Abstraction in the formalization of organization theory

Henk W.M. Gazendam and John L. Simons
Faculty of Management and Organization
Groningen University, The Netherlands

Abstract
In many organization theories, the interpretation frame is the most important part of the theory. Furthermore, organization theorists use a rich variety of reasoning mechanisms, for instance reasoning about actions using abstraction from time. In formalizing organization theories, one has to try to preserve the richness of the interpretation frames of those theories as well as the richness of the reasoning mechanisms used. This can be done by studying the abstraction mechanisms that organization theorists use:
- type abstraction
- function abstraction
- method abstraction (abstraction from time)

In order to represent the richness of interpretation frames, one can express the concepts in these interpretation frames as types using a suitable theory of types and type abstraction that enhances FOL. Function abstraction, for instance FP, an extension of lambda calculus, gives us the instruments to define powerful functions for aggregation. Abstraction from time can be understood using method abstraction, which distinguishes the method that describes an action in a timeless way from the situation or message that trigger the execution of this method.

1. Introduction

In many organization theories, the interpretation frame is the most important part of the theory. Furthermore, organization theorists use a rich variety of reasoning mechanisms, for instance reasoning about actions using abstraction from time. In formalizing organization theories, one has to try to preserve the richness of the interpretation frames of those theories as well as the richness of the reasoning mechanisms used. In order to do that, we will describe the richness of interpretation frames and of reasoning mechanisms first. We will pay some extra attention to reasoning about actions. This will be done based on the analysis of Fayol’s (1916/1956) classical theory in Gazendam (1993). Our strategy for understanding these reasoning mechanisms is the identification of the abstraction mechanisms used. A better understanding of these abstraction mechanisms can lead to a better formal description of organization theories. We will focus on type abstraction, function abstraction, and method abstraction.
2. The interpretation frame of an organization theory

Fayol’s organization theory offers an interesting example of a rich interpretation frame and a wealth of reasoning mechanisms. The specification of the interpretation frame has been done in ENBF (Gazendam, 1993). This specification comprises eight pages of ENBF-expressions. A translation of these specifications into logic, however, has not been done successfully thus far.

Interpretation frames can be described using ENBF. An example taken from the description of the interpretation frame of Fayol’s theory is given below.

\[
<Fayol's\ operation> ::= \\
\{ <corps\ social>, <material\ organism>, <contingency\ factor>^+, <symbol\ structure>^+, <person&task\ relation>^+, <person&action\ relation>^+, <management\ process>^+ \}.
\]

\[
<corps\ social> ::= \\
\{ <person>^+, <person&person\ quality>^+, <corps\ social\ quality>^+, <corps\ social\ network\ quality>^+ \}.
\]

After these eight lines of specification, another 8 pages of specifications follow in order to capture Fayol’s interpretation frame(Gazendam, 1993). This interpretation frame specification comprises the specification of several structures and relationships, of basic types, and of some functions.

The specifications of structures and relationships (at type level) encompass:
- whole – part;
- object – attribute;
- object – process;
- type – subtype;
- type – objects that can instantiate the type;
- process – next process.

The 12 basic types distinguished are:
- person;
- organization;
- action or process;
- symbol structure;
- thing;
- other object;
- time;
- location;
- nominal values (as values of attributes);
- ordinal values (as values of attributes);
- integer numbers (as values of attributes);
- real (rational or irrational) numbers (as values of attributes).

The functions that are important in Fayol’s theory are:
- corps social;
- material organism.
3. Reasoning mechanisms in organization theory

Fayol’s theory uses a variety of reasoning mechanisms (Gazendam, 1993), several of which encounter severe difficulties in the process of translating them into logic. Most of them can, however, be written down as rules in a notation that derives from ENBF and Prolog. In those reasoning mechanisms, five classes can be distinguished:

1. simple reasoning using comparative facts, calculations, and universal quantification;
2. reasoning about enabling a development, possibility, and necessity;
3. reasoning about probability;
4. reasoning about the relation between individual level and organization level using causation, aggregation and distribution;
5. reasoning about actions, including reasoning about tasks, preferences, equilibria and flows.

In this article, we will especially discuss Fayol’s reasoning about actions. In the discussion, we will focus on identifying the abstraction mechanisms that are used. With respect to reasoning about actions using abstraction from time, Fayol uses task analysis (reasoning about the composition and best way to accomplish tasks), opportunity-based reasoning (based on preferences), equilibrium-based reasoning, and flow-based reasoning.

4. Reasoning about actions and processes

In the reasoning mechanisms that concern actions, organization theorists often do not use time explicitly. In other words, they abstract from time. This is also the case in Fayol’s theory. Fayol uses four mechanisms for coping with actions without the necessity of explicitly representing time: task analysis, opportunity-based reasoning, equilibrium-based reasoning, and flow-based reasoning.

There are several steps from a description that is totally abstracted from time to a description in which time is explicit:
- totally abstracted form time;
- use of state;
- use of state transitions;
- explicit use of time.

The use of the concepts of state and state transition can be seen as a way of description that is between the total abstraction from time and the explicit use of time. The most fine-grained representation of processes that does not explicitly use time is discrete event simulation. The system hops from event to the next event, and from state to state; the event logic is dominant.

Port and Van Gelder (1995) describe a development in cognitive science stressing the importance of explicit representation of time in theoretical models. Masuch and Huang’s (1996) ALX3 is a logic of action that is predominantly based on abstraction from time using beliefs and preferences. The action operator in this logic connects possible worlds, and is mainly used for expressing preferences about actions to be taken.
4.1. Task analysis

Task analysis states that to perform a task, one has to perform certain subtasks, or that a task should be done in a particular way. For instance, the establishment of social order presupposes the following subtasks according to Fayol: (1) continuous observation of the human resources of the organization, and continuous estimation of the human requirements of the organization, (2) deciding about the necessary positions, and (3) selecting personnel for these positions. Task analysis can be seen as a strategy to split the time-dependent description of tasks into a time-independent part, a program-like description of action sequences, on the one hand, and on the other hand a system of triggers that can described as for instance schedules or as recurrent behavior patterns in time and space.

4.2. Opportunity-based reasoning

Opportunity-based reasoning uses tendencies or attitudes to describe what people would do when a certain situation would arise. For instance, if people get the chance to gain authority, they will do that, but they will try to get rid of responsibility:

"...authority is sought after, while responsibility is feared"

Opportunity-based reasoning distinguishes the more or less time-independent preferences of actors, from the decision situations that occur in time, in which decisions are actually made and executed. These decision situations may be the result from the system of triggers mentioned earlier, or arise in a more random way.

4.3. Equilibrium-based reasoning

Equilibrium-based reasoning uses the picture of an equilibrium that is defined in terms of sets of variables that have to match. In the type of matching that is used, aggregation mechanisms play an important role. The way equilibrium is defined and handled is time-independent.

In this type of reasoning, one abstracts from the necessity of maintaining the equilibrium by management while people tend to take actions that disturb this equilibrium. This is, for instance, the case in the principle of subordination of individual interest to general interest, and the maintenance of social order. In both cases, selfish interests or impulses such as ambition, ignorance or simply human passions lead to actions that disturb the equilibrium; management has to prevent and counteract these disturbing actions. The equilibrium concept, therefore, presupposes a time-dependent equilibrium-restoring mechanism that can be described by, for instance, a set of state equations that explicitly use time.

4.4. Flow-based reasoning

Flow-based reasoning uses flows to describe action patterns. Flows are used to describe the actions of persons that are dependent on each other by passing information or objects. For instance, he distinguishes a command flow from the top of the organization to the bottom, and an information flow from the bottom to the top. This picture is refined further by distinguishing lateral flows of communication. Flow-based reasoning uses steady state parallel and sequential processes in which recurrent behavior patterns in space and time are structured. Most flows can be seen as methods or programs describing action sequences that are triggered by stable systems of recurrent triggers.
5. Abstraction

Abstraction may be a fundamental characteristic of human thought, distinguishing concepts from individuals. The philosopher Plato (Kneale and Kneale, 1962: 17) distinguishes ideas or forms from souls or beings, a distinction we would nowadays designate as the distinction between concepts and objects (or individuals). These concepts are abstractions that can be applied in a multitude of situations and to a multitude of objects. Therefore concepts can be seen as parsimonious instruments that have to be remembered and are more permanent than the ever changing world of individual objects and situations. The application of a concept to an object in a situation gives a proposition (or a set of propositions). This scheme has been elaborated in several modalities by Peirce (1958), distinguishing the individual objects or situations (Firstness) from the concepts (Secondness) and from the propositions (Thirdness). This fundamental distinction between concepts, objects and propositions can be seen as underlying several logical and mathematical systems, for instance predicate logic (in which the predicate is the concept), functional programming (in which the function is the concept), object oriented modeling (in which the object type is the concept) and event-oriented programming (in which the program or method is the concept that can be triggered by a message in a trigger situation).

Resuming, we have the following examples of the application of concepts to individuals:

<table>
<thead>
<tr>
<th>Individual</th>
<th>Concept</th>
<th>Result of application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object I</td>
<td>Predicate II</td>
<td>Clause III</td>
</tr>
<tr>
<td>Object I</td>
<td>Object type II</td>
<td>Proposition III</td>
</tr>
<tr>
<td>Situation I</td>
<td>Situation type II</td>
<td>Proposition III</td>
</tr>
<tr>
<td>Object I</td>
<td>Object type II</td>
<td>Object instance I</td>
</tr>
<tr>
<td>Object I</td>
<td>Function II</td>
<td>Object I</td>
</tr>
<tr>
<td>Message I</td>
<td>Method II</td>
<td>Process (= Situation) I</td>
</tr>
</tbody>
</table>

Lambda calculus, extended by FP, gives us powerful operators for combining (sets of) functions with (sets of) objects. FP is a language for functional programming designed by John Backus (1978, 1981) that uses several operators for combining functions, and for combining structured collections of functions with structured collections of data. Using FP, problems of expressing distribution and aggregation in FOL can be solved elegantly.

We will use variant of FP for handling abstraction and application using several types of concepts and individuals. The operators we use are \( \lambda \) (lambda abstraction), \( (\ ) \) or :: (function application), \( \circ \) (composition of functions), \( \alpha \) (apply to all), \( \\backslash \) (selection), \( / \) (reduction), and \( \rightarrow \) (condition) (Eisenbach, 1987: 15, 47; Henson, 1987: 218). The following table gives an overview.
<table>
<thead>
<tr>
<th><strong>operator</strong></th>
<th><strong>symbol</strong></th>
<th><strong>example</strong></th>
<th><strong>meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>abstraction</td>
<td>$\lambda$</td>
<td>$f = \lambda xy$ (&lt;expression&gt;)</td>
<td>means that the expression using $x$ and $y$ is converted to a concept $f$ with the variables $x$ and $y$</td>
</tr>
<tr>
<td>application</td>
<td>( ) or :: or :</td>
<td>f(x, y) or f :: x, y or x, y : f</td>
<td>means that the concept $f$ is applied to the data structures $x$ and $y$, giving a proposition or an object</td>
</tr>
<tr>
<td>combination</td>
<td>°</td>
<td>$f^\circ g$</td>
<td>means that a new concept is composed of a sequence of first concept $g$, and after that, concept $f$; operator ° is pronounced as ‘after’</td>
</tr>
<tr>
<td>apply to all</td>
<td>$\alpha$</td>
<td>$f \alpha x$</td>
<td>means that the concept $f$ is applied to all members of $x$, giving a result that preserves the data structure of $x$</td>
</tr>
<tr>
<td>selection</td>
<td>\</td>
<td>$f \setminus x$</td>
<td>means that the concept $f$ is applied to all members of $x$, giving a result that is either TRUE or FALSE; if TRUE, the member of $x$ is selected as a member for the resulting data structure that is of the same type as $x$</td>
</tr>
<tr>
<td>reduction</td>
<td>/</td>
<td>$f[/x]$ (&lt;data structure&gt;)</td>
<td>means that the data structure with component $x$ is aggregated over the dimension $x$ using the concept $f$; operator / is pronounced as ‘over’</td>
</tr>
<tr>
<td>condition</td>
<td>$\rightarrow$</td>
<td>$p \rightarrow f; g :: x$</td>
<td>if $p :: x$ is true then $f :: x$; else $g :: x$</td>
</tr>
</tbody>
</table>

The main operators to be used are abstraction and application. Abstraction identifies concepts based on expressions and in which concepts and individuals are distinguished. In application, concepts and individuals are combined to give expressions that can be interpreted as:
- propositions;
- (descriptions of) individuals (constructive application).

The symbol $\bot$ (bottom) is used for undefined results of application.

The operator \ (select) is defined as follows:
\((\lambda x (L \rightarrow x; \text{nil})) \alpha = (\lambda x L)\) \\
where L is a logical expression containing x; L is used as a criterion for selecting elements from a data structure.

### 6. Type abstraction

Capturing an interpretation frame in logic requires the use of a theory of types. Contributions to such a theory of types have been given by Devlin (1991) and De Brock (1995). Fodor (1998) stresses the importance of concepts as fundamental building blocks of cognition that can be combined in order to give more complex structures. Date and Darwen (1998) state that types are orthogonal to relations. Relations are sets of propositions; types are domains of values occurring in these propositions, for which several kinds of characteristics can be defined, for instance the operators that are applicable.

Frege (Kneale and Kneale, 1962: 493) has made the distinction between Sinn (sense) and Bedeutung (reference), between the intension and the extension of an concept. The extension of a concept is always some kind of set that has to be enumerated or that has to be constructed from simpler sets using constructor functions. The intension of a concept can be described in terms of the propositions that are applicable to that concept. The extension of a concept can be grasped using set theory, the intension of a concept needs a system that is related to logic.

A type can be seen as a structured object consisting of a concept (the intension) and a corresponding set of individuals that belong to the type (the extension). The extension of a type is a set defined based on basic types and restricted by the conditions in the intension of that type. The intension of a type is a set of set-valued functions (De Brock, 1995). Each function maps an object to a set of propositions of a certain form applicable to that object (Devlin, 1991).

Type abstraction (Devlin, 1991) defines a type based on a set of propositions in which the parameters denoting individuals are identified. For the identification of these parameters, we use the \(\lambda\) operator.

\[ T = \lambda a (\langle\text{set of propositions containing } a \text{ in the role of object}\rangle) \]

According to Devlin, all propositions can be written in the form
\[ x : T \]
which means that object x is of type \(T\).

An alternative notation for this proposition uses the application symbol ::
\[ T :: x \]

This means that type \(T\) is applied to the object x.

A similar abstraction and application can be defined for situations and situation types.

Type construction defines a type in a more hierarchical way. using the following steps:
- define a type based on a set of functions;
- define each function by function abstraction.
Object types can be defined using enumeration (the extensional definition) or by stating logical conditions (the intensional definition). ENBF descriptions consisting of enumerations based on $|$ (logical exclusive OR) therefore can be used as extensional definitions. An object type definition generally consists of an enumeration of parts, of attribute types, and of processes. Standard (non-repeating) parts, attribute types, and processes of an object can be defined based on functions.

As an example of working with type construction, we will develop type definitions based on the following ENBF expression:

$<\text{Fayol’s organization}> ::= \{ <\text{corps social}>, <\text{material organism}>, <\text{contingency factor}>, <\text{symbol structure}>, <\text{person\&task relation}>, <\text{person\&action relation}>, <\text{management process}> \}$. 

Our first step is to define a type for Fayol’s organization based on a set of functions:

$Q = \lambda a ((\text{cs}(a) \neq \bot) \land (\text{mo}(a) \neq \bot) \land (\text{cf}(a) \neq \bot) \land (\text{st}(a) \neq \bot) \land (\text{pt}(a) \neq \bot) \land (\text{pa}(a) \neq \bot) \land (\text{mp}(a) \neq \bot)) \setminus P$

where:
- $P$ is an organization (a type)
- $Q$ is a Fayol’s organization (a type)
- $a$ is a parameter
- cs is a function giving a set of propositions concerning the corps social
- mo is a function giving a set of propositions concerning the material organism
- cf is a function giving a set of propositions concerning a contingency factor
- st is a function giving a set of propositions concerning a symbol structure
- pt is a function giving a set of propositions concerning a person-task relation
- pa is a function giving a set of propositions concerning a person-action relation
- mp is a function giving a set of propositions concerning the management process
- $\bot$ is the symbol for undefined results (bottom)
- $\setminus$ is the select operator

The organization type $Q$ is defined as an organization (the basic type $P$) for which the application of the functions in the set of functions \{cs, mo, cf, st, pt, pa, mp\} does not result in undefined results. The logical condition used for constructing the type $Q$ out of the basic object type $P$ is the function $L_Q$.

$L_Q = \lambda a ((\text{cs}(a) \neq \bot) \land (\text{mo}(a) \neq \bot) \land (\text{cf}(a) \neq \bot) \land (\text{st}(a) \neq \bot) \land (\text{pt}(a) \neq \bot) \land (\text{pa}(a) \neq \bot) \land (\text{mp}(a) \neq \bot))$

This function $L_Q$ can be seen as corresponding to (part of) the intension of the type $Q$. The intension of $Q$ is fully known if all elements occurring in the definition of $L_Q$, for instance the functions cs, mo, …, and so on, are sufficiently defined.

The extension of $Q$ is the set corresponding to the basic type $P$ restricted by the condition resulting from $L_Q$, which can be written as:

$Q = L_Q \setminus P$
The next step is to develop the functions appearing in the function $L_Q$ further. We take the corps social function $cs$ as an example. The corresponding ENBF expression reads:

\[
<\text{corps social}> ::= \\
\{<\text{person}>+, <\text{person}&\text{person quality}>+, <\text{corps social quality}>+, <\text{corps social network quality}>+\}.
\]

\[
cs(q) = (\lambda a \ (W(a, q, 1, \text{‘corps social’}) \land (p(a) \neq \bot) \land (ppq(a) \neq \bot) \land (csq(a) \neq \bot) \land (csnq(a) \neq \bot)) \setminus P)
\]

where:
- $q$ is an organization (a parameter)
- $a$ is a parameter
- $W$ is the whole-part predicate.
- $P$ is an organization (a type)
- $p$ is a function giving a set of propositions concerning a person
- $ppq$ is a function giving a set of propositions concerning a person-person quality
- $csq$ is a function giving a set of propositions concerning a corps social quality
- $csnq$ is a function giving a set of propositions concerning a corps social network quality

As an example of a specification where one reaches the lead nodes of the ENBF specification we take the specification of contingency factors.

\[
<\text{contingency factor}> ::= \\
\{\text{complexityOfProductionProcess(<the organization>)| development(<the corps social>, <value>)}| \\
\text{frequencyOfChangeOfWork(<a task>, <frequency>)}| \\
\text{size(<the corps social>)|} \\
\text{changeOfTechnology(<the organization>, <frequency>)}\}.
\]

\[
cf(q) = (\lambda a \ (\exists \alpha A(q, a, v, u)))
\]

Where:
- $A$ is the attribute predicate.
- $q$ is a Fayol organization (a parameter)
- $a$ is a parameter
- $v : \Re$ (real number)
- $u$ is a unit of measurement
7. Function abstraction

Another sort of abstraction can be found in lambda calculus. Objects can be written as function application:

\[ f :: x \]

an alternative notation is:

\[ y = f(x) \]

which means: the function \( f \) is applied to the object \( x \) giving an object \( y \).

The corresponding function abstraction has the form:

\[ f = \lambda a \ (<\text{construction of an object in which } a \text{ plays a role}>\) \]

For instance, the Law of Requisite Variety (Ashby, 1956) yields the following difference function:

\[ d_{\text{Ashby}} = \lambda uy (+/[y] ((\max/[u] (\lambda uy (v(u) \ast \text{effect}(u, y) - v(y)) \alpha uy)) - m)) \]

In this expression, \( m \) is some number \( \geq 0 \), expressing the optimal surplus variety. This difference function can be characterized as:

\[ d_{\text{Ashby}}: Y \times U \rightarrow D, \text{ where } D \text{ is the domain of the differences } d \]

The equilibrium condition simplifies to:

\[ \forall uy (x(t) \in X_e) \land (x = u \cup y) \land (d_{\text{Ashby}}(u, y) \geq 0) \Rightarrow x(t+1) \in X_e \]

8. Method abstraction

The third sort of abstraction is method abstraction, which abstracts from individual situations in space and time. Method abstraction is based on objects firing methods in response to messages, giving processes.

Methods are defined by method abstraction using an expression that describes a process that has been triggered in a situation by a message, and abstracting that specific message and its situation from it:

\[ m = \lambda a \ (<\text{description of a process that occurs triggered by message } a>) \]

Processes can be written as method application, that is, the firing of a method \( m \) based on a triggering message \( s \):

\[ m :: s \]
The message $s$ is characterized by its situation in terms of location and time, as well as by trigger conditions.

As an example, let us specify a task analysis of human resource management that is part of the ENBF expressions belonging to hypothesis 10.3 of Fayol.

\[
<\text{human resource management}> ::= \{<\text{estimates human requirements}>,
<\text{creates position}>;
<\text{selects person}>;
<\text{hires person}>;
<\text{assigns position, boss, and task to person}>*;
<\text{inspects personnel}>,
<\text{annihilates position}>;
<\text{lays off superfluous personnel}>*\}.
\]

We see that time does not occur in this ENBF specification. In order to be able to use method abstraction, we have to introduce time in the form of parameters.

\[
hrm(q) = \lambda t1t2t3t4 (\forall xn \exists tl t2 t3 t4 dym W(r, q, 1, \text{‘corps social’}) \land W(x, r, n, \text{‘member’}) \Rightarrow W(y, r, m, \text{‘member’}) \land EST(q, t1) \land CRE(q, d, t1) \land SEL(q, x, t2) \land HIR(q, x, t2) \land ASS(q, x, y, d, t3) \land INS(q, x, t3) \land ANN(q, d, t3) \land LAY(q, x, t4) \land (t1 < t2) \land (t2 < t3) \land (t3 < t4))
\]

Where:
hrm(q): a hrm method for organization q
q: a Fayol organization
r: a Fayol corps social
x, y: person
d: symbol structure (position, task)
t1, t2, t3, t4: time
n, m: integer as membership number

The application of the method hrm(q) to a set of situations characterized by their times \{ts1, ts2, ts3, ts4\} gives a process a (in the case of successful application):

\[
a = hrm(q) :: \{ts1, ts2, ts3, ts4\}
\]

Because of the definition of hrm(q), a has the following characteristics:

\[
\forall xn \exists ta1 ta2 ta3 ta4 dym W(r, q, 1, \text{‘corps social’}) \land W(x, r, n, \text{‘member’}) \Rightarrow W(y, r, m, \text{‘member’}) \land EST(q, ta1) \land CRE(q, d, ta1) \land SEL(q, x, ta2) \land HIR(q, x, ta2) \land ASS(q, x, y, d, ta3) \land INS(q, x, ta3) \land ANN(q, d, ta3) \land LAY(q, x, ta4) \land (ta1 < ta2) \land (ta2 < ta3) \land (ta3 < ta4)
\]

In this formula, q is to be interpreted as a constant.
References


