Chapter 2
Symbolic Artificial Intelligence

Basic methodological assumptions of methods which belong to *symbolic Artificial Intelligence* are presented in this chapter. The three following fundamental beliefs are common to these methods:

- a model representing an intelligent system can be defined in an *explicit* way,
- knowledge in such a model is represented in a symbolic way,[1]
- mental/cognitive operations can be described as formal operations[2] over symbolic expressions and structures which belong to a knowledge model.

In symbolic AI two subgroups of methods can be distinguished. Within the first subgroup researchers try to define *general (generic)* models of knowledge representation and intelligent operations. This subgroup includes cognitive simulation and logic-based reasoning.

By contrast, defining models which concern *specific* application areas is the goal of research within the second group. Thus, such models are based on representations of domain knowledge. This subgroup includes rule-based knowledge representation, structural knowledge representation, and an approach based on mathematical linguistics.

These groups of methods are presented briefly in the following sections.

[1] For example, knowledge is defined in the form of graphs, logic formulas, symbolic rules, etc. Methods of symbolic AI are developed on the basis of logic, theory of formal languages, various areas of discrete mathematics, etc.

[2] For example, operations in the form of inference rules in logic, productions in the theory of formal languages, etc.

M. Flasiński, *Introduction to Artificial Intelligence*,
DOI 10.1007/978-3-319-40022-8_2

2.1 Cognitive Simulation

Cognitive simulation is one of the earliest approaches to Artificial Intelligence. The approach was introduced by Newell and Simon, who used it to design their famous systems *Logic Theorist* and *General Problem Solver, GPS*. The main idea of cognitive simulation consists of defining heuristic algorithms[3] in order to simulate human cognitive abilities, e.g., reasoning, problem solving, object recognition, and learning. During such a simulation a sequence of elementary steps, which are analogous to those made by a human being, is performed by a computer. Therefore, in order to design such algorithms we try to discover elementary concepts and rules which are used by a human being for solving generic problems. Now, we introduce four basic concepts of cognitive simulation, namely state space, problem solving as searching state space, Means-Ends Analysis, and problem reduction.

Let us begin with the concept of *state space*. Its initial state represents the situation in which we begin problem solving. Let us consider the example of chess. The initial state represents the initial position of the pieces at the start of a game. Goal states (or the goal state, if the problem has only one solution) represent a problem at the moment of finding a solution. Thus, in chess goal states represent all the situations in which we checkmate the opponent. The remaining (intermediate) states represent all situations that are possible on the way to solving the problem. Thus, in chess they represent all situations that are allowed taking into account the rules of the game. A state space is a graph in which nodes correspond to states (initial, goal, and intermediate) and edges represent all allowable transitions from one state to another. Thus, for chess a state space can be defined in the following way. From the node of the initial state we define transitions (edges) to nodes that correspond to situations after the first "white" move. Then, for each such intermediate state we define transitions (edges) to nodes that correspond to situations after the first "black" move in response to a situation caused by the first "white" move, etc.

Problem solving as searching a state space is the second concept of cognitive simulation. The idea is straightforward: if we do not know how to formulate algorithmic rules of problem solving, we can try to solve this problem with the method of *"trial and error" ("generate and test", "guess and check")*. Of course, the use of this method in its "pure form" is not a good idea for, e.g., playing chess. However, sometimes this is the only method we can use in our everyday life. For example, we have forgotten the combination lock code of our suitcase or we have promised our fiancée to make a delicious omelet in the evening and we have lost the recipe. (However, we have a lot of eggs in the refrigerator, so we can make some experiments.) Let us notice that such generation of potential solutions can be "blind" (we do not

[3] A heuristic algorithm is an algorithm which can generate an accepted solution of a problem although we cannot formally prove the adequacy of the algorithm w.r.t. the problem. The term *heuristics* was introduced by the outstanding mathematician George Pólya. Newell attended lectures delivered by Pólya at Stanford University.

use codes which represent important dates) or can be limited to a certain subarea of the state space (we use important dates as codes). What is more, we can have a certain measure, called here a *heuristic function*, which tells us how close we are to a satisfactory solution (e.g., the "tasting measure", which tells us how close we are to a delicious omelet). Then, if our experimental omelet is almost delicious, we should modify the recent result only a little bit. A correct procedure to generate possible solutions should have three properties. Firstly, it should be *complete*, i.e., it should be able to generate *every* potential solution. Secondly, it should not generate a potential solution more than once. Thirdly, it should make use of information that restricts the state space to its subarea as well as measures/functions of the quality of the potential solutions generated.

If we have a function which assesses the quality of potential solutions, then we can use a technique called *Means-Ends Analysis, MEA*, for controlling the process of generating potential solutions. If we are at a certain state, then we use such a function for determining the difference/distance between this state and a goal state and we use a *means* (i.e., the means is usually in the form of an operator or a procedure) which reduces this difference. For example, if our experimental omelet is nearly as tasty as the ideal one and the only difference is that it is not sweet enough, then we should use a "sweetening operator" to reduce this difference. We use the MEA technique iteratively, starting from the initial state until we reach a goal state.

Problem reduction is the last concept. The idea consists of replacing a complex problem, which is difficult to solve at once, by a sequence of simpler subproblems. For example, if obtaining a driving license is our problem, then we should divide it into two subproblems: passing a theory test and passing a road (practical) test. Let us notice that for many problems their reduction to a sequence of simpler subproblems is necessary, because different state spaces must be defined for these subproblems and in consequence different quality functions and different operators must be determined.

2.2 Logic-Based Approach

As we have mentioned in a previous chapter, John McCarthy, who introduced a logic-based approach to AI, claimed that intelligent systems should be designed on the basis of formalized models of logical reasoning rather than as *simulators* of heuristic rules of human mental processes. This idea was revolutionary in computer science at the end of the 1950s, because of the following facts. At that time the standard methodology of designing and implementing information systems was based on the *imperative paradigm*.[4] It consists of defining a program as a sequence of commands[5] that should be performed by a computer. Thus, a computer programmer

[4]The Object-Oriented paradigm is, nowadays, the second standard approach.

[5]In Latin *imperativus* means *commanded*.

has to determine *how* computations should be performed in order to solve a problem. Meanwhile, according to McCarthy, a programmer who implements an intelligent program should determine only the required properties of a solution. In other words, he/she should specify *what* the solution of the problem is and not *how* the solution is to be obtained. The solution should be found by a universal problem solver (a generic program for solving problems). We say that such a methodology of designing and implementing information systems is based on the *declarative paradigm*.[6]

For example, if we want to write a declarative-paradigm-based program which solves the problem of determining whether two people are siblings, we can do it as follows:

$$\mathsf{siblings}(\mathsf{X}, \mathsf{Y}) :- \mathsf{parent}(\mathsf{Z}, \mathsf{X}) \ \underline{\mathsf{and}} \ \mathsf{parent}(\mathsf{Z}, \mathsf{Y}),$$

which is interpreted in the following way:

X and Y are siblings if there exists Z such that Z is a parent of X and Z is a parent of Y.

As we can see, in our program we declare only the required properties of a solution, i.e., *what it means* that two people are siblings, and we leave reasoning to a computer. Of course, we should define a database containing characteristics of people who can be objects of such reasoning.

There are two main approaches when we use the declarative paradigm in Artificial Intelligence,[7] namely logic programming and functional programming. In *logic programming* a specification of the required properties of a solution is expressed as a set of *theorems* in a logic language, usually in the language of First-Order Logic, FOL. A universal problem solver uses these theorems to perform reasoning according to FOL rules of inference. So, if we want to solve a specific problem with the help of such a solver, e.g., we want to check whether John Smith and Mary Smith are siblings, then we formulate a hypothesis of the form siblings(John Smith, Mary Smith) and the system tries to verify this hypothesis making use of facts stored in its knowledge base as well as theorems (like the one introduced above and saying what being siblings means).

Functional programming is based on *lambda calculus*, which was introduced by Alonzo Church and Stephen C. Kleene. Lambda calculus is a formal system of mathematical logic which is used for specifying computation functions defined with the help of highly formalized expressions. In functional programming a specification of the required properties of a solution is defined by just such a complex function. In this case, a universal problem solver should be able to interpret these expressions in order to perform an *evaluation* (i.e., symbolic computation) of the corresponding functions according to the principles of lambda calculus.[8]

---

[6] We *declare* required properties of a solution of a problem.

[7] This paradigm is also used nowadays beyond AI. For example, such programming languages as SQL and HTML are also based on the declarative paradigm.

[8] A more detailed description of the functional approach is included in Sect. 6.5.

2.3 Rule-Based Knowledge Representation

Newell and Simon continued research into models of cognitive processes after introducing their theory of cognitive simulation. In 1972 they proposed a *production system* model [208]. It consists of two basic components: a *production memory* and a *working memory*.

A production memory corresponds to *long-term memory* in psychology, and working memory to *short-term memory*. In long-term memory knowledge is represented with the help of *productions/rules*. A rule is of the following form: *If a certain* condition *holds, then perform a certain* action, where an action can be either a conclusion drawn from the condition or a certain action performed on the system environment (of course, only if the condition is fulfilled). Since rules are stored in long-term memory, we assume that they are available constantly.

A working memory contains information which changes in time. This is mainly information concerning the environment of the system. In one mode of the system reasoning information is checked continuously with respect to conditional parts of rules. If the condition of some rule is fulfilled,[9] then this rule is applied. If a rule application consists of drawing a certain conclusion, then this conclusion is stored in the working memory. If a rule application consists of performing a certain action on the system environment, then the system initializes this action, e.g., it switches off some device by sending a command to a processor which controls this device, sending a command to a robot actuator, which causes a movement of the robot arm, etc.

In spite of the fact that the production system model was introduced by Newell and Simon as a general theory to simulate generic cognitive processes, in Artificial Intelligence it is known mainly in a more specific version, namely as a model of an expert rule-based system. This model is presented in Chap. 9.

2.4 Structural Knowledge Representation

According to the theory of Gilbert Ryle[10] our taxonomy of knowledge includes *declarative knowledge*, which is a static knowledge concerning facts ("knowing that") and *procedural knowledge*, which is knowledge about performing tasks ("knowing how"). For example, a genealogical tree is a representation of declarative knowledge, and a heuristic algorithm, which simulates problem solving by a human being, corresponds to procedural knowledge.

Structural models of knowledge representation are used for defining declarative knowledge. They are usually in the form of graph-like hierarchical structures.

[9] In such a situation we say that the system has *matched* a certain fact (facts) stored in the working memory to the rule.

[10] Gilbert Ryle—a professor of philosophy at Oxford University, one of the most eminent representatives of analytic philosophy, the editor of the prestigious journal *Mind*.

Although originally they were used for *Natural Language Processing, NLP*, it turned out that they could be used in other AI application areas as well. *Conceptual dependency theory*, developed by Schank [263] at the end of the 1960s was one of the first such models. Schank claimed, contrary to the Chomsky generative grammar theory,[11] that a language syntax was rather a set of pointers to semantic information that could be used as a starting point for a direct semantic analysis. Conceptual dependency theory was formulated just for delivering convenient (structural) formalisms for performing a semantic analysis automatically. Since sentences of a (natural) language could not be used for performing an (automatic) semantic analysis in a direct way, Schank introduced a canonical, normalized representation[12] of semantic dependencies between language constructs (phrases, sentences).

Such a canonical representation is defined with the help of *conceptual dependency graphs*. Labeled nodes of such graphs correspond to *conceptual primitives*, which can be used for defining semantic representations. For example, a *primitive act PTRANS* means a transfer of the physical location of an object and *SPEAK* means an action generating a sound by a living object. Labeled edges of such graphs represent various relations (dependencies) between conceptual primitives. Conceptual dependency graphs are defined in an unambiguous way according to precise principles. These graphs can be analyzed in an automatic way, which allows the system to perform a semantic analysis of corresponding constructs of a natural language. Summing up, Schank has verified a hypothesis saying that we can try to perform semantic analysis automatically if we define concept-based language representations in an explicit and precise way.

Semantic networks, introduced by Collins and Quillian [56] is one of the first graph models defined on the basis of the assumptions presented above. Their nodes represent objects or concepts (classes of abstraction, categories) and their edges represent relations between them. Two specific relations are distinguished: *is subclass of* (e.g., *Triangle is subclass of Polygon*) and *is*—for denoting that some object belongs to a certain class (e.g., *John Smith is Human Being*).

Frames (frame-based systems), introduced by Minsky [203], can be treated as a substantial extension of semantic networks. A node of such a network is called a frame, and has a complex internal structure. It allows one to characterize objects and classes in a detailed way. Frame theory has the following psychological observation as a basic assumption. If somebody encounters a new unknown situation, then he/she tries to get a structure called a frame out of his/her memory. This structure, which represents a stereotyped situation similar to the current situation, can be then used for generating an adequate behavior. In AI systems a graph-like structure of frames is defined according to precise principles which allow the system to process and analyze it in an automatic way.

---

[11] The Chomsky theory is presented in the next section.

[12] A normalization of a semantic representation is necessary if it is to be performed automatically, because all sentences which have the same meaning, e.g., *John has lent a book to Mary.*, *Mary has borrowed a book from John.* should have the same representation.

Scripts have been proposed by Schank and Abelson [264] as a method for Natural Language Processing (NLP). Scripts can be defined with the help of conceptual dependency graphs introduced above. The model is also based on a certain observation in psychology. If one wants to understand a message concerning a certain event, then one can refer to a generalized pattern related to the type of this event. The pattern is constructed on the basis of similar events that one has met previously. Then it is stored in the human memory. One can easily notice that the concept of scripts is similar conceptually to the frame model.

In the past, structural models of knowledge representation were sometimes criticized for being not formal enough. This situation changed in the 1980s, when a dedicated family of formal systems based on mathematical logic called *description logics* were defined for this purpose. The foundations of description logics are presented in Appendix D, whereas a detailed presentation of semantic networks, frames, and scripts is included in Chap. 7.

2.5 Mathematical Linguistics Approach

As we have seen in previous sections, constructing a description of the physical world that can be used for intelligent system reasoning is one of the fundamental issues of Artificial Intelligence. We can try to solve this problem in a similar way as in cognitive simulation, that is to simulate a human being, which generates such a description with the help of natural language. However, the following crucial problem arises in such a case. Any natural language is an *infinite* set of sentences, which are constructed according to the principles of the corresponding grammar. Thus, automatically checking whether the syntax of a sentence is correct is a non-trivial problem.

A solution to this problem has been proposed by Noam Chomsky, who has defined *generative grammars* for the purpose of syntactic analysis of natural languages. Generative grammars are just *generators* of infinite languages. On the other hand, in Artificial Intelligence sometimes we are interested more in constructing systems that can be used for analysis of sentences than in their generation. Fortunately, *formal automata* can be used for this purpose. They are defined on the basis of generative grammars in mathematical linguistics. The task of a formal automaton can be described (in the minimalist case) as determining whether an expression is a sentence of a given language. In other words: Is the expression defined according to the principles of the grammar corresponding to the given language? In fact, a pair *generative grammar—formal automaton* can be treated as a (simple) physical symbol system.

Such a minimalist formulation of the task of an automaton (as a syntax analyzer only) can be generalized in two respects. Firstly, we can use a formal automaton for (semantic) interpretation of language sentences. Although the initial great expectations of *natural language* interpretation have not been fulfilled, in the case of *formalized languages* the Chomsky model has turned out to be very useful for an interpretation task. For the purpose of interpretation of formalized languages

the descriptive power of generative grammars has been increased considerably in Artificial Intelligence. Their standard modifications (extensions) have consisted of adding attributes to language components (*attributed grammars*) and defining "multi-dimensional" generative grammars. Standard Chomsky grammars generate sequential (string) structures, since they were defined originally in the area of linguistics. As we have discussed in the previous section, graph-like structures are widely used in AI for representing knowledge. Therefore, in the 1960s and 1970s grammars generating graph structures, called *graph grammars*, were defined as an extension of Chomsky grammars.

The second direction of research into generalizations of the formal language model concerns the task of formal language translation. A translation means here a *generalizing translation*, i.e., performing a kind of abstraction from expressions of a lower-level language to expressions of a higher-level language. Formal automata used for this purpose should be able to read expressions which belong to the basic level of a description and produce as their output expressions which are generalized interpretations of the basic-level expressions. Such automata are often called *transducers*.

The problem of automatic synthesis of formal automata is very important in Artificial Intelligence. To solve this problem *automata synthesis* algorithms, which generate the rules of an automaton on the basis of a generative grammar, have been defined. The successes in this research area have been achieved due to the development of the theory of programming language translation. At the same time, an even more fundamental problem, namely the problem of automatic *induction (inference) of a grammar* on the basis of a sample of language sentences has appeared. This problem is still an open problem in the area of Artificial Intelligence. All the issues mentioned in this section are discussed in detail in Chap. 8.

http://www.springer.com/978-3-319-40020-4

Introduction to Artificial Intelligence
Flasiński, M.
2016, X, 321 p. 70 illus., Hardcover
ISBN: 978-3-319-40020-4