Faculteit Wiskunde en Informatica Vrije Universiteit, Amsterdam Joris Hulstijn	**Re-Exam Knowledge-based Systems** Wednesday, 15th of August, 13.30 – 16.30

name: Joris
student number: 10 / 8 / 2001

Assessment:

Exercise 1	Exercise 2	Exercise 3	Exercise 4	Exercise 5	additional
10 +10	10 +10	10 +10	10 +10	10	10

Exercise 1: General

a. Applying knowledge-based systems in practice has *advantages* as well as *disadvantages*. Name at least four advantages and three disadvantages of the use of knowledge-based systems.

4 out of these 5 advantages (slide 1:10) gets you 5 pt

- **Larger distribution** and availability of expertise (in time, in location)
- (Relatively) **easy to modify**
- separation of knowledge & inference **Consistent results** **Retains Expertise**
  but: may be too rigid
- Solving problems with **incomplete or uncertain** data and knowledge
- **Explanation** of solutions

3 out of these 4 Disadvantages (slide 1:11) gets you 5pt

- **No guarantee on correctness** of results
- **Limited knowledge** without knowledge on the limitations
- lack of '**common sense knowledge**'
- **Ungraceful degradation** "breekbaar gedrag" brittle, performance cliff

b. Development of a knowledge-based system proceeds in a number of *phases*. Name the five development phases for knowledge-based systems. Describe the *end-product* of each phase, and their most characteristic *activities*.

See slices 2:27 – 2:30. There may be differences in terminology. However, using only the SE terminology is wrong. For each complete description 2pt; 1pt if aspects (end product / activities) are missing or confused. Note the testing!

phase	**end product**	**activities**
problem identification	problem identification document with requirements specification	determine vocabulary determine goals talks with clients
conceptualisation	conceptualisation document with domain model and task model	protocol analysis
formalisation	formal system specification architecture	determine types of knowledge, roles of knowledge, inference steps, model of search
implementation	implemented prototype	software engineering programming
evaluation (testing)	recommendations for re-design	user experiments verification

Exercise 2: Representation

a. There are roughly three types of spatial representation: *nearest first*, *Quadtree* and *hierarchical* representation. What are the distinguishing characteristics of these spatial representation types?

Nearest first: Divide space *uniformly* into regions.
For each object, represent: coordinates and other information (like color)
For each region, represent: objects in the region and neighbouring regions.
Objects must be of the some type. (at least at the description level)

Quadtree: The space is divided non-uniformly into regions. The distribution depends on object-density.
There is a special Addressing method, using quadrants: e.g. SE1.NE2.SE3
and there are useful navigation-operations: e.g. south(SE1.NE2.NE3) = SE1.NE2.SE3

Hierachical: use 'maps' with different levels of detail. While searching you can switch between maps. Levels can describe different types of objects. Using a recursive region representation like a Quadtree is not the same thing as having a hierarchical, multiple-level representation. The defining characteristic of a multi-level representation is that it represents different types of object at each level.

b. After your yearly check-up, the doctor has bad news and good news. The bad news is that you tested positive for some serious disease. The test is 99% accurate (i.e. the probability of testing positive given that you have the disease is 0.99, as is the probability of testing negative if you don't have the disease). The good news is that the disease is rare: it occurs for only 1 in every 10.000 people.
Why is this good news? In other words: what is the chance that you actually have the disease?

accuracy: P(test_pos | disease) = 0.99
P(test_neg | not disease) = 0.99, so P(test_pos | not disease) = 0.01
rare: P(disease) = 1 / 10.000

Use Bayes' rule (4pt)

$$P(\text{disease} \mid \text{test\_pos}) = \frac{P(\text{test\_pos} \mid \text{disease})\ P(\text{disease})}{P(\text{test\_pos})}$$

So we need P(test_pos). Use 'summation rule'. (4pt)

P(test_pos) = P(test_pos | disease) P(disease) + P(test_pos | not disease) P(not disease)
= 0.99* 1/10.000 + 0.01 * (1 -- 1/10.000) ~ 0.01

Fill in: (2pt)

$$P(\text{disease} \mid \text{test\_pos}) = \frac{P(\text{test\_pos} \mid \text{disease})\ P(\text{disease})}{P(\text{test\_pos})} = \frac{0.99 * 1/10.000}{0.01} \sim 0.0098$$

So even if you test positive, the chance that you have the disease is less than a percent due to the inaccuracy of the test. This is certainly good news.

Exercise 3: Classification and Data Abstraction

a. In classification raw data often first need to be combined and abstracted from. Name four types of *data abstraction*. Give an example of each type. (see slide 6:22)

- Qualitative abstraction (simplifies quantitative data)
  - 300 mm rain: wet
  - white blood-cell count (WBC) < 2500: low WBC
- Definitional abstraction
  - temperature > 37: fever temperature
  - low WBC: leukopenia condition
- Generalisation (superclass)
  - AI student: student
  - leukopenia condition: immunosuppressed condition
- Temporal abstraction (temp@T1 < temp@T2: rising temp)
- Anatomical abstraction

Consider the following classification scheme (fig 1). First raw data (R_i) are combined according to logical operators and transferred into abstract data (D_i), which are then used for hierarchical classification into a number of solution classes (S_i). Sub-classes with a common ‘parent’ are *mutually exclusive*.

Indicate for each of the following vectors of data, whether the solution classes ($S_{1\ ...}\ S_8$) are *matching*, *consistent* or *inconsistent* with the data.

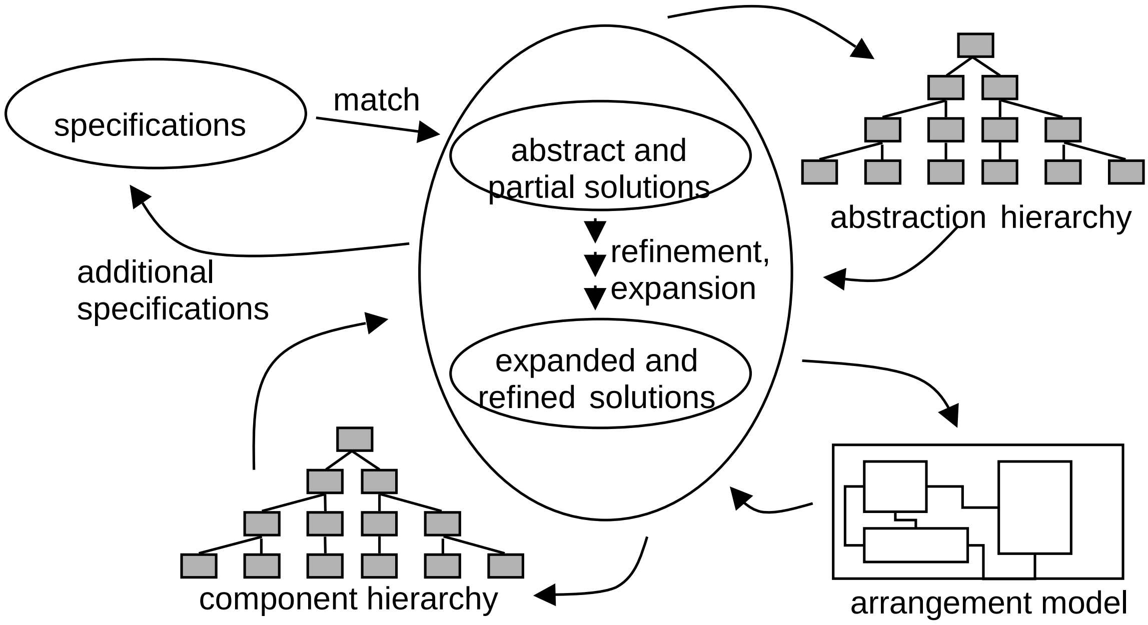
1. R = (? ? 0 1 1 ? ?), combine using $? \wedge ? = ?$, $? \vee \neg 0 = ? \vee 1 = 1$, $1 \wedge ? = ?$,
gives D = (1 1 ? ?), which produces

S1	1	(consistent and) match (because sub-class incons, only cons also OK)
S2	1	(consistent and) match
S3	11?	consistent
S4	11?	consistent
S5	11?	consistent
S6	10	inconsistent
S7	10?	inconsistent
S8	10?	inconsistent

2. R = (1 1 0 0 1 10), gives D = (1 0 10), which produces

S1	1	(consistent and) match (because sub-class incons, only cons also OK)
S2	10	inconsistent
S3	100	inconsistent
S4	100	inconsistent
S5	101	inconsistent
S6	11	(consistent and) match (because sub-class incons, only cons also OK)
S7	110	inconsistent
S8	111	(consistent and) match

Exercise 4: Scheduling



Making the daily work schedule for postmen and women is a complex task[1]. Postmen and women are assigned to a particular mail area. There are the following conditions. Every mail area must be done. Every postman or woman with a regular contract must be assigned some area. The other areas are divided among so called `temporary employees'. Postmen and women specialise in their 'own area'; in this way the job is done quicker and better. One area may have several such 'specialists'.

a. Explain that making a mail area schedule can be seen as an instance of a configuration task. Make use of a drawing to sketch how the specification and configuration space, and the various knowledge sources and models are filled in.

Use picture above (2 pt for general sketch) Each of the components can be filled in by data from the PTT example. In this way you can 'map' the scheduling problem onto the configuration problem. (2 pts for each component in the picture 'fits' the PTT problem; 1-4 crucial, one of the three at 5 is enough)

1. Specifications:
Use 'slot-filling' technique, where an area is a slot (functional component) with certain requirements. A postman or woman is a part, also with requirements that specify a 'good' solution.
2. Parts: (Finite sets of) areas (functional components) and postmen (actual parts)
3. Combinations: Mapping from area to postmen, such that requirements are met.
4. Configuration space:
Begin with general solution: All areas have to be done by some postman and woman.
End with detailed solution: Area 1081 is done by postman Jansen, ...
Multiple solutions are possible. Use algorithm of question b.

---

[1] Thanks to Marije Geldof, Ruud Schellekens, Nanja Smets en Gertjan Wijnalda for the example.

5. Abstraction model/hierarchy: areas with specialists
Arrangement model: areas with following requirements
1. All areas have to be done by postmen
2. Every postman with a regular contract must be assigned some area.
2. Area is done by specialist (preferred) or other.
Component model/hierarchy: employees with regular contract; temporary employees

b. In practice, making a mail area schedule proceeds in a number of steps. No backtracking is needed between steps.

Argue whether this configuration method is an example of MCF1, MCF2 of MCF3.

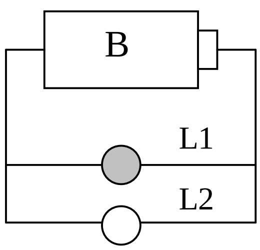
The mentioned configuration method is an example of MCF2. The four steps can be seen as ordered subtasks. For the selection of the subtasks domain-specific knowledge was used that look ahead in the configuration process to avoid the use of backtracking.

Exercise 5: Diagnosis

Consider the following electric circuit, with a battery B and two lights L1 and L2 in parallel. We predict that L1 and L2 are both on. However, we observe that L1 is off, but that L2 is on! We use the GDE method, in which there are no specific fault models.

a. What are the conflict sets for these observations?
b. According to the GDE method, what are the minimal covering sets of broken parts that would explain these observations?
c. Is this outcome in accordance with the expected behaviour of batteries?
d. Would a specific fault model for batteries change this outcome? If so, how? If not, why not?

a. Conflict set: <B, L1>
b. Minimal covering sets: <B>, <L1>
c. No, this solution is not in accordance with the expected behavior of batteries. It is silly for the battery to be a candidate. Either a battery is charged or it is not. If the battery is discharged (faulted), then there is no way it could still provide power to L2 without also powering L1.
d. Yes, a fault model for batteries could change the candidates. For example:
Battery: empty -> no current anywhere
Lamp: gets current -> no light



Observations:

L1 off

L2 on