Logics for OO information systems: a semantic study of object orientation from a categorial substructural perspective

de Haas, E.

Citation for published version (APA):
Chapter 4  

A semantics for object oriented information systems

Semantics is a strange kind of applied mathematics; it seeks profound definitions rather than difficult theorems. The mathematical concepts which are relevant are immediately relevant. Without any long chains of reasoning, the application of such concepts directly reveals regularity in linguistic behavior, and strengthens and objectifies our intuitions of simplicity and uniformity.

(J.C. Reynolds [Reynolds80])

The categorial graph language is a language for object oriented information systems for which we claim that the concepts it expresses are those concepts used in practical languages for object oriented information systems. The theory for the language of categorial graphs aims to show that these concepts are materialisable (in mathematics), and are furthermore so in a direct manner (i.e. without encoding). In this chapter we will present the semantics of the family of categorial-graph languages. The semantics will be constructed in the following manner:

We will translate a language of categorial graphs into a logical meta language. This meta language will itself have (again) a semantics and an inference system to reason about the semantic domain. The semantic domain consists of instances of object oriented information systems. These instances contain complex objects and partial descriptions of objects. The three ingredients: the meta language, the inference system, and the semantics, form a logic.

To interpret a categorial graph, the graph and the constraint phrases will be translated into a collection of phrases in the logical meta language. These phrases will constitute a theory in the given logic. Valid models in this theory will be models of valid information system instances of the given categorial graph.
In order to make this scheme work properly, we provide a meta language that inhabits language constructs that are (we are using another vague term here) 'very close' to the language constructs we want to interpret. This means that we provide a meta language that contains high level constructs that reflect directly the primitive structures in the categorial graph language. This way, we have a semantics in which the concepts that are expressed with the categorial graph are interpreted using primitive constructs of the semantics, and not by encoding in low level mathematical concepts.

Note that the distinction between signature and constraints becomes more vague if we turn to the meta language. If we translate the graphical (signatorial) syntax and the textual (constraint language) syntax to the logical language, we have only one textual logical language. Still we pursue the proposed distinction between signature and constraints by designing the logical language in such a way that the graphical ingredients of the object language are intrinsic features of the meta-language. We achieve this by the 'direct' translation of the graphical constructs in its logical constructs.

Summarizing, our semantics for categorial graphs looks as follows: We will provide a meta language in the form of a logical language to interpret the categorial graphs. A categorial graph, then, will be interpreted by a set of logical sentences in the meta language. All models of the theory of these logical sentences are object oriented information systems that satisfy the description of the categorial graph. In a picture:

\[ \begin{array}{c}
\text{Database Schema} \quad \Rightarrow \quad \text{Database Instance} \\
\text{Categorial Graph} \quad \Rightarrow \quad \text{Object Graph} \\
\text{Set of Logical Sentences} \quad \Rightarrow \quad \text{Discourse Model}
\end{array} \]

4.1 Desiderata for meta language for categorial graphs

The definition of the family categorial graphs languages presents us with a list of concepts that we need to be able to express in the meta language in order to be able to translate a categorial graph into the meta language. These concepts are coded in the graphical constructs of the categorial graph language, and we now need to make them explicit to mould them into a logical form.
4.1. Desiderata for meta language for categorial graphs

In the view of the categorial graph language information is present in the form of objects. In principle everything you model is an object. An object has an identity, essential properties -i.e. an object is of some type- and an object has some aspects -i.e. an object has some properties-. Objects and aspects of an object can appear with structure. For example one can say 'an object has two aspects of a certain type'. Moreover one can take objects together. For example one can say 'this object is the aggregation of two objects of some type'. Finally one can express complex constraints on the structure or content of an object.

In the categorial graph language we talk about the objects with graphical entities -edges- that denote types. As we saw in the previous chapter, a type denotes an essential property of an object that is assigned to that type by the typing functor (or inherits from that type because it is assigned to a specialization of that type).

Recall that aspects of an object are denoted by the adjacents of a category. An adjacent of a category types a property of an object of that category. In the object model (the instance) the adjacent object is a property of the object itself. The adjacents form a structure. If one wants to talk solely of 'having a certain kind of adjacent', a set structure is suited to express the adjacents of a category. If one wants to talk about a certain number of adjacents of some kind, the adjacency structure needs to be able to count. A multiset structure can denote this. If one wants to talk about the first adjacent and the second adjacent, one needs a list adjacency structure to express such structure.

The language of categorial graphs also provides an operation to take categories together. Aggregation of two categories delivers a category that is uniquely determined by its components.

Complex constraints can be formulated in a constraint language. This constraint language could, for example, enable one to use Boolean constructs to logically combine properties, formulating a complex constraint.

Below we list the desirables for what we want to be able to say about complex objects.

1. talk about essential properties of objects
2. talk about aspects of objects, which are in the OO philosophy information objects in their own right
3. talk about complex constraints on the objects using Boolean connectives like conjunction (∧), disjunction (∨), negation (¬) and implication (→)
4. talk about aggregations of objects (structurally and resource consciously)

5. talk structurally and resource consciously about aspects of objects

We will use the elaborated arsenal of modern formal logic to express the concepts listed in the desiderata. The first desiderata means that we need to be able to state (at least) propositions on whole objects. The second states that we need to be able to assert propositions about the adjacents of an object. Propositions about aspects are expressed with modal propositions (\(\Diamond P\) where \(P\) is a proposition). The third desiderata -complex constraints- introduces the need for Boolean connectives in order to make complex assertions about objects and their aspects\(^1\). The 4th item -aggregation- introduces the need for a connective that is interpreted as taking together objects. This will be the \(*\) (we use the same symbol as the related resource conscious conjunction of linear logic). The last item in our list requires an aggregation operation in a modal context. Although the items by them selves seem to introduce clear ingredients to the language, the combination of all the desiderata will appear to be problematic when we want to build a logic for the language with all the desired ingredients.

Note that when we look at the discussion in the previous chapters, we have even more desiderata. These are things we want to be able to express, but are not bound to language constructs. The most important of these are: labels, non-wellfoundedness and incomplete specification of objects (i.e. not all aspects known and/or the aspect structure not known). These are not desiderata that influence the language constructs, but are inherent to the interpretation of the logical connectives we use. This will become clear when we present the interpretation of the meta language.

In talking about the adjacency structure, we add the following remark. The languages we will define will have an interpretation in a semantic domain populated by complex objects. When issues like structural properties become important (due to items 4 and 5), the complex objects in the semantic domain need to have structural properties as well. This means that when we can say things about objects taken together, such aggregation needs to be defined on the objects in the semantic domain. We note that there are several variants on the result of taking objects together, which can all be accounted for in the structural rules for the logic that talks about the variant in focus. This will be elaborated when we present the logical calculus and the interpretation of the logic for categorial graphs below.

\(^1\)Note that we omitted the self operator here. Although very powerful, and important in the broad OO context, the self operator is not an intrinsic part of the core system. We will see that we can add it nicely to the core system in a logical context. We will spend some words on that when discussing the logical aspects of our system.
Recall the enumeration of desiderata (1-5). We summarize the possibilities\(^2\) we will consider in the table below.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>atomic typing of essential properties only</td>
</tr>
<tr>
<td>ii</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>atomic typing of aspects only</td>
</tr>
<tr>
<td>iii</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>atomic typing of essential properties and aspects</td>
</tr>
<tr>
<td>iv</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>Propositional logic for essential properties</td>
</tr>
<tr>
<td>v</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>Propositional logic for aspects</td>
</tr>
<tr>
<td>vi</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>atomic typing of essential properties only and possibility to say something about aggregated whole objects (structure)</td>
</tr>
<tr>
<td>vii</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>atomic typing of aspects only and possibility to say something about aggregated whole objects (structure)</td>
</tr>
<tr>
<td>viii</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>atomic typing of aspects only and possibility to say something about the adjacency structure</td>
</tr>
<tr>
<td>ix</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>atomic typing of essential properties and aspects and possibility to say something about aggregated whole objects (structure)</td>
</tr>
<tr>
<td>x</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>atomic typing of essential properties and aspects and possibility to say something about the adjacency structure</td>
</tr>
<tr>
<td>xi</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>propositional modal logic for essential properties and aspects</td>
</tr>
<tr>
<td>xii</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>propositional logic with structural (aggregation ,product) connective</td>
</tr>
<tr>
<td>xiii</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>propositional logic for aspects with structural connective (aggregation ,product)</td>
</tr>
<tr>
<td>xiv</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>Propositional modal logic for essential properties and aspects combined with a structural (aggregation, product) connective for taking together essential properties</td>
</tr>
<tr>
<td>xv</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>Propositional modal logic for essential properties and aspects combined with a structural (aggregation, product) connective for specifying structure of the aspects of an object</td>
</tr>
<tr>
<td>xvi</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Propositional modal logic for essential properties and aspects combined with a structural (aggregation, product) connective for taking together essential properties and a structural (aggregation, product) connective for specifying structure of the aspects of an object</td>
</tr>
</tbody>
</table>

The languages without logical connectives (without 3) are actually simple type systems, for which there are hardly any rules (we only type data with it).\(^2\)Note that some combinations make no sense, like combining propositions on whole objects (essentials) with a product for adjacents (and nothing else).
ertheless they are widely used as subsets of the modeling languages. Many Data
models are written with only the typing language. The calculus for these lan-
guages is nearly trivial, because we only need to have rules for the aggregation or
product connectives, in order to let them behave as the aggregation does in the
model. For example, if in the model we cannot count then the 'type' \( A \) will be
the same as \( A \ast A \). We will go formally into these matters after we have defined
the models.

For the languages with connectives things get complicated right away. Es-
pecially if we want to combine with these connectives the things we say about
essential properties of objects with things we say about aspects of objects. More-
over we have the classical problem of using negation in an information model,
where the description specified by \( \neg A \) is satisfied by information we may not
have in our information model.

In summary the meta language of categories has the following features:

- *Expresses propositions on the complex structure of an object using a modality
  for adjacency*, i.e. a proposition about the structure of an object that says
  that an object has some kind of adjacent is expressed with a modality. The
  modality will have an existential character, and will be denoted by the \( \diamondsuit \).
  This means we can express that an object has some structural properties
  without knowing its structure totally. For example the complex objects\(^3\) \( p, r \) and \( s \) will all be of type \( \diamondsuit A \).

![Diagram](image)

If we want to talk about a directed adjacency structure, i.e. about the
first adjacent, the second adjacent, etc. etc., we need to label the modal

\( ^3 \)A note on the informal notation for an object: the structure of a complex object is denoted
by a box using a circle as placeholder for an adjacent object and a line from the placeholder
to the box that denotes the adjacent. This line is called a *link*. The label for the object will
be a lower case letter \( (a,b,c,\ldots) \), and the type of an object will be denoted by a capital letter
\( (A,B,C,\ldots) \). The label \( a:A \) will mean 'object \( a \) of type \( A \)'.

---

\( \diamondsuit \)
adjacency operator: $\diamond^1, \diamond^2, \diamond^\text{role}$, etc.. Note that we may use symbolic names as labels, if we want.

- Has a linear aggregation operation. This means that we can express the 'taking together' of two objects resulting in a new complex object. For example if we have two objects $p$ and $r$, and $r$ has two adjacents $s$ and $t$, $p$ has an adjacent $q$, then their aggregation $p \cdot r$ will have three adjacents, $q$, $s$ and $t$. In a picture:

![Diagram showing adjacency and aggregation]

4.2 The meta language of categorial graphs

4.2.1. Definition. (The meta language of categorial graphs)
Let $S_{\text{cat}}$ be a set of operators containing one unary modal operator $\diamond$, together with the modal constant $1$. We define the language $L_{\text{cat}}$ to be the pair $(S_{\text{cat}}, Q_{\text{cat}})$, where $Q_{\text{cat}}$ is a set of propositional variables. The set $\Phi(L_{\text{cat}})$ of formulas in $L_{\text{cat}}$ is defined as usual, using, next to the modal operators, the connectives $\cdot, \land, \lor, \neg$.

The $\diamond$ modality will talk about the adjacency relation $R$, and 1 models the empty (or unknown) objects. Furthermore, the $\cdot$ denotes aggregation, or in other words, the multiplicative or resource conscious conjunction. The connectives $\land$ and $\lor$ are respectively additive (non resource conscious) conjunction and additive disjunction. Finally the $\neg$ denotes (non resource conscious) negation.
Let us give some informal interpretation of the connectives before getting formal. The $\cdot$ (multiplicative conjunction) is the resource conscious 'and' connective. Informally, an object of type $A \cdot B$ will be an object that is an aggregation of two objects, one of type $A$ and one of type $B$. The $\sqcap$ (additive conjunction) is the traditional 'and' ($\land$) connective. If an object is of type $A \sqcap B$ it informally means that the one object itself is both of type $A$ and of type $B$. The $\sqcup$ (additive disjunction) is similar to the disjunction ($\lor$) of classical logic. Informally, an object is of type $A \sqcup B$ if it is either of type $A$ or of type $B$. The $\neg$ (negation) is a non constructive and non-resource-conscious negation. An object is of type $\neg A$ if it is not of type $A$. The $\Diamond$ will model adjacency. An object of type $\Diamond A$ is an object that has an adjacent of type $A$. Finally $1$ will be the type of the empty object. We will make sure the type $A \cdot 1$ will have the same interpretation as the type $A$; in other words an object composed from an object of type $A$ and the empty object will be of type $A$, simply because composing (aggregating) with the empty object will be the identity operation.

We can extend the language by adding the following modalities that are induced by the adjacency relation:

1. the $\Diamond^+$ modality, which talks about the transitive closure of $R$.

2. the $\Diamond^{-1}$ modality, which talks about the inverse of $R$.

These modalities need axioms to be formally defined in the calculus.

4.2.2. Example. Recall the running example of chapter 3 (example 3.1.1). We can express the graph expressions in the meta language of categorial graphs as follows:

\[
pilot \Rightarrow \Diamond \text{name} \cdot \Diamond \text{empno} \cdot \Diamond \text{qualif} \\
pilot \Rightarrow \Diamond \text{roster} \\
roster \Rightarrow \Diamond (1 \sqcup \text{task}) \text{ (for set flavoured this suffices)} \\
roster \Rightarrow \Diamond (1 \sqcup \text{task} \sqcup (\text{task} \cdot \text{task}) \sqcup \ldots \sqcup (\text{task} \cdot \ldots \cdot \text{task})) \\
(roster \Rightarrow \Diamond (!\text{task}) \text{ when we introduce the bang (!)}) \\
\text{task} \Rightarrow \Diamond \text{task}\_\text{description} \\
\text{task} \Rightarrow \Diamond \text{start} \\
\text{task} \Rightarrow \Diamond \text{end} \\
pilot \Rightarrow \Diamond \text{person} \\
\text{task} \sqcap \Diamond (\text{task}\_\text{description} \sqcap \text{flying.duty}) \Rightarrow \Diamond^{-1}(\text{person} \rightarrow \text{pilot})
\]

when we introduce self and the functions we can tackle the other constraint:

\[
\text{start} \sqcap \Diamond^{-1}((\text{task} \sqcap \neg \Diamond \text{self}) \cdot (\text{task} \sqcap \Diamond (\text{task}\_\text{description} \sqcap \text{flying.duty}) \sqcap \Diamond \text{self})) \Rightarrow \Diamond^{-1}((\text{task} \sqcap \Diamond f_{3,11}(\text{start}, \text{self}) \cdot (\text{task} \sqcap \Diamond \text{self}))
\]
4.3 Calculus for the meta language of categorial graphs

In this section we present a variety of rules and axioms for the meta language of categorial graphs. This language, together with the calculus, constitutes a logic we call the logic of categories. Several rules and axioms directly correspond to the structural issues we presented in the previous sections. With this we mean that, in a multiset structure for example, we have rules that respect that we can count aspects.

The rules and axioms for our logic are based on the calculus for linear logic (Girard87, Troelstra92). The 'resource consciousness' paradigm of Girard's linear logic (Girard87) triggered the use of such a logic for categorial graphs. Logics such as linear logic emerged from a broader landscape of logics, which is the framework of substructural logics. In a Gentzen-style sequent formulation, a substructural logic distinguishes itself by the absence of some structural rules that are common in the Gentzen-style formulation of the most common logics like classical or intuitionistic logic. Well known substructural logics are 'relevance logic' (Dunn86), categorial logic (Lambek58, Benthem91) and 'BCK logic' (OnoKomori85). Linear logic differs from these substructural logics by allowing some limited or controlled use of the structural rules using logical (modal) operators.

The axioms and rules for the logic of categories will be presented in Gentzen-style sequent calculus with the restriction that the sequents have, exactly, one formula on the right. A sequent thus will have the following format:

\[ \Gamma \Rightarrow A \]

where \( \Gamma \) is a sequence of formulas \( A_1, \ldots, A_n \). The sequence of formulas can intuitively be interpreted as a comma separated list of resources. The \( \Rightarrow \) is interpreted as provability:

if \( \Gamma \Rightarrow B \) then from \( \Gamma \) we can prove \( B \)

For basic connectives \( \ast, \sqcup, \sqcap, 1, \bot, \) and \( T \), we present the usual axioms and rules. We also have modalities in the linear language that are different from the ones that are usually studied in the field of linear logic and modal logic (see e.g. Bucalo94). The fundamental difference lies in the fact the accessibility relations we consider are not only set-based, but are also multiset-based, or list-based.

\[4\]In order to avoid misunderstanding we note that for regulated use of the structural rules we, of course, use other modal operators than the modal operator for adjacency. In fact when we introduce a modal operator for regulated use of the structural rules, we get a multi modal logic.
4.3.1. Definition. (The basic rules for categories)

Rules and axioms for the non-modal part of the calculus:

\[(AX)\quad A \Rightarrow A\]

\[(CUT)\quad \Gamma \Rightarrow A, \Gamma', A \Rightarrow B \quad \therefore \Gamma, \Gamma' \Rightarrow B\]

\[(L\cap)\quad \Gamma, A \Rightarrow C \quad \Gamma, A \cap B \Rightarrow C \quad \Gamma, B \Rightarrow C \quad \Gamma, A \cap B \Rightarrow C\]

\[(L*)\quad \Gamma, A \ast B \Rightarrow C\]

\[(L\cup)\quad \Gamma, A \Rightarrow C \quad \Gamma, B \Rightarrow C \quad \Gamma, A \cup B \Rightarrow C\]

\[(L1)\quad \Gamma, \top \Rightarrow A\]

\[(L\bot)\quad \Gamma, \bot \Rightarrow A\]

\[(L\neg)\quad \Gamma, \neg A \Rightarrow \bot\]

\[(R\cap)\quad \Gamma \Rightarrow A, \Gamma \Rightarrow B \quad \therefore \Gamma \Rightarrow A\cap B\]

\[(R*)\quad \Gamma \Rightarrow A, \Gamma' \Rightarrow B \quad \therefore \Gamma, \Gamma' \Rightarrow A \ast B\]

\[(R\cup)\quad \Gamma \Rightarrow A \quad \Gamma \Rightarrow B \quad \therefore \Gamma \Rightarrow A \cup B\]

\[(R\bot)\quad \Gamma \Rightarrow \bot\]

\[(R\neg)\quad \Gamma \Rightarrow \neg A\]

For the adjacency modality we add the following:

\[(\Diamond I)\quad A \Rightarrow B \quad \therefore \Diamond A \Rightarrow \Diamond B\]

\[(\Diamond EXISTENTIAL)\quad \Gamma \Rightarrow \Diamond A \ast \Diamond B \quad \therefore \Gamma \Rightarrow \Diamond A\]

The basic set of axioms and rules axiomatizes a language that can talk about essential properties of objects (i.e. whole objects) and about aspects of objects (i.e. partial description of objects). Expressions on aspects of objects can be done using the \(\Diamond\) modality. The existential character of the \(\Diamond\) modality is axiomatized by the '\(\Diamond EXISTENTIAL\)' rule. This rules says (informally) that when an object of type \(\Gamma\) has an \(A\) and a \(B\) adjacent, then we may conclude that an object of type \(\Gamma\) has an \(A\) adjacent.

In the presentation of the models of discourse spaces we stated that the adjacency relation either consist of list, multiset, or set adjacency structures. Note that this basic set of rules can only be sound and complete for an interpretation that involves edges with a list adjacency relation because we cannot show with this basic collection of rules the behavior of a multiset adjacency relation,

\[\Diamond A \ast \Diamond B \Rightarrow \Diamond B \ast \Diamond A ,\]
nor behavior of a set adjacency relation,

$$\Diamond A * \Diamond A \Rightarrow \Diamond A .$$

For axiomatizing this kind of behavior we need structural rules. In the list below we will introduce rules that enable or disable certain expressivity of the language and behavior of the models.

- **rules for adjacency structure**
  1. Adjacency structure is undirected; i.e. there is no order in the adjacents of an object.

$$\frac{\Gamma, \Diamond A, \Diamond B, \Gamma' \Rightarrow C}{\Gamma, \Diamond A, \Diamond A, \Gamma' \Rightarrow C} \quad (\Diamond \text{EXCHANGE})$$

2. Adjacency structure is non-resource-conscious; i.e. there is no notion of counting adjacents (only existence of a type of adjacent matters)

$$\frac{\Gamma, \Diamond A, \Diamond A \Rightarrow \Delta}{\Gamma, \Diamond A \Rightarrow \Delta} \quad (\Diamond \text{CONTRACTION})$$

$$\frac{\Gamma, \Diamond A \Rightarrow \Delta}{\Gamma, \Diamond A, \Diamond A \Rightarrow \Delta} \quad (\Diamond \text{WEAKENING})$$

- **Rules for whole object structures**
  1. Aggregate structure is undirected; i.e. there is no order in the aggregates of an object.

$$\frac{\Gamma, A, B, \Gamma' \Rightarrow C}{\Gamma, B, A, \Gamma' \Rightarrow C} \quad \text{provided that } A \text{ is not of the form } \Diamond G \quad (\text{RESTRICTED-EXCHANGE})$$

2. Aggregation of whole objects is non-resource-conscious; i.e. there is no notion of counting aggregates.

$$\frac{\Gamma, A, A \Rightarrow \Delta}{\Gamma, A \Rightarrow \Delta} \quad \text{provided that } A \text{ is not of the form } \Diamond G \quad (\text{RESTRICTED-CONTRACTION})$$

$$\frac{\Gamma, A \Rightarrow \Delta}{\Gamma, A, A \Rightarrow \Delta} \quad \text{provided that } A \text{ is not of the form } \Diamond G \quad (\text{RESTRICTED-WEAKENING})$$

- **General rules**
  1. All structure is undirected (i.e. there is no order in the aggregates and neither in the adjacents of an object)

$$\frac{\Gamma, A, B, \Gamma' \Rightarrow C}{\Gamma, B, A, \Gamma' \Rightarrow C} \quad (\text{EXCHANGE})$$
2. All structure is non-resource-conscious; i.e. there is no notion of counting whole objects, neither is there a notion of counting adjacents (only existence of a type of adjacent matters)

\[(CONTRACTION)\]  \(\frac{\Gamma, A, A \Rightarrow \Delta}{\Gamma, A \Rightarrow \Delta}\)

\[(WEAKENING)\]  \(\frac{\Gamma, A \Rightarrow \Delta}{\Gamma, A, A \Rightarrow \Delta}\)

In order to illustrate the effect, the rules from above enable the following identifications in the categorial graphs language (specifying types):

\[(D\ EXCHANGE)\]

\[(D\ CONTRACTION)\]

\[(D\ WEAKENING)\]

\[(RESTRICTED-EXCHANGE)\]

\[(RESTRICTED-CONTRACTION)\]

\[(RESTRICTED-WEAKENING)\]
The rules from above have serious implications on the semantics of the language of categorial graphs. For example, by introducing $\Diamond \textit{EXCHANGE}$ the theory can not distinguish between objects with different order of adjacents. This means that the semantics of the language can not contain any reference to the order of the adjacents. In other words, the two expressions $\Diamond A \ast \Diamond B$ and $\Diamond B \ast \Diamond A$ will have the same meaning, just like the two graphs above at the $\Diamond \textit{EXCHANGE}$ label. We will typically interpret the language with the $\Diamond \textit{EXCHANGE}$ in a semantic domain with objects with an adjacency structure which has no order.

4.4 The semantic domain for object oriented information systems

The meta language of categories has an interpretation in a semantic domain. We can construct a landscape of semantic domains for which we can present different sets of axioms and rules that are sound with respect to the proposed semantics by adding or omitting these particular rules\(^5\). The semantic domains will (all) have the following features:

- Both 'whole objects' and 'partial description of objects' are members of the semantic domain. This means that in our semantics we can point at an

\(^5\)very much like the landscape of substructural logics is given its variety by adding and omitting structural rules
object $a$, but also at an aspect $b$ of an object $a$.

- **An adjacency structure models the complex structure of an object.** The adjacency structure of an object contains the adjacents of an object, and this structure is as rich as the rules of the calculus describe. In effect this means that the adjacency structure of an object is either a set, a multiset or a list. For example in the multiset case this means that if some object $a$ has three adjacents: $b$, $c$ and again $c$, then $\{(a, b), (a, c), (a, c)\} \in R_{\text{Adj}}$, where $R_{\text{Adj}}$ denotes the adjacency multiset relation.

- **A monoid structure over the object domain interprets aggregation.** This means that there is an operation $\cdot$ on complex objects that interprets aggregation, and that aggregated objects are members of the domain of objects.

- **Identity.** For handling the notion of identity properly we may not 'a priori' identify an object by its structure. In a world with an object notion there can be two different objects with the same adjacents (contrary to the relational world, which requires semantically meaningless attributes as key to force that two objects that in some knowledge state cannot be distinguished by there properties, but may turn out to be two different objects anyhow). This means that characterizations of objects by describing their structure are, in general, partial.

- **Links and partial descriptions.** In order to interpret formulas with an existential character, like "an $A$ object is an object that has at least two adjacents, one $B$-adjacent and one $C$-adjacent"$^8$, we will calculate in our models with partial descriptions of objects. These partial descriptions, called *aspects* or *links*, will be the witnesses of one particular object being the adjacent of another particular object. If we graphically write down an object, a link can be seen as the line between the object and its adjacent. In our semantics we denote an aspect or link by an ordered pair $(a, b)$ meaning the aspect witnessing that$^9$ $b$ is adjacent to $a$.

We interpret the connectives and operators of the meta language $L_{\text{cat}}$ of categorial graphs in the semantic domain. The construction of this semantic domain is a variation on a Kripke style semantic domain. The domain will be generated by a set of atomic objects $E_{\text{At}}$ and a function $f_R$ (e.g. the characteristic function

---

$^6$For example both the whole object named *Socrates* and its partial description the *white color of Socrates* can be members of the semantic domain

$^7$A remark on notation: We use two dots to distinguish the multiset symbols and operations from their normal set variants

$^8$in our language that is denoted by $\cap A \ast \cap B$

$^9$or more concrete: the aspect witnessing that *Socrates* is white
of a relation $R$) that describes the adjacency structure of the objects. The mathematical items $E_A$ and $f_R$ together generate a hybrid structure which is called a discourse frame with two dimensions of so called 'structures' (or monoidals) of mathematical elements:

1. one dimension is a 'structure' with mathematical elements behaving like whole objects (and aggregations of whole objects) which can interpret the propositions on whole objects; this structure will be called 'space of wholes'.

2. the other dimension contains for every (aggregation of) whole object(s) a lattice in the 'structure' of mathematical elements behaving like information-pieces (called 'aspects' or 'links' or 'infons') which can interpret the propositions on aspects or partial descriptions of whole objects; these structures will be called 'adjacency spaces'.

This discourse frame, together with a valuation (interpretation), form the so called discourse models that interpret the language of categorial graphs.

4.4.1. DEFINITION. (structure domain)

A structure domain is a triple $\langle C, \cdot, 1 \rangle$ where

- $C$ is a collection of elements (structures)
- $\cdot : C \times C \to C$ is a product
- $1 \in C$ is the unit element for \( \cdot \)

In the course of this section we distinguish 4 types of structure domains based on two properties:

1. commutativity: $e_1 \cdot e_2 = e_2 \cdot e_1$
2. idem-consuming/cloning: $e \cdot e = e$

The 4 variants of the structures are now:

1. non-commutative and non-idem-consuming/cloning (i.e. lists)
2. commutative and non-idem-consuming/cloning (i.e. multisets)
3. non-commutative and idem-consuming/cloning (i.e. lists with no identicals right after another)
4. commutative and idem-consuming/cloning (sets)
Within a structure we have an ordering $\leq$ called a substructure ordering that satisfies the following condition:

$$a \leq b \text{ implies } a \cdot c \leq b \cdot c \text{ and } c \cdot a \leq c \cdot b$$

$$a \leq b \begin{cases} 
    a = 1 \\
    a = b \\
    a = a_1 \cdot \ldots \cdot a_n \& b = b_1 \cdot \ldots \cdot b_n \& a_i \leq b_i (1 \leq i \leq n) \\
    \text{(where } \& \text{ means equivalence in the structure domain)}
\end{cases}$$

Both dimensions of the model, the space of wholes and all the adjacency spaces will be structures.

4.4.2. Definition. ('space of wholes')

Given a set $E^{At}$ of objects called 'atomic objects', a space of wholes $E$ is a structure domain $<E,.,1>$ freely generated from $E^{At}$. (The ordering in the space of wholes is simply the substructure ordering, which we do not need in the definitions further on)

The unit element 1 of the space of wholes will interpret the empty type 1 (i.e. the empty type is interpreted by the empty object). If the space contains objects $a_1, a_2$ and $a_3$, then it also contains aggregates $a_1 \cdot a_2$, and also $a_1 \cdot a_2 \cdot a_3$. But also $a_1 \cdot a_1 \cdot a_2 \cdot a_3 \cdot a_2$ etc. etc. etc. The multiplication operation is defined on objects. We will also define the multiplication on sets of objects (denoted by the same token ('.')) as usual: $X \cdot Y := \{x \cdot y \mid x \in X, y \in Y\}$.

Objects have adjacents. The adjacency structure of objects will be modeled by an adjacency function. This function maps a whole object to its (full) adjacency structure element, which is the structure element that models the complex structure of the object. Such a structure element is called an 'infon' and is an element of a structure with the 'information pieces' or 'links' as atomic elements. Such an atomic link is a witness to the fact that 'one object $b$ is in the adjacency structure of another object $a$', and will be denoted by an ordered pair $(a, b)$. The mathematical behavior of the infons (structure elements) will be determined by the rules that hold in the adjacency structure. For example if the adjacency structure is commutative, the behavior of the infons (structure elements) $(a, b) \cdot (a, c)$ and $(a, c) \cdot (a, b)$ containing both the two atomic links $(a, b)$ and $(a, c)$ will be identical. In effect an infon (structure element) representing the (whole or part of the) adjacency structure will be either a set (when both commutativity and idem-consumption hold), multiset (when only commutativity holds), or a list (when neither of the rules hold) of pairs. The function that maps an object to its adjacency structure element will interpret the $\Diamond$-modality. The proposition $\Diamond a$ will be interpreted as 'has an $A$-type adjacent'. In other words the adjacency operator talks about aspects (links or infons) of an object; i.e. partial descriptions.
4.4.3. **Definition.** (links)
Let $E = \langle E, \cdot, 1 >$ be a space of wholes and $e_1, e_2 \in E$. Then a *link* of object $e_1$ is a pair $(e_1, e_2)$. This pair witnesses the fact that $e_2$ is an adjacent of $e_1$. The pair $(e_1, e_2)$ is an *information piece* that partially describes object $e_1$. It is an *individual aspect* of object $e_1$.

4.4.4. **Definition.** ("Adjacency spaces")
Let $E$ be a space of wholes. Then for each element in $e \in E$ we define a structure domain $A_e$ freely generated from all possible links of $e$. i.e. let $A_e^{at} = \{e\} \times E \subseteq A_e$ be the set of all links of $e$; then

$$A_e = \langle A_e, \cdot, (e, 1), \leq_e >$$

is the structure domain freely generated by $A_e^{at}$. $A_e$ is called the *adjacency space* of $e$, and the elements of $A_e$ are called *partial descriptions* or *infons* of $e$.

4.4.5. **Definition.** (adjacency (structure) mapping) An adjacency structure mapping is a function $f_R : E \mapsto \bigcup_{e \in E} A_e$ that maps each atomic object to a structure element in its adjacency space. This structure element describes the full adjacency structure of the whole object. i.e.

$$f_R(e) \in A_e$$

The domain of $f_R$ can be extended to range over all objects including aggregates $(E)$ by putting (regularity):

$$f_R(a \cdot b) = (a \cdot b, c_1) \cdot a \cdot b \ldots a \cdot b (a \cdot b, c_n) \cdot a \cdot b (a \cdot b, d_1) \cdot a \cdot b \ldots a \cdot b (a \cdot b, d_n)$$

iff

$$f_R(a) = (a, c_1) \cdot a \ldots a (a, c_n)$$

and

$$f_R(b) = (b, d_1) \cdot b \ldots b (b, d_n)$$

4.4.6. **Definition.** (adjacency (structure) relation)
Given a space of wholes $E$ as above, we define an adjacency (structure) relation $R : E \times \bigcup_{e \in E} A_e$ by the adjacency structure mapping as follows:

$$e Ra \text{ iff } f_R(e) = a' \& a \leq_e a'$$

In other words, $R$ relates an object to a partial description (element from an adjacency space) iff the partial description is a substructure of the partial description that is the adjacency structure element of that object. i.e. the whole adjacency structure is related to its object, together with all the infons that are a substructure of that (total) adjacency structure element.
To clarify the definition of adjacency structure relations we list the 'special cases' where the structure is respectively a set, a multiset, and a list.

Case set: \( f_R \) maps an object to a set of partial descriptions. The adjacency structure of an object is a set. \( R \) is a set of pairs; pairs have only 'presence' as feature in \( R \). The adjacency space will have as elements sets of pairs with set inclusion as ordering.

Case multiset: \( f_R \) maps an object to a multiset of partial descriptions. The adjacency structure of an object is a multiset. \( R \) is a multiset of pairs; pairs will have 'presence' and 'arity' as feature in \( R \) (that is asserting arity=0 means 'not present', we could say they only have 'arity' as feature in \( R \)). The adjacency lattice (defined below) will have as elements multisets of pairs with multiset inclusion as ordering.

Case list: \( f_R \) maps an object to a list of partial descriptions. The adjacency structure of an object is a list. \( R \) consists of lists of pairs; pairs will have 'presence', 'arity', and 'positions' (one position for each occurrence) as feature in \( R \). The adjacency lattice will have as elements lists of pairs with list inclusion as ordering.

In this informal formulation the generic structure is formulated as follows:

Generic case structure: \( f_R \) maps an object to a structure of partial descriptions. The adjacency structure of an object is a structure. \( R \) consists of structures of pairs (a generalization of the relation concept, from which a traditional relation, a multiset relation and a relation consisting of lists are special cases). The structures of pairs will behave according to the rules of the structure. The adjacency lattice will have as elements structures of pairs with structure inclusion as ordering.

An adjacency lattice contains (unordered) sequences of pairs \((e_1, e_2)\), where \( e_1 \) is an object and \( e_2 \) is one of its adjacent objects. We will treat the elements of the adjacency lattices, i.e. the infons, as objects with a special property: they are 'extendible' to all elements that are more informative than themselves, with in the limit the 'whole' that they are a partial description of\(^{10}\). For example the infon \((e_1, e_2)\) is interpreted to be extendable to the 'whole' \( e_1 \). As an other example the aggregation of the infons \((e_1 \cdot e_3, e_2) \cdot (e_1 \cdot e_3, e_2)\) will be extendible to \( e_1 \cdot e_3 \). Whole objects will be extendible to themselves only, because they describe exactly one individual, and there is no description more informative or precise than the individual itself.

\(^{10}\)In a sense an infon has an existential character: 'There exists an edge object that I am a partial description of'.
We will now combine the two worlds, the world of infons (partial descriptions) and the world of whole objects. The combined structure is a hybrid structure containing as elements both the infons and the whole objects. This hybrid space will be called *discourse space*. The structure has two dimensions, one dimension playing at the level of the individual adjacency lattices (i.e. within the description of one whole object), and the other dimension playing at the level of (aggregation of) whole objects. Furthermore the interaction between these dimensions needs to be controlled. This means the following:

- The aggregation of infons from one individual adjacency lattice for an object \( e \) is the product in this lattice obeying the rules for taking together partial descriptions, and

- The aggregation of two whole objects is the product within the space of wholes, obeying the rules for taking together propositions about whole objects.

- the aggregation outside an individual adjacency lattice or involving both whole objects and infons is a product operation regulating the traffic between the two worlds.

The first two products are already defined. The product between elements of different worlds (i.e. between two infons of different adjacency lattices or between an infon and a whole object), called a *hybrid product*, needs some more discussion. In general the product operation is modeling the aggregation operation that is taking together two pieces of information. This taking together of two pieces of information can behave in different ways, it can for example be resource conscious or sensitive for the order in which the information is taken together. For a product on uniform behaving elements in our model, e.g. the product of two whole objects or the product of two infons of one object, this product can simply obey the rules in the specific subspace\(^{11}\). For the product of two differently behaving elements we need to take a little more care. The behavior of taking together elements in the different subspaces should be conservative w.r.t. the taking together of elements in the different subspaces, in the sense that it should not be possible to obtain equivalences in the separate subspaces by using the hybrid product which one can not obtain using soley the product in the different subspaces.

In effect this means that the product of two elements of different worlds will be a term that will have structural properties (like commutativity, associativity, idem-consumption, and cloning) only if the products in the other two worlds both have these properties.

4.4.7. DEFINITION. (hybrid product)
Let \( \mathcal{E} \) be a space of wholes and let \( \mathcal{A}_e \) (for all \( e \in E \)) be a collection of adjacency

\(^{11}\)The space of wholes and the separate adjacency lattices are all spaces with a product.
spaces generated from a collection of objects $E^A$ and an adjacency relation $R$. Let $o_1, o_2 \in E \cup \bigcup_{e \in E} A_e$, then

$$o_1 \cdot o_2 = \begin{cases} \ o_1 \cdot_A o_2 & \text{if } o_1, o_2, o_1 \cdot_A o_2 \in A_e \\ \ o_1 \cdot_E o_2 & \text{if } o_1, o_2 \in E \\ \ o_1 \cdot_H o_2 & \text{otherwise} \end{cases}$$

Now we formalize our notion of *extendibility* relating the separate worlds of infons to the world of whole objects. An infon will be extendable to the infons that are more informative than itself but still a partial description of the whole object, and to the (aggregation of) whole object(s) of which it is the witness of one or more aspects. Whole objects will be extendable to themselves only.

### 4.4.8. Definition. (extendibility)

Let $E$ be a space of wholes and let $A_e (e \in E)$ be a collection of adjacency spaces generated from a collection of objects $E^A$ and an adjacency relation $R$. For $o \in E \cup \bigcup_{e \in E} A_e$, we define

$$\text{Ext}(o) = \begin{cases} \ \{o\} & \text{if } e \in E \\ \ \{a \mid o \in A_e, o \leq a \& a \leq f_R(o)\} \cup \{e \mid o \in A_e\} & \text{if } a \in \bigcup_{e \in E} A_e \\ \ \emptyset & \text{otherwise} \end{cases}$$

(Note that the Ext operations filters out all undefined products)

### 4.4.9. Example. Look at the following objects $E^A = \{a, b_1, b_2, c, d\}$ with the adjacency multiset relation $R = \{(a, b_1), (a, b_2), (c, d)\}$. The following then are examples of adjacency lattices (in the case of a multiset adjacency structure, i.e. associative and commutative space for the adjacents):

- $A_a = \{(a, b_1), (a, b_2), (a, b_1) \cdot (a, b_2)\}$
  - where $(a, b_1) \leq (a, b_1) \cdot (a, b_2)$ and $(a, b_2) \leq (a, b_1) \cdot (a, b_2)$
- $A_e = \{(c, d)\}$
- $A_a \cdot e = \{(a \cdot c, b_1), (a \cdot c, b_2), (a \cdot c, d), (a \cdot c, b_1) \cdot (a \cdot c, b_2), (a \cdot c, b_1) \cdot (a \cdot c, d), (a \cdot c, b_2) \cdot (a \cdot c, d), (a \cdot c, b_1) \cdot (a \cdot c, b_2) \cdot (a \cdot c, d)\}$
  - where $(a \cdot c, b_1), (a \cdot c, b_2) \leq (a \cdot c, b_1) \cdot (a \cdot c, b_2)$
  - and $(a \cdot b_1), (a \cdot c, d) \leq (a \cdot c, b_1) \cdot (a \cdot c, d)$
  - and $(a \cdot c, b_2), (a \cdot c, d) \leq (a \cdot c, b_2) \cdot (a \cdot c, d)$
  - and $(a \cdot c, b_1) \cdot (a \cdot c, b_2), (a \cdot c, b_1) \cdot (a \cdot c, d), (a \cdot c, b_2) \cdot (a \cdot c, d) \leq (a \cdot c, b_1) \cdot (a \cdot c, b_2) \cdot (a \cdot c, d)$
and the following are examples of products and extensions:

\[
\begin{align*}
\text{Ext}(a) &= \{a\} \\
\text{Ext}(b_1) &= \{b_1\} \\
\text{Ext}((a, b_1)) &= \{(a, b_1), (a, b_1) \cdot (a, b_2), a\} \\
\text{Ext}((a, b_1) \cdot (a, b_2)) &= \{(a, b_1) \cdot (a, b_2), a\} \\
\text{Ext}((a, b_1) \cdot (c, d)) &= \emptyset \\
\text{Ext}((a, b_1) \cdot (a, b_1)) &= \emptyset
\end{align*}
\]

4.4.10. Definition. (discourse space)

Let \( E = < E, \cdot, 1_E > \) be a space of wholes and let \( A_e = < A_e, \cdot, 1_e, \leq > \) for all \( e \in E \) be adjacency lattices generated from a set of atomic objects \( E^{At} \) and a structure adjacency relation \( R \) on \( E^{At} \). Now define \( < O, \cdot_O > \) to be the associative and commutative monoid freely generated from \( E^{At} \cup \bigcup_{e \in E} A_e \) where

\[
a \cdot_O b = \begin{cases} 
    a \cdot_E b & \text{if } a, b \in E \\
    a \cdot_{A_e} b & \text{if } a, b \in A_e \text{ for any } e \in E \\
    a \cdot_H b & \text{otherwise}
\end{cases}
\]

where \( \cdot_E \) is the product of the space of wholes and \( \cdot_{A_e} \) is the product of the adjacency lattice of \( e \) and \( \cdot_H \) is the product for the hybrid terms.

We will say that \( O_{E,R} \) is the discourse space generated by \( E^{At} \) and \( R \). The rules for \( \cdot_E \) determine the product between whole objects, and the rules for \( \cdot \) in \( A_e \) determine the behavior of the product between infons of one world (built of links) and the rules for \( \cdot_H \) for the interaction between the different worlds. Again we are talking about behavior in terms of commutativity and idem-consumption and cloning.

4.4.11. Definition. (discourse frames)

Given a collection of atomic objects and an adjacency relation over these, we can determine uniquely a discourse space. In a Kripke style model this combination is called a frame. A collection of objects together with an adjacency relation constitutes what we will call a discourse frame. We will use the spaces generated from this frame to define models.

4.4.12. Definition. (Valuations)

Let \( F = (E^{At}, R) \) be a discourse frame and \( O_{E,R} = < O, \cdot > \) be the discourse space
determined by \( \mathcal{F} \). A pre-valuation \( \nu : \text{Q}_{\text{UE}} \mapsto P(\mathcal{E}) \) is a function that assigns to each propositional variable of \( \text{L}_{\text{UE}} \) a set of (whole) objects. We extend \( \nu \) to take arbitrary formulas of \( \text{L}_{\text{UE}} \) and extend its range to \( \mathcal{O}_{\mathcal{E},\mathcal{R}} \) (links and objects) by:

\[
\begin{align*}
\nu(P \cup Q) &= \text{Ext}(\nu(P) \cup \nu(Q)), \\
\nu(P \cap Q) &= \text{Ext}(\nu(P) \cap \nu(Q)), \\
\nu(P \ast Q) &= \text{Ext}(\nu(P) \cdot \nu(Q)), \\
\nu(\neg P) &= \text{Ext}[E - \text{Ext}(\nu(P))], \\
\nu(\Diamond P) &= \text{Ext}\{(q,p)|q,p \in \mathcal{R}, p \in \nu(P)\}, \\
\nu(\top) &= \emptyset, \\
\nu(\bot) &= \{1_{\mathcal{E}}\}.
\end{align*}
\]

A valuation \( V : \text{Q}_{\text{UE}} \mapsto P(\mathcal{E}) \) is defined as:

\[V(P) = \nu(P) \cap E\]

4.4.13. Definition. (Models for categorial graphs)

A pair \((\mathcal{F}, V)\) consisting of a discourse frame \( \mathcal{F} \) and a valuation \( V \) is called a discourse model. The notions of truth and validity given a (collection of) model(s) are defined as usual:

Let \( \mathcal{M} = (\mathcal{F}, V) \) be a discourse model, where \( \mathcal{F} = (\mathcal{E}, \mathcal{R}) \) is a discourse frame, \( \mathcal{E} = \langle E, \cdot, 1 \rangle \) is a space of wholes and \( a \in E \) is an object. A formula \( \phi \in \Phi(\text{M}_{\text{UE}}) \) is true at object \( a \) in model \( \mathcal{M} \), notation \( \mathcal{M}, a \models \phi \) if \( a \in V(\phi) \).

To end this section we will write down a very simple example of a model for the categorial graph language. This example illustrates the whole process of obtaining semantics for a categorial graph.

4.4.14. Example. Look at the very simple categorial graph \( G \) of figure 4.1, and its instance (object graph) \( O \). For \( G \) we get:

\[A \Rightarrow \Diamond B\]

For \( O \) we have the following model:

- space of wholes: \( \langle \{a, b, 1\}, \cdot_E, 1 \rangle \)
- links: \( \{(a, b)\} \)
- adjacency spaces:
  - \(< \{(a, b), (a, 1)\}, \cdot_a, (a, 1) \rangle \)
  - \(< \{(b, 1)\}, \cdot_b, (b, 1) \rangle \)
- adjacency structure mapping:
  - \( f_R(a) = (a, b)(= (a, b) \cdot (a, 1) \cdot (a, 1)) \)
  - \( f_R(b) = (b, 1) \)
- adjacency structure relation: \( R = \{(a, (a, b)), (a, (a, 1)), (b, (b, 1))\} \)
- discourse space: \( \langle \{a, b, 1, (a, b), (a, 1), (b, 1)\}, \cdot_O \rangle \)
- pre-valuation:
  - \( \nu(A) = \{a\} \)
  - \( \nu(B) = \{b\} \)
4.5 Summary

In this chapter we presented the semantics for the language of categorial graphs. The language together with its semantics provides a 'direct' formalization of expressions and concepts from the practical world of object oriented information systems. This system will be subject to further theoretical (logical) analysis in the subsequent chapters.

Figure 4.1: A very simple example of a categorial graph and a model