

# G53KRR

Development cycle of a knowledge-based system  
Knowledge acquisition

# Plan of the lecture

- 1 Development cycle of a knowledge-based system
- 2 Expert systems
- 3 Knowledge acquisition
- 4 Decision tables
- 5 Modern uses of rules: semantic web, business rules
- 6 Rules in Java

## Recommended reading for this lecture

- D. Partridge, K.M. Hussain. *Knowledge-based information systems*. London : McGraw-Hill, 1995, Ch.6,7. (development cycle, decision tables)
- E. Rich, K. Knight. *Artificial Intelligence*. McGraw Hill, 1991. Ch. 20.4 (Knowledge Acquisition).
- Semantic web <http://www.w3.org/2001/sw/>
- RuleML <http://www.ruleml.org/>
- Business rules and Java Rules Engine API (JSR)  
<https://www.oracle.com/technetwork/articles/java/javarule-139829.html>
- Jess <http://herzberg.ca.sandia.gov/jess/>

# Development cycle of a knowledge-based system

- 1. Plan knowledge base (the content of the knowledge base, relevant inputs and outputs, strategy for testing, knowledge dictionary, concepts etc. are identified.)
- 2. Select domain experts and knowledge sources
- 3. Acquire (elicit) knowledge
- 4. Formulate and represent knowledge (knowledge is formulated in the form suitable for inference)
- 5. Implement knowledge base (knowledge is encoded in machine-readable form.)
- 6. Test knowledge base
- depending on the results: continue with knowledge acquisition or go to 7.
- 7. Systems test

# Expert systems

- An **expert system** is a production systems which simulates behaviour of experts
- For example: MYCIN (diagnosis of bacterial diseases, 1970s), XCON (system for configuring VAX computers, 1978)
- Typical example of knowledge-based systems in the 80s

# Knowledge acquisition: non-automatic methods

- Interviews with domain experts
- (Extracting knowledge from a human is often called **knowledge elicitation**)
- Iterative process, hard to get right first time.
- Human experts usually find it very difficult to state all the data relevant for a given problem.

# Knowledge acquisition: automatic and semi-automatic methods (for expert systems)

- Programs which compile dependency networks during interviews with experts (MOLE, SALT)
- Programs using learning (META-DENDRAL)

# Using dependency networks to acquire knowledge

- MOLE (Elsheman, 1988) works for systems which classify cases as instances of fixed categories, such as a fixed number of possible diagnoses. It builds an inference network similar to belief networks we will see later in the module
- SALT (Marcus and McDermott, 1989) works for open-ended sets of solutions, such as design problems; builds a dependency network and compiles into a set of production rules.



# Using learning to acquire knowledge

- Learning decision diagrams from a set of positive and negative instances of a concept (e.g. when to approve a loan application)
- Learning rules from a set of positive and negative instances  
META-DENDRAL (Mitchell 1978) learned how to determine structure of complex chemical compounds

# Particular technique: decision tables

- A useful way of systematising knowledge preparatory to representing it using production rules
- can be compiled during interviews with experts or reading manuals or example sets

# Decision tables

- A **decision table** has the following structure:

Conditions	Decision rule
Condition stubs	Condition entries
Action stubs	Action entries

- where condition stubs are criteria relevant for a decision, action stubs are possible actions, condition entries are Y,N and - (should be true, should be false, not relevant) and action entries are X (for take this action) or blank.
- A decision rule is represented by a vertical column of condition and action entries.

# Example

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Else
cash	Y	Y	Y	N	N	
order > 100	Y	N	N	-	-	
order $\geq$ 50	Y	Y	N	-	-	
order < 50	N	N	Y	-	-	
credit record good	-	-	-	Y	N	
give 20% discount	X					
give 10 % discount		X				
accept order			X	X		
reject order					X	
exception report						X

## Example: rules

- Rule 1: **if** cash **and** order  $> 100$  **then** give 20% discount
- Rule 2: **if** cash **and**  $50 \leq \text{order} \leq 100$  **then** give 10 % discount
- Rule 3: **if** cash **and** order  $< 50$  **then** accept order
- Rule 4: **if not** cash **and** credit record good **then** accept order
- Rule 5: **if not** cash **and not** credit record good **then** reject order
- Else generate an exception report.

# Semantic web

- Aspiration: turn information available on the web into a huge knowledge base (integrated, readable and usable by machines ...)
- Formats for integration
- Languages for representing knowledge
- Ontology languages (description logics) in the following lecture

# Rule ML

- Rule Markup Language (RuleML): specifying Web interchange format for rules
- Motivation comes from various aspects of Semantic Web:
  - Rules marked up for e-commerce (business rules)
  - XML transformation rules
  - Rules used for declarative specification of web services
  - Intelligent agents using rules
- XML-like specification for each ruleset: rule conditions, rule conclusions, direction (backward, forward, bidirectional).

# Business rules

- A **business rule** is a statement that defines or constrains some aspect of the business
- Declarative, easy to modify; the idea is to separate dynamically changing rules which may apply for example only in the sales period from the application source code (for example on-line shop or rental business)
- Rules have a similar spirit to the discount example in the decision table
- Examples: IBM example of business rules:  
[https://www.ibm.com/support/knowledgecenter/en/SSPJVK/DigitalRecommendations/UserGuide/examplebusinessrules\\_nextgen.html](https://www.ibm.com/support/knowledgecenter/en/SSPJVK/DigitalRecommendations/UserGuide/examplebusinessrules_nextgen.html)



# Java Rule Engine API

- Java Rule Engine API (JSR-94) is a lightweight programming interface that constitutes a standard API for acquiring and using a rule engine.
- From the specification: 'Addresses the community need to reduce the cost associated with incorporating business logic within applications and the community need to reduce the cost associated with implementing platform-level business logic tools and services.'
- **javax.rules** and **javax.rules.admin** packages.

# Jess

- Jess is an expert system shell (can fill in your own rules, the engine already exists) written in Java
- Implemented using Rete algorithm (efficient incremental rule matching)
- Can be downloaded for free from <http://herzberg.ca.sandia.gov/jess> for educational use
- Rules can be specified in Jess rule language or XML; rule language is LISP-like:  

```
(defrule welcome-toddlers
  (person {age < 3})
  =>
  (println "Hello, little one!"))
```
- LHS is a pattern (if a person has age less than 3 years) and RHS is an action (function call, in particular can insert new facts).

# Exercise

- Construct a decision table for the following piece of Lenton local knowledge: *When a burglar alarm sounds, if it is in one of students' houses where alarm sounds every week, ignore it. Otherwise have a look outside and if the house looks not broken into and there is nobody moving inside it, ignore the alarm. Otherwise call police.*