

Chapter 2

Agent Versus Decomposition of an Algorithm

Abstract This chapter looks at the notions of the partial function and the Cartesian product. The presentation of the problem opens with a formal approach to the definition of the agent's properties. This part explores the reasons for the introduction of the concept of the agent and give an interpretation of such definitions as the *autonomy* of the agent or its *capability to observe the environment*.

2.1 The Genesis of an Agent

In this chapter we will try to illustrate the concept of an *autonomous agent* and especially the reasons why it was necessary to establish this notion. Briefly speaking, why the concept of the autonomous agent was invented and what should be understood by *autonomous*. Although the concept of the autonomous agent has existed in computer science for some time, it has not been clearly and precisely defined [154, 161, 174]. Particularly, there is no formal or at least more precise definition of the term *agent* and its basic characteristic features that could differentiate the agent from the object. The lack of this definition renders it difficult and often makes it impossible to carry out research on agent systems, not only in the area of formal research but also in practical applications.

There have been some attempts to solve these problems [81, 82, 98, 184], which concentrated on making reference to numerous examples illustrating notions introduced through analogy, or with reference to the analysis of the meanings of the notions, e.g., the term “autonomous”, used in the philosophical basis for the theory of evolution [122, 123]. Such an approach may intuitively have brought these notions closer, but it did not contribute to a more precise definition. Finally, the effort did not lead to satisfying results.

It may be accepted that considerations we present in this monograph, which are an attempt to analyse and find solutions to those problems, are the development of the previous suggestions or are inspired by them.

Below, we will present an attempt to define the agent based on an algorithm model, well-known from the literature [149, 180].

This approach is based on the concept of autonomy of the agent, which is considered in comparison with the concept of the object. Particularly, problems

concerning the interaction between agents (similar to those between objects) are considered here and the solution to these problems will be suggested with the use of a communication process and the operation of observation [31, 34, 60, 133].

The basic initial concept for further considerations is the idea that a particular problem (task) may be solved not by one algorithm but by a group of cooperating algorithms. In the beginning, the problem with cooperation of two (or more) cooperating algorithms appears. The model and then an attempt to define an agent will be illustrated in the following steps:

- Accepting as the starting point a general definition of an algorithm, used for solving a specified task, we will consider the possibility of application (in the simplest case two) of mutually cooperating algorithms in the realization of this task.
- Further, we will consider the problem of algorithm decomposition which is too complex (sophisticated) to be easily designed and realized; specifically we will analyze how that kind of decomposition can be realized with the use of a few cooperating simpler algorithms. The above- mentioned considerations on algorithm decomposition will make it possible to define the relationship between the cooperating algorithms.
- The analysis of these relationships between cooperating algorithms leads to providing a more formal definition of the notion *autonomy* of a particular algorithm towards other algorithms it cooperates with, as well as determining what consequences arise from the lack of this autonomy.

Summing up, the above considerations lead us to the following conclusions:

- Autonomy is not the characteristic (distinguishing) feature of the agent concept because an object, or generally speaking a component algorithm, may also be autonomous.
- The distinguishing feature of the agent concept is the capability to observe its surrounding environment, including other agents operating in a given environment, which makes that the agent acquires the property of autonomy in relation to other agents.

The analysis of these properties allows for decomposition of a particular algorithm into component algorithms such as objects or agents. By comparing the property of autonomy and the concept of encapsulation (in the sense of the object approach) it will be possible to define an object and an agent, as well as their basic distinguishing features.

2.2 The Model of an Algorithm and Problems with Its Decomposition

Let us consider the following definition of an algorithm Alg [149, 180]:

$$Alg = (U, F), \quad (2.1)$$

where U —is a non-empty set

F —is the function $F : U \rightarrow U$.

The function F is a partial function which means that the domain of the function F is a subset of the set U .

The components of the set U are called the states of algorithm Alg . The realization of algorithm Alg for the particular initial state $u^0 \in U$ (is denoted by $Exec(Alg, u^0)$) will be a finite or an infinite sequence [41]:

$$Exec(Alg, u^0) = (u^0, u^1, \dots, u^i, u^{i+1}, \dots, u^k) \quad (2.2)$$

such that

$$u^{i+1} = F(u^i), \quad u^k - \text{final state} \quad (2.3)$$

The above sequence, being the realization of the algorithm Alg , is finite if the final state u^k exists and it proceeds when the state u^k does not belong to the domain of the function F (but it belongs to the range of function F). So the final states of the algorithm Alg are those elements of the set U which belong to the domain of the function F . Further, we will take into account only these algorithms whose realization constitute the finite sequences (Formula 2.2) mentioned above.

Let us accept that the algorithm is used as a solution to a certain problem. With the use of the above denotations a given problem is represented by u^0 , however, the solution to a problem is represented by u^k and the sequence $Exec(Alg, u^0) = (u^0, u^1, \dots, u^i, u^{i+1}, \dots, u^k)$ is the schema or the method of solving a given problem.

In practical applications, when we form an algorithm which is too complicated, it is interesting to decompose it into a few simpler algorithms. The algorithm Alg will be referred to as a complex algorithm, and the algorithms it was decomposed (distributed) to—component algorithms $Alg_1, \dots, Alg_n$.

For this reason, it is necessary to define the decomposition process of a certain algorithm. We may say that a certain algorithm Alg was decomposed into component algorithms $Alg_1, Alg_2, \dots, Alg_n$, when by using component algorithms we receive the same result as with the use of a complex algorithm Alg . What is more, we expect component algorithms $Alg_1, Alg_2, \dots, Alg_n$ to be independent to such an extent that they could be created (designed, programmed) separately and in parallel—at the same time (which should accelerate the process of complex algorithms). It is possible in certain cases.

Along with further considerations we will often limit ourselves to two component algorithms Alg_a, Alg_b (sometimes referred to as Alg_1, Alg_2), which does not limit the generality of the above considerations. The problem of decomposition may be considered from different perspectives. Here, we will limit ourselves to two approaches: decomposition inspired by the division of the set of states into a few subsets (e.g. two), and decomposition inspired by the concept of the Cartesian product [180].

2.3 Decomposition Inspired by the Division of a Set

The algorithm $Alg = (U, F)$ is defined by two sets: the set of states U , and the set defining the transition between the states, which is the function F ; in other words the set of pairs (according to the theory of the set of the definition of the function).

The decomposition of the algorithm $Alg = (U, F)$ is a decomposition of the set of states U into the divided subsets (for instance two subsets U_1, U_2) and decomposition of the function F (considered as the set of pairs) into two functions F_1, F_2 :

$$U = U_1 \cup U_2, \quad U_1 \cap U_2 = \emptyset \quad (2.4)$$

satisfying the following conditions:

$$F = F_1 \cup F_2, \quad F_1 \cap F_2 = \emptyset \quad \text{and} \quad F_1 : U_1 \rightarrow U_1, \quad F_2 : U_2 \rightarrow U_2, \quad (2.5)$$

which in applications may be difficult or even impossible to realize.

The most frequent and possible decomposition in practice is the following:

$$F_1 : U_1 \rightarrow U, \quad F_2 : U_2 \rightarrow U, \quad (2.6)$$

where the sets of values of these functions is the set U .

Let us consider the problem denoted by u^0 to be solved with the use of decomposed algorithms. Then, we can accept that the algorithm $Alg = (U, F)$ is decomposed into two component algorithms $Alg_1 = (U_1 \cup \{u_2^0\}, F_1)$ and $Alg_2 = (U_2 \cup \{u_1^{l+1}\}, F_2)$.

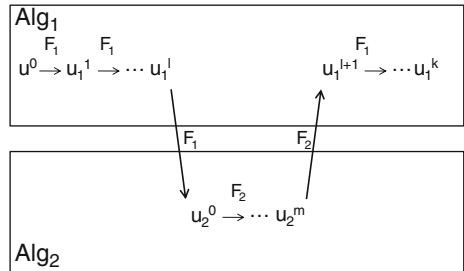
The completion of the set U_1 with the element u_2^0 and the set U_2 with the element u_1^{l+1} allows the realization of operation of calling a subprogram from the main program (*call*), and the return from the subprogram to the main program (*return*).

For a given problem with the initial value $u^0 \in U$ we may consider the following realization of a decomposed algorithm (Fig. 2.1):

$$\begin{aligned} Exec(Alg_1, u^0) &= (u_1^0, u_1^1, \dots, u_1^l, u_2^0), \\ Exec(Alg_2, u_2^0) &= (u_2^0, u_2^1, \dots, u_2^m, u_1^{l+1}), \\ Exec(Alg_1, u_1^{l+1}) &= (u_1^{l+1}, \dots, u_1^k) \\ u_2^0 &= F_1(u_1^l), \quad u_1^{l+1} = F_2(u_2^m), \end{aligned} \quad (2.7)$$

where the problem (u^0) is denoted by u_1^0 , and the result by u_k^1 .

Fig. 2.1 Schema of the decomposition of an algorithm through the division of the set U



We may consider this decomposition as follows: $Alg = (U, F)$ has been decomposed into the “main” algorithm $Alg_1 = (U_1, F_1)$ and the algorithm $Alg_2 = (U_2, F_2)$, which represents the subprogram or the service (depending on the perspective).

However, the function $F_1(u_1^l)$ represents the operation of calling the subprogram and the function $F_2(u_2^m)$ represents the return from the subprogram to the main program (an algorithm). The result of the decomposition is the cooperation of the algorithms Alg_1 i Alg_2) as the call through the main (calling) algorithm Alg_1 of the called algorithm Alg_2 . Summing up, this idea leads us to decomposition of an algorithm or rather decomposition of a program into subprograms—the procedures and functions known and used in programming.

2.4 Decomposition Inspired by the Concept of the Cartesian Product

In order to define the elements u of the certain set U ($u \in U$) we may use characteristic features of a given element u . It is connected with the fact that in practice it is easier to describe an element of a set with the use of a given set of features which take values from this defined set (e.g., the set of natural numbers, real numbers, etc.) These features may now be considered as the variables taking values from the defined sets (Fig. 2.2).

Then, we assume that the elements of the set U are associated with the elements of a given set X , that represent the sets of features characteristic of these elements x ($x \in X$), so they have the form of n -tuples:

$$u^i \text{ is equivalent to } x^i = (x_1^i, x_2^i, \dots, x_n^i), \quad (2.8)$$

where $u^i \in U$,

$$x^i \in X = X_1 \times X_2 \times \dots \times X_n.$$

Each element x_j^i defines the characteristic feature (or attribute) j of the element u^i (or corresponding to its element x^i). In consequence, instead of the set U we may use

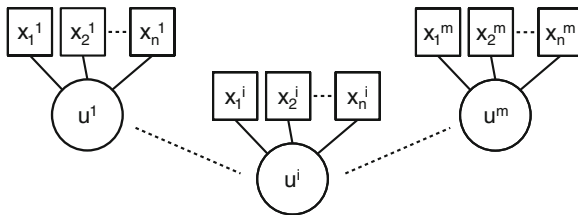


Fig. 2.2 Schema of the Cartesian product application—in the form of features of the elements of the set U ($(x_1^i, x_2^i, \dots, x_n^i)$)—the set of features describing the element u^i)

in further considerations the Cartesian product $X = X_1 \times X_1 \times \dots \times X_n$ (considered as a n-tuple x^i of variables x_j^i describing characteristic features of the element u^i) [69, 150, 151].

The concept of the Cartesian product may be applied not only to the set U but also to the algorithm Alg .

It means that through the algorithm $Alg = (U, F)$, we may consider decomposition of this algorithm, applying the concept of the Cartesian product to the algorithm Alg , or to the set of states of the algorithm U .

2.4.1 The Decomposition of an Algorithm Based on the Cartesian Product Versus Problem of Autonomy

Let us consider the Cartesian product $X = X_1 \times X_2 \times \dots \times X_m$, where the set X is associated with the set U . Considering the function F :

$$F(u^k) = u^{k+1} \quad (2.9)$$

and using the notation

$$u^k \text{ corresponds } x^k, \quad u^{k+1} \text{ corresponds } x^{k+1}, \quad (2.10)$$

where $x^k = (x_1^k, x_2^k, \dots, x_m^k)$, $x^{k+1} = (x_1^{k+1}, x_2^{k+1}, \dots, x_m^{k+1})$

it is noticeable that the partial function F may be replaced with the function (also partial) $f : X \rightarrow X$ whose domain is the set defined on the basis of characteristic features. The function f is defined by

$$f(x_1^k, x_2^k, \dots, x_m^k) = (x_1^{k+1}, x_2^{k+1}, \dots, x_m^{k+1}) \Leftrightarrow F(u^k) = u^{k+1}. \quad (2.11)$$

In further considerations without the loss of generality we may limit the decomposition of the set U into only to two sets X_1 and X_2 . The constraint to only two sets does not limit further considerations and all of the significant problems may be further successfully analyzed. The set X is the Cartesian product of two sets $X = X_1 \times X_2$.

This limitation may be treated as the result of grouping the elements of the Cartesian product: $\mathbf{X} = \mathbf{X}_1 \times \mathbf{X}_2$, where $\mathbf{X}_1 = X_1 \times X_2 \times \dots \times X_i$, $\mathbf{X}_2 = X_{i+1} \times X_{i+2} \times \dots \times X_m$ (Fig. 2.3).

The projection of the set X onto the set X_1 and X_2 may be introduced by

$$\begin{aligned} Proj_1 : X_1 \times X_2 &\rightarrow X_1 \quad \forall_{(x_1, x_2) \in X_1 \times X_2} \quad Proj_1(x_1, x_2) = x_1 \\ Proj_2 : X_1 \times X_2 &\rightarrow X_2 \quad \forall_{(x_1, x_2) \in X_1 \times X_2} \quad Proj_2(x_1, x_2) = x_2 \end{aligned} \quad (2.12)$$

The decomposition may now come down to the fact that it is necessary to form two component algorithms Alg_1 and Alg_2 as in Fig. 2.4. For this reason, it is necessary to

Fig. 2.3 Schema of the use of grouping to the Cartesian product in order to constrain the dimension of the problem

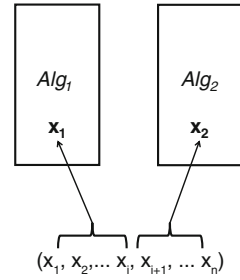
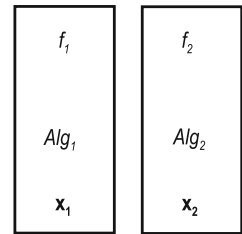


Fig. 2.4 Schema of the Cartesian product application to the decomposition of an algorithm



define the sets of states of these algorithms as well as their (partial) functions of the transition.

The following problems and questions appear:

- The component algorithms Alg_1 and Alg_2 resulting from the decomposition of the algorithm Alg should have defined sets of states X_1 and X_2 , specified with the use of the set of states of the algorithm Alg . Decomposition of the sets of states X based on the concept of the Cartesian product (Fig. 2.3) may be the starting point of realizing such decomposition.
- Another problem is that the algorithms should have the transition partial functions f_1 and f_2 , which should be formed on the basis of the transition function f , so the function f should be decomposed into two functions f_1 and f_2 . That will allow the algorithms $Alg_1 = (X_1, f_1)$ and $Alg_2 = (X_2, f_2)$ to create, with the appropriate transition functions (Fig. 2.4).
- It should also be analyzed whether the algorithms Alg_1 and Alg_2 are related to each other through mutual interactions. In particular, the mutual relationship of the algorithms should be analyzed. We need to define what the notion *autonomous* means (or should mean) and how we should really understand it. In other words, how the notion *autonomous* should be defined to make it clear that we deal with the *autonomous algorithm*.
- Further, it should be considered whether it is possible (and to what extent) to make algorithms Alg_1 and Alg_2 independent so that they can be autonomous algorithms.
- The question arises as to whether the decomposition of the algorithm Alg into the algorithms Alg_1 and Alg_2 ensures that they may solve the problems that are solved with the use of the algorithm $Alg = (X, f)$. The term of equivalence of algorithms, which is considered in Sect. 2.5.1, seems to be useful here.

In further considerations, we will try to answer these questions and solve the problems, and particularly discuss the notion of autonomy, define it more precisely and show that by using appropriate methods it is possible to realize decomposition of a given algorithm into component algorithms that are considered as autonomous.

2.4.2 The Autonomy of an Algorithm

In our considerations as well as in the literature in the field, we may encounter a statement that an algorithm is (or is not) autonomous. The notion has been introduced intuitively (see [174, 185]).

Let us try to define more precisely what should be understood by the notion autonomous and let us accept the following statement: *The notion of autonomy of a certain algorithm may only be considered towards another algorithm, which means that the autonomy of the algorithm Alg_1 may be considered towards the algorithm Alg_2 .* However, neither the autonomy of an algorithm can be defined without taking into consideration other algorithms, nor can the autonomy be defined only towards the environment (Fig. 2.5).

In order to define the notion of autonomy, we consider two algorithms $Alg_1 = (X_1 \times X_2, f_1)$ and $Alg_2 = (X_1 \times X_2, f_2)$. The transition function f_1 is denoted by $f_1 : X_1 \times X_2 \rightarrow X_1 \times X_2$, similarly the function f_2 is denoted by $f_2 : X_1 \times X_2 \rightarrow X_1 \times X_2$.

It should be emphasized that the function f_1 as well as the function f_2 are partial functions. Let us denote the domain of the function f_1 as Df_1 and the domain of the function f_2 as Df_2 .

The domain Df_1 and Df_2 are the subsets of the set $X_1 \times X_2$. Let us try to define what it means that the algorithm Alg_1 is autonomous (or non-autonomous) towards

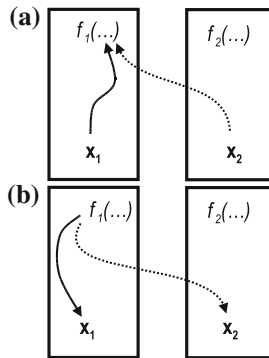


Fig. 2.5 Schema of the relationships between the algorithms; the case when the algorithm Alg_1 is not *autonomous* towards the algorithm Alg_2 (the interoperating relationship), **a** for calculating the transition function f_1 both states are indispensable; the state of the algorithm Alg_1 as well as the state of the algorithm Alg_2 , **b** the calculation of the transition function f_1 has an influence on the change of states of the algorithms Alg_1 and Alg_2

the algorithm Alg_2 . The problem of autonomy should be considered, taking account of the cooperation of algorithms, and the forms of the functions f_1 and f_2 are crucial.

The following cases of the autonomy of the algorithm Alg_1 towards the algorithm Alg_2 may be considered:

- The algorithm Alg_1 is autonomous towards the algorithm Alg_2 if for the transition function f_1 of the algorithm Alg_1 the following relationships occur:

$$\forall_{(x_1, x_2) \in Df_1, (x_1, x'_2) \in Df_1} : f_1(x_1, x_2) = f_1(x_1, x'_2) \quad (2.13)$$

and

$$\forall_{(x_1, x_2) \in Df_1} : (Proj_2 \circ f_1)(x_1, x_2) = x_2, \quad (2.14)$$

which can be briefly (informally) denoted by:

$$f_1 : X_1 \rightarrow X_1. \quad (2.15)$$

It means that the state of the algorithm Alg_1 is only needed for the calculation of the transition function f_1 whereas the state of algorithm Alg_2 is not necessary. However, the calculation of the transition function f_1 influences only the change of the state of the algorithm Alg_1 and does not have any influence on the state of the algorithm Alg_2 . The schema of that relationship is shown in Fig. 2.6.

- The algorithm Alg_1 is not autonomous towards the algorithm Alg_2 with *inter-information* relationship if for the transition function f_1 of the algorithm Alg_1 the following relationships occur:

$$\exists_{(x_1, x_2) \in Df_1, (x_1, x'_2) \in Df_1, (x_2 \neq x'_2 \Rightarrow f_1(x_1, x_2) \neq f_1(x_1, x'_2))} \quad (2.16)$$

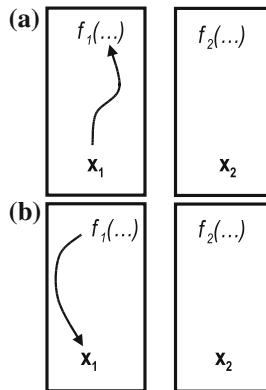


Fig. 2.6 Schema of the relationships between the algorithms; the case when the algorithm Alg_1 is *autonomous* towards the algorithm Alg_2 , **a** for calculating the transition function f_1 only the state of the algorithm Alg_1 is necessary, **b** the calculation of the transition function f_1 influences only the change of the state of the algorithm Alg_1 , and does not have any influence on the state of the algorithm Alg_2

and

$$\forall_{(x_1, x_2) \in Df_1} : (Proj_2 \circ f_1)(x_1, x_2) = x_2 \quad (2.17)$$

which can be briefly (informally) denoted by

$$f_1 : X_1 \times X_2 \rightarrow X_1. \quad (2.18)$$

It means that for the calculation of the transition function f_1 both states—the state of the algorithm Alg_1 as well as the state of the algorithm Alg_2 —are necessary, however, the calculation of the transition function f_1 influences only the change of the state of the algorithm Alg_1 and does not have any influence on the state of the algorithm Alg_2 . The schema of this relationship is shown in Fig. 2.7.

- The algorithm Alg_1 is not autonomous towards the algorithm Alg_2 , with *interaction* relationship if for the transition function f_1 of the algorithm Alg_1 the following relationships occur:

$$\forall_{(x_1, x_2) \in Df_1, (x_1, x'_2) \in Df_1} : f_1(x_1, x_2) = f_1(x_1, x'_2) \quad (2.19)$$

and

$$\exists_{(x_1, x_2) \in Df_1} : (Proj_2 \circ f_1)(x_1, x_2) \neq x_2 \quad (2.20)$$

which can be briefly (informally) denoted by

$$f_1 : X_1 \rightarrow X_1 \times X_2. \quad (2.21)$$

It means that for the calculation of the transition function f_1 only the state of the algorithm Alg_1 is necessary, however, the calculation of the transition function

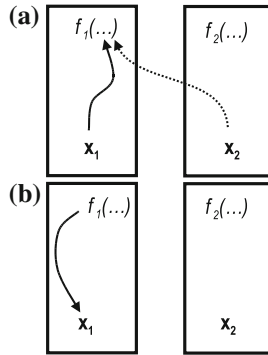


Fig. 2.7 Schema of the relationships between the algorithms; the case when the algorithm Alg_1 is not *autonomous* towards the algorithm Alg_2 with *inter-information* relationship, **a** for calculating the transition function f_1 both states—the state of the algorithm Alg_1 as well as the state of the algorithm Alg_2 —are necessary, **b** the calculation of the transition function f_1 influences only the change of the state of the algorithm Alg_1 , and does not have any influence on the state of the algorithm Alg_2

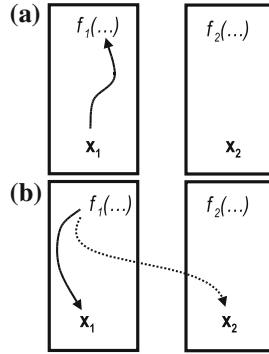


Fig. 2.8 Schema of the relationships between the algorithms; the case when the algorithm Alg_1 is not autonomous towards the algorithm Alg_2 with *interaction* relationship, **a** the sources of data indispensable to calculate the transition function f_1 , **b** the influence of the transition function f_1 on the change of the state of algorithms

f_1 influences both—the change of the state of the algorithm Alg_1 as well as the change of the state of the algorithm Alg_2 . The schema of that relationship is shown in Fig. 2.8.

- The algorithm Alg_1 is not autonomous toward the algorithm Alg_2 , with *interoperating* relationship (or completely non-autonomous) if for the transition function f_1 of the algorithm Alg_1 the following relationships occur:

$$\exists_{(x_1, x_2) \in Df_1, (x_1, x'_2) \in Df_1} (x_2 \neq x'_2 \Rightarrow f_1(x_1, x_2) \neq f_1(x_1, x'_2)) \quad (2.22)$$

and

$$\exists_{(x_1, x_2) \in Df_1} : (Proj_2 \circ f_1)(x_1, x_2) \neq x_2, \quad (2.23)$$

which can be briefly (informally) denoted by

$$f_1 : X_1 \times X_2 \rightarrow X_1 \times X_2. \quad (2.24)$$

It means that for the calculation of the transition function f_1 , both states are necessary—the state of the algorithm Alg_1 as well as the state of the algorithm Alg_2 —however, the calculation of the transition function f_1 influences both, the change of the state of the algorithm Alg_1 as well as the change of the state of the algorithm Alg_2 . The interoperating relationship exists if both—the inter-information relationship as well as the interaction relationship—take place at the same time.

The algorithm Alg_1 is not autonomous towards the algorithm Alg_2 if there is an inter-information, interaction or interoperating relationship.

The algorithms Alg_1 and Alg_2 are mutually autonomous if the algorithm Alg_1 is autonomous towards the algorithm Alg_2 and the algorithm Alg_2 is autonomous towards the algorithm Alg_1 .

Example 1

Let us consider the following example of an algorithm:

$$\begin{aligned} U &= \{a, b, c, d\}, \\ F(a) &= b, \quad F(b) = c, \quad F(c) = d \end{aligned} \quad (2.25)$$

Let us apply decomposition, using the concept of the Cartesian product applied to the set U :

$$\begin{aligned} X_1 &= \{0, 1\}, \quad X_2 = \{0, 1\} \\ X &= X_1 \times X_2 = \{(0, 0), (0, 1), (1, 0), (1, 1)\} \end{aligned} \quad (2.26)$$

We may define the following correspondence between the elements of the set U and the set X :

$$\begin{aligned} a &\text{ corresponds with } (0, 0), \quad b \text{ corresponds with } (0, 1), \\ c &\text{ corresponds with } (1, 0), \quad d \text{ corresponds with } (1, 1) \end{aligned} \quad (2.27)$$

The definition of the function f may be presented as follows:

$$f((0, 0)) = (0, 1), \quad f((0, 1)) = (1, 0), \quad f((1, 0)) = (1, 1) \quad (2.28)$$

The component algorithms Alg_1 and Alg_2 may be defined as follows—the algorithm Alg_1 :

$$\begin{aligned} Alg_1 &= (X, f_1) \\ f_1(0, 1) &= (1, 0) \end{aligned} \quad (2.29)$$

and the algorithm Alg_2 :

$$\begin{aligned} Alg_2 &= (X, f_2) \\ f_2(0, 0) &= (0, 1), \quad f_2(1, 0) = (1, 1) \end{aligned} \quad (2.30)$$

We note that for the algorithm Alg_2 :

$$\forall_{z \in X_1} : f_2(z, 0) = (z, 1) \quad (2.31)$$

(which allows to state that the algorithm Alg_2 is autonomous towards the algorithm Alg_1).

However, Alg_1 is non-autonomous (interaction relationship) because it changes the state x_2 of the algorithm Alg_2 .

The function f_1 or f_2 is sometimes informally denoted by $f_2 = (\bullet, 0) = (\bullet, 1)$, where $\bullet$ denotes an optional value (in example 0 or 1).

However, if we change the functions and the component algorithms will be defined as follows:

$$\begin{aligned} Alg_1 &= (X, f_1) \quad f_1(0, 0) = (0, 1), \quad f_1(1, 0) = (1, 1) \\ Alg_2 &= (X, f_2) \quad f_2(0, 1) = (1, 0) \end{aligned} \quad (2.32)$$

then neither the algorithm Alg_1 is autonomous towards the Alg_2 nor the algorithm Alg_2 is autonomous towards the Alg_1 . As we can see from the example, gaining autonomy of one component algorithm towards another component algorithm depends on the appropriate decomposition of the initial complex algorithm.

2.5 Decomposition with the Use of the Concept of the Cartesian Product Applied to the Set U

Let us consider decomposition of the algorithm Alg ($Alg = (U, F)$), applying the Cartesian product to the set of states U . Such a decomposition method seems to be useful in practical applications, nevertheless, it results in component algorithms which may often be mutually dependent and therefore that is more difficult for parallel (independent) forming. As was mentioned above, the set X was associated with the set of states U , which was the Cartesian product $X = X_1 \times X_2$ (in the general case $X = X_1 \times X_2 \times \dots \times X_m$, see Sect. 2.4). The function f , according to the set theory definition of the function notion, may be considered as a set of pairs (Sect. 2.4.1):

$$(x, x') \text{ such that } f(x) = x',$$

$$\begin{aligned} \text{where } x &= (x_1, x_2) \in X \\ x' &= (x'_1, x'_2) \in X \\ X &= X_1 \times X_2 \end{aligned}$$

As a result, we may divide the function f into two disjoint subsets (generally, a few disjoint subsets), each of them represents a certain component function being the result of decomposition (division) of the initial function f (Fig. 2.4).

Such functions as f_1 and f_2 map the set X onto X ($f_1 : X \rightarrow X, f_2 : X \rightarrow X$), and the function f is decomposed into two functions f_1 and f_2 such that $f = f_1 \cup f_2$, $f_1 \cap f_2 = \emptyset$. Because of the fact that the function F in the algorithm $Alg = (U, F)$ is a partial function, the functions f and f_1 as well as f_2 are also partial functions.

As a result, we receive two algorithms $Alg_1 = (X, f_1)$ and $Alg_2 = (X, f_2)$ for which x_1 and x_2 may be treated as variables gaining the values from the sets X_1 and X_2 that realize the evolution of the states of the algorithms Alg_1 and Alg_2 .

These two algorithms may be treated as the decomposition of the algorithm $Alg = (U, F)$ (or corresponding to it algorithm $Alg = (X, f)$) realized with the use of the Cartesian product applied to the set U .

Of course, from the practical point of view, in order to let the algorithms Alg_1 and Alg_2 be created independently (designed, programmed) these algorithms should be mutually autonomous. Their transition functions would have the following form: $f_1 : X_1 \rightarrow X_1$ and $f_2 : X_2 \rightarrow X_2$.

In practice these functions have the form $f_1 : X \rightarrow X$ and $f_2 : X \rightarrow X$ for $X = X_1 \times X_2$, which makes the component algorithms $Alg_1 = (X, f_1)$ and $Alg_2 = (X, f_2)$ non-autonomous.

We may improve this relationship of algorithms and so that the algorithms Alg_1 and Alg_2 become autonomous (mutually). It can be realized in two steps, which will be described further in later chapters.

2.5.1 Step One—The Introduction of the Environment Concept

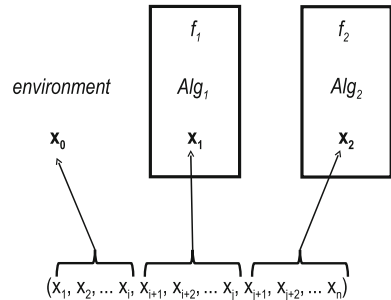
The first step is to extract a certain part of variables as the global variables, which involves the introduction of an environment, described by these global variables representing the state of that environment.

The inspiration for the introduction of the environment concept is derived from the observation of mobile robots' action. Their environment plays a crucial role in robots' action and it has an influence on the design of their algorithms of action, in a sense, the concept of the environment has accompanied the development of computer science sine operating systems were introduced. It was operating systems that constituted and constitute the environment for the realization of programs (algorithms) of the user. The introduction of the environment concept is realized as follows:

- We choose certain variables and consider them as the parameters describing the state of the environment. These variables (a set of these variables denoted by X_0) are available to each partial algorithm (see Alg_1 and Alg_2 in Fig. 2.9).
- Other variables are grouped in such a way that each group (of variables) may be considered as variables describing the state (also referred to as internal state) of individual partial algorithm (X_1, X_2 in Fig. 2.9).

We may now consider the component algorithms $Alg_1 = (X_0 \times X_1 \times X_2, f_1)$ and $Alg_2 = (X_0 \times X_1 \times X_2, f_2)$ cooperating (described in the next subchapter) in the environment X_0 . The introduction of the environment, the state which is available to both algorithms and can be changed by these algorithms, may create the impression that the algorithms to some extent become mutually dependent and lose autonomy towards each other. Nevertheless, new possibilities appear. Such dependence through the environment allows cooperation between the algorithms and a replacement of one algorithm for another, an equivalent one.

Fig. 2.9 Schema of decomposition of the set X which results in the partial algorithms Alg_1 and Alg_2 and the state of the environment (X_0)



It is necessary to analyze again and define such a property as the autonomy of one algorithm towards another it can cooperate with through the environment. Therefore, such terms as the cooperation between algorithms, the equivalence of algorithms, and the autonomy of algorithms should be considered, analyzed and updated.

2.5.1.1 The Cooperation Between the Algorithms

To analyze more complex methods of decomposition of algorithms we have to define precisely the term of autonomy which requires considering a cooperation process of algorithms. Therefore, let us consider two cooperating algorithms (X_0, Alg_1, Alg_2) . The set X_0 represents global data describing the state of the environment available to both cooperating algorithms. The algorithms have the following form:

$$\begin{aligned} Alg_1 &= (X_0 \times X_1, f_1), & Alg_2 &= (X_0 \times X_2, f_2), \\ f_1 &: X_0 \times X_1 \rightarrow X_0 \times X_1, & f_2 &: X_0 \times X_2 \rightarrow X_0 \times X_2 \end{aligned} \quad (2.33)$$

The concept of the environment used for the realization of cooperation between the algorithms may be presented with a scenario of cooperation of algorithms, which is shown in Fig. 2.10.

Let us accept that a problem is a pair $(x_0^0, \text{ and } x_0^k)$ where x_0^0 is the initial state, being the task of the problem, and x_0^k is the final state, representing a solution to the problem. It is necessary to emphasize that x_0^0 and x_0^k are the states of the environment and thus are not the states of either of algorithm.

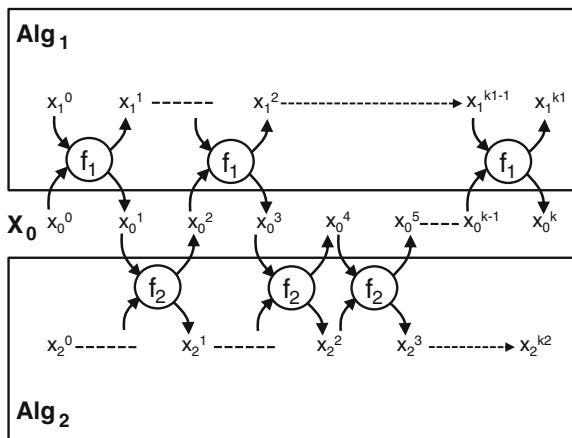


Fig. 2.10 The concept of the *environment* and the model of cooperation between the algorithms Alg_1 and Alg_2 through the environment X_0

The algorithm solves this problem if its application leads from the initial to the final state. This problem may be solved by two algorithms cooperating through the environment in the following way (Fig. 2.10):

- Let us accept that a problem which has to be solved is modelled with the use of the state of the environment X_0 and that the solution to the problem has to be achieved by two cooperating algorithms $Alg_1 = (X_0 \times X_1, f_1)$ and $Alg_2 = (X_0 \times X_2, f_2)$. It is necessary to adjust the problem in such a way that it could be solved by the algorithms that cooperate with each other by the appropriate encoding the problem (task) in the form of the chosen state of the environment $x_0^0 \in X_0$.
- Further, it is necessary to prepare the cooperating algorithms for an action properly choosing the initial internal states of the algorithms, which for Alg_1 may be denoted by x_1^0 , and x_2^0 for the algorithm Alg_2 . These states should be chosen in such a way that there would be a situation in which either the pair (x_0^0, x_1^0) belongs to the domain of the function f_1 , and then the algorithm Alg_1 starts the “action”, or the pair (x_0^0, x_2^0) belongs to the domain of the function f_2 , which makes the algorithm Alg_2 perform the first “move”.
- Cooperation in problem solving involves transformation of the state of the environment from the initial into the final state in the appropriate order made either by the algorithm Alg_1 or the algorithm Alg_2 . For example, in Fig. 2.10 we may observe a situation in which during the cooperation of the algorithms we have the state of the environment x_0^2 , and the pair (x_0^2, x_1^1) belongs to the domain of the function f_1 , which makes the algorithm Alg_1 undertake an “action” and transform the environment into the state x_0^3 . However, the state (x_0^3, x_2^1) belongs to the domain of the function f_2 , which makes the algorithm Alg_2 transform the state of the environment.

Synchronization of cooperation between the algorithms takes place as a result of appropriate states absorbed by the environment and algorithms. In practice, cooperating algorithms may not only be used for describing of different kinds of cooperation between algorithms, but also for decomposing more complex algorithms into autonomous component algorithms, which will be presented further in later chapters.

2.5.1.2 The Relationship and Equivalence of Cooperating Algorithms

We may consider the decomposition of an algorithm into cooperating component algorithms, presented in the previous chapter, in three basic cases:

- The algorithms $Alg_1 = (X_0 \times X_1, f_1)$ and $Alg_2 = (X_0 \times X_2, f_2)$ cooperate through the environment, however, some changes in the environment are realized by one algorithm and then by the other one (Fig. 2.10), as they are both necessary for solving a task.
- The algorithms $Alg_1 = (X_0 \times X_1, f_1)$ and $Alg_2 = (X_0 \times X_2, f_2)$, similar to the previous case, can affect the environment but each of them is able to make some changes independently in the environment and solve a given problem. It means there are

such internal states of algorithms as x_1^0 and x_2^0 that the realization of algorithms $Exec(Alg_1, (x_0^0, x_1^0))$ and $Exec(Alg_2, (x_0^0, x_2^0))$ exists in the following form:

$$Exec(Alg_1, (x_0^0, x_1^0)) = ((x_0^0, x_1^0), (x_0^1, x_1^1), \dots, (x_0^{k_1}, x_1^{k_1}))$$

$$Exec(Alg_2, (x_0^0, x_2^0)) = ((x_0^0, x_2^0), (x_0^1, x_2^1), \dots, (x_0^{k_2}, x_2^{k_2}))$$

If additionally it occurs that $x_0^{k_1} = x_0^{k_2}$, then these algorithms are equivalent in a given problem solving (internal states of the algorithms $x_1^{k_1}$ and $x_2^{k_2}$ are of no importance here, and only the final state of the environment is essential). If a solution to a given problem x_0^0 through the realization of the first algorithm Alg_1 is equivalent to the solution to this problem through the realization of the algorithm Alg_2 , then we may replace the algorithm Alg_1 with the equivalent algorithm Alg_2 and choose the one which is, e.g., faster or makes better use of the resources.

- The algorithms $Alg_1 = (X, f_1)$ and $Alg_2 = (X, f_2)$ cooperate through the environment but $X = X_0 \times X_1 \times X_2$. It means that certain changes in the environment, those necessary for problem solving, are realized by the former algorithm, and the other ones by the latter (Fig. 2.10), but simultaneously the algorithm Alg_1 affects the changes of the state x_2 of the algorithm Alg_2 and vice versa. It may be noted that the algorithm Alg_1 (or Alg_2) is non-autonomous, then it is necessary to broaden the definition of autonomy to algorithms cooperating through the environment.

The concept of the environment and the cooperation of algorithms through the environment, under the definition of autonomy presented above, is that the cooperating algorithms are not autonomous but dependent through the environment. However, apart from the states of the environment there are internal states of individual algorithms, and therefore we may broaden definitions of autonomy introduced earlier.

On the other hand, the concept of the environment allows consideration of the phenomenon of communication through the environment. For example, the algorithm Alg_1 may make such changes in the environment that can be “read” by the algorithm Alg_2 as changes connected with certain established information, which is a kind of sending a message by the algorithm Alg_1 to the algorithm Alg_2 . This way of communication allows for more complex forms of cooperation such as negotiations, planning and forming groups of cooperating algorithms, or realization of cooperation as such.

2.5.1.3 The Autonomy of Cooperating Algorithms Based on the Environment

The notion of *autonomy* has been defined and discussed in Sect. 2.4.2, but after the introduction of the environment concept and the possibility of cooperation between the algorithms based on these environments it needs certain modification.

Similarly the notion of autonomy of a given algorithm may be considered only towards another algorithm. For example, the autonomy of the algorithm Alg_1 may be considered towards the algorithm Alg_2 .

We may then, as in Sect. 2.4.2, introduce the following cases of the autonomy of the algorithm Alg_1 towards the algorithm Alg_2 , which both cooperate through the environment X_0 , taking into consideration the influence of the algorithm on this environment (simplified informal notation has been applied to formulas):

- The algorithm Alg_1 is autonomous if the function f_1 has the following form:

$$f_1 : X_0 \times X_1 \rightarrow X_0 \times X_1 \quad (2.34)$$

It means that for the calculation of the transition function f_1 only the state of the algorithm Alg_1 and the state of the environment X_0 are necessary and the state of the algorithm Alg_2 is not necessary at all. However, the calculation of the transition function f_1 influences only the change of state of the algorithm Alg_1 and the state of the environment X_0 , and does not influence the state of the algorithm Alg_2 .

- The algorithm Alg_1 is not autonomous with *inter-information* relationship towards the algorithm Alg_2 if the function f_1 has the following form:

$$f_1 : X_0 \times X_1 \times X_2 \rightarrow X_0 \times X_1 \quad (2.35)$$

It means that for the calculation of the transition function f_1 the state of the algorithm Alg_1 , the state of the algorithm Alg_2 and the state of the environment X_0 are necessary. However, the calculation of the transition function f_1 influences only the change of state of the algorithm Alg_1 and the state of the environment X_0 and does not influence the state of the algorithm Alg_2 .

- The algorithm Alg_1 is not autonomous with *inter-action* relationship towards the algorithm Alg_2 if the function f_1 has the following form:

$$f_1 : X_0 \times X_1 \rightarrow X_0 \times X_1 \times X_2 \quad (2.36)$$

It means that for the calculation of the transition function f_1 only the state of the algorithm Alg_1 and the state of the environment X_0 are essential and the state of the algorithm Alg_2 is not necessary at all. However, the calculation of the transition function f_1 influences the change of state of the algorithm Alg_1 , the state of the algorithm Alg_2 and the state of the environment X_0 .

- The algorithm Alg_1 is not autonomous with *inter-operation* relationship towards the algorithm Alg_2 (or completely non-autonomous) if the function f_1 has the following form:

$$f_1 : X_0 \times X_1 \times X_2 \rightarrow X_0 \times X_1 \times X_2 \quad (2.37)$$

It means that for the calculation of the transition function f_1 the state of the algorithm Alg_1 , the state of the algorithm Alg_2 and the state of the environment X_0 are essential. However, the calculation of the transition function f_1 influences the

change of state of the algorithm Alg_1 , the state of the algorithm Alg_2 and the state of the environment X_0 .

Summing up, it can be informally said that a given algorithm (e.g., Alg_1) is autonomous towards the other algorithm (e.g., Alg_2) when in order to appoint its next state, apart from the information about its own state, the only thing it needs is the information about the state of the environment, and the information about the state of the other algorithm is not necessary.

However, the algorithm Alg_1 is not autonomous towards the algorithm Alg_2 if there is an inter-information relationship, inter-action relationship or inter-operation relationship.

We may say that the algorithms Alg_1 and Alg_2 , cooperating through the environment X_0 , are mutually autonomous if the algorithm Alg_1 is autonomous towards the algorithm Alg_2 and the algorithm Alg_2 is autonomous towards the algorithm Alg_1 . If we deal with a large number of cooperating algorithms, then the property of autonomy may be extended to the whole group. A given algorithm is autonomous towards the whole group of algorithms if it is autonomous towards each algorithm in the group.

2.5.1.4 Summary of the Autonomy Problem and Algorithm Decomposition

A given problem denoted as the state of the environment $x^0 \in X$, whose solution is the state of the environment $x^k \in X$, may also be solved by two cooperating algorithms Alg_1 and Alg_2 (Fig. 2.10).

We may consider the algorithm $Alg = (X, f)$ which is decomposed into two component, autonomous, cooperating algorithms $Alg_1 = (X_1, f_1)$ and $Alg_2 = (X_2, f_2)$, and the set of parameters X_0 , representing the state of the environment, i.e., the global data. The sets X_1 and X_2 correspond to the internal data of the appropriate algorithms Alg_1 and Alg_2 which define their states and also X_0 —the global data. The partial functions f_1 and f_2 define the action of the algorithms, i.e., the evolution of their states. We may consider the following cases of the autonomy influence on the realization of decomposition:

- The algorithms Alg_1 and Alg_2 are mutually autonomous, which means that the functions accept the following forms: $f_1 : X_0 \times X_1 \rightarrow X_0 \times X_1$ and $f_2 : X_0 \times X_2 \rightarrow X_0 \times X_2$. In this case component algorithms may be formed (designed, realized) independently, separately and at the same time (Fig. 2.10).
- The component algorithms are not mutually autonomous, which means that there is inter-information, inter-action or inter-operation relationship. The algorithm Alg_1 needs for the assignment of its next state not only knowledge about the state of an environment but also information on the state of the other algorithm (in this case Alg_2), and changing its state it modifies not only the state of the environment but also the state of the algorithm Alg_2 . This case can be often found in practice, and

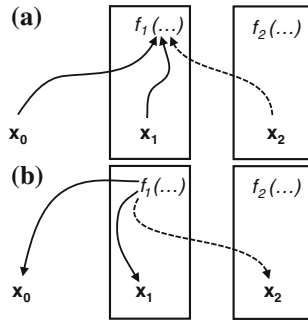


Fig. 2.11 Schema of decomposition of an algorithm based on the notion of autonomy, **a** for the calculation of the transition function f_1 the state of the algorithm Alg_1 , the state of the algorithm Alg_2 and the state of the environment X_0 are all indispensable, **b** calculation of the transition function f_1 has an influence on the change of states of the algorithms Alg_1 and Alg_2 and the change of state of the environment X_0

then decomposition of a given algorithm into component algorithms does not give possibilities to form component algorithms independently (and especially parallel in time).

The above-presented definition of the autonomy of cooperating algorithms has the principal meaning for further considerations.

In practice there are cases when the implementation of the concept of the environment does not solve completely the problem of algorithm decomposition into component algorithms. Although decomposition of a given algorithm Alg into component, mutually autonomous algorithms is not always possible (Fig. 2.11), decomposition into component algorithms, which not necessarily have to be mutually autonomous, is easier and more often possible.

The question arises whether it is possible to use that kind of “non-autonomous” decomposition as a starting point for finding a method of reducing (bringing) cooperating, non-autonomous, component algorithms to mutually autonomous (at least in some scope) algorithms.

These methods will be presented as another step of decomposition of algorithms and will be dealt with in later chapters.

2.5.2 Step Two-Modes of Access to Internal Data of Another Algorithm

Let us accept that a given algorithm $Alg = (X, f)$ may be decomposed into two component algorithms $Alg_1 = (X_1, f_1)$ and $Alg_2 = (X_2, f_2)$ cooperating through the external environment X_0 . In effect, a given problem encoded as $x^0 \in X_0$ and solved with the use of the algorithm Alg may also be solved with two cooperating algorithms Alg_1 and Alg_2 (Sect. 2.5, Fig. 2.10).

If cooperating algorithms Alg_1 and Alg_2 are mutually autonomous, then solving a problem x^0 with these algorithms does not cause any difficulties, as shown earlier.

However, a problem arises when the cooperating, component algorithms are not autonomous. It results from the fact that the algorithm Alg_1 (properly Alg_2) needs but has no access to the internal (local) data of the algorithm Alg_2 (properly Alg_1), so it is unable to read the value of variables which are essential for the calculation of its function f_1 (properly f_2) and for making the next step (Fig. 2.11).

However, there are possibilities of ensuring access of the algorithm Alg_1 (properly Alg_2) to the internal (local) data of the algorithm Alg_2 (properly Alg_1). It comes down to the replacement of algorithms which are mutually non-autonomous with the algorithms that gain autonomy to some extent, which will be presented further.

Two methods may be considered which enable access of one algorithm to the internal (local) data of the other algorithm, that is to say that non-autonomous algorithms are replaced with autonomous algorithms:

- with the use of *communication process* between the algorithms,
- with the use of *observation operation* of action of one algorithm (behaviour) by the other algorithm.

These methods are presented in later parts of the monograph.

2.5.2.1 The Application of a Communication Process Between the Algorithms

The application of a communication process enabling access to the internal data of an algorithm was the underlying reason for introduction of the term of object was introduced [180]. This term has been known and used for many years in the algorithm formation and programming technique and especially in object-oriented programming [67, 68].

The communication process between the algorithms that was used here enables access to the internal data of an algorithm.

Let us consider the algorithms Alg_1 and Alg_2 which use the method of communication (Fig. 2.12). The algorithms communicating in this particular way will be referred to as objects ($Obj_1 = (X_1, f_1)$ and $Obj_2 = (X_2, f_2)$).

The process of communication may proceed according to the following scenario:

- Let us accept that the component algorithm (object) Obj_1 needs for the calculation of the function f_1 access to the parameters X_2 , being the internal data of the algorithm (object) Obj_2 . It constitutes the fundamental difficulty in defining the next steps changing the state of the object Obj_1 .
- In order to receive the necessary data—the object Obj_2 may make its internal parameters accessible for the calculation of the function f_1 of the object Obj_1 with the use of mechanism referred to as a *method*. The object Obj_1 starts up the appropriate method (the *method*₁₂) of the object Obj_2 (Fig. 2.12a, dashed arrow).

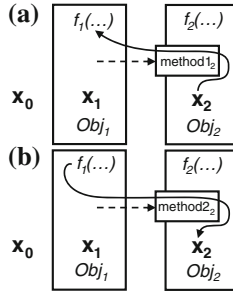


Fig. 2.12 Schema of the concept of the object, formed as the communicating algorithms which are to some extent autonomous, **a** the object Obj_1 receives the information about the state of the object Obj_2 , necessary for the calculation of the transition function f_1 through calling an appropriate method, the $method_{12}$, **b** the result of calculation of the function f_1 causes changes of the state of object Obj_2 realized through calling the $method_{22}$

- As a result of calling (starting up) the method—the method in response to calling gives back essential data (internal parameters) of the object Obj_2 (Fig. 2.12a, solid arrow).
- After receiving the essential data, the function f_1 may calculate new values of global internal parameters for the object Obj_1 , as well as internal parameters (all or some) for the object Obj_2 .
- The algorithm (object) Obj_1 may with the use of another method (the $method_{22}$) affect the state of internal parameters of the object Obj_2 and transfer the new values calculated with the use of the function f_1 (Fig. 2.12b).

Summing up, it may be ascertained that in order to solve the above-presented problem mechanisms of communication referred to as *methods*, are applied, which enable access to the parameters. Communication realized in this way is characteristic of cooperating component algorithms referred to as objects.

This information exchange (communication) between the algorithms (objects) with the use of methods is the underlying reason for the object concept and so-called object-oriented approach.

2.5.2.2 The Application of an Observation Mechanism for the Algorithm Behaviour

The solution using the process of communication is burdened with some imperfections, which results from the fact that the mechanism of communication does not guarantee full independence (and at the same time autonomy) of cooperating algorithms.

An algorithm which is asked by another algorithm to send (and similarly receive transferred) data should agree to do that. It means that the algorithm Alg_1 while calling a method, the $method_2$ of the algorithm Alg_2 must be provided with access to the appropriately working method by the algorithm Alg_2 .

A similar situation takes place in the case of transferring results of the function f_2 with the use of the *method*₂.

A different approach to the realization of cooperation in problem solving may be proposed, however, the cooperating algorithms realizing this cooperation will be more independent (autonomous) than in the presented object-oriented approach which has already been presented.

This approach is possible due to the observation operation which a given partial algorithm may be equipped with. With the use of observation one algorithm may trace the behaviour (action in the environment) of another cooperating algorithm. A given component algorithm observes the environment and especially changes that occur in that environment resulting from the action of another component algorithm. On the basis of these observations, it may learn (indirectly and probably approximately) about the internal state of another algorithm, and through the change of the state of the environment it may influence (indirectly) the change of the internal state of another algorithm (Fig. 2.13).

This method of solving a problem leads us to the *agent* notion (Ag) which will be identified with the algorithm equipped with the capability of observing. Particularly, this algorithm (agent) will be denoted by $Ag = (X, f)$ with the appropriate indices, if necessary.

The approach to cooperation between the agents may be specified with the use of the following reasoning:

Let us consider cooperation between the agents $Ag_1 = (X_1, f_1)$ and $Ag_2 = (X_2, f_2)$ through the environment X_0 .

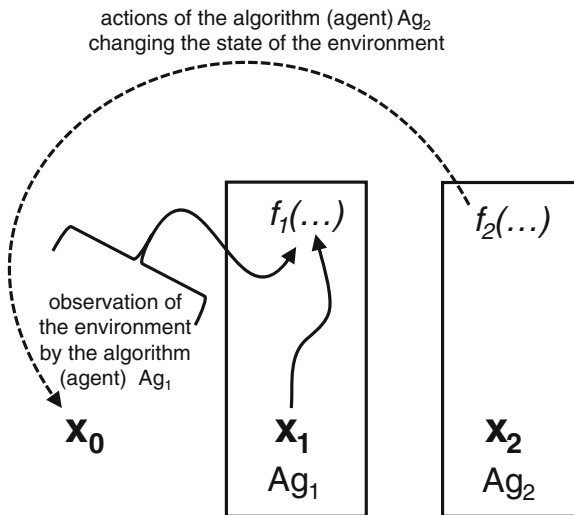


Fig. 2.13 Schema of decomposition of an algorithm with the use of the agent concept. The agent Ag_1 observes changes occurring in the environment through the agent Ag_2 , which gives him the capability to define the internal state x_2 of this agent

- For the calculation of values, the function f_1 , which is responsible for the realization of the agent Ag_1 algorithm, needs internal parameters of that agent (x_1), global data (x_0), and internal parameters (all or some) (x_2) of the agent Ag_2 . It should be emphasized that the agent Ag_1 does not have access (direct) to parameters (x_2).
- For the purpose of achieving the appropriate data, the agent Ag_1 observes the behaviour of the agent Ag_2 . It means that the agent Ag_1 observes changes in the environment (global data x_0) which result from the action (realization of the action) of the agent Ag_2 . On the basis of the observation result, the agent Ag_1 may define (estimate) the state of the agent Ag_2 , in other words the state of (values) its internal parameters. In order to do that, the agent Ag_1 must possess some knowledge about the agent Ag_2 , and especially some knowledge about the function f_2 and its effect on the changes of the state of the environment X_0 , as well as its influence on the state of that agent X_2 (Fig. 2.13).

The process of defining the values of parameters (x_2) may be realized with greater or lesser precision, depending on specific, practical possibilities. In effect observed (estimated) data do not have to give precise, complete information about the state of the agent Ag_2 , but they should be sufficient for continuing actions by the agent Ag_1 (for the calculation of the values of the function f_1).

- The agent Ag_1 possessing essential information (values of parameters x_1 , x_0 and indirectly x_2), using (calculating) the function f_1 may change its state (parameters x_1), and the state of the environment (global data- parameters x_0). The changes of the environment state are realized through calling subsequent actions (actions of the agents Ag_1 and Ag_2). However, the agent Ag_1 does not have the access to the internal data of the agent Ag_2 and it is not capable of effecting directly the change of its state, though it should be done as a result of the calculation of the function f_1 . Nevertheless, it is possible to achieve it indirectly with the use of changes of the environment state which forces the change of the state of the agent Ag_2 . The agent Ag_2 similar to the agent Ag_1 observes changes in the environment (parameters x_0) resulting from the actions of the agent Ag_1 , and on the basis of the information makes changes of its state, in other words modifies parameters x_2 .

The procedures of gaining autonomy by an agent give greater independence than that received in the object-oriented approach because it is not engaged directly in the internal states of another agent. The solution based on the observation process is more difficult in realization, however, the range of interaction and cooperation between the agents gives greater possibilities in the field of forming agent systems (multi-agent).

Numerous problems occur such as intentionality, suitability of actions, awareness, cooperation between the agents, as well as problems of interaction between the agents and others which remain open. Some of them will be discussed further in later chapters. The two approaches to the realization of autonomy of cooperating algorithms (with the use of communication and observation) provide a basis for distinguishing between the object notion and the agent notion.

Summing up, it may be noted that the source of information for the agent (Ag_1) is the state of its local data (x_1), the state of the environment (global data x_0) and the information received as a result of observation of behaviour of other agents (for

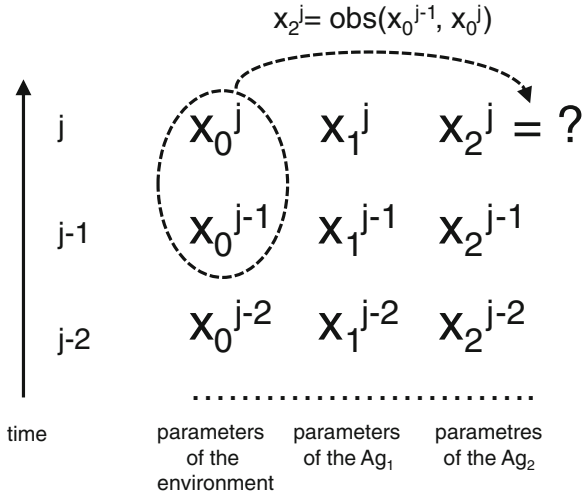


Fig. 2.14 Schema of the concept of the application of observation process in defining the parameters of the agent

instance, the agent Ag_2). This concept of the agent action has been schematically presented in Fig. 2.14. Changes in time of one parameter (x_0) make it possible to estimate data values of the other parameter (x_2).

Example 2

Let us consider the following example:

$$U = \{a, b, c, d, e\}, \quad F(a) = b, \quad F(b) = c, \quad F(c) = d, \quad F(d) = e \quad (2.38)$$

Let us apply the Cartesian product:

$$\begin{aligned} X_0 &= \{0, 1\}, \quad X_1 = \{0, 1\}, \quad X_2 = \{0, 1\} \\ X &= X_1 \times X_2 \times X_3 = \{(0, 0, 0), (0, 0, 1), \dots, (1, 1, 1)\} \end{aligned} \quad (2.39)$$

Between the elements of the sets U and X there is the following relation:

$$\begin{aligned} a &\text{ corresponds to } (0, 0, 0), & b &\text{ corresponds to } (0, 0, 1), \\ c &\text{ corresponds to } (0, 1, 0), & d &\text{ corresponds to } (1, 1, 0), \\ e &\text{ corresponds to } (1, 0, 0) \end{aligned} \quad (2.40)$$

The function f may be defined as follows:

$$\begin{aligned} f((0, 0, 0)) &= (0, 0, 1), & f((0, 0, 1)) &= (0, 1, 0), \\ f((0, 1, 0)) &= (1, 1, 0), & f((1, 1, 0)) &= (1, 0, 0) \end{aligned} \quad (2.41)$$

However, the algorithms Alg_1 and Alg_2 have the following forms:

$$\begin{aligned}
Alg_1 &= (X_1, f_1) \\
f_1 : X_0 \times X_1 \times X_2 &\rightarrow X_0 \times X_1 \times X_2 \\
f_1(0, 0, 1) &= (0, 1, 0), \quad f_1(1, 1, 0) = (1, 0, 0)
\end{aligned} \tag{2.42}$$

$$\begin{aligned}
Alg_2 &= (X_2, f_2) \\
f_2 : X_0 \times X_1 \times X_2 &\rightarrow X_0 \times X_1 \times X_2 \\
f_2(0, 0, 0) &= (0, 0, 1), \quad f_2(0, 1, 0) = (1, 1, 0)
\end{aligned} \tag{2.43}$$

The observation function Obs_{12} realizing (modelling) the process of recognition of the behaviour of the algorithm Alg_2 by the algorithm Alg_1 may be denoted as follows:

$$\begin{aligned}
Obs_{12} : X_0 \times X_0 &\rightarrow X_2 \\
Obs_{12}(x_0^{i-1}, x_0^i) &= x_2 \\
Obs_{12}(0, 0) &= 1, \quad Obs_{12}(0, 1) = 0
\end{aligned} \tag{2.44}$$

In effect, the algorithm Alg_1 may be considered as autonomous towards the algorithm Alg_2 due to the observation function Obs_{12} , which informally can be denoted as follows:

$$\begin{aligned}
Alg_1 &= (X_1, f_1, Obs_{12}) \\
f_1(x_0, x_1, x_2) &= f_1(x_0, x_1, Obs_{12}(x_0^{i-1}, x_0^i)) = (x_0, x_1, x_2)
\end{aligned} \tag{2.45}$$

It corresponds to taking into consideration stored “historical” global data (data on the state of the environment) along with the next step of an algorithm, which may be presented as follows:

$$\begin{aligned}
Alg_1 &= (X_1, f_1, Obs_{12}) \\
f_1(x_0^i, x_1^i, x_2^i) &= f_1(x_0^i, x_1^i, Obs_{12}(x_0^{i-1}, x_0^i)) = (x_0^{i+1}, x_1^{i+1}, x_2^{i+1})
\end{aligned} \tag{2.46}$$

In conclusion, in this example we may observe the manner of gaining autonomy due to the observation operation, which causes the algorithm Alg_1 to be considered as autonomous towards the algorithm Alg_2 (because of cooperation between the algorithms through X_0).

2.6 Multi-agent System as a Result of Decomposition of an Algorithm

The problem of decomposition of an algorithm may be considered from many perspectives.

As mentioned above, the problem of decomposition of the algorithm $Alg = (U, F)$ may be considered as decomposition of the set U and the function F , or may be the result of the Cartesian product application in which case decomposition of the set X is used.

(a)

$X_1, X_2, \dots X_i \dots X_{m-1}, X_m$

$$\begin{aligned}
 u^1 \sim x^1 &= (x_1^1, x_2^1, \dots, x_i^1, \dots, x_{m-1}^1, x_m^1) \\
 u^2 \sim x^2 &= (x_1^2, x_2^2, \dots, x_i^2, \dots, x_{m-1}^2, x_m^2) \\
 &\dots \\
 u^j \sim x^j &= (x_1^j, x_2^j, \dots, x_i^j, \dots, x_{m-1}^j, x_m^j) \\
 &\dots \\
 u^{n-1} \sim x^{n-1} &= (x_1^{n-1}, x_2^{n-1}, \dots, x_i^{n-1}, \dots, x_{m-1}^{n-1}, x_m^{n-1}) \\
 u^n \sim x^n &= (x_1^n, x_2^n, \dots, x_i^n, \dots, x_{m-1}^n, x_m^n)
 \end{aligned}$$

(b)

$$\begin{aligned}
 u^1 \sim x^1 &= (x_1^1, x_2^1, \dots, x_i^1, \dots, x_{m-1}^1, x_m^1) \\
 u^2 \sim x^2 &= (x_1^2, x_2^2, \dots, x_i^2, \dots, x_{m-1}^2, x_m^2) \\
 &\dots \\
 u^j \sim x^j &= (x_1^j, x_2^j, \dots, x_i^j, \dots, x_{m-1}^j, x_m^j) \\
 &\dots \\
 u^{n-1} \sim x^{n-1} &= (x_1^{n-1}, x_2^{n-1}, \dots, x_i^{n-1}, \dots, x_{m-1}^{n-1}, x_m^{n-1}) \\
 u^n \sim x^n &= (x_1^n, x_2^n, \dots, x_i^n, \dots, x_{m-1}^n, x_m^n)
 \end{aligned}$$

(c)

$$\begin{aligned}
 u^1 \sim x^1 &= (x_1^1, x_2^1, \dots, x_i^1, \dots, x_{m-1}^1, x_m^1) \\
 u^2 \sim x^2 &= (x_1^2, x_2^2, \dots, x_i^2, \dots, x_{m-1}^2, x_m^2) \\
 &\dots \\
 u^j \sim x^j &= (x_1^j, x_2^j, \dots, x_i^j, \dots, x_{m-1}^j, x_m^j) \\
 &\dots \\
 u^{n-1} \sim x^{n-1} &= (x_1^{n-1}, x_2^{n-1}, \dots, x_i^{n-1}, \dots, x_{m-1}^{n-1}, x_m^{n-1}) \\
 u^n \sim x^n &= (x_1^n, x_2^n, \dots, x_i^n, \dots, x_{m-1}^n, x_m^n)
 \end{aligned}$$

Fig. 2.15 Schema of two approaches to the problem of decomposition of an algorithm: **a** the schema of a column i corresponding to the parameter X_i , **b** linear division leading to decomposition into subprograms, **c** column decomposition being the basis of the concept of the object and the agent

The set X may be presented as the lines of the parameters $(x_1^i, x_2^i, \dots, x_m^i)$ corresponding to the individual states u^i , which may be presented as a table form, shown in Fig. 2.15a. Both approaches—decomposition based on the division of the set and decomposition based on the Cartesian product concept—may be presented in the form of two kinds of division of the table (Fig. 2.15):

- Method of decomposition, based on the *linear* division of the table of parameters (Fig. 2.15b). This method of decomposition based on the concept of division of the set U and F leads to the concept of a subprogram.
- The *column* decomposition of the table of parameters inspired by the notion of the Cartesian product (Fig. 2.15c) provides a basis for the decomposition of an algorithm with the use of the concept of an object as well as the concept of an agent.

Summing up, we consider two methods of decomposition—*linear* decomposition leading to the notion of a subprogram, and *column* decomposition that makes it possible to define the notion of an object and the notion of an agent.

Using both methods we may define the manner of decomposition of an algorithm that leads to receiving the multi-agent system.

This process may be realized as follows:

Let us consider the algorithm $Alg = (U, F)$. The set of states of this algorithm U may be presented as the Cartesian product of the sets of parameters—in other words, the set X , where $X = X_1 \times X_2 \times \dots \times X_n$ given in the table form (Fig. 2.16a). When analyzing the table, we will present the way of constructing the multi-agent system.

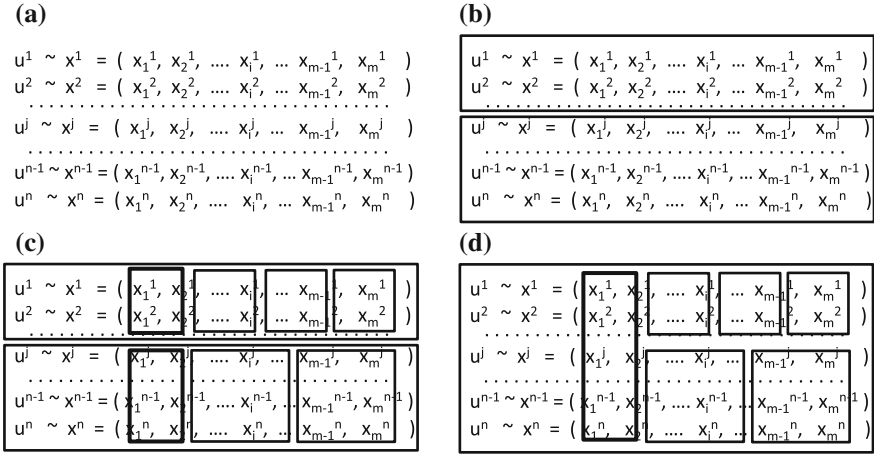


Fig. 2.16 Schema of the process of decomposition of an algorithm, which in effect defines the agent system (multi-agent), **b** the division of the system into subprograms, **c** the division into agents within each subprogram, **d** the integration of the agent systems into one multi-agent system

- The first step is the decomposition of the set U , which corresponds to the *linear* division of the table. In effect, we receive the decomposition of the algorithm Alg into subprograms (Fig. 2.16b).
- In the next step, each component algorithm (or subprogram) may be considered as an algorithm and subjected to another process of decomposition. In that case, we use the process of division based on the concept of the Cartesian product, and the component algorithms we receive may take the form of agents, creating decomposition into agent systems within each subprogram (Fig. 2.16c).
- The multi-agent systems resulting from decomposition (Fig. 2.16c) may be connected and create the multi-agent system with different kinds of agents—so called heterogenic multi-agent system (Fig. 2.16). The agents resulting from decomposition of one subprogram may observe through the environment other agents from “a different world” i.e., another subprogram. This make cooperation of these agents possible.

The connection of environments from separate agent systems provides a basis for connecting these systems together. Two agent systems have defined environments X_0^1 and X_0^2 (Fig. 2.17a) and connecting them into one environment (X_0^1 and X_0^2) provides a basis for the realization of cooperation between agents from different agent systems (Fig. 2.17b).

The agent system (multi-agent) realized in this way is a kind of algorithm (the set of algorithms) which can be considered as a result of decomposition of a certain complex algorithm.

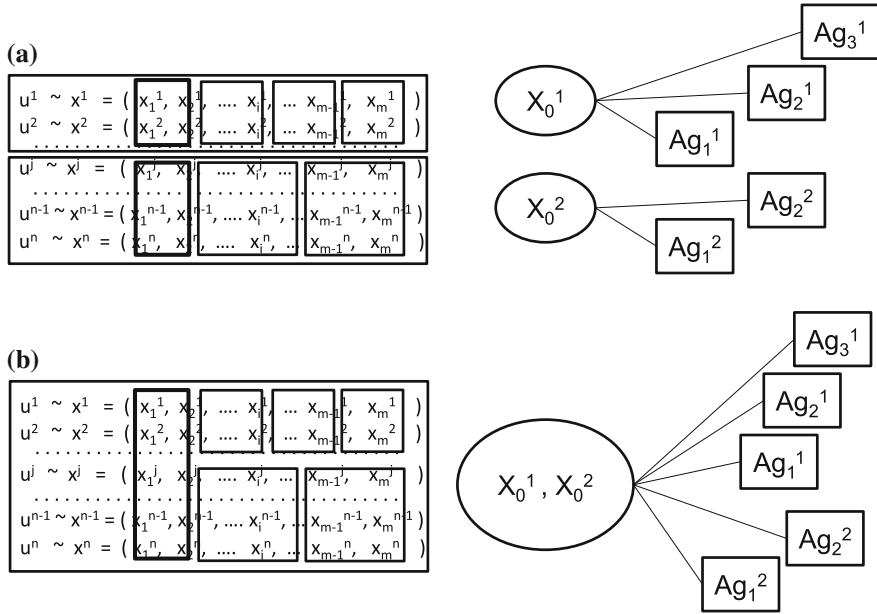


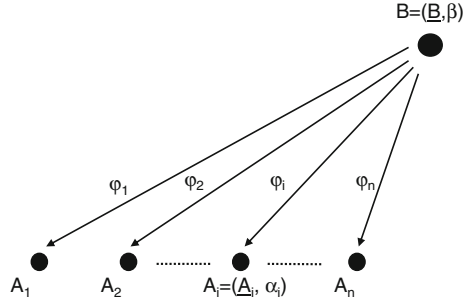
Fig. 2.17 Practical interpretation of the process of decomposition of an algorithm leading to the agent system (multi-agent) using the concept of connecting the environments X_0^1 and X_0^2

2.7 Decomposition with the Use of the Cartesian Product in the Category of the Algorithm Alg

Considering the above-presented decomposition of an algorithm based on the application of the Cartesian product, a question may arise whether there is any other way of using the Cartesian product application in the decomposition of an algorithm. And so, we may consider a more general approach and apply the Cartesian product in its generalized version, and specifically the decomposition of the algorithm Alg with the use of the Cartesian product in the theory of a category. In this theory, a definition of the Cartesian product is formulated in such a way that the definition we have used until now is a particular form (in the category of sets) of a general definition. We may consider the category of algorithms and use the form of the Cartesian product in this category for the realization of decomposition.

For this purpose, it is necessary to define the category of algorithms Alg . The category Alg , like any other category, is defined as a set of objects and a set of morphisms. Therefore, it is necessary to consider the category of algorithms Alg consisting of a set of objects $obj(Alg)$ and the set of morphisms $\sigma(Alg)$ [1, 93, 152]. The objects of the category Alg may be defined as follows:

Fig. 2.18 Schema of the Cartesian product in the category of algorithms



$$A = (\underline{A}, \alpha) \in \text{obj}(\text{Alg}) \quad (2.47)$$

$$\underline{A} - \text{set}, \quad \underline{A} \neq \emptyset, \quad \alpha : \underline{A} \rightarrow \underline{A}$$

The function α corresponding to the function F is a partial function. However, the function ϕ which is a morphism in the category Alg (we denote as $\phi \in \sigma(\text{Alg})$) may be defined as follows:

$$A \in \text{obj}(\text{Alg}), \quad C \in \text{obj}(\text{Alg}), \quad A = (\underline{A}, \alpha), \quad C = (\underline{C}, \gamma)$$

$$\phi : \underline{A} \rightarrow \underline{C} \text{ as that } \gamma(\phi(a)) = \phi(\alpha(a)) \text{ and } a, \in \underline{A}$$

Summing up, the category Alg consists of the set of objects $\text{obj}(\text{Alg})$ and a set of morphisms $\sigma(\text{Alg})$ as defined above.

The Cartesian product of objects that belongs to the category Alg above the set of indices $I = \{1, 2, \dots, i, \dots, n\}$ may be defined as follows (Fig. 2.18):

$$B = \prod_{i \in I} A_i \quad (2.48)$$

where:

$$B \in \text{obj}(\text{Alg}), \quad B = (\underline{B}, \beta), \quad \beta : \underline{B} \rightarrow \underline{B},$$

$$\forall_{i \in I} A_i \in \text{obj}(\text{Alg}), \quad A_i = (\underline{A}_i, \alpha_i), \quad \alpha_i : \underline{A}_i \rightarrow \underline{A}_i$$

$$\forall_{i \in I} \phi_i \in \sigma(B, A_i) \quad \text{i.e. } \phi_i \text{ is a morphism from } B \text{ to } A_i$$

$$\text{and } \forall b \in \underline{B} \text{ as that } \forall_{i \in I} \phi_i(b) = a_i \in \underline{A}_i, \quad \phi_i(\beta(b)) = \alpha_i(\phi_i(b)) \in \underline{A}_i$$

The application of this Cartesian product to a given algorithm (in the category of algorithms) enables decomposition of the algorithm Alg into component algorithms $\text{Alg}_i = (\underline{A}_i, \alpha_i)$. Each of the component algorithms (Alg_i) obtained in this way is autonomous towards another from the rest of component algorithms (Alg_j for $i \neq j$), which results from the definition of the Cartesian product in the category of algorithms.

On the basis of the above considerations we may conclude that there are two approaches to the application of the Cartesian product to decompose an algorithm:

- Decomposition based on the application of the Cartesian product to the set U of the algorithm $Alg = (U, F)$. It enables decomposition of an algorithm into component algorithms, but the component algorithms we obtain are not usually mutually autonomous. The decomposition itself is in that case easier than the decomposition presented in the category of algorithms and more often possible in practice. Lack of autonomy of the component algorithms creates a certain problem, however, with the use of methods presented in the previous chapters, the autonomy of component algorithms may get and guarantee the possibility of creating component algorithms, which is of great importance in practice.
- Decomposition realized by the application of the concept of the Cartesian product concerning the “whole” algorithm Alg considered as an object in the category of algorithms. As a result, we receive component algorithms that are independent of one another, which would enable the creation (designing, programming) of the algorithms independently (in parallel). However, for the problems occurring in practice the realization of such decomposition of an algorithm into component algorithms may turn out to be difficult because it requires to the application of the concept of the Cartesian product to the whole structure which is an algorithm. It is necessary to meet certain demands (the form of the Cartesian product in the theory of a category: [1, 93, 152]) which are difficult to provide in practical applications. Particularly, these difficulties result from the fact that our considerations presented in this chapter only confirm the possibility of decomposition of an algorithm within the category of algorithms, but they do not provide practical suggestions on how for a given algorithm (object of a category) B we can find algorithms (as objects of a category) A_i , into which we decompose the initial algorithm, that is how the sets A_i or the functions ϕ_i should look like (see works on the theory of a category, for instance [1, 93, 152]).

It cannot be excluded that the development of research on the properties of the Cartesian product in the theory of the category (and especially its role in the decomposition of an algorithm in the category of algorithms) may lead to the possibility of application of algorithm decomposition in practice, on the basis of the application of the Cartesian product concept to the category of algorithms.

2.8 Summary—Decomposition, Agent, Autonomy

This chapter presented the concept of decomposition of a given algorithm into component algorithms. The analysis of the process of decomposition and different methods of its realization resulted in the concept of an object and an agent, as well as the agent system. A basic role was played by the analysis of such property as autonomy and the notion of an autonomous agent and its capability to observe the behaviour of another agent in the environment, which constitutes a method of gaining autonomy.

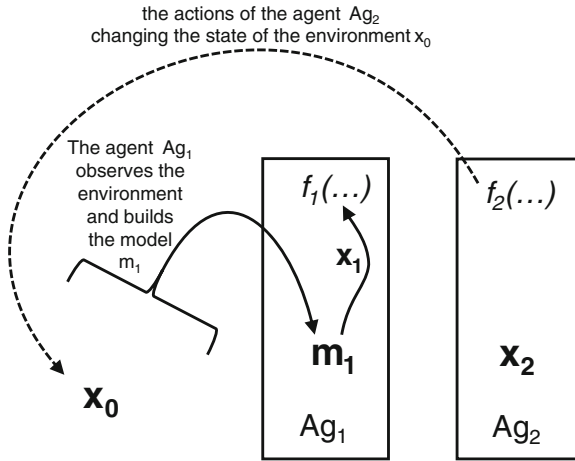


Fig. 2.19 Schema of actions of the algorithm of the agent on the basis of the model of the environment. The agent Ag_1 on the basis of observation of the environment builds the model m_1 with which it may define the internal state of the agent Ag_2

These considerations lead to the following scenario of actions of agent:

- The function realizing the actions of the agent Ag_1 must use the data x_1 defining the state of the agent, the data applying to the state of the environment (global data) x_0 , as well as the data x_2 of the agent Ag_2 .
- In order to gain information defining x_2 , the agent Ag_1 observes the behaviour of the agent Ag_2 (Fig. 2.19), i. e., the agent Ag_1 registers changes which are made by the agent Ag_2 in the environment (x_0). The task of this observation is to build by the agent Ag_1 the model m_1 of its surrounding environment, which makes it possible to receive indispensable information on local data x_2 of the agent Ag_2 (the model m_1 will be discussed in details further in later chapters). It should be emphasized that the agent Ag_1 may realize its observation (and on that basis the construction of the model m_1) independently of the agent Ag_2 . When data are received from the observation of the environment (about the agent Ag_2) do not give complete information on the state of the agent Ag_2 (data x_2), they may be replaced with approximate data (received from the analysis of the model m_1).
- The agent Ag_1 , which makes use of the model m_1 , may foresee states of other algorithms (agents) cooperating in the system and make appropriate decisions concerning its action in the agent system.
- Having all the necessary data the function f_1 of the agent Ag_1 realizes an appropriate action which results in the modification of the state of the agent x_1 and the state of the environment x_0 (which is not shown in Fig. 2.19). The situation may change and, now it is the agent Ag_2 that observes the changes of the environment made by the agent Ag_1 , and on this basis builds its model m_2 . Accordingly, the agent Ag_2 may change its state (x_2) on the basis of the model realized with the use of observation of changes of the environment (x_0) evoked by the action of the agent Ag_1 .

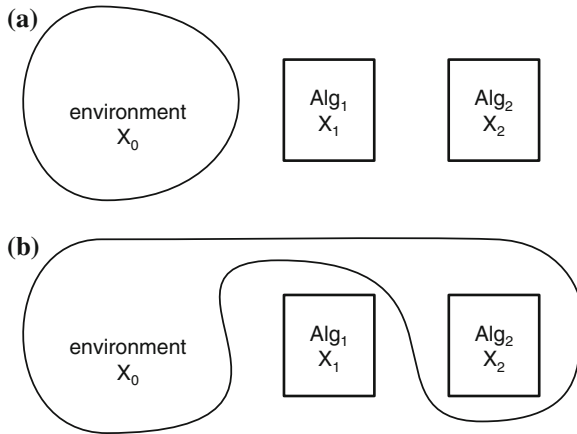


Fig. 2.20 Schema of perceiving the environment by the algorithm Alg_1 : **a** perceiving the non extended environment, **b** inclusion of Alg_2 in the perceived environment (extended environment)

Thus, for the agent Ag_1 information on its state x_1 , the state of the environment x_0 , and the state x_2 of the agent Ag_2 provides a basis for creating the model m_1 . On the basis of this model (m_1) the agent Ag_1 plans and realizes its actions. It is concerned with the extension of the environment perceived by the algorithm Alg_1 . So far, the algorithm Alg_1 has been able to perceive the environment as the variables X_0 (Fig. 2.20a). Becoming an agent, it incorporates other agents (algorithms), e.g. Alg_2 (Fig. 2.20b) into the range of the environment it observes. The above mentioned concept ensures considerable independence of agents' actions, with the possibility of their mutual interaction.

A given agent receives information on the state of the environment and on behaviour of other agents by means of the observation operation of its surrounding environment. It is the main difference between the notion of an agent and the notion of an object which does not possess that kind of capability.

Summing up the features of the agent, it may be concluded that:

- The property of autonomy is not the feature distinguishing the notion of an agent from other notions in the field of computer science, and especially from the notion of an object, as both the agent and the object may be considered as autonomous.
- The distinguishing feature of the agent is its capability to observe behaviour of other agents operating in the environment.
- The agent may possess the capability to communicate with other agents through the environment, but also through direct communication (similar to the communication between the objects), but this capability is not the distinguishing feature for the agent with reference to the object.

The agent Ag_1 observes the environment X_0 (Fig. 2.20) and if we accept the above concept of an algorithm, it can observe neither the agent Ag_2 , nor its influence on

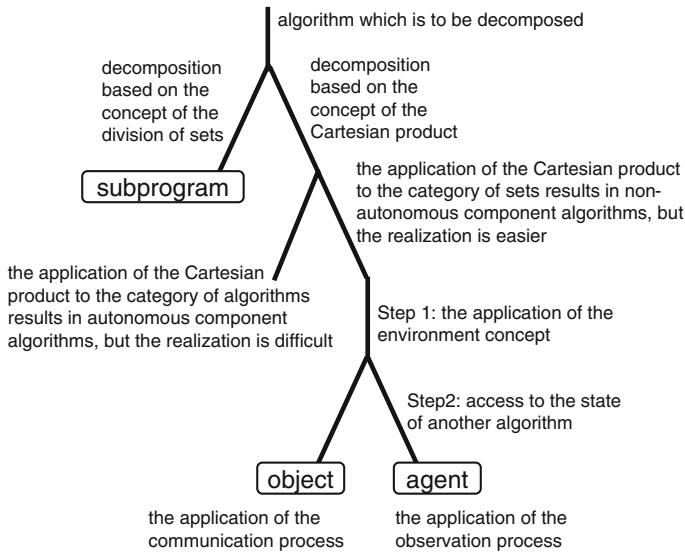


Fig. 2.21 Schema of decomposition of an algorithm into autonomous component algorithms

the environment or changes that occur in that environment (Fig. 2.20a). Under the concept of the agent, the agent Ag_1 incorporates the agent Ag_2 into the area of the environment it observes, and in particular:

- The agent Ag_1 is able to observe not only the environment X_0 (Fig. 2.20), but also the fact that the agent Ag_2 exists in the environment,
- The agent Ag_1 is able to observe changes in the environment caused by the agent Ag_2 , which means that the agent Ag_1 is able to associate the event occurring in the environment that changes the state of the environment with a specific agent), the doer of this event (e.g., with the agent Ag_2 , Fig. 2.20b). It can be said that the agent Ag_1 extends its own model of the environment it observe by incorporating the agent Ag_2 into it, as presented in the Fig. 2.20.

Summing up the process of decomposition of a component algorithm that leads to the concept of an object, and especially to the concept of an agent, we may use a schema from Fig. 2.21, which shows the following possibilities of transformation of an algorithm:

- An algorithm we are to realize (design, program, activate) is too complicated and has to be decomposed into more or less independent component algorithms. There are two possible ways of decomposition: one based on the concept of the set division, and on the other on the Cartesian product.
- The decomposition based on the division of sets leads to the concept of a subprogram. This approach is often successfully applied in practice but not always effectively.

- The application of decomposition based on the Cartesian product gives two possibilities: it can be used in the category of algorithms and in the category of sets and mapping.
- The use of the concept of the Cartesian product in the category of algorithms makes it possible to receive autonomous component algorithms, which are independent, but this process of decomposition is difficult and not always possible in practice.
- The application of the concept of the Cartesian product in the category of a set and mapping is easier in practical applications, but it usually does not result in autonomous partial algorithms. However, it is possible to regain (at least to some certain extent) this autonomy.
- One method of gaining autonomy is the application of the communication process between the partial algorithms. It leads us to the concept of an object.
- The other method of gaining autonomy is the use of operation of observation of the environment (including other agents) which makes it possible to formulate the concept of an *agent*. It is noteworthy that this observation may be used for some kind of communication between agents, so that one agent makes a certain change of the chosen attribute of environment (or a characteristic modification of the state of the environment), and the other agent observes this change and interprets it.

The aim of the considerations was to define such notions as decomposition and autonomy, which resulted in defining an agent as an autonomous (to a certain extent) algorithm (program) equipped with the capability to communicate and cooperate whose distinguishing feature is the capability to observe the environment in order to recognize actions undertaken by other agents existing in this environment.

The formulations we obtained may be used for defining agents operating in computer systems as well as for constructing agent systems, oriented towards defined areas of applications, which will be presented later in further chapters (Chaps. 3 and 4).

A Perspective on Agent Systems

Paradigm, Formalism, Examples

Cetnarowicz, K.

2015, XI, 140 p. 72 illus., 2 illus. in color., Hardcover

ISBN: 978-3-319-13196-2