Faculteit der Exacte Wetenschappen	Tentamen Ontwerp van Multi-agentsystemen
Vrije Universiteit Amsterdam	1 april 1999

Opgave	**1**	**2**	**3**	**4**	**bonus**
Punten	15	25	30	30	10

Normering:
Het **tentamencijfer** T is gelijk aan (het totaal behaalde punten voor de tentamenopgaven plus 10 punten bonus) gedeeld door 11.

Het **eindcijfer** voor het hoorcollege Ontwerp van Multi-agentsystemen wordt als volgt berekend:

Eindcijfer = (T + H + P) / 3,

waarbij
T = tentamencijfer,
H = cijfer huiswerkopgaven,
P = cijfer voor 1 punts practisch werk.

Opgave 1 (15 punten)

Deze opgave bestaat uit twee onderdelen. Motiveer Uw antwoorden. Deze opgave gaat over het spel Memory (zie Appendix 1).

Opgave 1a (5 punten)

Beschrijf in het kort in je eigen woorden hoe jij graag zou willen dat een software agent het spel Memory speelt.

Opgave 1b (10 punten)

In het generieke agentmodel (hoofdstuk 6 van de syllabus) komen een aantal componenten voor. Stel dat je het generieke agentmodel zou gebruiken om een artificiële agent te ontwerpen die het spel Memory zou kunnen spelen. Gebruik je antwoord op vraag 1a om te beslissen welke van deze componenten je nodig hebt in je ontwerp. Motiveer waarvoor een component nodig is, motiveer ook waarom een component eventueel niet nodig is.

Opgave 2 (25 punten)

Deze opgave bestaat uit drie onderdelen. Bestudeer het probleem van de Torens van Hanoi (zie Appendix 2).

Opgave 2a (5 punten)

Formuleer de requirements voor dit probleem.

Opgave 2b (10 punten)

Formaliseer de kennis die door de agent gebruikt wordt in een kennisbank. Je mag je beperken tot de kennisregels die in de Appendix reeds informeel zijn opgeschreven, je mag ook juist de ontbrekende kennis opschrijven.

Opgave 2c (10 punten)

Geef specificaties van informatietypen voor de kennis die je in opgave 2b geformaliseerd hebt.

Opgave 3 (30 punten)

Deze opgave bestaat uit drie onderdelen. Het onderwerp van deze opgave betreft informatie toestanden. Bestudeer de partiële specificatie van Appendix 3. Ga uit van de volgende twee informatietoestanden S1 en S2 van de component mouse_a.

```
S1 = [ observation_result(at_position(self, p0), pos),
       observation_result(at_position(food, p1), pos),
       observation_result(at_position(screen, p0), neg),
       observation_result(at_position(food, p0), pos) ]

S2 = [ to_be_performed(goto(p0)) ]
```

Opgave 3a (10 punten)

Geef een volledige verfijning (complete refinement) van de publieke **input** informatietoestand S1 voor component mouse_a. (Je hoeft je dus niet bezig te houden met de **out**put van de component.) Note: je mag gebruik maken van afkortingen, en doe het tabelsgewijs, zodat het een snelle invul oefening wordt.

Opgave 3b (10 punten)

Geef een coherent levelled public **output** information state van component mouse_a, ga daarbij uit van informatietoestand S2. (Je hoeft je dus niet bezig te houden met de **in**put van de component.)

Opgave 3c (10 punten)

Geef een object level information state die S1 verfijnt en bovendien consistent en gesloten (refines, consistent and closed) ten opzichte van de kennisbank van component mouse_a.

Opgave 4 (30 punten)

Deze opgave bestaat uit drie onderdelen. In deze opgave wordt U gevraagd een specialisatie te maken van een bestaand model (zie Appendix 4) voor het monitoren van de bakkers van een koekjesfabriek.

Opgave 4a (10 punten)

Formaliseer de kennis die door het systeem gebruikt wordt in kennisbanken en geef aan welke component welke kennisbank gebruikt. Motiveer Uw antwoord.

Opgave 4b (10 punten)

Geef specificaties van de informatietypen voor de kennis die je in opgave 4a geformaliseerd hebt. (Hint: zoek uit welke domein specifieke informatietypes wel in Appendix 4 genoemd worden, maar daar niet zijn gespecificeerd.)

Opgave 4c (10 punten)

Geef een voorbeeld trace van het systeem, waarin voor de bakker in de loop van de tijd eerst de manager moet worden gewaarschuwd en later een ambulance moet worden geroepen. Geef duidelijk aan hoe de informatie toestanden elkaar opvolgen en binnen welke component(en) veranderingen optreden.

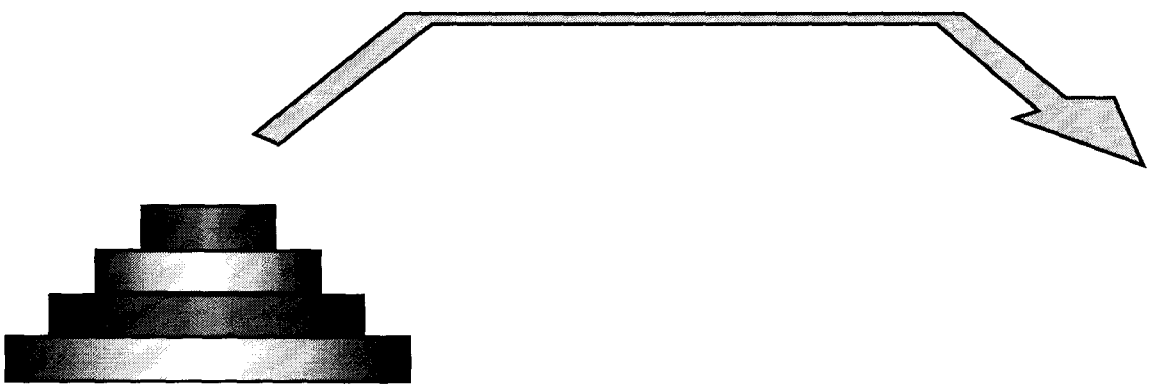
Appendix 1 Memory

Memory is a card game with an even number of cards. For every card in the deck there is exactly one other card in the deck that has the same picture on it. The cards are spread out face down on the table. At least two players participate in the game. During his turn, the player is allowed to to subsequently look at two cards. If the second card has the same picture as the first card, both cards are removed from the gave, the player gets a point, and the player can again look at two cards, etcetera. If the second card differs from the first, the player's turn ends and the next player's turn begins.

Although these rules are simple, the game requires a good memory, and a good strategy in order to score well.

Appendix 2 Torens van Hanoi

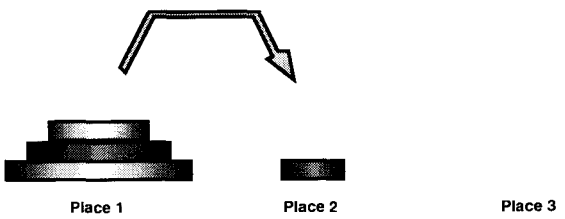
An agent is needed that is capable of controlling the execution of a solution for the problem of the towers of Hanoi for towers of hight four. A tower of Hanoi is a tower of blocks of increasing size (seen from the top down). The goal of the exercise is to move the tower from place 1 to place 3. There are four blocks, the smallest one is called block a, the next one is called block b, the next is block c, and the largest block is block d. The agent can only move one block at a time, that block has to be on top, and a block can only be placed on the ground or on top of a larger block.



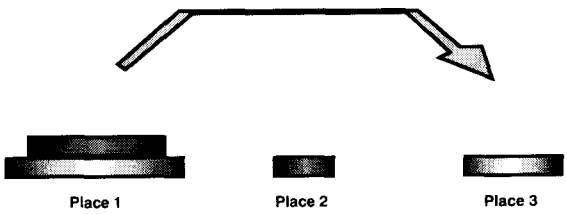
Place 1 Place 2 Place 3

Furthermore, blocks may only be placed on place 1, place 2, or place 3. Part of the knowledge used by the agent is the following:

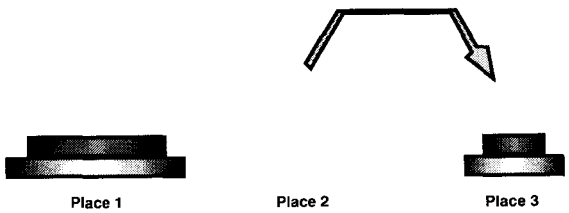
1. If I observe that all blocks are at place 1,
   then I decide to move block a to place 2.



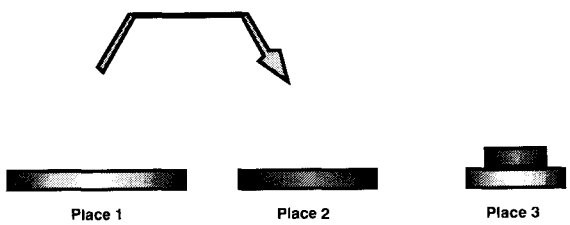
2. If I observe that blocks b, c, and d are at place 1,
   and I observe that block a is at place 2,
   then I decide to move block b to place 3.



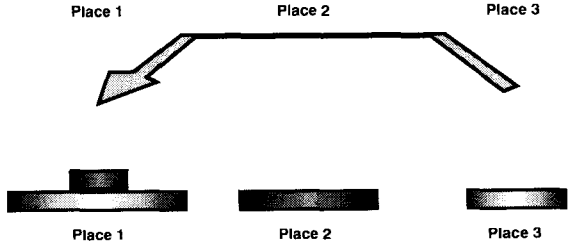
3. If I observe that blocks c and d are at place 1,
   and block a is at place 2
   and block b is at place 3
   then I decide to move block a to place 3.



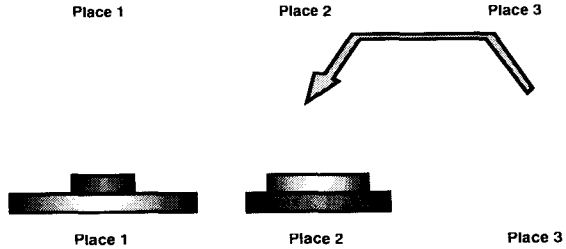
4. If I observe that blocks c and d are at place 1,
   and blocks a and b are at place 3
   then I decide to move block c to place 2.



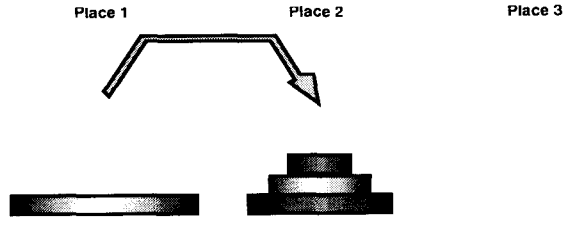
5. If I observe that block d is at place 1,
   and blocks a and b are at place 3
   and block c is at place 2
   then I decide to move block a to place 1.



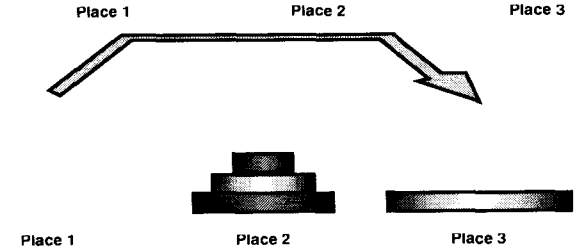
6. If I observe that blocks a and d is at place 1,
   and block b is at place 3
   and block c is at place 2
   then I decide to move block b to place 2.



7. If I observe that blocks a and d are at place 1,
   and blocks b and c are at place 2
   then I decide to move block a to place 2.



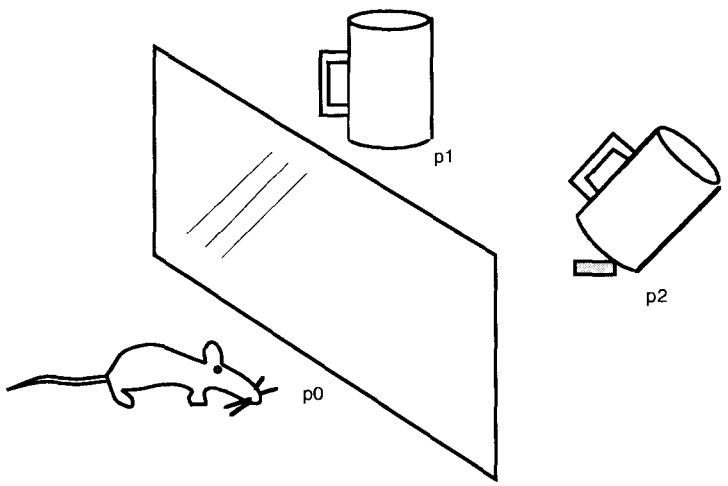
8. If I observe that block d is at place 1,
   and blocks a, b and c are at place 2
   then I decide to move block d to place 3.



Appendix 3 Mouse A

3.1 Problem Description

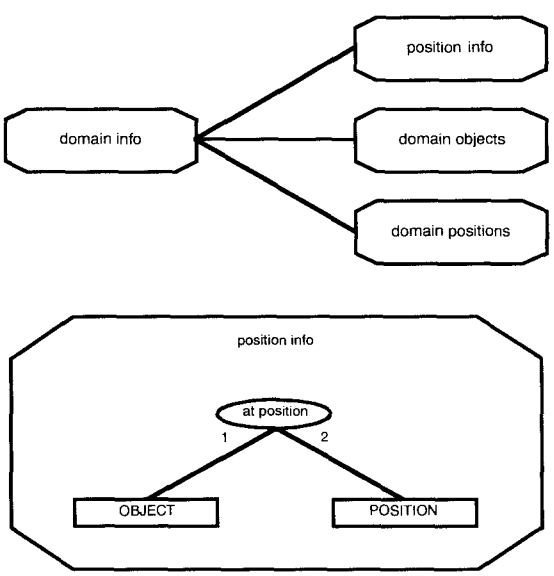
Separated by a transparent screen (a window, at position p0), at each of two positions p1 and p2, a cup (upside down) and/or a piece of food can be placed. At some moment (with variable delay) the screen is raised, and the mouse is free to go to any position. A genuine mouse is known to go to food and eat it.

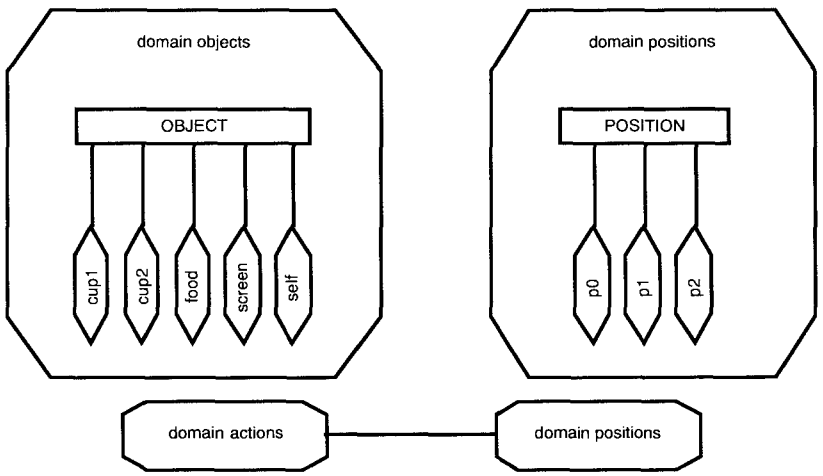


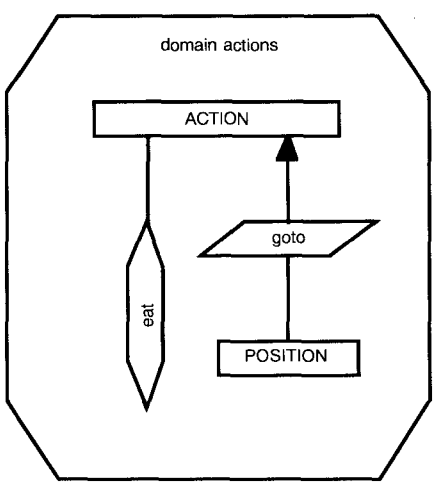
In sections 3.2 and 3.3 of this Appendix a partial specification can be found of the artificial mouse.

3.2 Informatie typen

De informatie typen die in het hele systeem gebruikt worden zijn:







information type truth_indication
sorts SIGN ;
objects pos,
neg : SIGN;
end information type

information type observation_results
sorts INFO_ELEMENT, SIGN ;
relations observation_result: INFO_ELEMENT * SIGN;
end information type

information type domain_meta_info
sorts INFO_ELEMENT ;
meta-descriptions domain_info : INFO_ELEMENT;
end information type

```
information type observation_result_info
        information types     truth_indication,
                              observation_results,
                              domain_meta_info;
end information type

information type actions_to_be_performed
        sorts                                 ACTION ;
        relations       to_be_performed:      ACTION;
end information type

information type action_info
        information types     actions_to_be_performed,
                              domain_actions;
end information type
```

3.3 Fragmenten van specificatie van de component

De component is primitief en wordt hier kort beschreven.

De component mouse_a

De interfaces worden gedefiniëerd door:
input interface: de informatietypen observation_result_info;
output interface: het informatietype action_info;

De targets bij task control focus determine_action zijn:
target(determine_action, to_be_performed(X: ACTION), confirm);

De initial extent is: all-p

De kennisbank is:

```
if      observation_result(at_position(food, P:POSITION), pos)
  and   observation_result(at_position(screen, p0), neg)
  and   observation_result(at_position(self, P:POSITION), neg)
then    to_be_performed(goto(P:POSITION))

if      observation_result(at_position(self, P:POSITION), pos)
  and   observation_result(at_position(food, P:POSITION), pos)
then    to_be_performed(eat)
```

Appendix 4 Koekenbakker

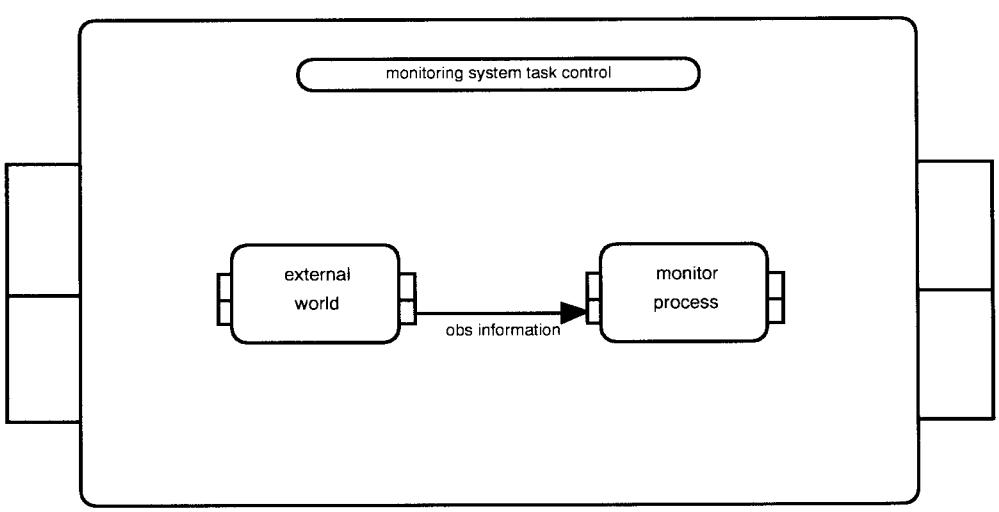
Een grote fabrikant van speculaasjes klaagt dat veel beginnende bakkers meer speculaas eten dan bakken. Naast dat dit gedrag niet bevordelijk is voor de productiecijfers heeft het een ander ongewenst effect: soms moeten de van buikpijn krimpende bakkers met een ambulance worden afgevoerd om hun maag te laten leegpompen.

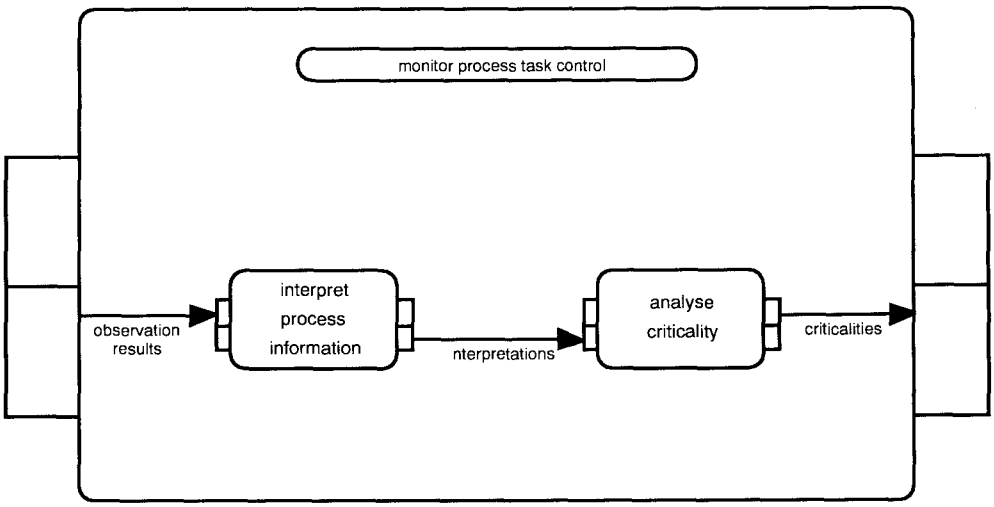
De fabrikant wil graag een software systeem laten ontwerpen die in de gaten houdt hoe de gezondheidstoestand van de bakkers is. De taak van het systeem lijkt sterk op de taak van het proces monitor_process zoals dat in de hoofdstukken 8 en 9 is beschreven. Het systeem kan gebruik maken van de gegevens van een aantal sensoren. De binnenkomende informatie betreft de houding van de bakker en de geluiden die zij voorbrengt: de bakker kan liggen, rechtop staan of krom staan. De bakker kan stil zijn, kreunen of fluiten. Op grond van de interpretaties moet het systeem analyseren of de toestand van de bakker kritiek is.

Het systeem moet eerst de signalen en de houding van de bakker interpreteren: als hij kreunt, krom staat of ligt, dan heeft hij buikpijn, als hij fluit voelt hij zich goed, als hij ligt en stil is, dan is hij in een coma.

Op grond van deze interpretatie moet het systeem een beslissing nemen of de toestand van de bakker zo erg is dat de agent een ambulance moet bellen, of dat de toestand wel inhoudt dat de manager geroepen wordt.

Omdat deze taak sterk lijkt op de taak van het proces monitor_process zoals dat in de hoofdstukken 8 en 9 is beschreven, volgt hieronder een partiële specificatie van deze component.





Informatietypen

De generieke informatie typen die in het hele systeem gebruikt worden zijn:

```
information type  domain_info
        information  types
        ......
end  information  type

information type  domain_meta_info
        sorts  INFO_ELEMENT
        meta-descriptions     domain_info     :          INFO_ELEMENT
end  information  type

information type  truth_indication
        sorts  SIGN
        objects          pos, neg : SIGN
end  information  type

information type  observation_results
        sorts  INFO_ELEMENT
        relations          observation_result : INFO_ELEMENT * SIGN
end  information  type

information type  observation_result_info
        information  types
                observation_results,
                truth_indication,
                domain_meta_info
end  information  type
```

Information links

De informatie links die in het hele systeem gebruikt worden zijn:

```
private link obs_information : object-object;
        domain external_world
                level EW_object_level
                        information types      observation_result_info ;

        co-domain monitor_process
                level MP_object_level
                        information types      observation_result_info;
        identity
end link

private link interpretations : object-object;
        domain interpret_process_information
                level IPI_object_level
                        information types      interpretations ;
        co-domain analyse_criticality
                level AC_object_level
                        information types      interpretations;
        atom links
....
end link

mediating link observation_results : object-assumption;
        domain monitor_process
                level MP_object_level
                        information types      observation_result_info;
        co-domain interpret_process_information
                level IPI_meta_level
                        information types      assumption_domain_info;
        sort links
                (INFO_ELEMENT, IA)
                rest identity
        atom links
                ( observation_result(E:INFO_ELEMENT, S:SIGN),
                        assumption(E:IA, S:SIGN)          ) :
                        < <true, true>, <false, false> >
end link
```

```
mediating link criticalities : epistemic-object;
        domain  analyse_criticality
                level  AC_meta_level
                        information  types      epistemic_criticality ;
        co-domain  monitor_process
                level  MP_object_level
                        information  types      alert ;
        sort  links
                (OA, CRIT_INFO_ELEMENT)
        atom  links
        ..... .
end  link
```

Interface Informatietypen van de verschillende componenten

Component	Input informatietype	Output informatietype
External World		observation result info
Monitor process	observation result info	alert
Interpret process information	domain info	interpretations
Analyse criticality	interpretations	criticality