - MYCIN (~1975 Stanford Medical School):  
    - **Rule base:** 600 rules (coded in LISP)  
    - (if-then logic with confidence factors)  
    - **Input:** set of symptoms  
    - Desired **result:** diagnosis of disease or other malady  
    - Tests showed 69% accuracy  
    - (That's comparable to medical specialists)
  
- Effective especially in diagnosing *rare* diseases.  
  http://en.wikipedia.org/wiki/Mycin

**Why?**

**Confidence factors**

- A weight between 0 and 1* assigned to a proposition, which can be either the premise of a rule or the conclusion of a rule.

  **Confidence factor** = 1 or not specifying a confidence factor means **certainty;** a rule.

- They combine like probabilities. See reference.

  * between 0 and 100 in some treatments

**Logical expressions**

- Clojure supports the usual logical connectives, but in a slightly different way from most other programming languages.

  See http://clojure.org/cheatsheet for details. This is in lieu of a textbook.

- Logical **and** and **or** work in the usual way, except:
  - Any number of terms may be given, e.g.  
    (and c1 c2 c3 c4)
  - Short circuit goes left to right. If the last term is reached it is the value of the expression.
Evaluating logical expressions

- Any number of terms may be given in the list, e.g. `and c1 c2 c3 c4`
- Short circuit goes left to right. Evaluation stops:
  - For `and` when the first `false` (or `nil`) term is encountered.
  - For `or` when the first `true` term is encountered.
  - If the last term is reached it becomes the value of the expression.

Clojure conventions for logical constants

- In Clojure anything that isn't `false` or `nil` is considered to be true.
- Note that even `0` is true!
- Predicates yield either true or false as expected.
  - Relations (`,. `<.` etc.)
  - Forms that yield true or false (conventionally using suffix question mark in the name).