

# Ten Project Proposals in Artificial Intelligence

Keld Helsgaun



*Artificial intelligence* is the branch of computer science concerned with making computers behave like humans, i.e., with automation of intelligent behavior. Artificial intelligence includes game playing, expert systems, natural language, and robotics.

The area may be subdivided into two main branches.

The first branch, *cognitive science*, has a strong affiliation with psychology. The goal is here to construct programs for testing theories that describe and explain human intelligence.

The second branch, *machine intelligence*, is more computer science oriented and studies how to make computers behave intelligent. It doesn't matter whether or not the mental processes of humans are simulated as long as the constructed systems behave intelligent.

This paper presents ten proposals for projects in the latter branch.

## 1. **Machine Learning** by Building Decision Trees

Since the invention of computers a lot of efforts have been made in order to make them learn. If we can program a computer to *learn* – that is, to improve its performance through experience - the consequences will be far-reaching. For example, it will be possible to make a computer ordinate an optimal treatment for a new disease using its past experience from treatment of a series of related diseases. Providing learning capabilities to computers will lead to many new applications of computers. Furthermore, knowledge about automation of learning processes may give insight in humans' ability to learn.

Unfortunately, we don't know yet how to make computers learn as well as humans. However, in recent years a series of algorithms has appeared which now makes it possible to successfully automate learning in some application areas. For example, one the most efficient algorithms for speech recognition are based on machine learning.

Today the interest in machine learning is so great that it is the most active research area in artificial intelligence.

The area may be divided into to sub areas, *symbolic* and *non-symbolic* machine learning. In symbolic learning the result of the learning process is represented as symbols, either in form of logical statements or as graph structures. In non-symbolic learning the result is represented as quantities, for example as weights in a *neural network* (a model of the human brain).

In recent years the research in neural networks has been very intensive and remarkably good results have been achieved. This is especially true in connection with speech and image recognition.

But research in symbolic machine learning has also been intensive. An important reason for this is that humans can understand the result of a learning process (in contrast to neural networks), which is significant if one should trust the result.

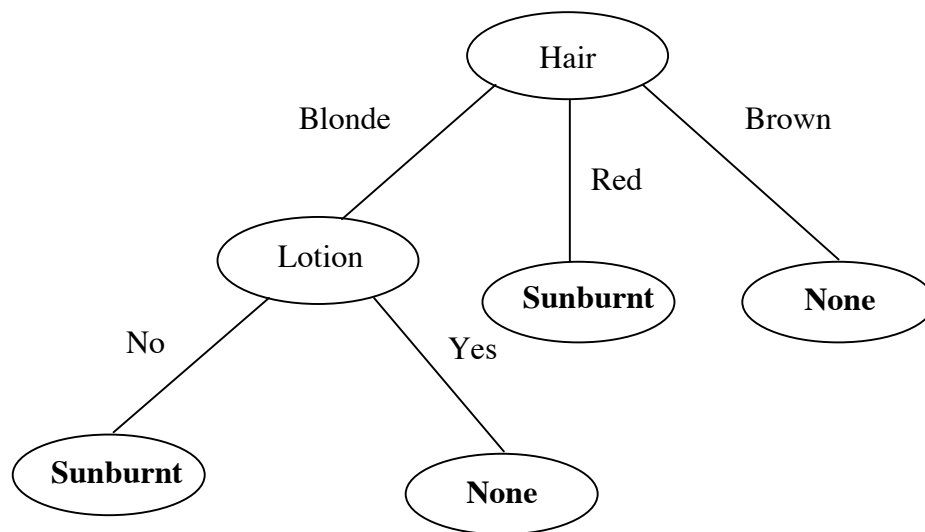
The following two projects deal with symbolic machine learning and are both using so-called *induction*.

Logical deduction is the process of learning from examples. The goal is to reach a general principle or conclusion from some given examples. Here is a simple example of induction. Suppose you see a set of letterboxes that are all red. By induction you may conclude that *all* letterboxes in the world are red (including letterboxes that you haven't seen).

*Decision trees* are one of the most applied methods for leaning by induction. The general principle of decision trees is best illustrated through and example. Suppose that you don't know what causes people to be sunburnt. You observe a series of persons and register some of their features, among these whether or not they are sunburnt. The observations are given in the table blow.

Name	Hair	Height	Weight	Lotion	Effect
Sarah	Blonde	Average	Light	No	Sunburnt
Dana	Blonde	Tall	Medium	Yes	None
Alex	Brown	Short	Medium	Yes	None
Annie	Blond	Short	Medium	No	Sunburnt
Emily	Red	Average	Heavy	No	Sunburnt
Pete	Brown	Tall	Heavy	No	None
John	Brown	Medium	Heavy	No	None
Kate	Blonde	Small	Light	Yes	None

From this table we can construct the following decision tree.



This decision tree may be used to classify a given person, i.e., to predict whether or not he will get sunburnt. Start at the top of the tree and answer the question one after another, until you reach a leaf. The leaf stores the classification (Sunburnt or None).

In the present case the decision tree agrees with our intuition about factors that are decisive for getting sunburnt. For example, neither a person's weight nor height plays a role.

It is often possible to construct more than one decision tree that agrees with the observed data. However, not all of them may be equally well suited for making generalizations, i.e., to classify examples outside the set of examples used to build the tree. How do we build the best tree? Using the so-called *ID3 algorithm*, one of the most effective algorithms for induction, may solve this problem. The algorithm builds a tree while striving at as simple a tree as possible. The assumption is that a simple tree performs better than a complex tree when unknown data are to be classified. The simplification algorithm is based on *information theory*.

In this project the ID3 algorithm is implemented and tested with some chosen example problems. If time permits the project group could study some improvements of the original algorithm, for example the C4.5 algorithm, or the group could compare the ID3 algorithm with the so-called *version spaces algorithm*.

## References

- [1] J. R. Quinlan:  
Discovering rules by induction from large collection of examples,  
in  
D. Michie (editor),  
*Expert systems in the micro electronic age*,  
Edinburgh University Press (1979).
- [2] J. R. Quinlan:  
Induction of decision trees,  
*Machine Learning*, Vol. 1(1) (1986), pp. 81-106.  
<http://www.cs.toronto.edu/~roweis/csc2515/readings/quinlan.pdf>
- [3] J. R. Quinlan:  
*C4.5: Programs for Machine Learning*,  
Morgan Kaufman (1993).
- [4] T. M. Mitchell:  
*Machine Learning*,  
McGraw-Hill (1997),  
Chapter 1-3, pp. 1-78.
- [5] E. Rich & K. Knight:  
*Artificial Intelligence*,  
McGraw-Hill (1991),  
Section 17.5, pp. 457-471.
- [6] P. Winston:  
*Artificial Intelligence*,  
Addison-Wesley (1992),  
Chapter 20-21, pp. 411-442.
- [7] S. J. Russell & P. Norvig:  
*Artificial Intelligence - A Modern Approach*,  
Prentice Hall (1995),  
Section 18.3-18.4, pp. 531-544.

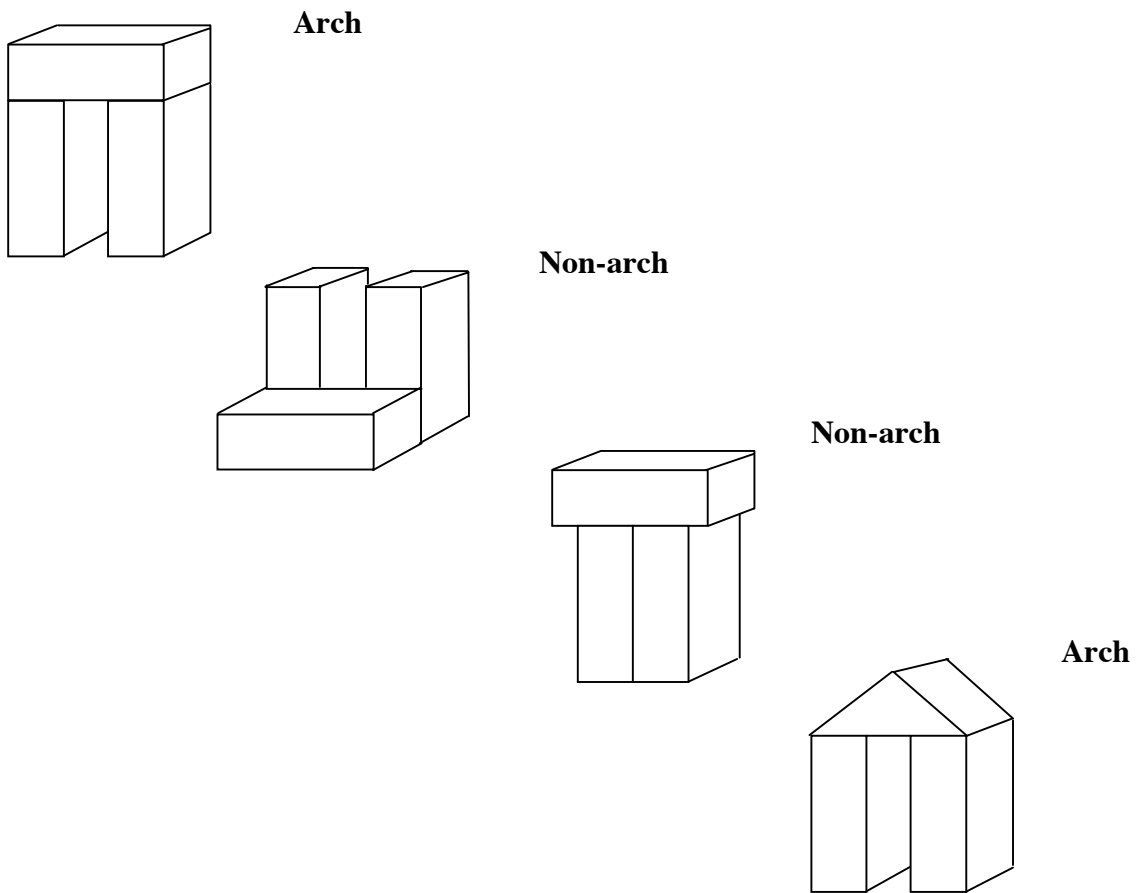
- [8] G. F. Luger & W. A. Stubblefield:  
*Artificial Intelligence - Structures and Strategies for Complex Problem Solving*,  
Addison-Wesley (1997),  
Section 13.0-13.3, pp. 603-632.
- [9] M. Ginsberg:  
*Essentials of Artificial Intelligence*,  
Morgan Kaufmann (1993),  
Chapter 15, pp. 300-320.
- [10] P. Clark & T. Niblett:  
The CN2 Induction Algorithm.  
*Machine Learning*, Vol. 3 (4), (1989), pp. 261-283.
- [11] *ID3kerne og Vindue – to generelle klasser til induktion ud fra ID3-algoritmen*,  
RUC-rapport. Datalogi, modul 1 (1990/91).

## 2. Machine Learning by Building Semantic Nets

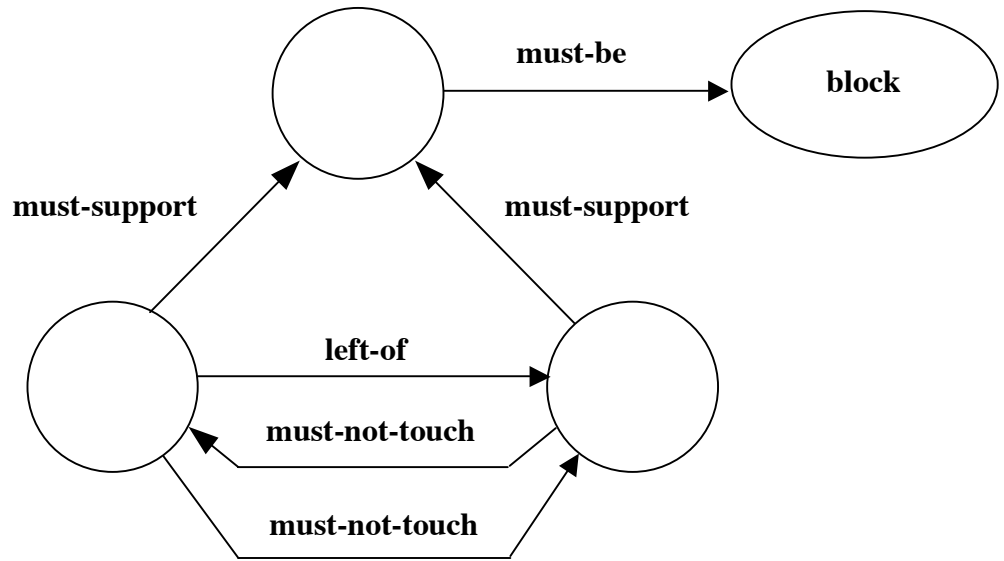
One of the first successful machine learning algorithms was developed in 1972 by P. H. Winston. The algorithm is able to learn a concept by analyzing the differences that occur in a sequence of observations. For example, the concept of an “arch” can be learned as follows.

First the algorithm is presented for a series of examples of arrangements of blocks. Some of the arrangements represent an arch. These are called *positive examples*. Others do not represent an arch. They are called *negative examples*.

Suppose the following examples are presented to the algorithm:



Then the algorithm will conclude that an arch consists of a block that is supported by two non-touching box-formed blocks. The conclusion is represented by a graph as the following (a so-called *semantic net*).



The goal of this project is to implement Winston's original algorithm and test it on one or more example problems (including the arch learning problem).

## References

- [1] P. H. Winston,  
*Learning structural descriptions from examples*,  
Ph.D. dissertation, MIT (1970).
  
- [2] P. H. Winston,  
Learning structural descriptions from examples,  
in  
*Psychology of Computer Vision*,  
P. H. Winston (editor),  
MIT Press (1975).  
<http://portal.acm.org/citation.cfm?id=889456&dl=GUIDE>  
<http://citeseer.ist.psu.edu/115629.html>
  
- [3] P. Winston:  
*Artificial Intelligence*,  
Addison-Wesley (1992),  
Chapter 16, pp. 349-363.
  
- [4] R. S. Michalski,  
Pattern Recognition as Rule-Guided Inductive Inference,  
*IEEE Transactions on Pattern Analysis and Machine Intelligence*,  
Vol. 2 (4), (1980).  
<http://mars.gmu.edu/dspace/bitstream/1920/1550/1/80-05.pdf>
  
- [5] E. Rich & K. Knight:  
*Artificial Intelligence*,  
McGraw-Hill (1991),  
Section 17.5.1, pp. 458-462.
  
- [6] G. F. Luger & W. A. Stubblefield:  
*Artificial Intelligence - Structures and Strategies for Complex Problem Solving*,  
Addison-Wesley (1997),  
Section 12.1, pp. 606-612.
  
- [7] *Maskinindlæring. Winstons porte.*  
RUC-rapport, Datalogi, speciale (1986/87).

### 3. Automatic Theorem Proving

Representation and use of *knowledge* plays a central role in artificial intelligence. Use of knowledge is a prerequisite for intelligent behavior.

Many formalisms for representing knowledge have been developed. Among these, *logic* is the most widely used. The logical formalism is often preferred since it makes it possible in a simple way to deduce new knowledge from existing knowledge. By this formalism it is possible to prove that a statement logically follows from a set of given statements.

Consider the following two statements

*All humans are mortal.*  
*Socrates is a human.*

From these we can deduce that

*Socrates is mortal.*

The logical statements above may be expressed as follows in the logical formalism called *first-order predicate logic*.

$$\begin{array}{l} \forall x: \text{human}(x) \Rightarrow \text{mortal}(x). \\ \text{human}(\text{Socrates}). \\ \hline \text{mortal}(\text{Socrates}) \end{array}$$

The first statement has the following meaning: for every  $x$ , if  $x$  is a human, then  $x$  is mortal.

A proof is found by applying one or more logical inference rules. For example, the proof above is found by the inference rule called *modus ponens*: given the following two statements

$$\begin{array}{l} P \Rightarrow Q, \\ P. \end{array}$$

we may conclude

$$Q.$$

A breakthrough in automatic theorem proving was made in 1965 when J. A. Robinson [1] showed that all proofs in first-order predicate logic can be made by application of only one inference rule, *resolution* (a generalization of modus ponens).

Resolution is a method of proof by contradiction. In order to prove that a conclusion follows from a set of statements the conclusion is negated, and attempts are made to show that the negated conclusion contradicts with the other sentences. If it can be shown that the negated conclusion is inconsistent with the other sentences, then we can conclude that the original conclusion must be valid.

In this connection it is worth mentioning that resolution forms the basis of the logic programming language PROLOG – one of the most used languages for programming artificial intelligence systems.

Space does not allow a detailed description of first-order predicate logic and resolution. See the references below.

The goal of present project is an implementation of a program for automatic theorem proving. The program should be able to read a set of logical sentences, convert these to an internal form, called *clause form*, and prove that one of the statements logically follows from the other statements. The proof is made by resolution.

Dependent on the resources of the project group input and conversion may be included or excluded. The paper by Boyer and Moore [2] is a good starting point for the implementation of the proof part of the program.

## References

- [1] J. A. Robinson:  
A machine-oriented logic based on the resolution principle,  
*Journal of the ACM*, Vol. 12 (1965), pp. 23-41.  
<http://portal.acm.org/citation.cfm?id=321253&dl=ACM&coll=portal>
  
- [2] R. S. Boyer & J. S. Moore:  
The sharing of structure in theorem-proving programs,  
in  
*Machine Intelligence 7*,  
D. Mitichie, editor,  
Elsevier (1972), pp. 101-116.
  
- [3] J. A. Robinson,  
The generalized resolution principle,  
in  
*Machine Intelligence 3*,  
D. Mitichie, editor,  
Elsevier (1968).
  
- [4] R. S. Boyer & J. S. Moore:  
*Computational Logic*,  
Academic Press (1979).
  
- [5] L. Wos,  
*Automated Reasoning*,  
Prentice-Hall (1988).
  
- [6] N. J. Nilsson,  
*Problem-Solving Methods in Artificial Intelligence*,  
McGraw-Hill (1971).
  
- [7] S. J. Russell & P. Norvig:  
*Artificial Intelligence - A Modern Approach*,  
Prentice Hall (1995),  
Chapter 9, pp. 265-296.
  
- [8] G. F. Luger & W. A. Stubblefield:  
*Artificial Intelligence - Structures and Strategies for Complex Problem Solving*,  
Addison-Wesley (1997),  
Chapter 12, pp. 559-602.

- [9] E. Rich & K. Knight:  
*Artificial Intelligence*,  
McGraw-Hill (1991),  
Chapter 5, pp. 131-169.
- [10] T. Østerby:  
*Kunstig intelligens - metoder og systemer*,  
Polyteknisk Forlag (1992),  
Kapitel 2, s. 11-33,  
Kapitel 7, s. 117-140.
- [11] J. K. Siekmann & G. Wrightson (editors):  
*Automation of reasoning, Vol I and 2*,  
Springer Verlag (1983).
- [12] M. Ginsberg:  
*Essentials of Artificial Intelligence*,  
Morgan Kaufmann (1993),  
Chapter 6-8, pp. 105-172.
- [13] R. J. Cunningham & S. Zappacost-Amboldi:  
*Software Tools for First-Order Logic*,  
*Software - Practice and Experience*,  
Vol. 13 (1983), pp. 1019-1025.
- [14] M. Fitting:  
*First-order logic and automated theorem proving*,  
Springer-Verlag (1990).
- [15] *En SIMULA-klasse til automatisk bevisførelse*,  
RUC-rapport. Datalogi, modul 3 (1977).
- [16] *Automatisk bevisførelse med resolution*.  
RUC-rapport. Datalogi, modul 1 (2003).
- [17] K. Helsgaun:  
*Automatisk bevisførelse*,  
DIKU-rapport. Speciale (1972/73).

#### 4. Expert Systems

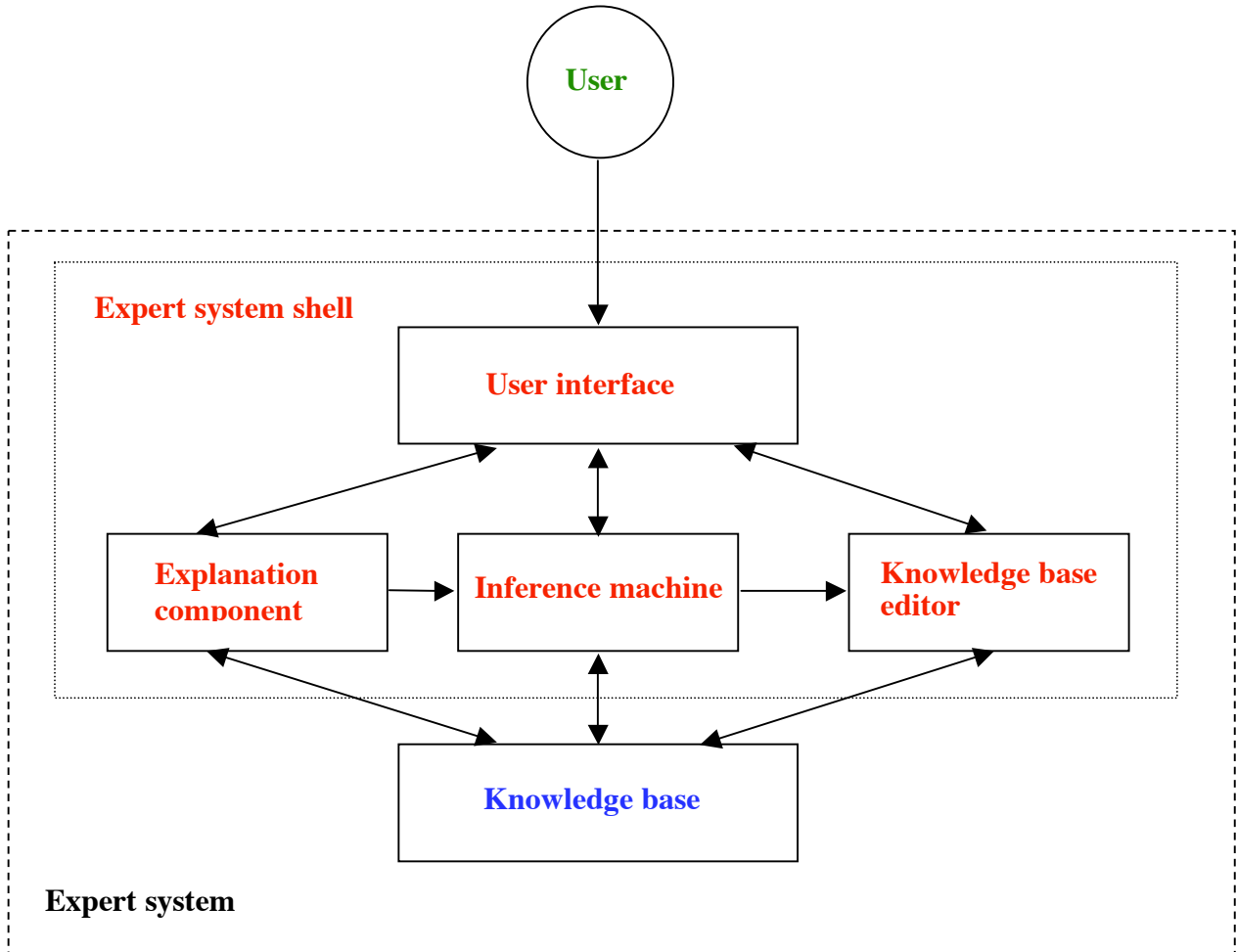
Human experts are able to solve problems at a high level because they exploit knowledge about their area of expertise. This fact provided the background for the design of programs with *expert-based* problem solving capabilities. An *expert system*, which such a program is often called, uses specific knowledge about an area in order to obtain competence comparable to that of a human expert. The specific knowledge may be obtained by interviewing one or more experts in the area in question.

The area “expert systems” is probably the sub-area of artificial intelligence that has achieved the greatest commercial success. Today expert systems are used in a large number of subject areas, ranging from medicine, chemistry, and geology to law, politics, and economics. Any area in which decisions are to be made is a potential application area for expert systems.

Below are listed examples of application areas [1].

- |                          |   |
|--------------------------|---|
| 1. Interpretation:       | drawing high-level conclusions based on data.         |
| 2. Prediction:           | projecting possible outcomes.                         |
| 3. Diagnosis:            | determining the course of malfunction, disease, etc.  |
| 4. Design:               | finding best configuration based on criteria.         |
| 5. Planning:             | proposing a series of actions to achieve a goal.      |
| 6. Monitoring:           | comparing observed behavior to the expected behavior. |
| 7. Debugging and Repair: | prescribing and implementing remedies.                |
| 8. Instruction:          | assisting students in learning.                       |
| 9. Control:              | governing the behavior of a system.                   |

The figure on the next page shows the most important components of an expert system.



The *knowledge base* is the central component of an expert system. It stores knowledge in the form of facts and rules. Usually predicate logic is used for this purpose. The knowledge base may also store the *confidence level* that a fact or rule is true or valid.

The *inference machine* is used to make logical inferences from stored knowledge.

The *user interface* is used to exchange information with the user of the system, for example, for input of questions and output of answers.

The *explanation component* provides the user with information about the reasoning of the system, including a report of how answers were found by the system.

The *knowledge base editor* is used for building and maintaining a knowledge base.

The explicit separation of the knowledge base component from the rest of system makes it possible to reuse all other components in the development of expert systems. An *expert system shell* contains all components in the figure above, except for the knowledge base. Thus, a developer of an expert system may use an “empty” shell and just add a relevant knowledge base. This has the advantage that the developer may fully concentrate in gathering knowledge and building the knowledge base. All other components are available and need not to be changed.

In the present project a rule-based expert system shell is to implemented and tested. As a starting point for the implementation reference [2] may be used with advantage.

## References

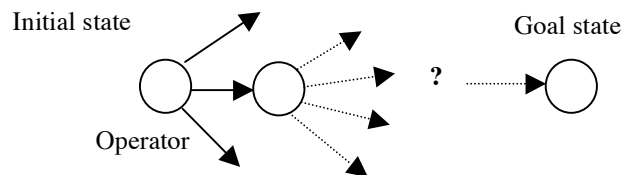
- [1] D. A. Waterman:  
*A Guide to Expert Systems*,  
Addison-Wesley (1986).
- [2] B. Swayer & D. Foster:  
*Programming Expert Systems in Pascal*,  
Wiley Press (1986).
- [3] T. Østerby:  
*Kunstig intelligens - metoder og systemer*,  
Polyteknisk Forlag (1992),  
Kapitel 10, s. 179-198.
- [4] E. Rich & K. Knight:  
*Artificial Intelligence*,  
McGraw-Hill (1991),  
Chapter 20, pp. 547-557.
- [5] S. J. Russell & P. Norvig:  
*Artificial Intelligence - A Modern Approach*,  
Prentice Hall (1995),  
Chapter 5, pp. 122-148.
- [6] G. F. Luger & W. A. Stubblefield:  
*Artificial Intelligence - Structures and Strategies for Complex Problem Solving*,  
Addison-Wesley (1997),  
Chapter 6, pp. 207-246.
- [7] M. Ginsberg:  
*Essentials of Artificial Intelligence*,  
Morgan Kaufmann (1993),  
Chapter 18, pp. 381-401.
- [8] F. Hayes-Roth, D. A. Waterman & D. B. Lenat (editors):  
*Building expert systems*,  
Addison-Wesley (1983).
- [9] *En ekspertsystemskal i SIMULA*.  
RUC-rapport. Datalogi, modul 1 (1995/96).
- [10] *Nemesis - kernen i en skal. Ekspertsystemer og ekspertsystemskaller*,  
RUC-rapport. Datalogi, speciale (1986/87).

## 5. Problem Solving by Means of Macro-operators

Problem solving is one of the most central subjects in artificial intelligence. The concept *problem solving* is broadly understood and covers any situation where one wants to find out if it is possible to get from a given initial state to some wanted goal state. A *problem* is defined abstractly by the following three components:

1. An *initial state*
2. One or more *goal states*
3. A set of *operators*

When an operator is applied to a state, it yields a new state. Solving a problem is equivalent to finding a sequence of operator applications that transforms the initial state into one of the goal states.



A simple example of problem solving is the so-called *8-puzzle*. Eight tiles labeled 1-8 have been placed on a  $3 \times 3$  board, for example, as shown below.

6	2	8
/	3	5
4	7	1

One of the squares of the board is empty (the hatched square in the figure above). At each move a player can slide a tile adjacent to the empty square to the empty square. The problem is to find a sequence of moves that leads from the initial state to a given goal state, for example, the one given below.

1	2	3
8	/	4
7	6	5

This problem, and many others, may be solved by means of the so-called *A\*-algorithm*. Suppose a problem is given together with some *heuristic function*, i.e., a function that for

every possible state is able estimates how “close” it is to a goal state. Then a solution of the problem may be determined by a variant of *best-first-search*. At each step the algorithm chooses to work with the state that actually appears to be closest to a goal state, in other words, the most promising state.

However, not all problems are suited for the A\*-algorithm. This is the case for *Rubik's cube*.



Each of the six sides of the cube is made of nine squares. In its initial state each side of the cube is a different color. But the rotation of each face allows the squares to be rearranged in many different ways. The problem is to return the cube from a given state to its original state.

The difficulty in using the A\*-algorithm for the solution of this problem is that no effective heuristic function is known today. There are apparently no simple connection between the appearance of the cube and the number of rotations that may lead to the goal state. Even if the cube is a jumble of colored squares it could be quite close to the goal state.

One remedy for this difficulty is to use so-called *macro-operators*. A macro-operator is a fixed sequence of simple operators. In the case of Rubik's cube a macro-operator is a fixed sequence of rotations. A problem is decomposed into independent subproblems, and each of these subproblems is solved. A macro-operator is useful if it solves a given subproblem without destroying the solutions of the preceding subproblems.

This goal of this project is to solve Rubik's cube by means of macro-operators as described in the publications [1, 2, 3].

## References

- [1] R. E. Korf:  
Macro-operators: A weak method of learning,  
*Artificial Intelligence*,  
Vol. 26, (1985), pp. 35-77.
- [2] R. E. Korf:  
Depth-first iterative deepening: An optimal admissible tree search,  
*Artificial Intelligence*,  
Vol. 17, (1985), pp. 97-109.
- [3] R. E. Korf:  
*Learning to Solve Problems by Searching for Macro-operators*,  
Ph.D. Thesis, Carnegie-Mellon University (1983).
- [4] L. Kanal & V. Kumar, editors:  
*Search in artificial intelligence*,  
Springer-Verlag (1988).
- [5] *Søgealgoritmer, makrooperatorer og kvadratspil*,  
RUC-rapport, Datalogi, modul 1 (1997/98).
- [6] G. A. Iba:  
A Heuristic Approach to the Discovery of Macro-Operators,,  
*Machine Learning*, Vol. 3 (1989).  
<http://www.springerlink.com/content/h1wwttk5l8k5mjqp/>
- [7] I. T. Hernádvölgyi:  
*Searching for Macro Operators with Automatically Generated Heuristics*,  
*Lecture Notes in Computer Science*, Vol. 2056 (2001).  
<http://www.springerlink.com/content/v1ugh76edgnewcu3/>

## 6. Problem Solving by Constraint Satisfaction

A constraint satisfaction problem, also called a CSP, is a problem type with the following structural properties. Each problem state is defined by a set of values of *variables*, and the goal state satisfies a set of given *constraints*. The goal is to find values of the variables that will satisfy the set of constraints.

A CSP can more formally be described as follows. A CSP consists of

- a set of *variables*,  $\{x_1, x_2, \dots, x_n\}$ ,
- for each variable,  $x_i$ , a *domain*,  $D_i$ , of values that  $x_i$  can take, and
- a set of *constraints*, i.e., a set of relations between variables.

The goal is for each variable,  $x_i$ , to find a value in  $D_i$ , such that all constraints are satisfied.

A simple example of CSP is the so-called *8-queens problem*. We want to place 8 queens on a chessboard so that no queen attacks any other. Here the variables represent the placement of the eight queens, and the constraints express that the queens must not attack each other (that is, no two queens must share the same row, column, or diagonal).

Another example is the following *cryptarithmic* puzzle.

$$\begin{array}{r} + \text{ S E N D} \\ \text{ M O R E} \\ \hline \text{ M O N E Y} \end{array}$$

The goal is to assign unique digits to the letters in such a way that the above sum is correct.

Due to its generality CSP has been applied to solve a wide range of problems. Many design and planning problems may be expressed as CSPs. For example, timetable problems.

The goal of this project is to implement a general tool for solving CSPs and test it on some selected example problems.

## References

- [1] A. K. Macworth:  
Consistency in Networks of Relations,  
*Artificial Intelligence*, Vol. 8 (1), (1977).
  
- [2] E. C. Freuder:  
Synthesizing Constraint Expressions,  
*Communications of the ACM*, Vol. 21 (11), (1978).  
<http://portal.acm.org/citation.cfm?id=359654>
  
- [3] L. Kanal & V. Kumar, editors:  
*Search in artificial intelligence*,  
Springer-Verlag (1988).
  
- [4] S. J. Russell & P. Norvig:  
*Artificial Intelligence - A Modern Approach*,  
Prentice Hall (1995),  
Section 3.7, pp. 83-84.
  
- [5] T. Østerby:  
*Kunstig intelligens - metoder og systemer*,  
Polyteknisk Forlag (1992),  
Afsnit 6.8-6.9, s. 108-112.
  
- [6] *KP - en generel klasse til Kombinatorisk Problemløsning*,  
RUC-rapport. Datalogi, modul 1 (1990/91).

## 7. Game Playing

Games have over the years fascinated many people. As long as there have been computers efforts have been made to provide them with abilities to play games. As early as in the eighteenth century Charles Babbage considered the possibility of getting a machine to play chess. In the beginning of the 1950's Claude Shannon and Alan Turing made the first sketches of chess programs.

Starting from Shannon and Turing's ideas Arthur Samuel in 1959 wrote the first program that was able to play checkers. The program was able to learn from its errors and achieved an impressive playing strength. It was capable of beating the world champion of checkers at that time.

Chess is a more complex game than checkers. Not until during the last decade one has succeeded in developing programs that can beat the best human chess players.

Game playing is a popular application area for artificial intelligence. There are a number of reasons for this popularity, but probably the most important reason is that games are suitable for evaluating some of the central techniques of artificial intelligence, such as search and use of heuristic knowledge. Normally, a game is based on a few, simple rules, and it is easy to measure success and failure.

The goal of this project is to develop a program, which is able to play a given game. The game is assumed to have the following two properties. It must be a *two-person game*, i.e., a game where two persons take turns in making moves, and it must be a *game of perfect information*, i.e., a game where both players know the current state of the game (nothing is hidden to the players). Among games with these properties may be mentioned: Chess, Othello, Go, Tic-Tac-Toe, and Kalaha.

As a part of the project a general program package is developed which may be used to play any two-person game of perfect information. The package must provide a search method (for example, the so-called *Alpha-Beta method*) for producing a "good" move in a given position. The user of the package must provide information about the game to be played, including representation of a position, a specification of legal moves, and a position evaluation function.

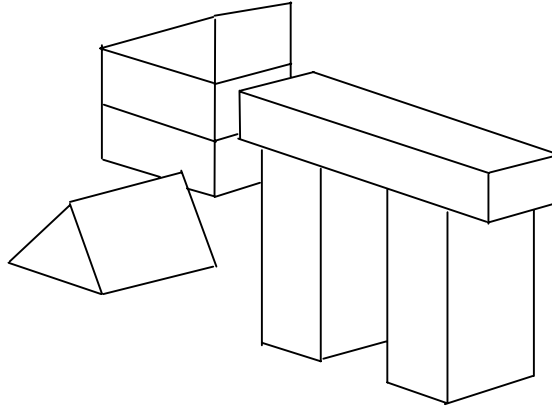
Use of the package is demonstrated through a game chosen by the project group. If the group has additional resources may be provided with a graphical user interface.

## References

- [1] D. Waltz:  
Understanding line drawings with shadows,  
in  
*Computer Vision*,  
P. H. Winston (editor),  
McGraw-Hill (1975), pp. 19-91.
  
- [2] E. Rich & K. Knight:  
*Artificial Intelligence*,  
McGraw-Hill (1991),  
Section 14.3, pp. 367-375.
  
- [3] P. Winston:  
*Artificial Intelligence*,  
Addison-Wesley (1992),  
Chapter 12, pp. 249-272.
  
- [4] M. Ginsberg:  
*Essentials of Artificial Intelligence*,  
Morgan Kaufmann (1993),  
Chapter 16, pp. 323-350.
  
- [5] *Billedanalyse. Implementering af Waltz algoritme*,  
RUC-rapport. Datalogi, modul 1 (1989/90).

## 8. Computer Vision

Consider the picture below.



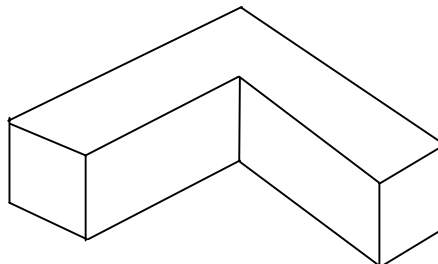
We humans can easily see what the picture represents. For example, we can see that the picture contains several objects, and we can point these out. A computer, on the other hand, has difficulties. Given only the line segments of the picture the computer cannot easily deduce that the picture contains objects.

In 1975 Walz [1] showed that it was possible to construct an algorithm that solves this problem. His algorithm labels each line segment of a picture. The labeling is then used to solve the problem. For each line segment, edge, is determined whether it is

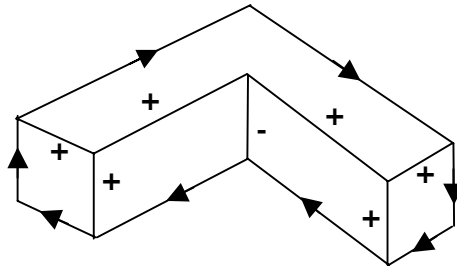
- an *obscuring edge* (a boundary between objects, or between objects and the background),
- a *concave edge* (an edge between two faces of an object that forms an *acute* angle when viewed from outside it), or
- a *convex edge* (an edge between two faces of an object that forms an *obtuse* angle when viewed from outside it).

An obscuring edge is labeled with an arrow, so that if one moves in the direction of the arrow the object will be to the right of the arrow. A concave edge is labeled with a minus (-), whereas a convex edge is labeled with a (+).

Suppose, for example, that we are given the following picture of a brick.



Then the result of Waltz' algorithm is that the edges are labeled as shown below.



The goal of the project is to implement and test Waltz' algorithm. An easily understood description of the algorithm is given in [2].

### References

- [1] D. Waltz:  
Understanding line drawings with shadows,  
in  
*Computer Vision*,  
P. H. Winston (editor),  
McGraw-Hill (1975), pp. 19-91.
- [2] E. Rich & K. Knight:  
*Artificial Intelligence*,  
McGraw-Hill (1991),  
Section 14.3, pp. 367-375.
- [3] P. Winston:  
*Artificial Intelligence*,  
Addison-Wesley (1992),  
Chapter 12, pp. 249-272.
- [4] M. Ginsberg:  
*Essentials of Artificial Intelligence*,  
Morgan Kaufmann (1993),  
Chapter 16, pp. 323-350.
- [5] *Billedanalyse. Implementering af Waltz algoritme*,  
RUC-rapport. Datalogi, modul 1 (1989/90).

## 9. Natural Language Processing

The ability to use a language for communication is what separates humans from other animals. Humans master the complexity of spoken language. To understand speech is a difficult task, especially because it requires processing of analog signals, which normally are encumbered with noise. But even the simpler task of understanding a written text is very difficult. To fully understand a written statement about a subject it not only necessary to know a lot about the language itself (vocabulary, grammar, etc.) but also a lot about the subject in question.

Processing of natural language is an area of artificial intelligence, which was initiated in the beginning of the sixties. The area may be divided into the following subareas:

- Text understanding
- Text generation
- Speech understanding
- Speech generation
- Machine translation (translation from one natural language to another)
- Natural language interfaces for databases

In most applications the text or the speech is transformed into some internal representation. Thus, transformation of sentences into an internal representation is central in natural language processing.

The goal of the present project is to implement the basic element of a system for understanding sentences written in a natural language (for example, English or Danish). The system might be realized by solving the following subproblems:

1. Morphological analysis
2. Syntactic analysis
3. Semantic analysis

The *morphological analysis* splits a sentence into words. Each word is looked up in a dictionary in order to determine its word class and inflection. This information is used as input for the syntactic analysis.

The *syntactic analysis* checks that the words form a legal sentence. The rules of which sentences are legal in the language are expressed in the form of a grammar. In this project are used so-called *extended state diagrams* for the purpose. They are also called *ATN-grammars* (an abbreviation of *Augmented Transition Networks*). The result of the syntactic analysis is input for the semantic analysis.

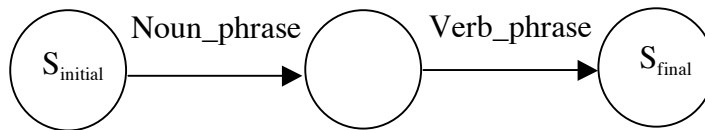
The *semantic analysis* determines the meaning of a sentence, among other things by looking up the meaning of the individual words of the sentence.

To give an impression of the method we sketch of an analysis of the sentence

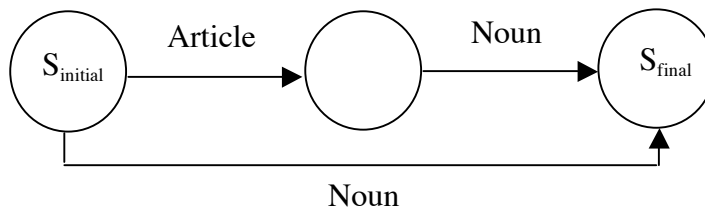
*The dog likes a man.*

Let the following state diagram represent the grammar of a simple subset of English.

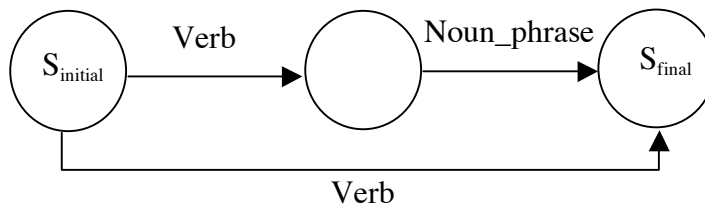
Sentence:



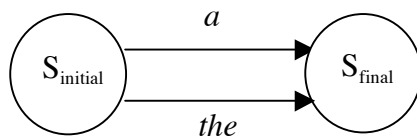
Noun\_phrase:



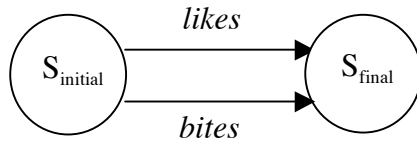
Verb\_phrase:



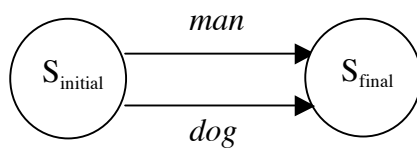
Article:



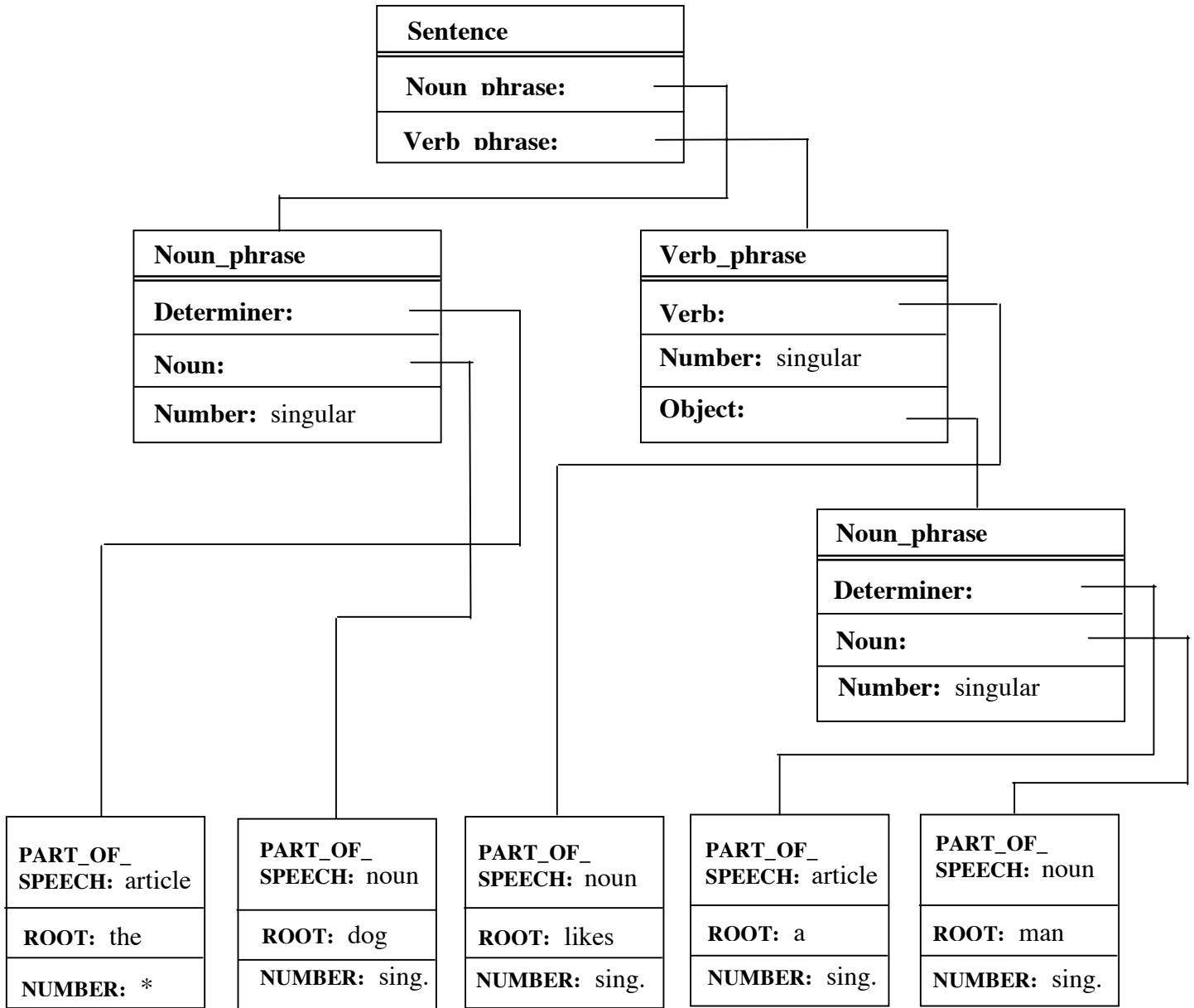
Verb:



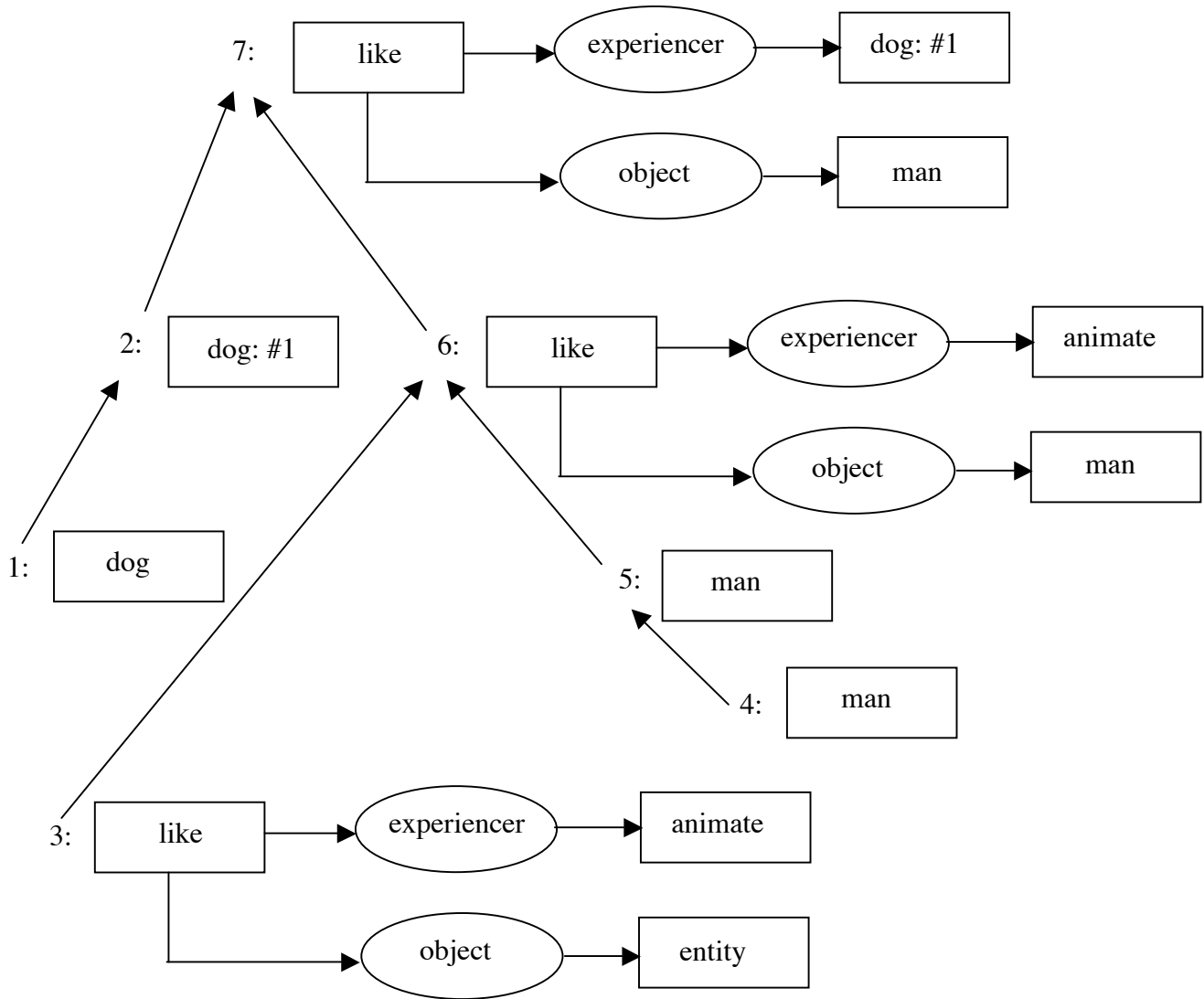
Noun:



By extending the state diagram with suitable actions the original sentence may now be transformed into the *parse tree* below.



This parse tree is now used as input to the semantic analysis, which produces the following semantic representation.



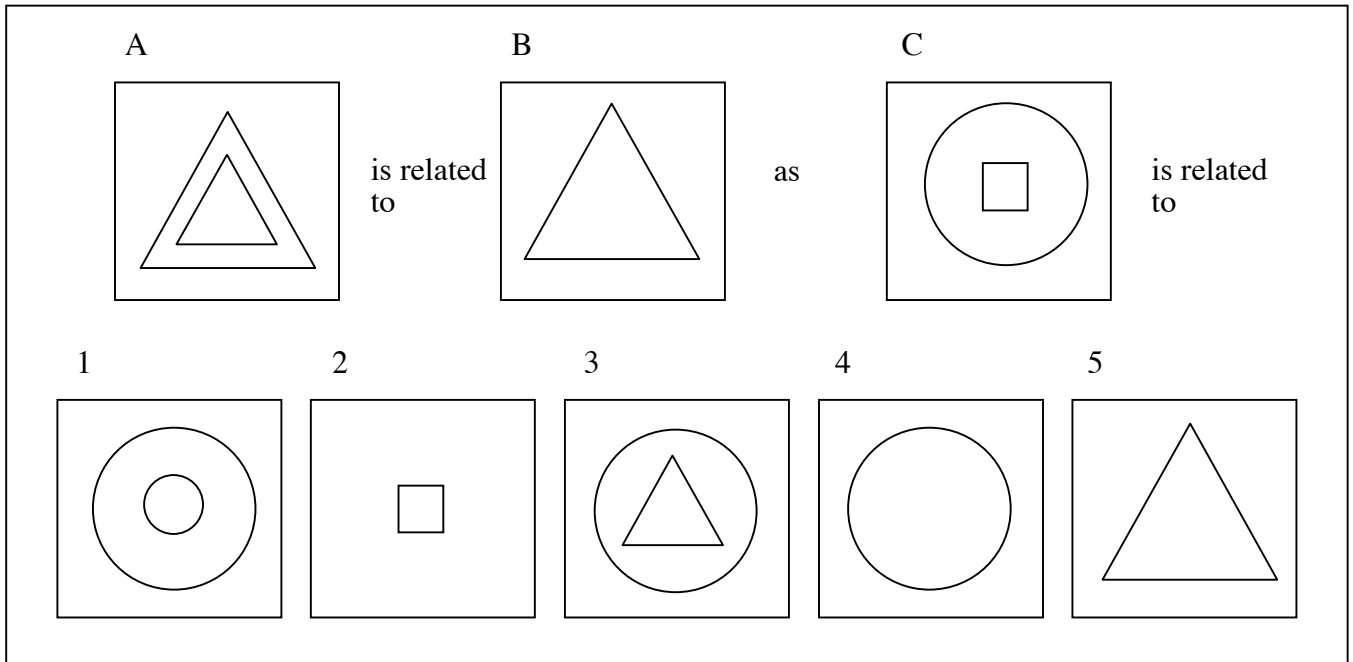
A specified explanation of these figures may be found in [1].

## References

- [1] G. F. Luger & W. A. Stubblefield:  
*Artificial Intelligence - Structures and Strategies for Complex Problem Solving*,  
Addison-Wesley (1997),  
Chapter 11, pp. 519-558.
- [2] J. Allen:  
*Natural Language Understanding*,  
Benjamin/Cummings (2. edition), 1995.
- [3] T. Winograd:  
*Language as a Cognitive Process*,  
Addison-Wesley (1983).
- [4] M. D. Harris:  
*Introduction to Natural Language Processing*,  
Reston (1985).
- [5] E. Rich & K. Knight:  
*Artificial Intelligence*,  
McGraw-Hill (1991),  
Chapter 15, pp. 377-428.
- [6] P. Winston:  
*Artificial Intelligence*,  
Addison-Wesley (1992),  
Chapter 29, pp. 599-616.
- [7] M. Ginsberg:  
*Essentials of Artificial Intelligence*,  
Morgan Kaufmann (1993),  
Section 7.2-7.3, pp. 354-371.
- [8] T. Østerby:  
*Kunstig intelligens - metoder og systemer*,  
Polyteknisk Forlag (1992),  
Kapitel 16, s. 275-303.
- [9] *Natursprogsgrænseflade til SQL*,  
RUC-rapport. Datalogi, modul 2 (1992/93).

## 10. Analogy Formation

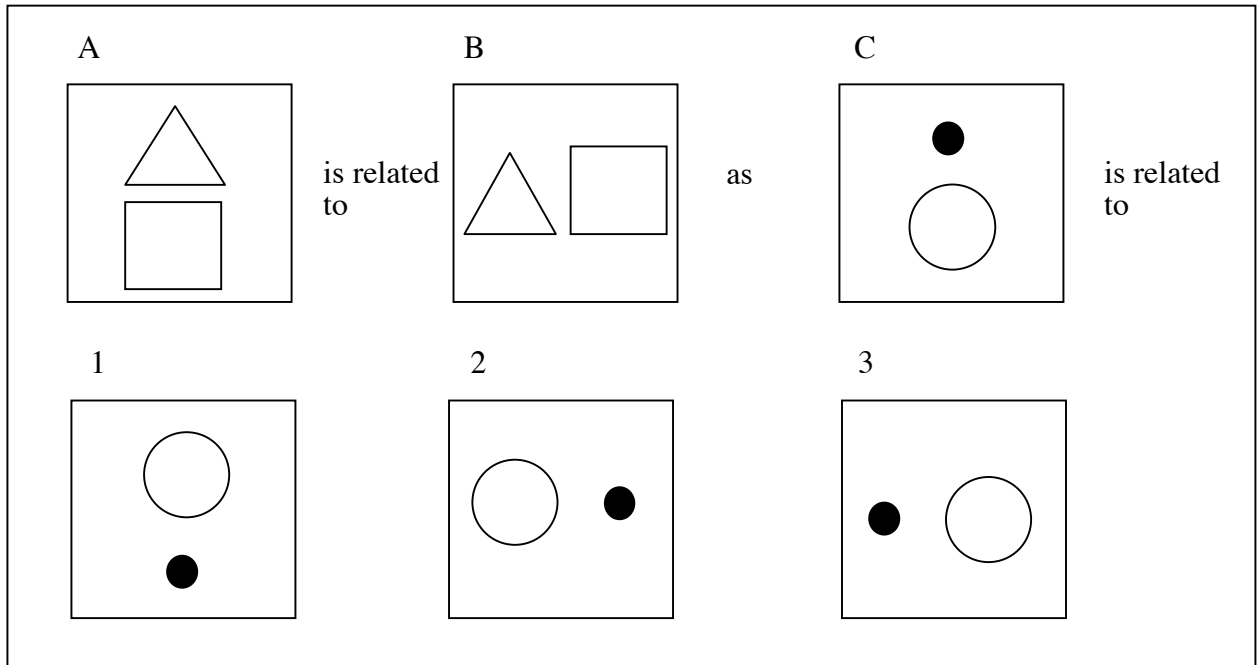
Many intelligence tests contain so-called *geometric analogy problems*. The test person is presented with a figure as the following.



Now the task is to choose one of the figures 1 to 5 such that figure A is related to figure B as figure C is related to the chosen figure.

The goal of the present project is write and test a program that can solve this type of problems. A possible algorithm is described in [1] and [2]. Below the idea of the algorithm is sketched.

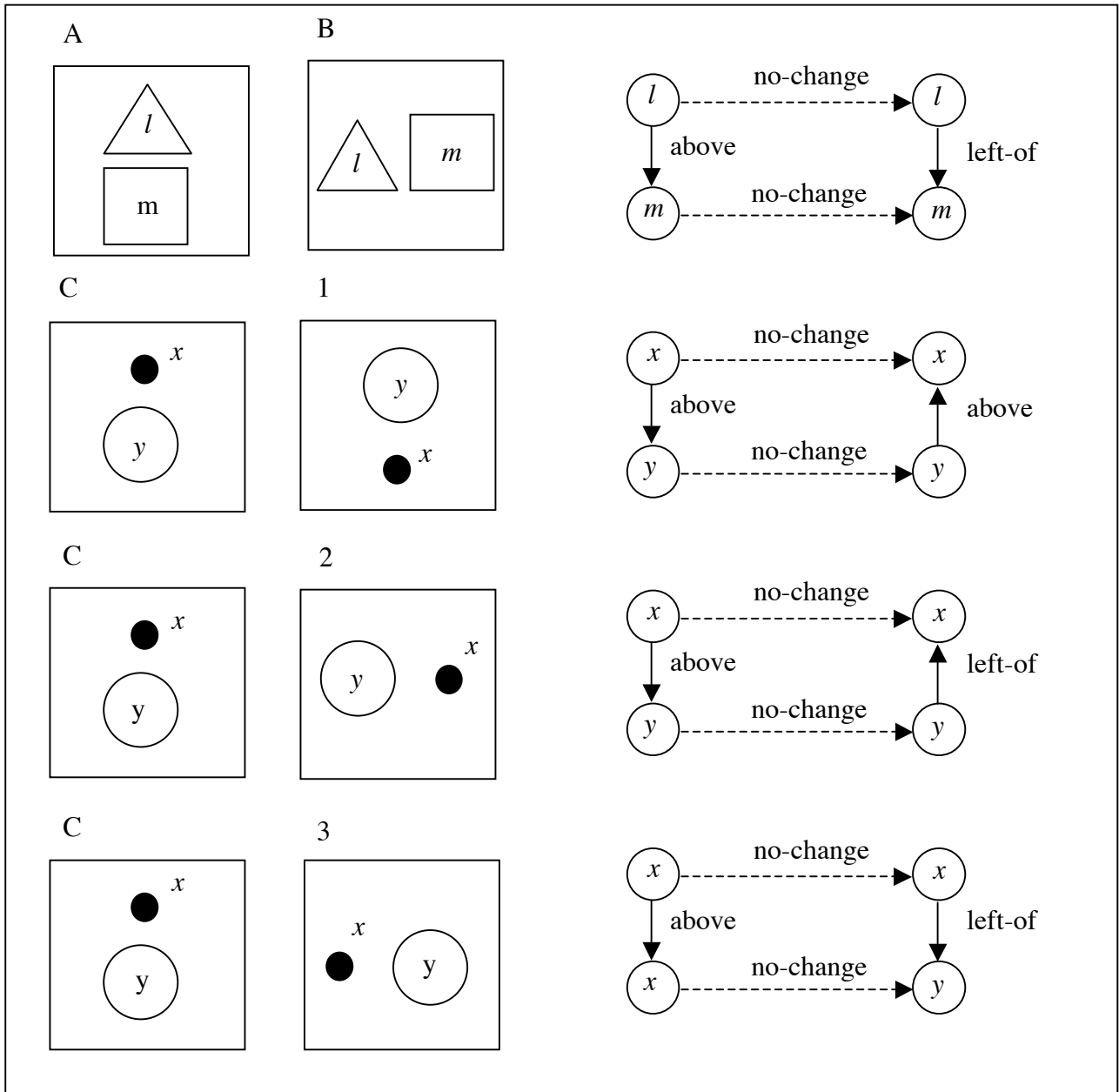
Suppose the algorithm is presented with following simple problem.



Then the program will make a rule for how A is related to B, and rules for how C is related to each of the possible answer figures. Among the rules that tell how C is related to the possible answers the rule that best matches the A to B rule is used to identify the best answer.

Each rule consists of two parts. The first part describes the relations between the individual objects of a figure. An object may be to the left, inside, or above another object. The second part describes how the objects in one figure may be transformed into the objects in another figure. An object may be scaled, rotated, reflected, or undergo a transformation, which is a combination of these transformations.

The rules are represented by means of so-called *semantic nets*. The semantic net for the example problem is shown on the next page.



By comparison of the semantic nets one can see that the rule *C-is-related-to-3* best matches the rule *A-is-related-to-B*. *l* and *m* only has to be replaced by *x* and *y*, respectively. Therefore, the chosen answer is 3.

## References

- [1] T. G. Evans:  
A Heuristic Program to Solve Geometry Analogy Problems,  
in  
*Semantic Information Processing*,  
M. Minsky (editor),  
Academic Press (1968).
  
- [2] P. Winston:  
*Artificial Intelligence*,  
Addison-Wesley (1992),  
Section 2, pp. 15-45.