
Production Systems and Information Engineering
Volume 5 (2009), pp. 51-66.

DEVELOPING MODELS BASED ON REAL ENVIRONMENTS

TAMÁS BÁKAI
University of Miskolc, Hungary
Regional University Knowledge Centre
retbakai@gold.uni-miskolc.hu

[Received January 2009 and accepted April 2009]

Abstract. All of the applications of information engineering are based on a correct model of the environment the applications represent. Unlike artificial environments, real environments cannot be modelled correctly. Firstly because the behaviour of the real environments also depends on the behaviour of other environments also. These relations cannot be revealed at the time the model is constructed. Therefore the boundaries of a real environment cannot be defined correctly. The only thing the model-designer can do is to define a model describing the behaviour of the environment to be modelled with no inconsistencies at the time the model is constructed. Therefore each model based on real environments needs to be redesigned continuously as the time passes to provide their consistency. It is only feasible using adaptive knowledge-intensive modelling tools. This paper shows a new concept for modelling real environment.

Keywords: system identification, behaviour description, knowledge-intensive modelling tool

1. Problem declaration

Nowadays more and more fields of our life are supported by applications of information engineering. These applications have to work not only in artificial environments, but in real environments as well. The main difference between artificial and real environments based on the boundaries of environment needs to be modelled. Both the boundaries and the behaviour of an artificial environment are defined by the model designer only. The object structure and the set of rules describing the behaviour of these environments can be defined at the time the model is constructed. Some models designed for working in artificial environments can learn rules and can extend their knowledge-base with those rules, but cannot invalidate a rule defined by the model designer. It is because the behaviour of artificial environments does not change. In artificial environments the set of rules is finite and therefore the behaviour of these environments can be transformed into a deterministic model easily. Unfortunately the applications based on artificial environments cannot work in real environments correctly. It is because the object

structure and the set of rules representing the behaviour of a real environment cannot be determined at the time the model is constructed. The behaviour of a real environment depends on the behaviour of other real environments also and these dependencies cannot be revealed correctly until they appear. Therefore the set of rules of a real environment cannot be described correctly. The only thing the model designer can do is to define a model which represents the correct behaviour of the real environment needed to be modelled at the time the model is constructed. Besides the indefinable boundaries the modelling of real environments also suffers from the complexity of their state space. For revealing the problem based on the complexity of real environments let us consider an environment with *n* pieces of state descriptor variables. If each of these variables has only two values, then the state space of this environment has 2^n independent states. The rules representing the behaviour of the real environment need to be revealed in such a large state-space. In the worst case it means 2^n pieces of rules for describing the behaviour of one of the state descriptor variables of the environment. For describing the behaviour of the whole environment $n*2^n$ pieces of rules have to be modelled. Table 1 shows an example of an environment with three state descriptor variables, each with two values.

Table 1. Complexity of the behaviour description of an environment containing three state descriptor variables (A, B, C), each with two values

conditions			conclusion
A	B	C	A
0	0	0	?
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

a,

conditions			conclusion
A	B	C	B
0	0	0	?
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

b,

conditions			conclusion
A	B	C	C
0	0	0	?
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

c,

Usually the state descriptor variables have more than two values. In the general case the complexity of behaviour description can be calculated by multipling the number of the values of each state descriptor variable of that environment and multipling the result by the number of the state descriptor variables of the environment. For example if the environment consists of three state descriptor variables with three, two and four values, then the complexity of the behaviour description of this environment is (3*2*4)*3 = 72 pieces of rules in the worst case. For example a medium sized environment has some hundreds of state description variables. The behaviour revealing cannot be modelled with the common

algorithms in so large a state-space. Artificial environments with large state spaces (like game of chess) can be grasped by common algorithms because both the object structure and the set of rules of these environments are determined by the model designer only. Therefore the large state space of these environments cannot be revealed to find the necessary rules.

If the application is required to work in a real environment then the object structure and the set of rules of the environment need to be revealed. Unfortunately, these features cannot be determined correctly because of the complexity of the real environments and their dependencies on other environments. Therefore the real environment cannot be transformed into a deterministic model correctly. At this point I need to mention that theoretically there is a deterministic real environment (see [2]). Its state space consists of all of the elementary particles in the universe. Probably information engineering will never handle this complex environment so the real environments can be held as these would be stochastic. It means that the state of a real environment at time *t* cannot be determined by the state of that environment at time *t-1* and the set of rules of that environment only. The most the designer can do in order to eliminate the inconsistencies is to widen the boundaries of the real environment needed to be modelled until the environment behaves if it were deterministic. By this the behaviour of that environment at time *t* can be described correctly. To provide the consistency of the model as the time passes the model is needed to be redefined continuously. It is only possible using knowledge-intensive learning concept. This paper shows a new concept for solving this problem efficiently.

2. Modelling the object structure of the environment

The object structure of an environment can be modelled in several ways. This paper uses the most common and popular object-oriented designing concept [3]. According to this concept the object-structure of the environment needed to be modelled is described by object-attribute-value triple. Each object in this model represents a set of properties belonging to an elementary unit of the environment. For example in the game of chess it can be a figure on the chess-board. Each property belongs to an object named the attribute of that object. Each attribute represents an observable property in the environment. An attribute of the figure on the chess-board is its position. Each attribute has some values. While the objects and the attributes are virtual elements in the object-structure (whose only rule is to categorize the observable elements) a value can be observed by a sensor. For example a value can be a colour value (black, white, red, ...), a temperature value (0°C, 20°C, ...) or in case of the position attribute of the figures on the chess-board it can be a coordinate point. Each attribute has one marked value among it values at any time. This value represents the state of the attribute it belongs to at that time. The set of the active values of each attribute in the environment at a given time represents the state of the environment at that time. As the time passes a marked

value may lose its marked state and an unmarked value may become marked, but at any time each attribute has one and only one marked value. This process takes place at discrete time periods determined by the sampling frequency of the modelling tool only. Each sample is a state of the environment. Two consecutive samples give a change of states of the environment.

Some of the objects of an environment can do activities. For example a figure on the chess-board can move. An object which can do an activity is called an agent. The activities are similar to attributes from the point of view of the description of the states of the environment. The state of the environment at time t is given not only by the state of the attributes of the environment at time t but by the state of the activities at that time also. The attributes and the activities represent the dimensions of the environment. The values of the dimensions represent the points that can be reached along these dimensions. Each activity has two values. One for representing the fact the activity is done and one for representing the fact the activity is not done. The main difference between the attributes and the activities is that the values of the activities cannot be observed. Each agent knows the state of its activities but cannot observe the state of the activities of another agent. Therefore at first approach each agent considers the objects of the environment as these would be agents with no activities. The state of the activities can be deduced from their effect on the state of the attributes of the environment.

The designer tool in this paper is modelled as an intelligent agent trying to reveal the behaviour of the real environment needed to be modelled. The designer tool may have activities to interact with the environment it reveals. Using these activities the agent can lead the environment into states that have not observed yet. This possibility allows the agent to reveal new parts of the environment. The designer tool is modelled with one agent (containing all of the activities the designer tool can do) and a set of objects (representing the sensors with which the designer tool can observe the environment). Unlike this, the environment is modelled with a set of agents without objects. It is because any of the objects of the real environment may have activities.

Figure 1 shows the object structure of an environment with two objects. The first one is an agent which represents the inner properties of the developing tool and the second one is an object in the environment. The agent has a sensor called $A_1.At_1$ for describing the state of the agent, and two activities called $A_1.Ac_1$ and $A_1.Ac_2$ for describing the activities the agent can do. The developing-tool has a sensor called $O_1.At_1$ for describing the state of the environment. Both of the attributes in the object structure have two values.

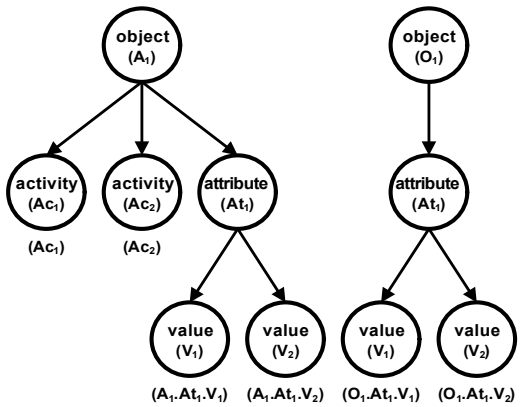


Figure 1. Object structure of an environment with two objects

In general the attributes of a real environment have many values. For example if one of the sensors of the designer tool is a thermometer then it can be modelled with an attribute. If this thermometer can measure the temperature in the range [-20...40°C] with the precision of 0.1°C, then that attribute has 600 values. The great amount of values enlarges the state space of the environment significantly, therefore makes the designer tool unworkable. To handle this problem only the values that have become active values are modelled. It decreases the complexity of the object structure considerably but does not make the model incorrect. The values which never become active are not existing values of the attribute.

The designer tool records the state of the environment to be modelled at each discrete time period according to a sampling rate. The resulting sequence is the history of the environment. The behaviour of a real environment can be revealed correctly using a history with infinite pieces of records. In practice it is not possible, but in general the more pieces of records there are in the history, the more correct the model. On the other hand, increasing the sampling frequency increases the correctness of the model as well. Each of these possibilities increases the size of the history significantly. One of the greatest problems the designer must eliminate in modelling a real environment is the great amount of information needed to be handled. The continuously growing number of records in the history requires storage and processing units with continuously growing capacity. For modelling the values that have become active at least once can decrease the amount of information needed to be stored in the history but in case of real environments it is not enough. The solution of this problem is oblivion. Each adaptive knowledge-intensive system needs oblivion to follow the changes in the behaviour of the real environment it models. For example imagine a railroad schedule. After it has changed, the old version has to be forgotten in order for the designer-tool to work properly. The history with oblivion in this paper is modelled as follows. Each value records a timestamp each time it becomes the active-value of the attribute it belongs to. This time-stamp is called satisfaction. Each satisfaction represents the time it occurred, the duration it is memorable in the short-term memory and the duration it is memorable in the history. The duration a satisfaction is memorable in

the history is greater than or equal to the duration that satisfaction is memorable in the short-term memory. A satisfaction is removed from the history if the duration it is memorable in the history expires. If none of the satisfactions of a value is memorable in the short-term memory, it removes all of the satisfactions of that value from the history. This rule models the oblivion process of intelligent beings. According to this if an event occurs frequently, then the occurrences of this event in the past are memorable for a long time, but if an event does not occur for a while, then all of the occurrences of this event become forgotten. The developing tool can handle different durations for each satisfaction. Therefore the oblivion of intelligent beings can be modelled with higher precision. For determining these durations is a developing possibility for the future. At this point constant durations are determined for all of the satisfactions of the values. A value which loses all of its satisfactions will be removed from the object structure. Similarly an attribute which loses all of its values will be removed from the object structure. An activity does not lose any of its values but if it is not done for a while (defined by the short-term memory) then that activity is removed from the object structure. An object or an agent which loses all of its dimensions will be removed from the object structure. Modelling of the values that have became active at least once only and oblivion extend the abilities of the designer tool to model real environments.

3. Revealing the behaviour of the environment

3.1. The concept of the revealing behaviour

In case of deterministic environments the state of the environment at time *t* can be calculated by the state of that environment at time *t-1*. Therefore the set of rules that describes the behaviour of a deterministic environment can be revealed correctly. For example a wrist-watch is a deterministic environment because its state at time *t* is determined unambiguously by its state at time *t-1*. No matter how complex a deterministic environment is there is a possibility (at least theoretically) to reveal the set of rules that represents the behaviour of that environment correctly. Unlike this, the state of a stochastic environment at time *t* cannot be calculated by its state at time *t-1* only. It is because stochastic environments are partly observable only. It means that there is always a set of state descriptor variables which produce an effect on the environment but are not modelled. For example the traffic lights are a stochastic environment from the point of view of the traffic. If its state at time *t* is red then its state at time *t+1* can be red or green equally. Because the traffic lights are an artificial environment (that is its set of rules is defined only by its designer) then its stochastic behaviour can be transformed into deterministic unambiguously. For example a counter can be put beside the traffic-lights which counts downwards until zero. The zero induces the switch of colours. Therefore if the boundaries of the traffic lights environment are extended by this counter (as a new state descriptor variable), then the state of the traffic lights at time *t* can be calculated by its state at time *t-1* correctly.

Unfortunately real environments remain always stochastic. The most the designer can do is to determine the boundaries of the environment needed to be modelled with which the behaviour of the real environment can be described as if it were deterministic up to the actual date. Unfortunately the rules that describe the behaviour of a real environment up to time *t* correctly may become incorrect at time *t+1*. In real environments no one can guarantee that the rules revealed are correct. The most that can be guaranteed is that the rules revealed describe the behaviour of the environment up to the actual date correctly. Intelligent beings trying to reveal the behaviour of the environment around them work on this basis. We cannot be sure that our equations that describe the environment around us are correct. Each time we encounter a contradiction modify the equations to describe the environment correctly up to the actual date.

Therefore if according to experience up to time *t* each time the sun shines a rainbow can be seen then the following rule that describes the visibility of the rainbow is correct.

IF the sun shines **THEN** a rainbow can be seen. (3.1)

According to our experience we know that this rule is not correct. It is because we have a large amount of experience in our history. Our experience makes us to find a more correct formula but no one can guarantee that our formula is correct and that then cannot come new experience bringing contradiction. If new experience contains a sun shining and a rainbow cannot be found in the sky then the previous rule has to be reconstructed. For this the agent tries to extend the condition part of the rule with one state descriptor variable it can observe. For example if the agent has a thermometer then it tries to use the temperature to eliminate the contradiction of the rule. If the history contains that each time the sun shone and the temperature was lower than 20°C and the rainbow was visible then the following rule can be constructed.

IF the sun shines **AND** the temperature is lower than 20°C **THEN** a rainbow can be seen. (3.2)

Taking this with each state descriptor variable the agent can observe, the agent creates a set of new rules which describe the visibility of the rainbow correctly. If the agent has a sensor for observing the rain then maybe one of these rules is the following.

IF the sun shines **AND** it rains **THEN** a rainbow can be seen. (3.3)

If the agent does not have a sensor for observing the rain then it is possible that the agent cannot find another sensor among the sensors the condition part of the inconsistent rule is extended with that rule becoming correct. In this case the agent tries to find two sensors to extend the condition part of the incorrect rule. If it is

possible then the problem is solved up to the actual date, but if it is not possible then a state of more and more sensors is needed to be taken in the condition part of the rule. Following this concept some problems arise:

- the states of all of the sensors the agent has are in the condition part of the rule but the contradiction cannot be eliminated
- the contradiction of the incorrect rule is eliminated successfully but there are numerous state descriptor variables in the condition part of the rule.

The solution of these problems implemented in the designer tool will be shown in Section 4 of the paper.

3.2. Using initial knowledge in learning systems

Implementing initial knowledge into learning-applications is popular practice today. It makes the application more efficient in revealing the behaviour of the environment because it protects it from learning numerous misleading rules. In spite of this the developing tool described in the paper avoids applying initial knowledge. It is because the application designer is an agent too whose knowledge about the real environment cannot be correct. The initial knowledge describes some of the rules of the real environment and these rules describe the behaviour of the environment at the time the application is constructed only. Therefore it helps the application in the near future only. As time passes the initial rules become more and more inconsistent and these inconsistencies make the application unable to work.

3.3. Methods implemented methods for revealing the behaviour

This section shows the method implemented in the developing tool for revealing the behaviour of real environments. For this let us consider the object structure shown in Figure 1. This object structure contains two values for each dimension. It is the object structure of one of the simplest models that can be constructed and it is to demonstrate the learning process of the developing tool in the simplest way only. The concept shown is applicable for handling attributes with more than two values also. As Figure 1 shows, this environment has four independent states. These are [$A_1.At_1.V_1$, $O_1.At_1.V_1$], [$A_1.At_1.V_1$, $O_1.At_1.V_2$], [$A_1.At_1.V_2$, $O_1.At_1.V_1$], [$A_1.At_1.V_2$, $O_1.At_1.V_2$]. The agent in this environment can do four activities. These are [nothing], [Ac_1], [Ac_2], [Ac_1 and Ac_2]. Let the behaviour of the environment the developing-tool has to reveal be as follows:

- Ac_1 changes the state of the attribute [$A_1.At_1$]
- Ac_2 changes the state of the attribute [$O_1.At_1$]

Let the state of the environment at the beginning of the investigation is [$A_1.At_1.V_1$, $O_1.At_1.V_1$]. Figure 2 shows the time sequence of the changes of states in the environment the developing tool will observe.

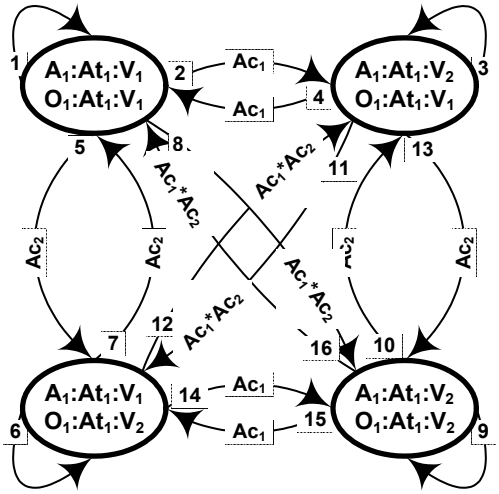
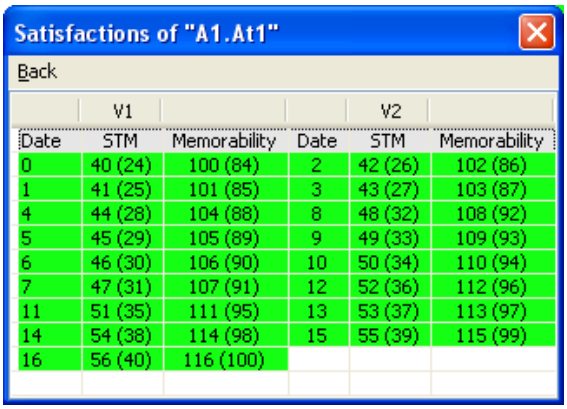


Figure 2. Time sequence of the changes of states in the environment the developing tool will observe

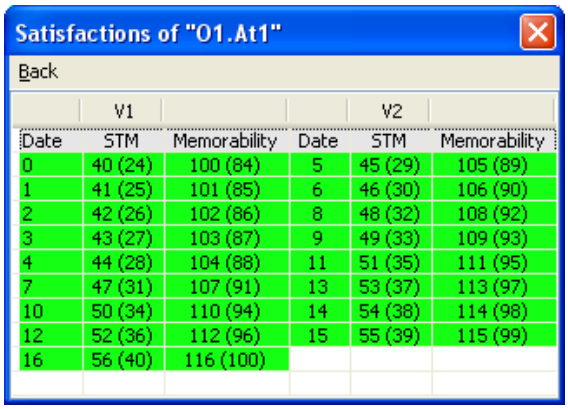
In Figure 2 the four ellipses represent the states of the environment and the arrows represent the activities the agent does. At the middle of each arrow the name of the activity - that the arrow represents - can be seen. An arrow with no activity name represents the null activity, that is no activity is done during that change of states. The number at the beginning of each arrow represents the serial number of the change of states according to the time sequence. Following the sequence of these changes of states the values in the object structure of the environment are satisfied as shown in Figure 3.



Satisfactions of "A1.At1"

Back

	V1			V2	
Date	STM	Memorability	Date	STM	Memorability
0	40 (24)	100 (84)	2	42 (26)	102 (86)
1	41 (25)	101 (85)	3	43 (27)	103 (87)
4	44 (28)	104 (88)	8	48 (32)	108 (92)
5	45 (29)	105 (89)	9	49 (33)	109 (93)
6	46 (30)	106 (90)	10	50 (34)	110 (94)
7	47 (31)	107 (91)	12	52 (36)	112 (96)
11	51 (35)	111 (95)	13	53 (37)	113 (97)
14	54 (38)	114 (98)	15	55 (39)	115 (99)
16	56 (40)	116 (100)			



Satisfactions of "O1.At1"

Back

	V1			V2	
Date	STM	Memorability	Date	STM	Memorability
0	40 (24)	100 (84)	5	45 (29)	105 (89)
1	41 (25)	101 (85)	6	46 (30)	106 (90)
2	42 (26)	102 (86)	8	48 (32)	108 (92)
3	43 (27)	103 (87)	9	49 (33)	109 (93)
4	44 (28)	104 (88)	11	51 (35)	111 (95)
7	47 (31)	107 (91)	13	53 (37)	113 (97)
10	50 (34)	110 (94)	14	54 (38)	114 (98)
12	52 (36)	112 (96)	15	55 (39)	115 (99)
16	56 (40)	116 (100)			

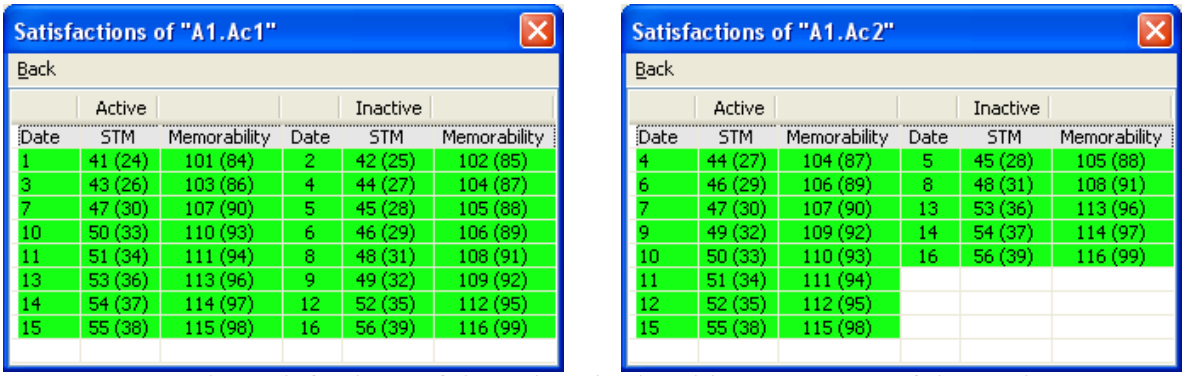


Figure 3. The satisfactions of the values in the object-structure of the environment

The header of the table in each screenshot contains the name of the values of the dimension the screenshot represent. Each column belongs to a value representing the satisfactions of that value. The columns called *Date* contain the date in the environment the satisfaction appeared. The number without brackets in the columns called *STM* represents the date the satisfaction remains memorable in the short-term memory. The number within brackets in these columns represents the time that remains until this date. The number without brackets in the columns named *Memorability* represents the date the satisfaction remains memorable. The number within brackets in these columns represents the time that remains until this date.

This presentation deals with the topic of revealing the rules that represent the behaviour of the attributes in the object structure of the environment only. At the first approach the environment is regarded as if it were deterministic. That is the state of the environment at time t can be calculated by the state of the environment at time $t-1$ and the activities done at time $t-1$. If at time t this supposition fails that makes the environment deterministic until time t by the methods described in Section 4. Revealing the rules that represent the behaviour of the activities of the agents is a development possibility. At each state of the environment each agent calculates the set of next states thet can be reached by its activities. Supposing that each agent chooses the state best for itself the set of activities can be determined.

According to the concept of the revealing behaviour this paper follows, the rules that describe the behaviour of the attributes of the environment are determined by the hypothesis space of these attributes. A hypothesis space is a set of consequences the agent observes during its experiences [1]. In the hypothesis-space the agent can classify the different consequences using the conditions that caused those consequences in order to form distinct subsets of the same consequences. The consequences in each subset represent the same consequence which was observed at different time. The condition of each subset can be obtained by the logical AND relation needed to form that subset. The rule describing a consequence can be formed by the logical OR relations of the condition of each subset representing that consequence. Figure 4 shows an example of a hypothesis space.

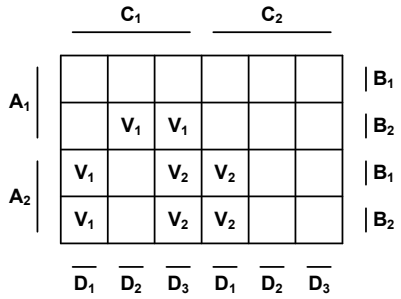


Figure 4. Example of a hypothesis space

Each place in the hypothesis space shown in Figure 4 represents an experience. The letters in the places represent the consequences of those experiences. The consequence in this case is the value that became the active value of the attribute whose hypothesis space is represented in Figure 4. The letters next to the table represent the dimensions (attributes or activities) of the environment. The index of the dimensions denotes the serial number of the value of that dimension. Each of these values represent the condition (observed by sensors) at the time the experience was recorded. In this example the letters *A*, *B* and *C* represent the dimensions of the environment with two values. Letter *D* represent a dimension of the environment with three values. Each dimension divides the hypothesis space into so many pieces as the number of its values. The number of places represented by the logical AND relation of values belonging to different dimensions can be calculated by dividing the number of places in the hypothesis space by the number that can be calculated by multiplying the number of values of the dimensions those values belong to. For example the number of places represented by A_1 **AND** D_1 in case of the hypothesis space shown in Figure 4 is 24/(2*3) = 4 pieces of places. The condition of the subsets containing the same consequences is as follows:

$$\begin{aligned} &C_1 \text{ AND } D_1 \Rightarrow V_1 \\ &A_1 \Rightarrow V_1 \\ &A_2 \text{ AND } D_3 \Rightarrow V_2 \\ &C_2 \Rightarrow V_2 \end{aligned} \tag{3.4}$$

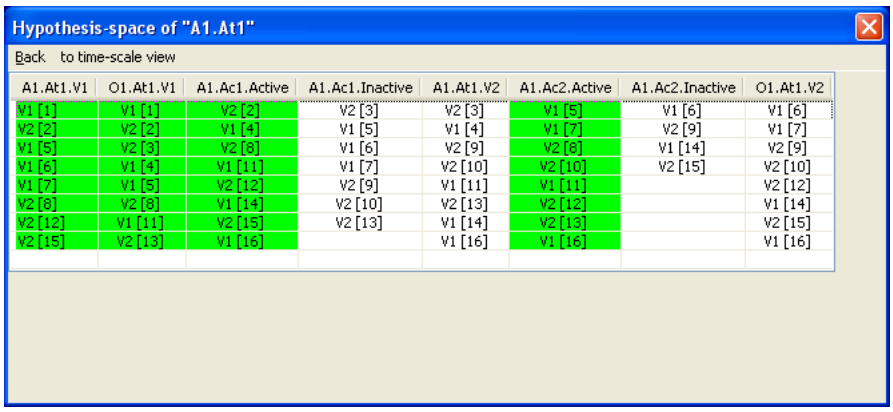
As this example shows the places that are not experienced can be taken into any subset. It follows the concept intelligent beings reveal the behaviour of the environment. If each time the sun shone a rainbow was visible then the state of the sensor indicating the state of the sun is enough to describe the visibility of the rainbow. Other sensors are not needed for this until a consequent experience appears.

The rule representing the behaviour of a value of an attribute can be determined by the logical OR relation of the condition of the subsets containing the satisfactions of that value. For example the rules as describing the behaviour of the attribute whose hypothesis-space is shown in Figure 4 are the follows.

IF (C_1 **AND** D_1) **OR** A_1 **THEN** V_1
IF (A_2 **AND** D_3) **OR** C_2 **THEN** V_2 (3.5)

The rules generated by this concept can be simplified using existing algorithms therefore this paper does not detail this issue.

Figure 4 shows the simplest representation of hypothesis spaces for understanding its role only. However this representation wastes memory significantly. It is because it reserves places for all of the experiences with different conditions. In general cases the number of the experiences appearing in an environment as much smaller than the number of the experiences with different conditions in that environment. Therefore the representation of the hypothesis spaces shown in Figure 4 can be used for environments with few state descriptor variables only. For handling environments with as many state descriptor variables as possible a new representation of the hypothesis spaces had to be developed. This new representation reserves places for the experiences that have been appeared at least once only. This new representation eliminates the waste of memory and therefore maximizes the number of the state descriptor variables. In this new representation the consequences of the experiences are organized in lists according to its conditions. The hypothesis spaces of the experiences shown in Figure 3 can be seen in Figure 5.



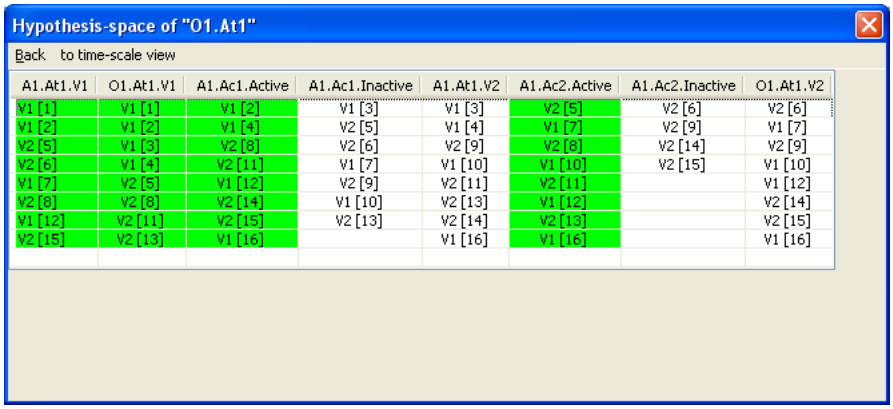


Figure 5. Hypothesis-spaces of the attributes of the environment shown in Figure 3

This representation uses no more memory space than needed for describing the hypothesis spaces. The header of the tables contains the conditions of the experiences. In each column the satisfactions of the values of the attribute whose hypothesis space is represented by the table are listed under the condition which was satisfied at the previous state of the environment where the satisfactions appeared. The number behind each consequence in square brackets represents the date that the satisfaction appeared. The green background of a column indicates that the condition associated with that column is satisfied at the actual state of the environment. If a column contains the satisfactions of the same value then the condition of that column represents the cause of the satisfaction of that value. For example according to the hypothesis spaces shown in Figure 5 each time the activity called $A_1.Ac_2$ was not done the value called $O_1.At_1.V_2$ was the active value of its attribute at the next time. Therefore the following rule can be constructed.

IF $A_1.Ac_2$.Inactive **THEN** $O_1.At_1.V_1$ (3.6)

If a column in this table contains the satisfactions of different values of the attribute whose hypothesis space that table represents then two or more columns had to be combined in order to obtain a new column with the satisfactions of the same value.

For example in Figure 5 the column with condition $A_1.At_1.V_1$ contains the satisfactions of the values $A_1.At_1.V_1$ and $A_1.At_1.V_2$. If this column is combined with the one whose condition is $A_1.Ac_1.Active$, then the resulting column contains the satisfactions of the value $A_1.At_1.V_2$ only. Therefore the following rule can be constructed.

IF $A_1.At_1.V_1$ **AND** $A_1.Ac_1$.Active **THEN** $A_1.At_1.V_2$ (3.7)

If the condition of a resulting column contains at least two values belonging to the same dimension, then that column does not contain a satisfaction. If the resulting column contains the satisfactions of the same value, then this column does not need to be combined with other columns. The rule represents the behaviour of a value of an attribute hat be formed by taking the condition of the resulting rules leading to the consequence of that value into logical OR relation with each other. Using this concept the logical rules representing the behaviour of the environment shown in Figure 3 will be as follows.

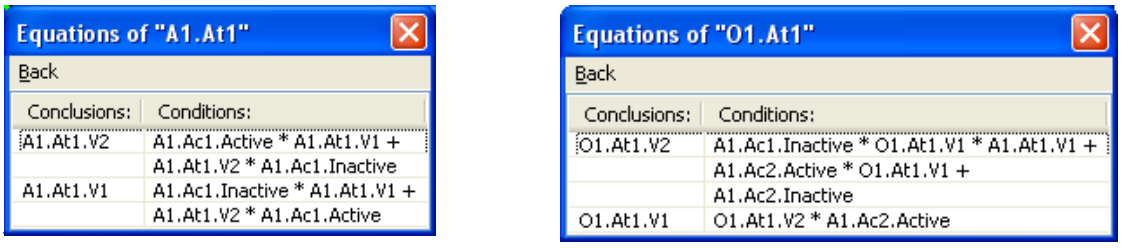


Figure 6. Rules representing the behaviour of the environment

The rules shown in Figure 6 are not equal to the ones expected according to the sequence of the changes of states shown in Figure 2. It is because the agent handles the values observed at least once. The value $A_1.At_1.V_2$ can be observed at time 2 first and the value $O_1.At_1.V_2$ can be observed at time 5 first. Therefore the behaviour of these values cannot be observed until that time. For modelling the behaviour of the values observed at least once from the initial state of the environment is a development possibility. At this point the rules describing the planned behaviour can be revealed by recording more changes of states. Figure 7 shows the rule representing the completed behaviour of the environment.

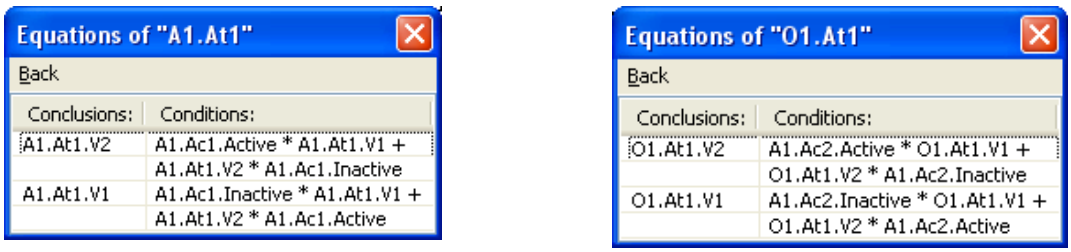


Figure 7. Rules representing the completed behaviour of the environment

4. Eliminating the inconsistencies of the observable environment

If two states of the environment with the same state description lead to states with different state descriptions, then the environment is stochastic. It is because the state of the environment at time *t* cannot be determined by the state of this environment at time *t-1* only. In practice the environment is handled as if it were be stochastic if the condition part of the rules representing the behaviour of the

environment contains numerous state descriptor variables also. This helps to avoid misleading rules. If the agent does not have a sensor to measure a relevant attribute of the environment, then the changes of states become inconsistent. If these sensors cannot be installed, then the agent has to eliminate the inconsistencies by itself. Figure 8 shows a sequence of the changes of states containing inconsistencies. In this Figure each letter with uppercase represents a state of the environment and each letter with lowercase represents the set of activities done by the agents.

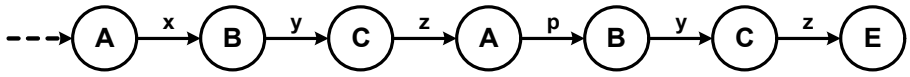


Figure 8. Example of inconsistent changes of states

As Figure 8 shows two different states can be reached from the state denoted by *C* with the same activity. This makes the environment inconsistent. For eliminating the inconsistency an inner state descriptor variable has to be created to distinguish the states be denoted by *C*. Let the values of the inner state descriptor variables denoted by numbers. Therefore the states denoted by *C* can be differentiated into a state denoted by C_1 and a state denoted by C_2.

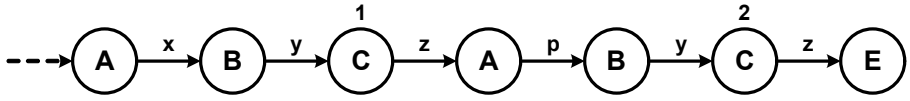


Figure 9. Elimination of some inconsistencies

It eliminates the inconsistency appearing at the states *C* but creates a new inconsistency at states denoted by *B*. These states also have to be distinguished by the inner state descriptor variable as shown in Figure 10.

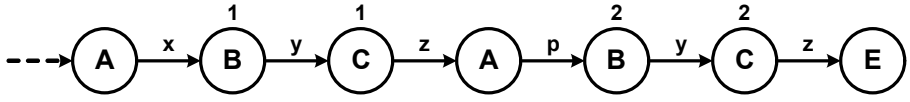


Figure 10. Elimination of some inconsistencies

This method has to be followed until all of the inconsistencies of the changes of states become eliminated.

REFERENCES

[1] RUSSELL, S., NORVIG, P.: *Mesterséges intelligencia modern megközelítésben (2., átdolgozott kiadás). Published by Pearson Education Inc. 2003.*

[2] Szabó, L.: *A nyitott jövő problémája, Véletlen, kauzalitás és determinizmus a fizikában. Typotex Kiadó. 2004.*

[3] KONDOROSI, K., LÁSZLÓ, Z., SZIRMAY-KALOS, L.: *Objektum-orientált szoftverfejlesztés*. ComputerBooks, Budapest, 1999.

[4] GILL, A.: *Introduction to the Theory of Finite-state Machines. New York, McGraw-Hill,1962*.

[5] AHO, A.V., HOPCROFT, J.E., ULLMAN, J.D.: *The Design and Analysis of Computer Algorithms*. Menlo Park, CA: Addison-Wesley, 1974.

[6] CHOW, T.S.: *Testing software design modelled by finite-state machines.* In IEEE Trans. Software Eng., vol. SE-4, no.3, pp. 178-187, Mar.1978.

[7] GOBERSHTEIN, S.M.: *Check words for the state of a finite automation*. In Kibernetika, No. 1, pp. 46-49, 1974.

[8] FRIEDMAN, A.D., MENON, P.R.: *Fault Detection in Digital Circuits*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1971.