

6 FUNCTOR-BASED CONSTRUCTIONS

Functors provide us not only with the ability to investigate relationships between categories, as shown in the previous chapter and continued in the first section of this chapter, but also to build new categories based on such relationships. In this chapter, we present some of the functor-based constructions that we have found useful in our day-to-day.

6.1 Functor-distinguished kinds of categories

In this section, we show how the properties of certain categories can be derived from the properties of functors that relate them to other categories. That is to say, we investigate functors as a means of revealing structural properties of categories and of their objects.

We start with the notion of concrete category, which appears already in [79, page 26] and is extensively explored in [1], as the title suggests. It is also a notion that we have found to be extremely useful in Computing, namely because we typically build new categories over old ones by adding some structure but without interfering with the structure of the original category. This construction by "conservative extension" can be captured by a faithful functor $u: \mathbf{D} \rightarrow \mathbf{C}$ where \mathbf{C} is the old category and \mathbf{D} is the new one. Typically, the functor "forgets" the structure that is being added to \mathbf{C} to produce \mathbf{D} .

6.1.1 DEFINITION – concrete categories

A *concrete category* over a category \mathbf{C} is a pair $\langle \mathbf{D}, \sqsubset \rangle$ where $\sqsubset: \mathbf{D} \rightarrow \mathbf{C}$ is a faithful functor.

Notice that we are using the terminology introduced in [1]. The notion of concrete category introduced in [79] is a particularisation of the one above to the case where the category \mathbf{C} is **SET**. Concrete categories over **SET** are called *constructs* in [1]. The category \mathbf{C} is sometimes called the *base category* of $\langle \mathbf{D}, \sqsubset \rangle$ and \sqsubset is called the *forgetful* or *underlying functor*.

Because the underlying functor is faithful, i.e. injective on hom-sets, for each pair of \mathbf{D} -objects $\langle x, y \rangle$, $\text{hom}_{\mathbf{D}}(x, y)$ is usually regarded as a subset of $\text{hom}_{\mathbf{C}}(\sqsubset(x), \sqsubset(y))$. Following [1], we shall often use the expression " $f: \sqsubset(x) \rightarrow \sqsubset(y)$ is a *\mathbf{D} -morphism*" to mean that there exists a (necessarily) unique \mathbf{D} -morphism $x \rightarrow y$ whose image by \sqsubset is f .

We have already come across a few concrete categories:

6.1.2 EXAMPLES

1. The category **CLOS** of closure systems defined in section 3.6 is concrete over **SET**: the forgetful functor maps closure systems to the underlying sets (languages).
2. The categories **PRES_{LTL}**, **SPRES_{LTL}** and **THEO_{LTL}** defined in 3.5.4 are all concrete over the category **SET** of sets. The underlying functors, which we will name **sign_{LTL}**, "forget" the sets of axioms/theorems, mapping theories and their presentations to the corresponding signatures.
3. Another example is the category **AUTOM** of automata as defined in 2.1.10. This category is concrete over the product **SET[SET][SET]**. The underlying functor forgets the input, output and transition functions and projects every automaton to its sets of inputs, states and outputs.

A typical situation in which concrete categories arise is one in which the underlying functor represents some sort of classification or typing mechanism that is strong enough to extend to the morphisms and, hence, to the structure defined over the objects by the morphisms. In such circumstances, one is usually interested in studying the way all the objects that share the same classification or type relate to one another.

6.1.3 DEFINITION – fibres

Given a concrete category $\langle \mathbf{D}, \sqsupset \rangle$ over \mathbf{C} and a \mathbf{C} -object c , the fibre of c is the pre-order that consists of all the objects d of \mathbf{D} that are mapped to c , i.e. such that $\sqsupset(d)=c$, ordered by $d_1 \leq_c d_2$ iff $id_c: \sqsupset(d_1) \rightarrow \sqsupset(d_2)$ is a \mathbf{D} -morphism.

The structure of the fibres reveals a lot about the properties of the concrete category. A detailed discussion can be found in [1]. We shall limit our study to three cases that will be used further on in the book.

6.1.4 DEFINITION

A concrete category $\langle \mathbf{D}, \sqsupset \rangle$ over \mathbf{C} is called:

1. *amnesic* provided that its fibres are partially ordered, i.e. $d_1 \leq_c d_2$ and $d_2 \leq_c d_1$ implies $d_1 = d_2$ for all \mathbf{C} -objects c and objects d_1, d_2 in the fibre of c .
2. *fibre-complete* if its fibres are complete lattices (i.e. admit arbitrary meets and joins).
3. *fibre-discrete* if its fibres are ordered by equality.

Amnesic concrete categories are such that the morphisms of \mathbf{D} do not introduce additional properties over the structures that are being superposed on \mathbf{C} . For instance, **SPRES_{LTL}** and **THEO_{LTL}** are amnesic because their morphisms do not introduce any structural properties over the sets of axioms/theorems. However, **PRES_{LTL}** is not am-

nestic because its morphisms capture the notion of closure. Two presentations over the same temporal signature can be isomorphic without being equal. On the other hand, strictly isomorphic presentations and isomorphic theories over the same temporal signature are necessarily equal: the former because the consequences of the axioms are not taken into consideration, and the latter because they are already closed under consequence. In other words, morphisms of theories just take into account the elements of a set (the theorems) whereas morphisms of presentations have to compute the closure of a set.

Concrete categories that are fibre-complete superpose over the objects of \mathbf{C} information that the morphisms organise in a "convenient" way, allowing for operations to be performed internally within each fibre. We shall see some examples further on. Hence, being fibre-complete is a step further than amnesticity in the degree of interaction that exists between the existing and the superposed structure. Fibre-discrete categories present still a step further: they are such that the extension that \mathbf{D} makes over the objects of \mathbf{C} is inessential, i.e. it has no intrinsic structure or meaning – it acts just like a comment.

Because concrete categories have "added structure", we should provide a notion of functor that reflects that structure:

6.1.5 DEFINITION – concrete functors

A *concrete functor* \square between two concrete categories $\langle \mathbf{D}_1, \square_1 \rangle$ and $\langle \mathbf{D}_2, \square_2 \rangle$ over the same underlying category \mathbf{C} is a functor $\square: \mathbf{D}_1 \rightarrow \mathbf{D}_2$ such that $\square_1 = \square_2 \circ \square$.

Because the morphisms of a concrete category are "imported" from its underlying category, concrete functors cannot act on them and, hence, are fully determined by their values on objects:

6.1.6 PROPOSITION

Given concrete functors \square and \square' between two concrete categories $\langle \mathbf{D}_1, \square_1 \rangle$ and $\langle \mathbf{D}_2, \square_2 \rangle$, $\square = \square'$ if, for every \mathbf{D}_1 -object d , $\square(d) = \square'(d)$.

PROOF

Let $f: d \rightarrow d'$ be a morphism of \mathbf{D}_1 . Given that both functors agree on objects, both $\square(f)$ and $\square'(f)$ have the same source and target. Because both \square and \square' are concrete, we have $\square_1(f) = \square_2(\square(f)) = \square_2(\square'(f))$. Finally, because \square_2 is faithful, we can conclude that $\square(f) = \square'(f)$. Hence, both functors agree on morphisms as well.

6.1.7 REMARK – concrete subcategories

Given that subcategories determine embeddings (see 5.1.10), and that the composition of faithful functors is also faithful (see 5.1.9), every subcategory \mathbf{D}' of a category \mathbf{D} that is concrete over \mathbf{C} with underlying functor \square can be regarded also as a concrete category over \mathbf{C} whose underlying functor is the composition $\square_{\mathbf{D}', \mathbf{D}} \circ \square$. We then say that $\langle \mathbf{D}', \square_{\mathbf{D}', \mathbf{D}} \circ \square \rangle$ is a concrete subcategory of $\langle \mathbf{D}, \square \rangle$.

Consider now the notions of reflective and co-reflective subcategories that we discussed in section 3.3. Intuitively, for the (co)reflection to be "concrete", i.e. to be consistent with the classification that the underlying functor provides, we would like to remain within the same fibre, i.e. we would like that the (co)reflection arrows be identities:

6.1.8 DEFINITION – concretely (co)reflective subcategories

A concrete subcategory $\langle D_1, \sqsubset_1 \rangle$ of $\langle D_2, \sqsubset_2 \rangle$ is *concretely (co)reflective* iff, for each object of D_2 , there is a (co)reflection arrow that is mapped by \sqsubset_2 to the identity.

For instance, although **REACH** is a co-reflective subcategory of **AUTOM**, it is not concretely co-reflective when both categories are considered as being concrete over **SET[SET][SET]**. This is because the reachable automata are not necessarily in the same fibre as the automata from which they are computed – their state space may have been reduced.

"Concreteness" also extends to universal constructions in the sense that, once we "look" at a category D as being concrete through $\sqsubset:D \rightarrow C$, we can classify the way universal constructions in D relate to C through \sqsubset :

6.1.9 DEFINITION – concrete universal constructions

Consider a concrete category $\langle D, \sqsubset \rangle$ over C and a diagram $\sqsubset:I \rightarrow D$. A (co)limit of \sqsubset is said to be a *concrete (co)limit* of \sqsubset in $\langle D, \sqsubset \rangle$ iff it is preserved by \sqsubset .

That is to say, universal constructions are concrete when they map to the underlying category. In some categories, all universal constructions are concrete. We shall see an example in 6.3.6. In other concrete categories, some universal constructions may be concrete and others not so.

6.1.10 EXERCISE

Workout examples of limits and colimits in **AUTOM** that are not concrete over **SET[SET][SET]**.

The construction of concrete (co)limits, when they exist, can be systematised through a process that consists in projecting the (co)cones to the base category where a (co)limit is computed and then lifted back. This process can be extended to more general notions of fibre as discussed below.

The notion of fibre is not exclusive to concrete categories. It can be generalised to arbitrary functors as follows:

6.1.11 DEFINITION – fibres

Consider a functor $\sqsubset:D \rightarrow C$.

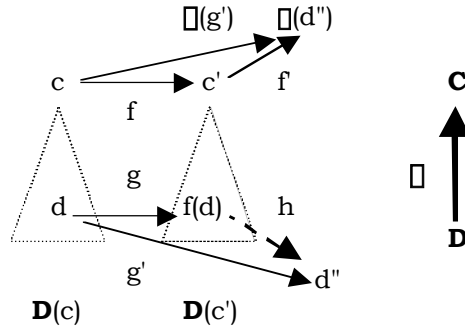
1. Given a C -object c , the *fibre of c* , which we denote by $D(c)$, is the subcategory of D that consists of all the objects d of D that are mapped to c , i.e. such that $\sqsubset(d)=c$, together with the D -morphisms $f:d_1 \rightarrow d_2$ such that $\sqsubset(f)=id_c$.

2. The functor \square is said to be *amnesic* if in its fibres no two distinct objects are isomorphic. That is to say, if an isomorphism $f:d_1 \square d_2$ in \mathbf{D} is such that $\square(f)=id_c$ for some object c of \mathbf{C} , then f is itself an identity.

The discussion around concrete categories focused basically on the structure of fibres. Another interesting aspect worth discussing is the relationships that morphisms on the underlying category induce over the fibres corresponding to their sources and targets.

For instance, consider again the category \mathbf{THEO}_{LTL} of temporal theories defined in 3.5.4 and the forgetful functor \mathbf{sign}_{LTL} that maps theories and their morphisms to their signature components. Consider an arbitrary signature \square . A signature morphism $f:\square \square \square'$ provides a translation from the symbols of one signature to symbols of the other. We have seen in 3.5.7 that such a translation extends to the temporal language associated with \square . Can we further extend this translation to the fibres of \square , i.e. can we define a mechanism that translates theories with signature \square to theories over \square' ? What about the reverse? Can we define a mechanism that translates theories with signature \square' to theories over \square ?

To answer these questions, we have first to provide a reasonable definition of "translation" (and inverse translation) between fibres induced by a morphism. Consider a functor $\square:\mathbf{D} \square \mathbf{C}$, a \mathbf{C} -morphism $f:c \square c'$, and a \mathbf{D} -object d in the fibre of c , i.e. $\square(d)=c$. By the "image" of d under f we mean some object $f(d)$ in the fibre of c' that is "closest" to d . By "closest" we mean the following: f can be lifted to a morphism between d and $f(d)$, i.e. there must exist some morphism $g:d \square f(d)$ such that $\square(g)=f$; for any other object d'' and morphism $g':d \square d''$ that leaves d at a "distance" $f':c \square \square(d'')$ of f , i.e. such that $\square(g')=f'$, this distance can be covered in \mathbf{D} in a unique way by a morphism $h:f(d) \square d''$ such that $\square(h)=f'$ and $g'=h \circ g$.



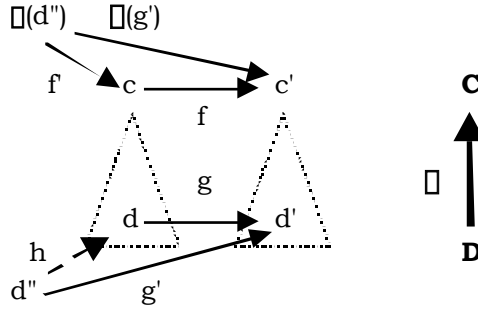
By the inverse translation, we mean the dual notion (what else...).

The following definitions are adapted from [12], a good source for the study of fibre-related matters.

6.1.12 DEFINITION – (co)cartesian morphisms

Let $\square:\mathbf{D} \square \mathbf{C}$ be a functor and $f:c \square c'$ a \mathbf{C} -morphism.

1. Let $d':\mathbf{D}(c')$, i.e. a \mathbf{D} -object such that $\square(d')=c'$. A \mathbf{D} -morphism $g:d \square d'$ is said to be *cartesian* for f and d' iff (1) $\square(g)=f$ and (2) for every $g':d' \square d''$ and $f':\square(d'') \square c$ such that $\square(g')=f'$, there is a unique morphism $h:d' \square d''$ such that $\square(h)=f'$ and $g'=h \circ g$.



2. Let $d: \mathbf{D}(c)$. A \mathbf{D} -morphism $g: d \rightarrow d'$ is said to be *cocartesian* for f and d iff (1) $F(g)=f$ and (2) for every $g': d \rightarrow d''$ and $f': c \rightarrow F(d'')$ such that $F(g')=f;f$, there is a unique morphism $h: d' \rightarrow d''$ such that $F(h)=f'$ and $g'=g;h$.

6.1.13 DEFINITION – (co)fibration

Let $F: \mathbf{D} \rightarrow \mathbf{C}$ be a functor.

1. We say that F is a *fibration* provided that, for every \mathbf{C} -morphism $f: c \rightarrow c'$ and \mathbf{D} -object d' in the fibre of c' , there is a cartesian morphism for f and d' .
2. We say that F is a *cofibration* provided that, for every \mathbf{C} -morphism $f: c \rightarrow c'$ and \mathbf{D} -object d in the fibre of c , there is a cocartesian morphism for f and d .

6.1.14 EXAMPLE – specifications as (co)fibrations

The forgetful functors that define \mathbf{PRES}_{LTL} , \mathbf{SPRES}_{LTL} and \mathbf{THEO}_{LTL} as concrete categories over the category \mathbf{SET} are all fibrations and cofibrations at the same time, but with different (co)cartesian morphisms among themselves. Given a signature morphism $f: \Sigma \rightarrow \Sigma'$:

1. a cartesian morphism for a theory $\langle \Sigma', \Sigma' \rangle$ is $f: \langle \Sigma, f^1(\Sigma) \rangle \rightarrow \langle \Sigma', \Sigma' \rangle$ and a cocartesian morphism for a theory $\langle \Sigma, \Sigma \rangle$ is $f: \langle \Sigma, \Sigma \rangle \rightarrow \langle \Sigma', c(f(\Sigma)) \rangle$.
2. a cartesian morphism for a presentation $\langle \Sigma', \Sigma' \rangle$ is $f: \langle \Sigma, f^1(c(\Sigma)) \rangle \rightarrow \langle \Sigma', \Sigma' \rangle$ and a cocartesian morphism for a presentation $\langle \Sigma, \Sigma \rangle$ is $f: \langle \Sigma, \Sigma \rangle \rightarrow \langle \Sigma', f(\Sigma) \rangle$.
3. a cartesian morphism for a strict presentation $\langle \Sigma', \Sigma' \rangle$ is $f: \langle \Sigma, f^1(\Sigma) \rangle \rightarrow \langle \Sigma', \Sigma' \rangle$ and a cocartesian morphism for a strict presentation $\langle \Sigma, \Sigma \rangle$ is $f: \langle \Sigma, \Sigma \rangle \rightarrow \langle \Sigma', f(\Sigma) \rangle$.

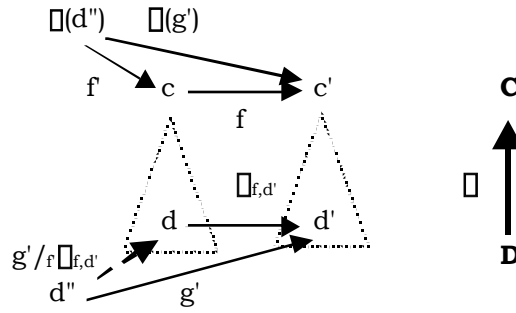
The differences that we can witness between these three cases have to do with both the properties of the closure operator and the morphisms that characterise the categories. For instance, even if Σ is closed for consequence, $f(\Sigma)$ is not necessarily so. Hence, the cocartesian morphisms for theories have to return the closure of the image set. This is not necessary for presentations because it is already implicit in the morphisms, nor for strict presentations because they do not involve the consequence operator at all. The cartesian morphisms, on the contrary, do not need to compute the closure of the inverse image set. This is because, if Σ is closed, so is $f^1(\Sigma)$. However, presentations do need to compute the closure explicitly because they only do it implicitly to the target, not the source.

These examples illustrate the fact that there may be more than one cocartesian morphism for a given signature morphism and theory presentation. This is because the set of axioms of the specification can be determined only up to logical equivalence. In the case of theories, the fact that the set of sentences must be closed ensures that the lifts are unique. Hence, although it is possible to generalise the translation induced by signature morphisms to presentations, there may be more than one way of doing so. This is the general case of any (co)fibration. Amnestic concrete (co)fibrations, however, guarantee uniqueness of the lifting and, hence, of the choice for (co)cartesian morphisms.

6.1.15 DEFINITION – cleavage, cloven fibration

Let $\square: \mathbf{D} \rightarrow \mathbf{C}$ be a functor. A choice of a cartesian morphism for every \mathbf{C} -morphism $f: c \rightarrow c'$ and \mathbf{D} -object d' is called a *cleavage*. A fibration equipped with a cleavage is called *cloven*.

We shall often denote by $\square_{f,d'}$ the cartesian morphism selected for f and d' by the cleavage. Given $g': d'' \rightarrow d'$ and $f': \square(d'') \rightarrow c$ such that $\square(g') = f'; f$, we know that there is a unique morphism $h: d'' \rightarrow d$ such that $\square(h) = f'$ and $g' = h; \square_{f,d'}$. We shall denote h by $g' /_f \square_{f,d'}$. When f is id_c , we omit the subscript.



The dual notion is called cocleavage, and a cofibration equipped with a cocleavage is also said to be cloven.

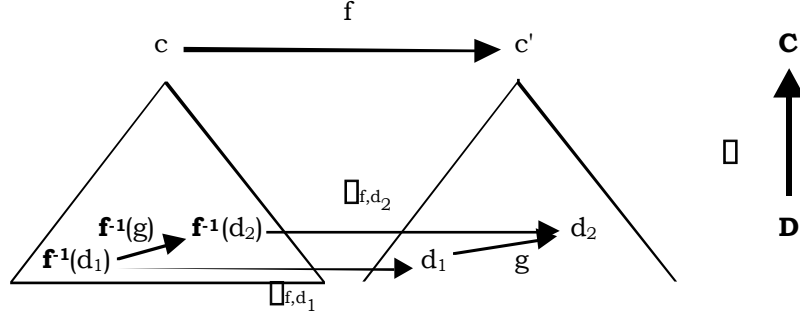
Summarising, a cloven (co)fibration $\square: \mathbf{D} \rightarrow \mathbf{C}$ provides us with a way of lifting morphisms to translations between the objects of their fibres in the sense that every \mathbf{C} -morphism $f: c \rightarrow c'$ defines a map from the objects of $\mathbf{D}(c')$ to the objects of $\mathbf{D}(c)$ in the case of a fibration, and from the objects of $\mathbf{D}(c)$ to the objects of $\mathbf{D}(c')$ in the case of a cofibration. Can these translations be generalised to the morphisms of the fibres? That is, can we generalise this mapping to a functor between the fibres?

6.1.16 PROPOSITION

Let $\square: \mathbf{D} \rightarrow \mathbf{C}$ be a functor and $f: c \rightarrow c'$ a \mathbf{C} -morphism.

1. If \square is a cloven fibration, then f defines a functor $f^1: \mathbf{D}(c') \rightarrow \mathbf{D}(c)$ as follows
 - given $d': \mathbf{D}(c')$, $f^1(d')$ is the source of the cartesian morphism $\square_{f,d'}: d' \rightarrow d$ that the cleavage associates with the fibration;
 - given $g: d_1 \rightarrow d_2$ in $\mathbf{D}(c')$, $f^1(g)$ is the morphism $f^1(d_1) \rightarrow f^1(d_2)$ that results from the universal property of the cartesian morphism $\square_{f,d_2}: f^1(d_2) \rightarrow d_2$ when ap-

plied to $\square_{f,d};g$ and id_c . That is to say, $f^1(g)$ is the morphism that we also denote by $(\square_{f,d_1};g)/\square_{f,d_2}$. Notice that this morphism is the only one that satisfies $\square_{f,d_1};g=f^1(g);\square_{f,d_2}$ and $\square(f^1(g))=id_c$.



2. If \square is a cloven cofibration, then f defines a functor $f:D(c)\rightarrow D(c')$ in the dual way, i.e. by working on the target side of the cocartesian morphism.

PROOF

1. There are three properties to prove. The reader is invited to fill-in the details:
 - the functor is well defined in the sense that the image objects and morphisms exist and are of the right types;
 - because $id_{f^1(d)}$ satisfies the properties of the universal property that characterises $f^1(id_d)$, the uniqueness associated with that universal property guarantees that the cleavage has no other choice;
 - the same line of reasoning applies to conclude that $f^1(g_1;g_2)=f^1(g_1);f^1(g_2)$
2. By duality.

This definition raises an immediate question: what if $f=id_c$? Are f^1 and f the identity functor? What if $f=f_1;f_2$? Are f^1 and f the compositions $f^1_1;f^1_2$ and $f_1;f_2$, respectively? The answer is twofold: they can, but they do not have to. Consider the first part of the answer, this time instantiated for cofibrations.

6.1.17 PROPOSITION

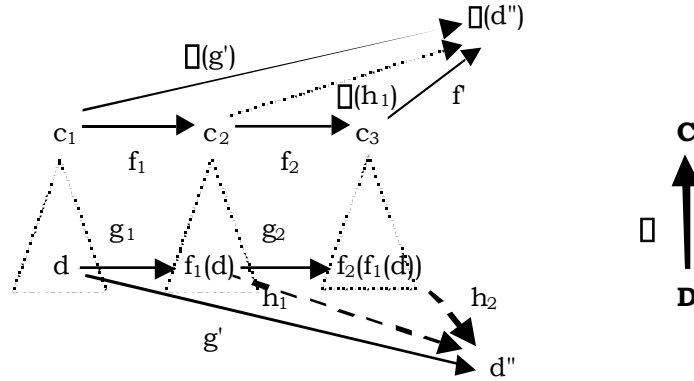
Let $\square:D\rightarrow C$ be a functor.

1. Given a C -object c and an object d in the fibre of c , the identity id_d is both a cartesian and a cocartesian morphism for id_c and d .
2. Given C -morphisms $f_1:c_1\rightarrow c_2$ and $f_2:c_2\rightarrow c_3$, an object d in the fibre of c_1 , and "translations" (cocartesian morphisms) $g_1:d\rightarrow f_1(d)$ and $g_2:f_1(d)\rightarrow f_2(f_1(d))$, the composition $g_1;g_2$ provides a cocartesian morphism for $f_1;f_2$ and d .

PROOF

1. Exercise.
2. There are two properties to prove:

- The composite translation lifts the composition in \mathbf{C} :
 $\square(g_1;g_2)=\square(g_1);\square(g_2)=f_1;f_2$
- The lift has the required couniversal property:
 Let $g':d \rightarrow d''$ and $f:c_3 \rightarrow \square(d'')$ be such that $\square(g')=f_1;f_2;f$. The couniversal property of g_1 applied to g' and $(f_2;f)$ gives a unique $h_1:f_1(d) \rightarrow d''$ such that $g_1;h_1=g'$ and $\square(h_1)=f_2;f$. We can now apply the couniversal property of g_2 to h_1 and f to infer the existence and uniqueness of $h_2:f_2(f_1(d)) \rightarrow d''$ such that $g_2;h_2=h_1$ and $\square(h_2)=f$. We are now going to prove that h_2 has the required properties. On the one hand, $\square(h_2)=f$. On the other hand, if we take $h'_2:f_2(f_1(d)) \rightarrow d''$ such that $g_1;g_2;h'_2=g'$ and $\square(h'_2)=f$, we derive $g_2;h'_2=h_1$ because we have $g_1;(g_2;h'_2)=g'$ and $\square(g_2;h'_2)=\square(g_2);\square(h'_2)=f_2;f$, and h_1 is the unique morphism satisfying these two properties; but, then, we conclude that $h'_2=h_2$ because h_1 is the unique morphism satisfying $g_2;h_2=h_1$ and $\square(h_2)=f$.



The fact that cleavages do not need to choose (co)cartesian morphisms satisfying these properties leads to the following definition.

6.1.18 DEFINITION – split fibration

Let $\square: \mathbf{D} \rightarrow \mathbf{C}$ be a cloven fibration. If, for every \mathbf{C} -object c , id_c^{-1} is $\text{id}_{c(c)}$ and, for every decomposition $f=f_1;f_2$, f^1 is the composition $f^1_2;f^1_1$, then the fibration is said to be *split*.

6.1.19 REMARK

The more alert reader may have noticed that the universal property of the (co)cartesian morphism is taken over a space that is larger than the fibre of either c or c' . This is why the "distance" f is introduced as a kind of "type converter". This means that the translation or inverse translation is chosen not only over the objects that have the exact type but also those that "type check". The reason we point this out is to recall that the "categorical way" of defining a concept, namely of choosing a universal property, has to be "morphism-oriented" in order to yield "good" properties such as the one that we have just proved. The reader is encouraged to experiment with the weaker notion of (co)cartesian morphism for which the universal property is required only over the fibres, and check which of the properties that we have proved still hold.

Split (co)fibrations allow us to wrap up the idea of translation between fibres with which we started in neat categorical clothing:

6.1.20 PROPOSITION

Let $\mathcal{F}:\mathcal{D}\rightarrow\mathcal{C}$ be a functor.

1. If \mathcal{F} is a split fibration, then it defines a functor $\mathbf{ind}(\mathcal{F}):\mathcal{C}^{op}\rightarrow\mathbf{CAT}$ by mapping every \mathcal{C} -object c to its fibre $\mathcal{D}(c)$ and every morphism $f:c\rightarrow c'$ to the functor $\mathbf{f}^1:\mathcal{D}(c')\rightarrow\mathcal{D}(c)$ as defined above.
2. If \mathcal{F} is a split cofibration, then it defines a functor $\mathbf{ind}(\mathcal{F}):\mathcal{C}\rightarrow\mathbf{CAT}$ by mapping every \mathcal{C} -object c to its fibre $\mathcal{D}(c)$ and every morphism $f:c\rightarrow c'$ to the functor $\mathbf{f}:\mathcal{D}(c)\rightarrow\mathcal{D}(c')$ as defined above.

PROOF

Left as an exercise.

Functors of the form $\mathcal{C}^{op}\rightarrow\mathbf{CAT}$ are called *indexed-categories*, a structure widely applied in Computing (see, for instance, [103]) precisely because it generalises what are usually called indexed-sets, i.e. mechanisms for indexing given collections of objects with other kind of objects (indexes). We shall focus on indexed categories in section 6.4.

One of the reasons to study fibrations is the close relationship that exists between their universal properties and those of the fibres.

6.1.21 DEFINITION – fibre completeness

A cloven fibration $\mathcal{F}:\mathcal{D}\rightarrow\mathcal{C}$ is said to be *fibre-complete* if its fibres are complete categories and the inverse translation functors induced on the fibres preserve limits.

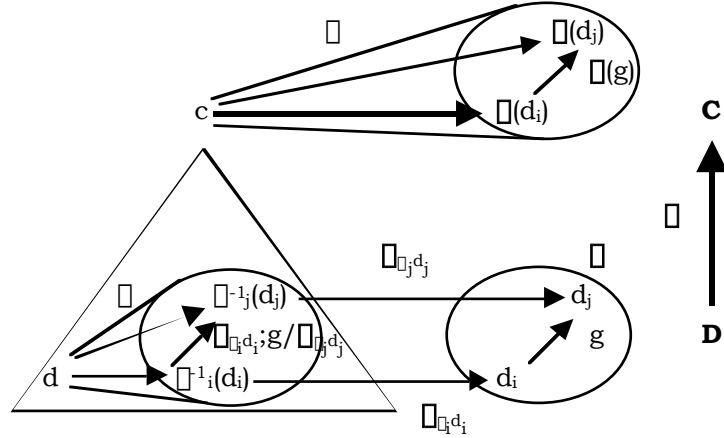
6.1.22 PROPOSITION

Let $\mathcal{F}:\mathcal{D}\rightarrow\mathcal{C}$ be a split fibration.

1. If \mathcal{F} is fibre-complete, then it lifts limits.
2. If, in addition, \mathcal{F} is amnestic, the lift is unique.

PROOF

1. Consider a diagram $\mathcal{I}\rightarrow\mathcal{D}$ and a limit $\mathcal{L}\rightarrow\mathcal{D}$ for the underlying \mathcal{C} -diagram. We are going to provide a step-by-step construction of a limit for \mathcal{F} that lifts \mathcal{L} .



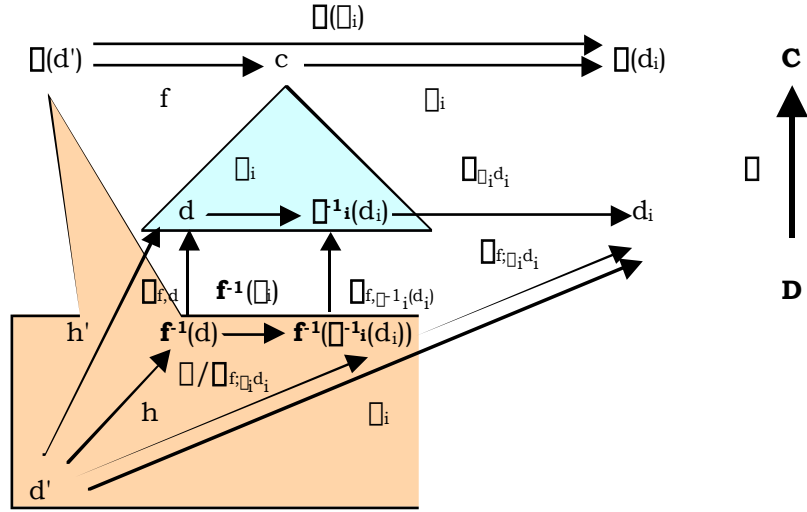
- first, we take the inverse image of ϖ induced by ϖ . We obtain a diagram within the fibre of c , related to the original one by the cartesian morphisms $\varpi_{\varpi_i d_i}$. Notice that morphisms $g: d_i \rightarrow d_j$ are translated to $\varpi_{\varpi_i d_i} g / \varpi_{\varpi_j d_j} : \varpi^{-1}_i(d_i) \rightarrow \varpi^{-1}_j(d_j)$
- then, given that the fibre of c is complete, we compute the limit $\varpi: d \rightarrow \varpi^{-1}(\varpi)$. We are going to prove that the cone obtained through the compositions $\varpi_i: \varpi_{\varpi_i d_i}$ is a limit for ϖ
- the cone is commutative: consider $g: d_i \rightarrow d_j$ in ϖ we have $\varpi_i: \varpi_{\varpi_i d_i} g = \varpi_i: \varpi_{\varpi_i d_i} g / \varpi_{\varpi_j d_j} \varpi_{\varpi_j d_j} = \varpi_j: \varpi_{\varpi_j d_j}$.
- the universal property of the cone is satisfied: consider another commutative cone $\varpi': d' \rightarrow \varpi$ because the image of ϖ under ϖ is a commutative cone, the universal property of $\varpi: c \rightarrow \varpi$ implies the existence of a unique $f: \varpi(d') \rightarrow c$ such that $f \varpi_i = \varpi'(\varpi_i)$ for every $i \in I$; unfortunately, d' is not necessarily in the fibre of c and, hence, we cannot use the universal properties of the limit.

Because the inverse translation induced by f preserves limits, the cone $\mathbf{f}^1(\varpi): \mathbf{f}^1(d) \rightarrow \mathbf{f}^1(\varpi^{-1}(\varpi))$ is itself a limit in the fibre of $\varpi(d')$; given that $\varpi_i / \varpi_{\varpi_i d_i} : d' \rightarrow (f; \varpi)^{-1}(\varpi)$ is also a commutative cone and $(f; \varpi)^{-1}(\varpi) = \mathbf{f}^1(\varpi^{-1}(\varpi))$ is a consequence of ϖ being split, we can infer the existence of a unique $h: d' \rightarrow \mathbf{f}^1(d)$ such that $h \mathbf{f}^1(\varpi_i) = \varpi_i / \varpi_{\varpi_i d_i}$.

We now prove that $h: \varpi_{f,d}$ satisfies the required properties.

$$(h; \varpi_{f,d}) (\varpi_i: \varpi_{\varpi_i d_i}) = h \mathbf{f}^1(\varpi_i); \varpi_{f, \varpi^{-1}_i(d_i)} \varpi_{\varpi_i d_i} = \varpi_i / \varpi_{\varpi_i d_i}; \varpi_{f, \varpi^{-1}_i(d_i)} \varpi_{\varpi_i d_i} = \varpi_i / \varpi_{\varpi_i d_i}; \varpi_{\varpi_i d_i} = \varpi_i.$$

Let $h': d' \rightarrow d$ satisfy $h'(\varpi_i: \varpi_{\varpi_i d_i}) = \varpi_i$. We then have $\varpi_i = h' / \varpi_{f,d}; \varpi_{f,d}; \varpi_i: \varpi_{\varpi_i d_i} = h' / \varpi_{f,d}; \mathbf{f}^1(\varpi_i); \varpi_{f, \varpi^{-1}_i(d_i)} \varpi_{\varpi_i d_i} = h' / \varpi_{f,d}; \mathbf{f}^1(\varpi_i); \varpi_{\varpi_i d_i}$ which implies $h' / \varpi_{f,d}; \mathbf{f}^1(\varpi_i) = \varpi_i / \varpi_{\varpi_i d_i}$ and, hence, $h' / \varpi_{f,d} = h$, and $h; \varpi_{f,d} = h'$



2. Left as an exercise.

6.1.23 COROLLARY

Let $\square: \mathbf{D} \rightarrow \mathbf{C}$ be a split fibration. If \square is fibre-complete and \mathbf{C} is complete, then \mathbf{D} is also complete.

These results and their duals tell us how to compute universal constructions over (co)fibrations that are fibre-(co)complete: the diagram is projected to the underlying category and its (co)limit is computed; the original diagram is (co)translated to the fibre of the apex and its (co)limit is computed within the fibre.

6.1.24 EXAMPLE – colimits of specification diagrams

The fibres of \mathbf{PRES}_{LTL} , \mathbf{SPRES}_{LTL} and \mathbf{THEO}_{LTL} as concrete categories over \mathbf{SET} are all complete and co-complete as ordered sets. It is also easy to see that the (co)translations are (co)continuous. Taking into account the operations that define these universal constructions, we have the following procedure for calculating limits and colimits of a diagram \square with $\square = \langle \square_i, \square_i \rangle$ in these categories.

1. Calculate the limit $\square: \square \square \square$ or colimit $\square: \square \square \square$ of the underlying diagram of signatures.
2. Lift the result by computing the specification-component according to the following rules:

	limit	colimit
\mathbf{PRES}_{LTL}	$\square: \langle \square, \square_{i \in I} \square_i^{-1}(c(\square_i)) \rangle \square \square$	$\square: \square \square \langle \square, \square_{i \in I} \square_i(\square_i) \rangle$
\mathbf{SPRES}_{LTL}	$\square: \langle \square, \square_{i \in I} \square_i^{-1}(\square_i) \rangle \square \square$	$\square: \square \square \langle \square, \square_{i \in I} \square_i(\square_i) \rangle$
\mathbf{THEO}_{LTL}	$\square: \langle \square, \square_{i \in I} \square_i^{-1}(\square_i) \rangle \square \square$	$\square: \square \square \langle \square, c(\square_{i \in I} \square_i(\square_i)) \rangle$

In order to illustrate these constructions, consider the specification of the vending machine that we studied in section 3.5. Therein, we saw how the original specification (3.5.6) could be extended to include a mechanism for regulating the sale of cigars, which was captured by a morphism (3.5.11)

$$\text{vending machine} \xrightarrow{\text{no cigars}} \text{regulated vending machine}$$

The need for such kind of extensions arises whenever there is a change in the original requirements. Such changes may be very frequent in business domains that are very volatile, for instance as a result of fierce competition forcing companies to maintain a level of service that matches or beats the offer of their rivals. This prompts the need for mechanisms that allow systems to evolve in ways that localise the impact of changes. For instance, a better way of accounting for the need to regulate the sale of cigars is to interconnect the original vending machine with an external device (regulator). This would indicate that the required change does not need to be intrusive of the existing system in the sense that it is not necessary to change the way the existing system is implemented. Instead, we just need to implement the regulator and connect it to the system, even while it is running, i.e. without interruption of service.

The required regulator can be specified as follows:

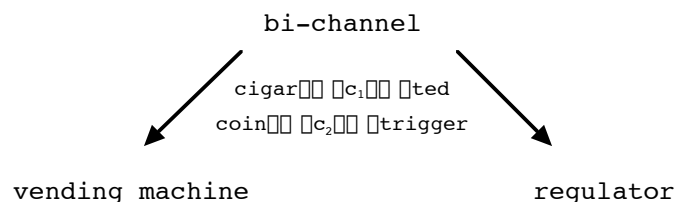
```

specification regulator is
signature      trigger, ted, tor
axioms        beg    ( $\neg$ tor)
                  trigger ( $\neg$ ted)  $\mathbf{W}$ tor
                  tor    ( $\neg$ ted)

```

The rationale is the following. When the trigger occurs, action *ted* (the one being regulated) is blocked until the regulator occurs.

As discussed in chapter 4, we use diagrams to express systems as configurations of interconnected components, the (co)limits of which return the object that represents the system as a component, all the interconnections having been encapsulated. The regulated vending machine can be put together by synchronising action *trigger* of the regulator with action *coin* of the vending machine, and the regulated action with *cigar*. The corresponding configuration diagram is:



where

```

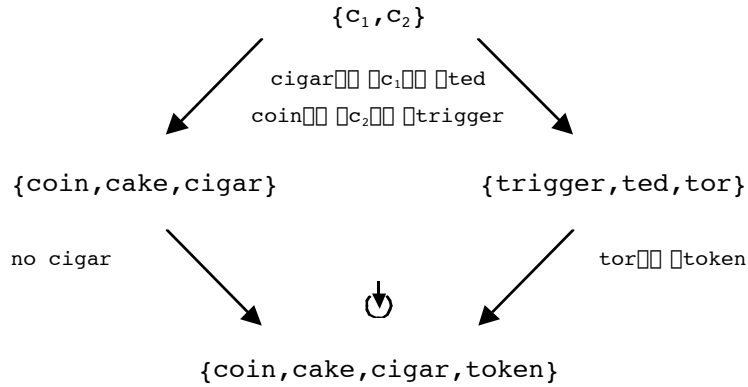
specification bi-channel is
signature       $c_1, c_2$ 
axioms

```

Notice that there is nothing in the regulator that names the vending machine: the regulator is intended to be a component that can be reused in different contexts. The same applies, a fortiori, to the vending machine because it was developed before

the need to be regulated was determined. As a result, there is no implicit interaction between the two components. The interconnection through which the vending machine becomes regulated is completely externalised in the bi-channel and the two morphisms. This is one basic difference between object and service-oriented development as already mentioned: whereas objects, through clientship, code up interactions in the way features are called, services cannot name other services because they are not specified with particular interactions in mind. Moreover, service integration is only performed when it is needed, in run-time, and for the services that will have been identified or selected *at that time* from the current configuration.

According to the rules above, the pushout of this configuration diagram can be calculated in two steps. First, we compute the pushout of the underlying diagram of signatures:



Notice that, because pushouts are determined only up to isomorphism, we chose the signature that matches that of the regulated vending machine. The second step consists in lifting the result back to the category of specifications which, according to the rules, can be achieved by choosing as axioms the translations of the axioms of the component specifications:

```

axioms  beg    (¬cake[]¬cigar) []
            (coin    (¬cake[]¬cigar)Wcoin)
  coin    (¬coin)W(cake cigar)
  (cake cigar)  (¬cake[]¬cigar)Wcoin
  cake    (¬cigar)
  beg    (¬token)
  coin    (¬cigar)Wtoken
  token   (¬cigar)

```

Notice that, because of the renaming imposed by the interconnection, the axioms are now in the shared language and, as a result, they interfere and make new properties (the new requirements) to emerge at the level of the resulting system. As we have already mentioned, this is why the categorical approach brings software systems into the realm of “general, complex systems”, the global behaviour of which is characterised in terms of properties or phenomena that emerge from components and interactions. This is a view that is now shared across different Sciences, from biological to economical and sociological systems.

It is easy to see that this set of axioms is logically equivalent to the one we gave originally for the regulated vending machine, which further shows that the lifting performed through the forgetful functor is not unique. Also notice that we recover the original extension (morphism) – *no cigar* – as one of the co-cone projections. Hence, basically, what we have done is factorise the extension by externalising the regulator that was coded over the original specification.

We should emphasise that, having performed the externalisation, the view of the system that interests us is the one given by the (configuration) diagram. The colimit construction is "only" useful as a semantics for the diagram, namely as a means of checking that the required properties will emerge from the interconnections. Further evolution of the system should be performed on the configuration, not the global specification resulting from the colimit.

Other examples of the application of these techniques in Computing can be found in the area of Concurrency Theory, namely in the work of G.Winskel [111], and F.Costa [21]. The idea is that operations on processes like synchronisation can be defined at the level of the actions that the processes can perform (their alphabet) through some algebraic operations and then lifted to the category of processes using a (co)fibration. An example in this area will be given in section 6.3. Finally, the second part of the book will be dedicated to a systematisation of this process of structuring complex systems through what have been called "software architectures".

6.2 Structured objects and morphisms

One of the best examples of the added expressive power that functors bring into the categorical discourse is the ability to work in frameworks that involve more than one category. An exhaustive study of distinguished objects and arrows with respect to a functor can be found in [1]. In the sequel, we shall present some of the concepts that we have found to provide a good introduction and, thus, motivate the reader to search more about this topic.

6.2.1 EXAMPLE – realisations of a specification

For instance, we argued in 5.1.3 that the notion of satisfaction of specifications by programs can give rise to functors **spec: PROG** **SPEC** that map programs to the strongest specification that they satisfy. The notion of satisfaction of a specification S by a program P can then be captured by the existence of a morphism $\square: S \square \mathbf{spec}(P)$. Indeed, the morphism expresses the fact that the properties specified through S are entailed by the strongest specification of P .

Such a "structured" morphism $\square: S \square \mathbf{spec}(P)$ accounts for representations of abstract concepts of the specification in terms of the syntax of the program. That is to say, the morphism reflects design decisions taken during the implementation of S in terms of P , say the choice of specific representations for the state of a system. Hence, there is an interest in manipulating morphisms of the form $\square: S \square \mathbf{spec}(P)$ as capturing the possible realisations of specifications in terms of programs.

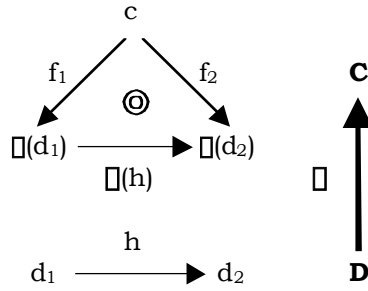
6.2.2 DEFINITION – structured morphisms

Given a functor $\square: \mathbf{D} \square \mathbf{C}$, a \square -structured morphism is a \mathbf{C} -morphism $f: c \square \square(d)$ where c is an object of \mathbf{C} and d is an object of \mathbf{D} .

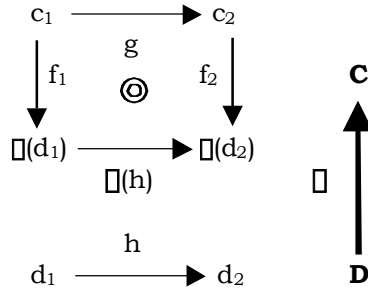
Structured morphisms can be organised in categories that generalise the notion of comma-category that we studied in section 3.2:

6.2.3 DEFINITION – comma-categories

1. Comma-categories, as defined in 3.3.2, can be generalised to the case of a functor $\square: \mathbf{D} \rightarrow \mathbf{C}$ and an object $c: \mathbf{C}$, to define a category $c \square \square$ that has for objects all pairs $\langle f: c \square \square(d), d \rangle$ where f is a \square -structured morphism with domain c . The morphisms between $\langle f_1: c \square \square(d_1), d_1 \rangle$ and $\langle f_2: c \square \square(d_2), d_2 \rangle$ are all the \mathbf{D} -morphisms $h: d_1 \rightarrow d_2$ such that $f_1; \square(h) = f_2$. These categories are called under-cone categories in [22, page 48].



2. The construction above can be further generalised to define the category $\square \square$ (or $\mathbf{C} \square \square$) that has for objects all the \square -structured morphisms, i.e. triples $\langle c, f: c \square \square(d), d \rangle$. The morphisms between $\langle c_1, f_1: c_1 \square \square(d_1), d_1 \rangle$ and $\langle c_2, f_2: c_2 \square \square(d_2), d_2 \rangle$ are all the pairs $\langle g, h \rangle$ of \mathbf{C} -morphisms $g: c_1 \rightarrow c_2$ and \mathbf{D} -morphisms $h: d_1 \rightarrow d_2$ such that $f_1; \square(h) = g; f_2$.



PROOF

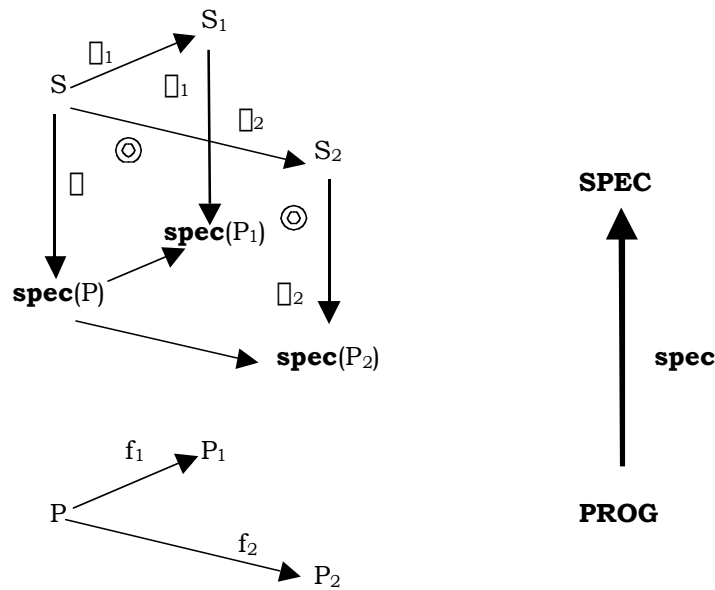
The proof that these constructions do yield categories is left as an exercise. The reader is also invited to consult [79, pages 46-48] for further generalisations of the notion of comma category, including the one that justifies their name!

6.2.4 REMARK – compositionality in software development

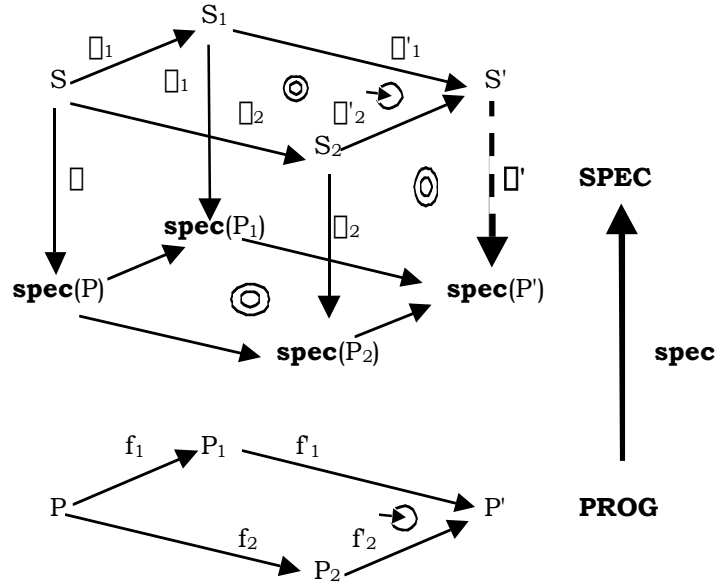
In chapter 4, we showed how universal constructions could be used to formalise notions of composition that occur in system development. These were applied to system specification in 6.1.24 including an illustration of their role in supporting evolution through the interconnection, in run-time, of new components that can bring about new emergent properties.

However, ideally, such constructions should apply to realisations and not to specifications alone in the sense that, when we compose specifications for which we have already correct implementations, we should be able to obtain a correct implementation for the resulting specification as a composition of the implementations of its components. This is usually called *compositionality* of system development. We are now going to show how this form of compositionality can be formulated in this setting and derived from the existence of an abstraction functor.

First of all, we are now interested in extending the notion of realisation to diagrams as capturing configurations of complex systems: a realisation of a specification diagram $\square: I \rightarrow \mathbf{SPEC}$ by a program diagram $\square': I \rightarrow \mathbf{PROG}$ is an $|I|$ -indexed family $(\square_i: \square(i) \rightarrow \text{spec}(\square'(i)))_{i \in I}$ of realisations such that, for every $f: i \rightarrow j$ in I , $\square(f); \square_j = \square_i; \text{spec}(\square'(f))$. That is to say, in addition to the individual components, the interconnections have to be realised as well.



Compositionality of the relationship between programs and specifications can be expressed through the fact that the colimit of \square' – the diagram of programs – is a realisation of \square – the diagram of specifications – in an essentially unique way.



We can actually prove that, as long as **spec** is a functor, this property holds regardless of the nature of the specifications and programs. Indeed, because functors preserve the commutativity of diagrams, the image of the program diagram commutes, i.e. in the case of the diagram in the figure,

$$\mathbf{spec}(f_1); \mathbf{spec}(f'_1) = \mathbf{spec}(f_2); \mathbf{spec}(f'_2)$$

This entails that $\square_1; \square'_1; \mathbf{spec}(f'_1) = \square; \mathbf{spec}(f_1); \mathbf{spec}(f'_1) = \square; \mathbf{spec}(f_2); \mathbf{spec}(f'_2) = \square_2; \square'_2; \mathbf{spec}(f'_2)$. Because $\langle \square'_1, \square'_2, S' \rangle$ is a pushout of $\langle S, \square_1, \square_2 \rangle$, we can conclude that there exists a unique morphism $\square': S \rightarrow \mathbf{spec}(P')$ such that $\square_1; \mathbf{spec}(f'_1) = \square'_1; \square'$ and $\square_2; \mathbf{spec}(f'_2) = \square'_2; \square'$. Notice that these two equations express the fact that the pairs $\langle \square'_i; \mathbf{spec}(f'_i) \rangle$ are, indeed, morphisms of realisations. Also notice that **spec** is not required to preserve the pushout of programs!

Because compositionality is a property that is not exactly easy to achieve, as more than 20 years of research in the area as shown, this means that the existence of a functor relating two domains of system modelling expresses a very strong structural relationship between them. In [31], we have given examples of situations in which compositionality fails because some of the properties required of functors are not met by the way specifications relate to programs. Basically, what is at stake is the balance that must be struck between the nature of the properties that the abstraction map is able to derive, and the ability of program morphisms to preserve such properties. In [31] it is also shown that, in the absence of a functorial relationship, properties of systems may emerge that result from the need to regulate the interconnections between the components. This happens when the "semantics" of the programming language is not strong enough to ensure the preservation of the properties made observable through the specification language. This is similar to what happens in social systems in which regulators (e.g. the police) are necessary to enforce laws that are not "natural" but imposed by the society for coordinating the behaviour of individuals.

An example of such a situation can be given in terms of Eiffel class specifications and linear temporal logic. If we had chosen to restrict the behaviours of Eiffel classes to those that are normative in the sense that routines are only executed when their pre-conditions hold, then the following sentence would be included in the strongest specification of every routine:

$$r \text{ } pre_r$$

That is to say, if the routine r is about to happen, its pre-condition holds. It is easy to see that this property is not preserved by class morphisms. Indeed, given a class morphism f , all that f guarantees is that $F(pre_r) \vdash pre_{F(r)}$. Hence, from $(F(r) \text{ } pre_{F(r)})$ we cannot infer the translation through F of $(r \text{ } pre_r)$, i.e. the property $(F(r) \text{ } F(pre_r))$. This means that an implementation of a class specification that refuses to execute every routine when the pre-condition fails, cannot be reused, with the same semantics of refusal, when the specification is inherited into a larger one. In other words, inheritance does not preserve the property of refusing execution of routines for which pre-conditions fail. This is probably why the semantics of pre-conditions in Eiffel does not include such refusals...

Although the construction in 6.2.4 was motivated by the extension of the notion of realisation to diagrams as compositions of complex systems, it admits a dual reading as extending composition to realisations, i.e. moving to the category $\mathcal{I}\mathcal{S}\mathcal{P}\mathcal{E}\mathcal{C}$ whose objects are, precisely, the realisations. Actually, this dual reading is supported by a dual visualisation of the figures above: whereas we analysed them from the point of view of "vertical" relationships (the realisations) between "horizontal structures" (the specification and the program configurations), we are now analysing them as horizontal relationships (a configuration) between vertical structures (the realisations of the component specifications).

Indeed, because, for every $f: i \rightarrow j$ in \mathcal{I} , $\mathcal{I}(f); \mathcal{I}_j = \mathcal{I}_i \mathcal{S}\mathcal{P}\mathcal{E}\mathcal{C}(\mathcal{I}(f))$, \mathcal{I} and \mathcal{I}_i define a diagram $\langle \mathcal{I}, \mathcal{I}_i \rangle: \mathcal{I} \rightarrow \mathcal{I}\mathcal{S}\mathcal{P}\mathcal{E}\mathcal{C}$ such that, for every i , $\langle \mathcal{I}, \mathcal{I}_i \rangle(i)$ is $\langle \mathcal{I}(i), \mathcal{I}_i(i) \rangle$ and, for every $f: i \rightarrow j$, $\langle \mathcal{I}, \mathcal{I}_i \rangle(f)$ is $\langle \mathcal{I}(f), \mathcal{I}_i(f) \rangle$. Compositionality expresses the existence (and uniqueness) of a colimit for this diagram of realisations when \mathcal{I} and \mathcal{I}_i admit colimits.

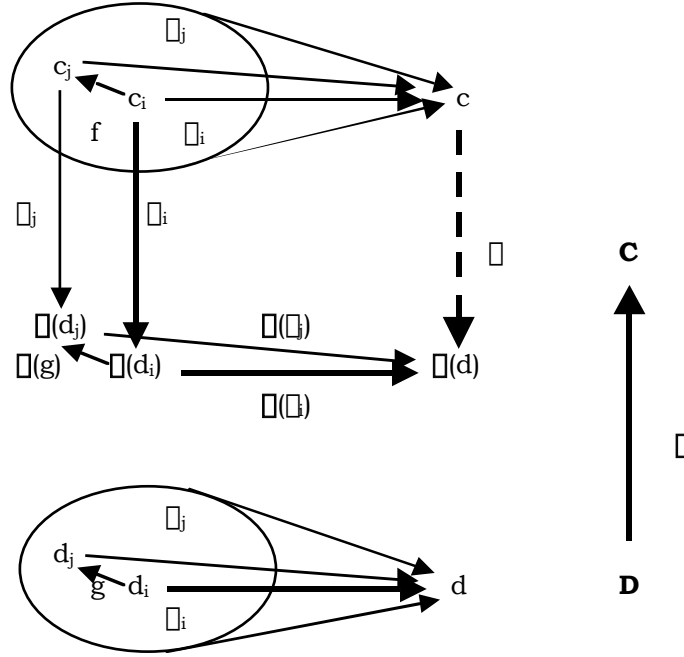
Compositionality in the sense that we have just described is a property that can be stated and proved in the general setting of comma-categories:

6.2.5 DEFINITION/PROPOSITION – projection functor creates colimits

1. Given a functor $\mathcal{I}: \mathcal{D} \rightarrow \mathcal{C}$, we define two functors $\mathcal{I}_c: \mathcal{I}\mathcal{C} \rightarrow \mathcal{C}$ and $\mathcal{I}_d: \mathcal{I}\mathcal{D} \rightarrow \mathcal{D}$ by projecting objects and morphisms of the comma-category $\mathcal{I}\mathcal{I}$ to their \mathcal{C} and \mathcal{D} components, respectively.
2. The functor $\langle \mathcal{I}_c, \mathcal{I}_d \rangle: \mathcal{I}\mathcal{I} \rightarrow \mathcal{C}\mathcal{D}$ creates colimits.

PROOF

1. Left as an exercise.
2. Consider a diagram $\mathcal{I}: \mathcal{I} \rightarrow \mathcal{I}\mathcal{I}$ and colimits $\mathcal{I}(\mathcal{I})_c: \mathcal{I}\mathcal{C} \rightarrow \mathcal{C}$ and $\mathcal{I}(\mathcal{I})_d: \mathcal{I}\mathcal{D} \rightarrow \mathcal{D}$ for its projections \mathcal{I}_c and \mathcal{I}_d , respectively.



- *existence of a co-cone in $\underline{\mathcal{C}}$ whose image is the pair of projections*: basically, we have to show that there is a morphism $\square: c \rightarrow \square(d)$ such that $\square_i; \square(\square_i) = \square_i; \square$. To prove this, consider the \mathbf{C} -co-cone defined by $\{\square_i; \square(\square_i)\} \rightarrow \square(d)$. This co-cone is commutative because, given any morphism $f: c_i \rightarrow c_j$ in the base, if $g: d_i \rightarrow d_j$ is the morphism in \mathbf{D} that, together with f , constitutes a morphism in $\underline{\mathcal{C}}$, we have that

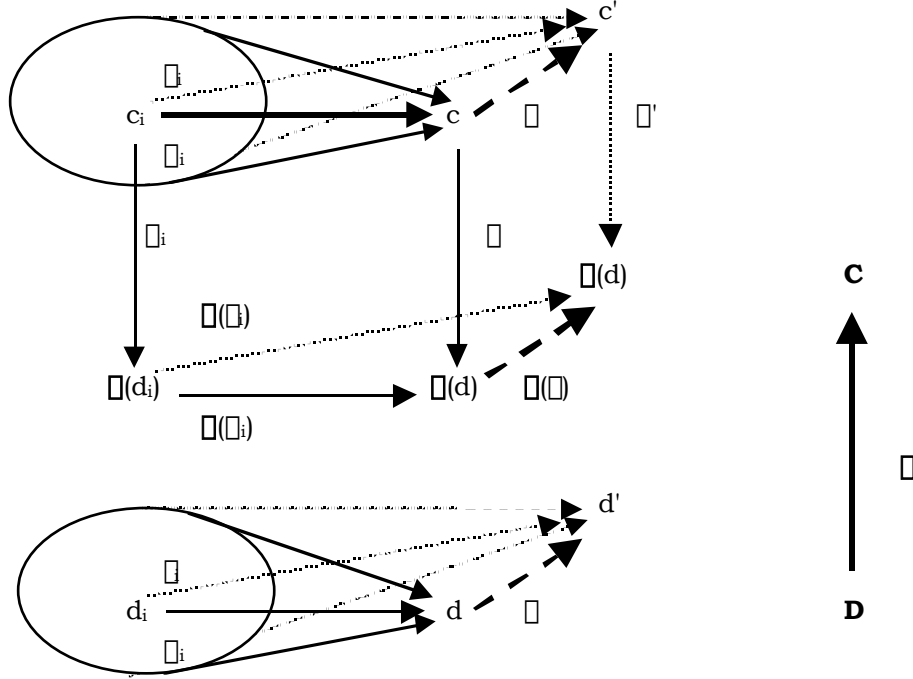
$$\begin{aligned}
 \square_i &= g; \square_j && \text{because the } \mathbf{D}\text{-co-cone is commutative} \\
 \square(\square_i) &= \square(g); \square(\square_j) && \text{because functors preserve composition} \\
 f; \square_j &= \square_i; \square(g) && \text{because } \langle f, g \rangle \text{ is a morphism in } \underline{\mathcal{C}} \\
 f; \square_j; \square(\square_j) &= \square_i; \square(g); \square(\square_j) && \text{from the previous equality} \\
 f; \square_j; \square(\square_j) &= \square_i; \square(\square_i) && \text{applying the second equality}
 \end{aligned}$$

Hence, because $\square: \mathbf{C} \rightarrow \mathbf{D}$ is a colimit, there is a (unique) morphism $\square: c \rightarrow \square(d)$ such that $\square_i; \square(\square_i) = \square_i; \square$.

- *uniqueness of the co-cone*: for the co-cone to be in $\underline{\mathcal{C}}$, the equalities $\square_i; \square(\square_i) = \square_i; \square$ have to hold and the fact that the co-cone in \mathbf{C} is a colimit implies that \square is the only morphism $c \rightarrow \square(d)$ that satisfies that property.
- *the co-cone is a colimit*: let $\langle \square, \square' : \mathbf{C} \rightarrow \mathbf{D} \rangle$ be a commutative co-cone. Its projections $\square: \mathbf{C} \rightarrow \mathbf{D}$ and $\square': \mathbf{C} \rightarrow \mathbf{D}$ are also commutative. Therefore, because \square and \square' are colimits, there are unique $\square: c \rightarrow c'$ and $\square': d \rightarrow d'$ such that $\square_i = \square_i; \square$ and $\square_i = \square_i; \square'$. We want to prove, first of all, that $\langle \square, \square' \rangle$ is a morphism of $\underline{\mathcal{C}}$, i.e. $\square; \square' = \square; \square(\square')$. For that purpose, we are going to prove that the co-cone defined by $(\square_i; \square(\square'))$ is commutative: given an arbitrary $\langle f: c_i \rightarrow c_j, g: d_i \rightarrow d_j \rangle$ in the base for a structured object $\langle c_j, \square_j, d_j \rangle$,

$$\begin{aligned}
 f; \square_j; \square(\square_j) &= && \langle \square_j, \square_j \rangle \text{ being a morphism,} \\
 &= f; \square_i; \square' && \square \text{ being commutative, } f; \square_i = \square_i \\
 \square_j; \square(\square_j) &= \square_i; \square' && \langle \square_j, \square_j \rangle \text{ being a morphism,} \\
 &= \square_i; \square(\square') && \\
 \square_i; \square(\square_j) &= \square_i; \square' &&
 \end{aligned}$$

Hence, there is a unique morphism $\varphi: c \rightarrow \varphi(d)$ such that $\varphi_i \circ \varphi = \varphi_i \circ \varphi(\varphi)$. But both $\varphi \circ \varphi'$ and $\varphi \circ \varphi(\varphi)$ satisfy that property. Therefore, they are equal. Finally, we have to prove that $\langle \varphi, \varphi' \rangle$ is the only morphism satisfying $\langle \varphi_i, \varphi_i \rangle = \langle \varphi_i, \varphi_i \rangle; \langle \varphi, \varphi' \rangle$. But this is because any morphism $\langle \varphi, \varphi' \rangle$ satisfying the same equation is such that $\varphi_i = \varphi_i \circ \varphi$ and $\varphi' = \varphi_i \circ \varphi'$, which implies $\varphi = \varphi$ and $\varphi' = \varphi$.



6.3 Functor-structured categories

In this section, we present a construction that yields concrete categories of a very simple nature but with wide applicability:

6.3.1 DEFINITION – functor-structured categories

Let $\varphi: \mathbf{C} \rightarrow \mathbf{SET}$ be a functor. We define the category $\mathbf{spa}(\varphi)$ whose objects are the pairs $\langle c, S \rangle$ where $c: \mathbf{C}$ and $S \subseteq \varphi(c)$ and whose morphisms $f: \langle c, S \rangle \rightarrow \langle d, T \rangle$ are the morphisms $f: c \rightarrow d$ of \mathbf{C} such that $\varphi(f)(S) \subseteq T$.

PROOF

As already illustrated so many times in section 3.2, we have to prove that the composition law and the identity map inherited from \mathbf{C} are applicable.

- Given morphisms $f: \langle c, S \rangle \rightarrow \langle d, T \rangle$ and $g: \langle d, T \rangle \rightarrow \langle e, R \rangle$, we have

- $\varphi(f)(S) \subseteq T$ because f is a morphism of $\mathbf{spa}(\varphi)$
- $\varphi(g)(\varphi(f)(S)) \subseteq \varphi(g)(T)$ from (a)

3. $\Box(g)(T)\Box R$ because g is a morphism of **spa**(\Box)
 4. $\Box(g)(\Box(f)(S))\Box R$ from (b) and (c) and transitivity of inclusion
 5. $(\Box(f);\Box(g))(S)\Box R$ from (d) and function composition in **SET**
 6. $\Box(f;g)(S)\Box R$ from (e) and the properties of functors.
- The case of the identity map is trivial: given a pair $\langle c, S \rangle$, the identity on c is such that $\Box(id_c)(S) = id_{\Box(c)}(S) = S$.

Functor-structured categories, sometimes also called *spa-categories*, are studied in detail in [1] where some additional terminology is introduced. The objects of such categories are usually called \Box -spaces and their morphisms \Box -maps.

6.3.2 EXAMPLE – processes

We have already mentioned that we may regard a pointed set as a signature or alphabet of a process: the proper elements of the set denote actions or events in which the process can get involved, and the designated element denotes an action of the environment, i.e. an action in which the process is not involved. We can associate with every pointed set A_\Box , the set of possible trajectories over A_\Box : $\mathbf{tra}(A_\Box) = \{\Box : \Box \Box A_\Box\}$. That is, a trajectory for an alphabet is an infinite sequence of actions. Notice that finite behaviours can be represented by infinite sequences that, after a certain point, consist of the designated element, i.e. consist only of environment steps. This association defines a functor $\mathbf{tra} : \mathbf{SET}_\Box \rightarrow \mathbf{SET}$ if we extend it to morphisms as follows: $\mathbf{tra}(f : A_\Box \rightarrow B_\Box)(\Box) = \Box; f$, i.e. $\mathbf{tra}(f)(\Box)(i) = f(\Box(i))$. That is, every morphism of pointed sets induces a translation between the corresponding trajectories by pointwise application of the function between the base sets. We usually denote $\mathbf{tra}(f)$ by f^\Box .

We can now define the category **PROC** of processes as being **spa(tri)**: a process consists of a pair $\langle A_\Box, \Box \rangle$ where $\Box \in \mathbf{tra}(A_\Box)$, i.e. a process consists of an alphabet and a set of trajectories that capture its behaviour; a process morphism $f : \langle A_{\Box_1}, \Box_1 \rangle \rightarrow \langle A_{\Box_2}, \Box_2 \rangle$ is a morphism $f : A_{\Box_1} \rightarrow A_{\Box_2}$ between the underlying pointed sets such that $f^\Box(\Box_1) \Box \Box_2$, i.e., such that every trajectory of the source is translated to a trajectory of the target.

Intuitively, a morphism $f : P_1 \rightarrow P_2$ identifies process P_2 as a component of process P_1 : it identifies, for every action of P_1 , what is the participation of P_2 in that action; if the event of P_1 is mapped to the designated event of P_2 , this means that P_2 does not participate in it, making it an environment step for P_2 . Notice that the preservation of the designated element, as enforced through the morphism, is consistent with the view of P_2 as a component of P_1 : an environment step for P_1 must necessarily be an environment step for P_2 . Hence, P_1 identifies part of the environment of P_2 .

The condition on the trajectories requires that every possible behaviour of P_1 (the system) be mapped to one of the allowed behaviours of P_2 (the component). That is to say, the system cannot exhibit behaviours that are not allowed by the component. However, it is possible that behaviours of the component do not show up in the system, for instance because interactions with other components within the system prevent them from occurring.

Functor-structured categories provide us examples of some of the kinds of categories that we discussed in section 6.1.

6.3.3 PROPOSITION

Let $\square: \mathbf{C} \rightarrow \mathbf{SET}$ be a functor.

1. The functor $\square: \mathbf{spa}(\square) \rightarrow \mathbf{C}$ that forgets the **SET**-component of each object defines $\mathbf{spa}(\square)$ as a concrete category over \mathbf{C} .
2. The forgetful functor $\square: \mathbf{spa}(\square) \rightarrow \mathbf{C}$ is both a split fibre-complete fibration and a split fibre-co-complete cofibration. Given a \mathbf{C} -morphism $f: c \rightarrow c'$, the cartesian morphism for $\langle c', S' \rangle$ is $f: \langle c, \square(f)^{-1}(S') \rangle \rightarrow \langle c', S' \rangle$ and the cocartesian morphism for $\langle c, S \rangle$ is $f: \langle c, S \rangle \rightarrow \langle c', \square(f)(S) \rangle$.

PROOF

Left as an exercise.

6.3.4 EXAMPLE – processes

We call **alph** the forgetful functor defined by 6.3.3(1) on **PROC**: it projects processes and their morphisms to the underlying alphabets (pointed sets).

Notice how the (co)cartesian morphisms have an intuitive interpretation in **PROC**. The cartesian morphism returns, for a given process and alphabet morphism having the process as target, the least deterministic system over that alphabet in which the process can fit, through the morphism, as a component: any additional behaviours of the system would violate the allowed behaviours of the given process. On the other hand, the cocartesian morphism returns, for a given process and alphabet morphism having the process as source, the most deterministic process over the target alphabet that it admits, through the morphism, as a component: any additional behaviours that the component may have will not be able to be observed in the given system. That is to say, the cartesian and cocartesian morphism allow us to gauge the roles that a given process can play through a morphism as a system or as a component.

The characterisation that 6.3.3 provides for (co)cartesian morphisms allows us to derive some useful properties of functor-structured categories:

6.3.5 COROLLARY

Let $\square: \mathbf{C} \rightarrow \mathbf{SET}$ be a functor and $\square: \mathbf{spa}(\square) \rightarrow \mathbf{C}$ the forgetful functor induced by the **spa**-construction⁹.

1. \square lifts limits uniquely. Any limit $\square: c \rightarrow \square$ for $\square: \mathbf{I} \rightarrow \mathbf{spa}(\square)$ with $\square = \langle c_i, S_i \rangle$ is lifted uniquely to $\square: \langle c, S \rangle \rightarrow \square$ where $S = \prod_{i \in \mathbf{I}} \square(\square_i)^{-1}(S_i)$.
2. \square lifts colimits uniquely. Any colimit $\square: \square \rightarrow c$ for $\square: \mathbf{I} \rightarrow \mathbf{spa}(\square)$ with $\square = \langle c_i, S_i \rangle$ is lifted uniquely to $\square: \square \rightarrow \langle c, S \rangle$ where $S = \prod_{i \in \mathbf{I}} \square(\square_i)(S_i)$.

6.3.6 COROLLARY

Let $\square: \mathbf{C} \rightarrow \mathbf{SET}$ be a functor.

⁹ For readability, we shall omit the forgetful functor when referring to objects and morphisms of \mathbf{C} that are projected from **spa**(\square).

1. If \mathbf{C} is complete so is $\mathbf{spa}(\square)$. Moreover, all limits in $\mathbf{spa}(\square)$ are concrete (see 6.1.9).
2. If \mathbf{C} is co-complete so is $\mathbf{spa}(\square)$. Moreover, all colimits in $\mathbf{spa}(\square)$ are concrete.

Therefore, we can apply the recipe for calculating limits and colimits in $\mathbf{spa}(\square)$ that we discussed in section 6.1: (1) project the diagram to the base category and calculate its (co)limit; (2) lift the result by computing the intersection of the inverse images of the set components (for a limit) or the union of the direct images of the set components (for the colimit).

It is useful to see how this construction can be instantiated to particular universal constructions:

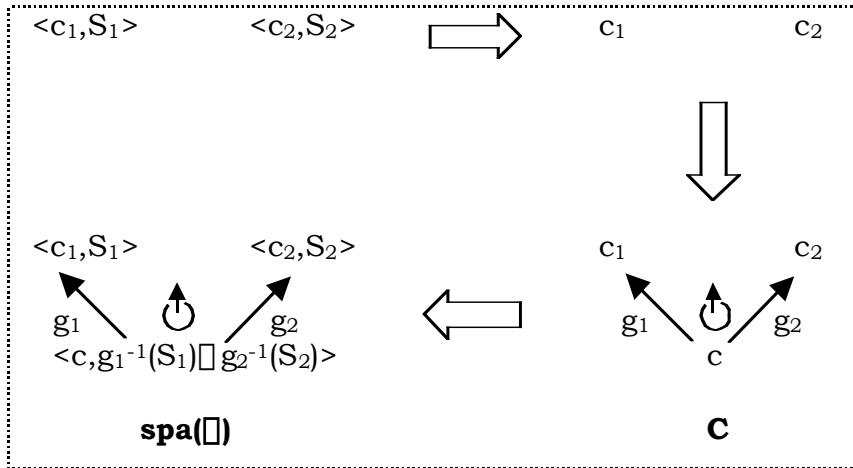
1. Lifting of the initial and terminal objects

Let $0_{\mathbf{C}}$ be an initial object of \mathbf{C} . Then $\langle 0_{\mathbf{C}}, \emptyset \rangle$ is initial for $\mathbf{spa}(\square)$.

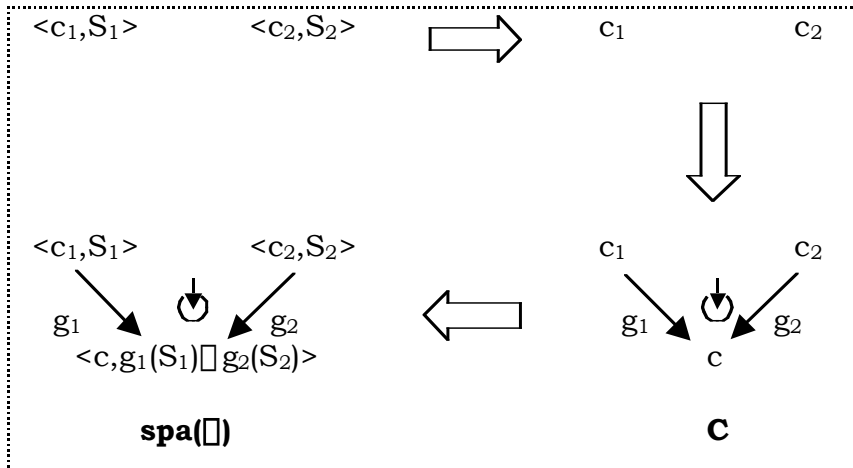
Let $1_{\mathbf{C}}$ be a terminal object of \mathbf{C} . Then $\langle 1_{\mathbf{C}}, \square(1_{\mathbf{C}}) \rangle$ is terminal for $\mathbf{spa}(\square)$.

2. Universal constructions in three steps: (1) projection of the $\mathbf{spa}(\square)$ -diagram to \mathbf{C} ; (2) universal construction performed in \mathbf{C} ; (3) lifting the result back to $\mathbf{spa}(\square)$.

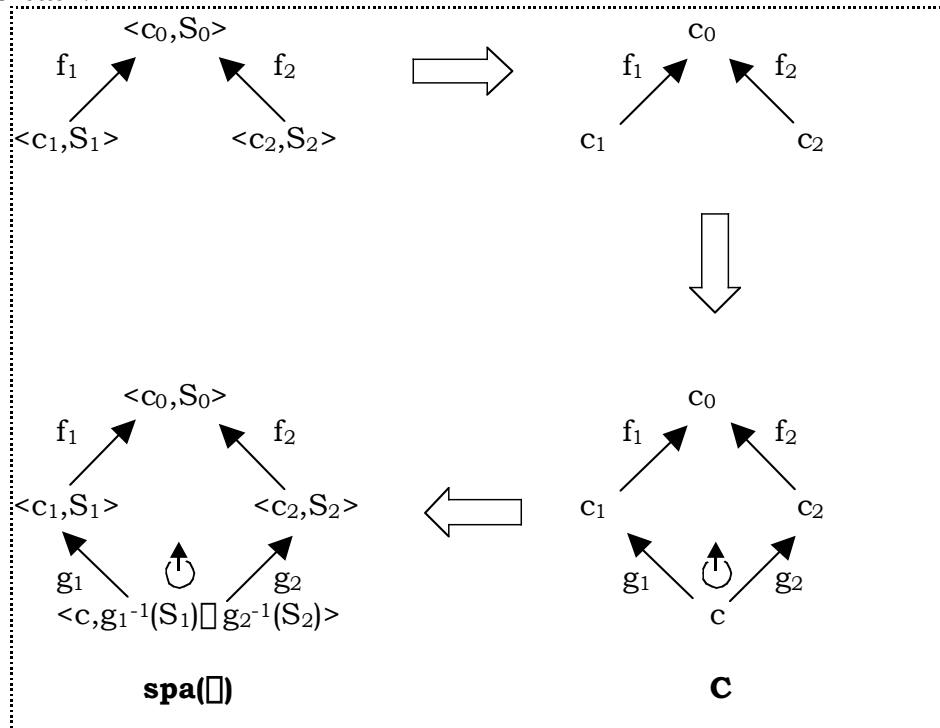
Product:



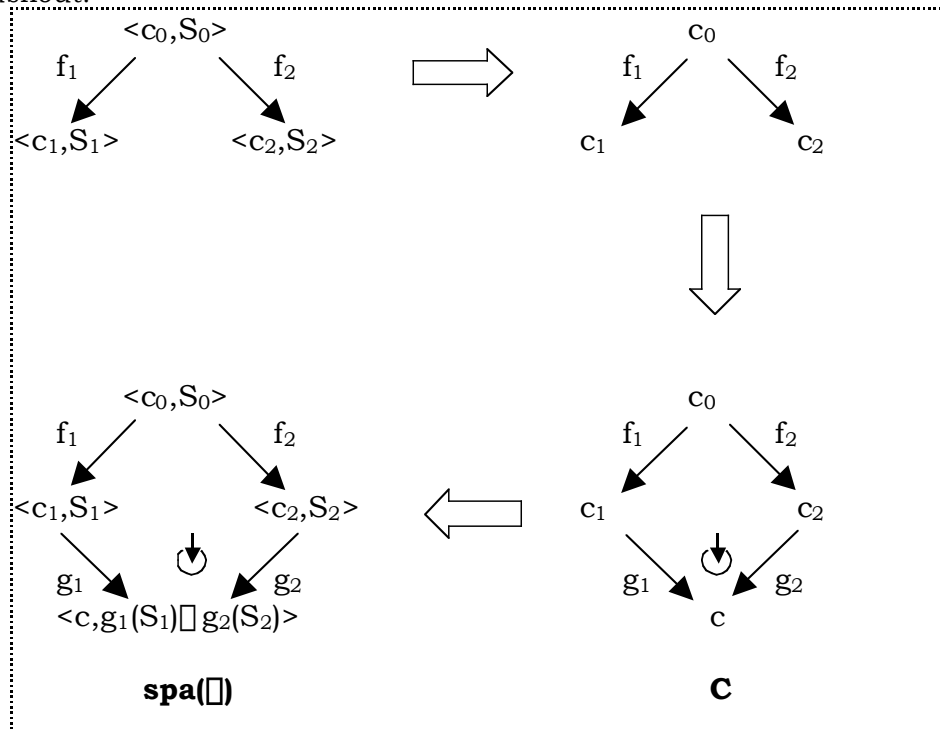
Sum:



Pullback:



Pushout:



Notice that the lifting of the product and the pullback are exactly the same: the "shared" object is only used for determining the **C**-component of the apex of the pullback and does not interfere with the calculation of the **SET** component. This is another example of the kind of separation that can be achieved between "coordination"

and "computation" that we motivated already in section 5.2. We shall return to these aspects later on but this is a simple illustration of the principle.

6.3.7 EXAMPLE – parallel composition of processes

When applied to processes, these constructions provide us with a mathematical semantics for typical operations of process calculi. Recalling the universal constructions that we studied in chapter 1 for pointed sets, we have:

1. The terminal process is $\langle \{\square_\emptyset\}, \{\square_\emptyset \} \rangle$: its alphabet contains only the witness for actions of the environment, and its behaviour reflects exactly that – it witnesses life go by. That is, we have an idle process.
2. The initial process is $\langle \{\square_\emptyset\}, \emptyset \rangle$: its alphabet is exactly the same as for the terminal process, but its behaviour is completely different – it does nothing, not even witnessing life go by! That is, we have a blocking process that deadlocks any system to which it is interconnected (see below).
3. We have already seen in section **Error! Reference source not found.** that products of pointed sets model the interleaving of alphabets in the sense that they compute the set of all the synchronisation pairs of actions between the components plus the individual actions themselves. When we take into account the behaviour of processes, products return the infinite sequences of such parallel actions that, once projected into the components, result into behaviours of the components. That is to say, the product of processes $\langle A_1, \square_1 \rangle$ and $\langle A_2, \square_2 \rangle$ is obtained by computing the product $\langle A, g_1, g_2 \rangle$ of the alphabets and by taking as set of behaviours the intersection of the inverse images of the sets of behaviours of the components. This intersection consists of the sequences $\square : \square \square A$ such that $g_1(\square) \square \square_1$ and $g_2(\square) \square \square_2$. Hence, what we obtain is the traditional trace-based semantics of the parallel composition of processes.

Notice the difference between the idle and the blocking process. When put in parallel with another process, the idle process is "absorbed", i.e. the result of the parallel composition is the other process: this is because the behaviour of the idle process is already present in any other process (except the blocking one). Indeed, the product of a terminal object with any other object is, up to isomorphism, that object. Hence, the idle process does nothing and lets the others do as they please.

On the contrary, the blocking process "absorbs" any other process with which it is put in parallel: it does nothing and does not let the others do anything. This is because the product of the initial object with any other process P returns a process with the alphabet of P but with an empty behaviour. As we said at the very beginning of the book, Category Theory is all about the social behaviour of objects...

4. Pullbacks allow us to select only the behaviours that satisfy certain synchronisation requirements. We also saw in section **Error! Reference source not found.** that such requirements are expressed through the morphisms that connect the components to the "channel" that interconnects them. Each action of the channel acts as a point for *rendez-vous* synchronisation in the sense that the actions that participate in a *rendez-vous* are not allowed to occur in isolation, i.e. are not part of the alphabet of the resulting process. In what concerns the re-

sulting behaviour, its computation is exactly as for products: it consists of the sequences of actions determined by the pullback of alphabets that are projected into behaviours of the components. Hence, all the synchronisation mechanism is achieved at the level of the alphabets. In summary, pullbacks model parallel composition with synchronisation.

5. What we have just observed about pullbacks can be generalised to limits in general. The limit of the diagram of alphabets internalises all the interconnections established via the morphisms as synchronisation sets: each component may be involved with at most one action in a synchronisation set. That is, internal synchronisations cannot be established and a given component may not participate in a given *rendez-vous*. When we take into account the behaviour of the processes involved, limits return the infinite sequences of such synchronisation sets that are projected into allowed behaviours of the components.

6.4 The Grothendieck construction

In section 6.1, we showed how every split fibration $\square:D \rightarrow \mathbf{C}$ defines a functor $\mathbf{ind}(\square): \mathbf{C}^{op} \rightarrow \mathbf{CAT}$ that maps every object of \mathbf{C} to its fibre. In the case of a split cofibration, we obtain $\mathbf{ind}(\square): \mathbf{C} \rightarrow \mathbf{CAT}$. The objects of \mathbf{C} can be seen as indexes, or types, that are used for classifying the objects of \mathbf{D} . The functor \square makes the type assignments and the functor $\mathbf{ind}(\square)$ groups the objects according to their types. On the morphism side, the functor $\mathbf{ind}(\square)$ tells us how to translate between fibres when moving from one type to another through a type morphism. For fibrations, this translation is contravariant (like when taking inverse images), whereas for cofibrations the translation is covariant (like when taking direct images).

For instance, if \mathbf{C} is the inheritance hierarchy of an object-oriented system, and \mathbf{D} models the population of objects, we can think of a cofibration that assigns to every object the type of which it is a direct instance. For every inheritance morphism, the cocartesian morphisms define how each object of the child class is also an instance of the parent type (which feature renaming applies, etc). The \mathbf{CAT} -based functor returns, for each type, the population of direct instances associated with the type and, for each inheritance morphism, the functor that maps instances of the child to instances of the parent.

The fact that we work with a fibration or a cofibration, and that the associated \mathbf{CAT} -based functor is contravariant or covariant, is not very important. After all, it is all a matter of arrow direction. However, as we have already pointed out, it is important that the direction of the arrows is chosen so as to be "natural" in the context in which they are going to be used. For instance, in the case of inheritance, hierarchies are traditionally depicted as graphs with the arrows pointing to the parents, and it would be counterintuitive to formalise them with morphisms going in the opposite direction. All this to say that, in this section, one direction will be preferred for definitions and results, but examples will be given in their "natural" direction. The reader is encouraged to detail the dual constructions as an exercise.

We have also mentioned that indexed categories are very much part of the categorical folklore danced in Computing [e.g. 12,22,103]. We will follow [103] more closely

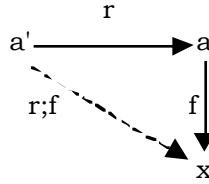
because it shares with us the same kind of "engineering" inspiration that we set ourselves to promote. The reader will also find in [103] many more examples and a much more extended coverage of this topic. Take this section as an appetiser to what can be a much bigger meal.

6.4.1 DEFINITION – indexed categories

An indexed category \mathcal{I} over a category \mathbf{I} (of indexes) is a functor of the form $\mathcal{I}^{\mathbf{I}^{\text{op}}} \mathbf{CAT}$.

6.4.2 EXAMPLE – comma-categories

In 3.2.2 we showed how, given a category \mathbf{C} and an object $a \in \mathbf{C}$, we can define the category of objects under a – a/\mathbf{C} – as consisting of all the morphisms f of \mathbf{C} of the form $f: a' \rightarrow x$ organised in such a way that the morphisms between $f: a' \rightarrow x$ and $g: a' \rightarrow y$ are all the morphisms $h: x \rightarrow y$ such that $fh = g$. Given a morphism $r: a' \rightarrow a$, there is a natural way of translating objects under a to objects under a' : given $f: a \rightarrow x$, we map it to $(r;f): a' \rightarrow x$. It is easy to prove that this translation is indeed a functor $a/\mathbf{C} \rightarrow a'/\mathbf{C}$ so that we obtain an indexed category $\mathcal{I}^{\mathbf{C}: \mathbf{C}^{\text{op}}} \mathbf{CAT}$.



6.4.3 EXERCISE

Complete the previous example by doing the proofs and generalising it to the comma-categories of the form a/\mathcal{I} as defined in 6.2.3.

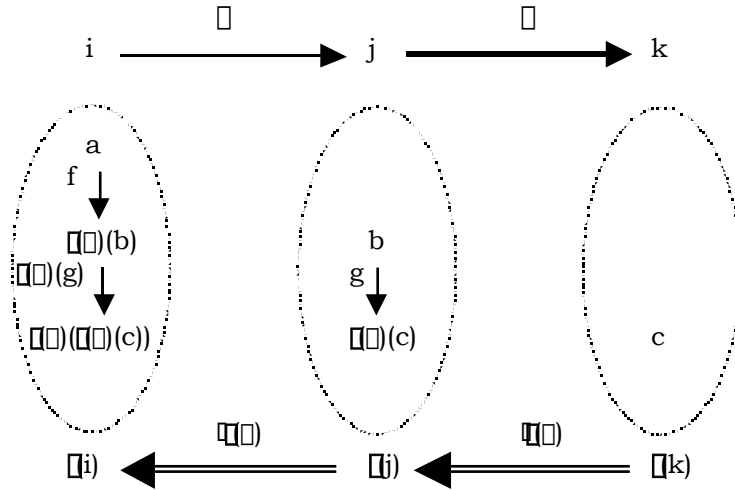
Once we look at functors $\mathcal{I}^{\mathbf{I}^{\text{op}}} \mathbf{CAT}$ as performing an indexing of certain classes of objects, it seems natural to think about the amalgamation of all these classes into a single population, using the indexes to distinguish between the objects according to their provenance. This operation is traditionally called the "Grothendieck construction" [12,22] in honour of A.Grothendieck. We will use the less "flattering" and, perhaps, more "flatfooted" terminology proposed in [103], and refer to this operation as "flattening" the indexed category.

6.4.4 DEFINITION – flattening an indexed category

Given an indexed category $\mathcal{I}^{\mathbf{I}^{\text{op}}} \mathbf{CAT}$ we define a category $\mathbf{FLAT}(\mathcal{I})$ as follows:

1. The objects of $\mathbf{FLAT}(\mathcal{I})$ are all the pairs $\langle i, a \rangle$ where $i \in \mathbf{I}$ is an index and $a \in \mathcal{I}(i)$ is an object of "type" i .
2. The morphisms $\langle i, a \rangle \rightarrow \langle j, b \rangle$ are all the pairs $\langle \square, f \rangle$ where $\square: i \rightarrow j$ is a morphism of indexes and $f: a \rightarrow \square(b)$ is a morphism in $\mathcal{I}(i)$.

3. The composition of morphisms is defined componentwise: given $\langle \square, f \rangle : \langle i, a \rangle \rightarrow \langle j, b \rangle$ and $\langle \square, g \rangle : \langle j, b \rangle \rightarrow \langle k, c \rangle$, we define $\langle \square, f \rangle ; \langle \square, g \rangle = \langle \square, \square(f; \square(g)) \rangle$.
4. The identity for an object $\langle i, a \rangle$ is $\langle id_i, id_a \rangle$.

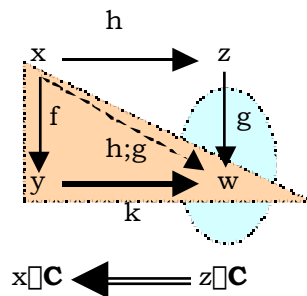


PROOF

The proof that a category is defined in this way is left as an exercise.

6.4.5 EXAMPLE

Consider the flattening of the indexed category $\square \mathbf{C} :: \mathbf{C}^p \rightarrow \mathbf{CAT}$ defined by the comma-category construction as studied above. The objects of $\mathbf{FLAT}(\square \mathbf{C})$ are triples $\langle x, f : x \rightarrow y, y \rangle$ and the morphisms $\langle x, f : x \rightarrow y, y \rangle \rightarrow \langle z, g : z \rightarrow w, w \rangle$ are the pairs $\langle h, k \rangle$ such that $h : x \rightarrow z$, $k : y \rightarrow w$ and $h;g = f;k$.



The equation results from the requirement that k be a morphism in $x \square \mathbf{C}$ between $f : x \rightarrow y$ and $h;g : x \rightarrow w$ which is the translation of $g : z \rightarrow w$ defined by h . This category is usually called the "arrow category of \mathbf{C} ".

6.4.6 EXERCISE

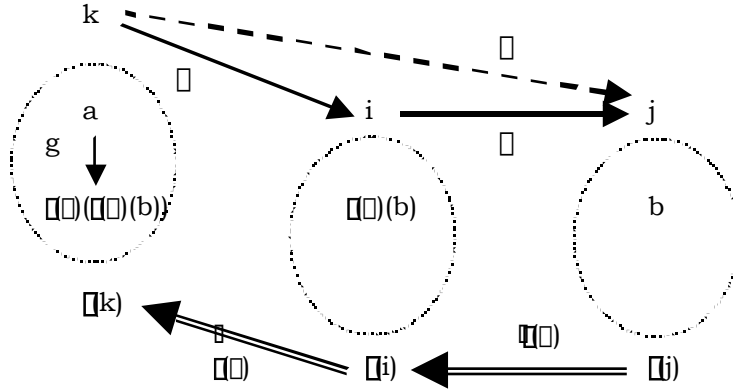
Prove that the category $\square \square$ defined in 6.2.3 "is" the flattening of the indexed-category of exercise 6.4.3.

The flattened indexed category comes equipped with some structure:

6.4.7 PROPOSITION

Consider an indexed category $\square^{I^P} \mathbf{CAT}$.

1. We define a functor $\mathbf{fib}(\square): \mathbf{FLAT}(\square) \rightarrow \mathbf{I}$ by projecting objects and morphisms to their \mathbf{I} -components.
2. This functor is faithful, defining $\mathbf{FLAT}(\square)$ as a concrete category over \mathbf{I} .
3. This functor is a split fibration: the cartesian morphism associated with $\square: i \rightarrow j$ and $\langle j, b \rangle$ is $\langle \square, id_{\square(\square)(b)} \rangle: \langle i, \square(\square)(b) \rangle \rightarrow \langle j, b \rangle$. That is to say, we simply use the translation mechanism of the indexed category to perform the required lift.



PROOF

We leave the proof of the first two properties to the reader.

3. We have to prove that the proposed cartesian morphisms satisfy the required properties.
 - First of all, it is easy to see that the morphism is well defined: $\square(\square)(b)$ is indeed indexed by i . Moreover, its projection over \mathbf{I} is \square as required.
 - Consider now the universal property. Let $\langle \square, g \rangle: \langle k, a \rangle \rightarrow \langle j, b \rangle$ and $\square: k \rightarrow i$ satisfy $\square = \square; \square$. Because, by definition, $g: a \rightarrow \square(\square)(b)$ and $\square(\square)(b) = \square(\square; \square)(b) = \square(\square)(\square(\square)(b))$, we obtain $\langle \square, g \rangle: \langle k, a \rangle \rightarrow \langle i, \square(\square)(b) \rangle$. On the one hand, this morphism is projected to \square as required. On the other hand, $\langle \square, g \rangle = \langle \square, g \rangle; \langle \square, id_{\square(\square)(b)} \rangle$ as required. The uniqueness property results from the fact that the second component of the cartesian morphism is an identity and the projection over \mathbf{I} is faithful.

The fact that the fibration is split is trivially checked.

This result allows us to derive properties of flattened indexed categories as corollaries of the general results that we proved about fibrations. A useful result is the following:

6.4.8 PROPOSITION

Consider an indexed category $\square^{I^P} \mathbf{CAT}$. If \mathbf{I} is complete, $\square(i)$ is complete for every $i: \mathbf{I}$, and every translation functor $\square(\square)$ preserves limits, then $\mathbf{FLAT}(\square)$ is complete.

Examples of indexed-categories, their properties, and some of their uses in Computing are given in the next section.

6.5 Institutions

I am proud to say that I grew up into Category Theory when trying to learn the concept of Institution put forward by Goguen and Burstall in the early 80s! However, and although the fascination is still there, the reasons for including this section in this book are more than sentimental. The Theory of Institutions is a brilliant piece of "engineering mathematics", both in form and content. It is the result of a process of abstraction that has allowed computer scientists to classify and relate the specification formalisms that they have been developing or working with, as well as construct new formalisms in a systematic and integrated way. That is to say, it is a piece of Science, one of the few that Computing can boast.

Like every abstraction, it has its limitations. But these limitations have inspired many other computer scientists to develop ramifications that have further enlightened the path of those of us who are in the business of systematising and supporting the process of software development. Hence, our own account of the basics of institutions is left here both for exemplifying some of the constructions that we developed so far in the book, and as a source of inspiration for the reader.

Some readers, and hopefully users of this book, will complain that this topic comes too late: for instance, it does not require the material that we introduced so far in this section. While this is clearly true, the reason is; like for all the other examples, that we deliberately placed Institutions where we think they "belong". The point, once again, is that the purpose of this section is not to promote institutions but, rather, illustrate the constructions that we have been defining with the best examples we know. The community is still longing for a book on Institutions and this section is no substitute for it.

We start by defining not the original notion of institution as it appears in [61], but a variant that we developed in [45] called π -institutions. This is not for self-promotion but because it will allow us to capitalise on some material already introduced in previous sections. We defined in section 3.6 the notion of closure system as consisting of a pair $\langle L, c \rangle$ where L is a set and $c: L \rightarrow L$ is a total function satisfying the properties of a closure operator – reflexivity, idempotence and monotonicity. The idea is that the elements of L are the well-formed formulae that result from a specific choice of vocabulary symbols and a given grammar. The closure operator captures the notion of consequence of the logic at stake, instantiated to that particular language.

The idea of (π -)institutions is to add to this abstract view of Logic the notion that the notion of consequence is independent of the specific choice of vocabulary symbols that determine the language L . More precisely, it makes explicit the distinction between logical and non-logical symbols in a language: whereas the former (usually called the *connectives* of the logic) are captured by the grammar, the latter are given by (typed) sets. The notion of consequence is then constrained to be invariant under changes of non-logical symbols in order to make it depend only on the connectives. That is to say, we group together several of these closure systems that we regard as

being different instantiations of the same logic through different choices of non-logical symbols, and take that set as defining that logic.

Basically, this is what we did in section 3.5 when defining linear temporal logic: we introduced the notion of signature as a means of modelling the instantiations of the general structures of temporal logic, and made explicit the grammar that defines the connectives. For every specific signature, we obtain a closure system whose language is determined by the application of the grammar of temporal logic to the signature, and whose closure operator is obtained through a notion of consequence derived from the traditional Kripke semantics of temporal logic. The idea that all these closure systems are related in a way that makes them instantiations of the same logic can be captured by the following property:

6.5.1 PROPOSITION

The mapping **prop** defined in 3.5.2 and the translations induced by signature morphisms as defined in 3.5.7 extend to a functor **ltl:SET** \rightarrow **CLOS**.

PROOF

The crux of the proof is in showing that signature morphisms map to morphisms of closure systems. But this is a direct consequence of the presentation lemma 3.5.10. The reader is invited to fill-in the details.

Through the functor that maps closure systems to their underlying languages, **ltl** maps each signature to the set of well-formed formulae over that signature. The fact that we have a functor means that changes of vocabulary induce corresponding translations at the level of the languages, thus capturing the idea of "uniformity" that we tend to associate with a grammar. The same applies to the closure operator. The fact that changes of vocabulary induce morphisms between the corresponding closure systems captures the flavour of uniformity and continuity that we associate with a logic. Naturally, we can make these "flavours" concrete by making the grammar explicit rather than implicit. It all depends on the level of abstraction at which one wants to work. The reader interested in these aspects should consult [62].

6.5.2 DEFINITION – π -institution

A π -institution consists of a pair $\langle \mathbf{SIGN}, \mathbf{clos} \rangle$ where **SIGN** is a category (of signatures) and **clos:SIGN** \rightarrow **CLOS** is a functor.

6.5.3 REMARK

An equivalent definition of a π -institution, and the one given in [45], consists of

- a category **SIGN**;
- a functor **gram:SIGN** \rightarrow **SET**;
- for every $\square : \mathbf{SIGN}$, a relation $\vdash_{\square} : 2^{\mathbf{gram}(\square)} \rightarrow \mathbf{gram}(\square)$ satisfying the following properties:
 - for every $p \in \mathbf{gram}(\square)$, $p \vdash_{\square} p$;
 - for every $p \in \mathbf{gram}(\square)$ and $\square_1, \square_2 \in \mathbf{SIGN}$, if $\square_1 \sqsubseteq \square_2$ and $\square_1 \vdash_{\square_1} p$ then $\square_2 \vdash_{\square_2} p$;
 - for every $p \in \mathbf{gram}(\square)$ and $\square_1, \square_2 \in \mathbf{SIGN}$, if $\square_1 \vdash_{\square_1} p$ and $\square_2 \vdash_{\square_2} p'$ for every $p' \sqsubseteq \square_1$, then $\square_2 \vdash_{\square_2} p$;

- for every morphism $\square:\square\square\square'$, $p\square\mathbf{gram}(\square)$ and $\square\square\mathbf{gram}(\square)$, $\square \vdash_{\square} p$ implies $\mathbf{gram}(\square)(\square) \vdash_{\square} \mathbf{gram}(\square)(p)$.

Notice that the functor **gram** is the composition of **clos** with the forgetful functor that maps closure systems to the underlying languages. The closure operator itself is derived from the consequence relation as follows: for every $\square\square\mathbf{gram}(\square)$, $c_{\square}(\square) = \{p\square\mathbf{gram}(\square) : \square \vdash_{\square} p\}$. On the other hand, every closure operator defines a consequence relation: $\square \vdash_{\square} p$ iff $p\square c_{\square}(\square)$.

Actually, the definition given in [45] further requires the consequence relation to be compact, a property that we will not need for the constructions that we are going to present.

6.5.4 REMARKS

1. Given the equivalence between the two definitions, we shall often use the consequence relation and the grammar function without notifying the reader.
2. Because the notation can become quite cumbersome, we shall often omit the reference to the grammar functor when applied to morphisms and write $\square(p)$ instead of $\mathbf{gram}(\square)(p)$.

When defining a π -institution, the consequence relation or the closure operator can be presented in many different ways. One of the possible ways is the one we adopted in the definition of linear temporal logic: by providing a notion of model and a satisfaction relation. This is what institutions, as defined by Goguen and Burstall [61], consist of.

6.5.5 DEFINITION – institution

An *institution* is a quadruple $\langle \mathbf{SIGN}, \mathbf{gram}, \mathbf{mod}, \models \rangle$ where

- **SIGN** is a category;
- $\mathbf{gram}:\mathbf{SIGN} \rightarrow \mathbf{SET}$ is a functor;
- $\mathbf{mod}:\mathbf{SIGN}^{\text{op}} \rightarrow \mathbf{CAT}$ is a functor;
- for every $\square:\mathbf{SIGN}$, $\models_{\square}:\mathbf{mod}(\square)\mathbf{gram}(\square)$ satisfies for every morphism $\square:\square\square\square'$, $p\square\mathbf{gram}(\square)$ and $M'\square\mathbf{mod}(\square')$, $\mathbf{mod}(\square)(M') \models_{\square} p$ iff $M' \models_{\square} \mathbf{gram}(\square)(p)$.

The functor **mod** provides, for every signature, the category of models that can be used to interpret the language defined by **gram** over that signature. Signature morphisms induce translations between these classes of models in the opposite direction. The idea is that, as seen in 3.5.9 for linear temporal logic, sentences in the language of the source signature can be interpreted in a model for the target signature through the interpretation of their translations. The condition that is required on the satisfaction relation, which is usually called the *satisfaction condition*, states precisely this property – that satisfaction is invariant under change of notation.

When reasoning about institutions, we normally use the simplified notation for the **gram** functor that we mentioned for π -institution. We also adopt a similar simplification for the **mod** functor: for every morphism $\square:\square\square\square'$ and $M'\square\mathbf{mod}(\square')$, we normally write $M'|_{\square}$ instead of $\mathbf{mod}(\square)(M')$. For every signature \square , $\square\square\mathbf{gram}(\square)$, and $M\square\mathbf{mod}(\square)$, we usually write $M \models_{\square} p$ meaning that $M \models_{\square} p$ for every $p\square\square$.

6.5.6 PROPOSITION – π -institution presented by an institution

An institution $\langle \mathbf{SIGN}, \mathbf{gram}, \mathbf{mod}, \models \rangle$ presents the π -institution $\langle \mathbf{SIGN}, \mathbf{gram}, \vdash \rangle$ where, for every signature Σ , $p \in \mathbf{gram}(\Sigma)$ and $M \in \mathbf{mod}(\Sigma)$, $\Sigma \vdash p$ iff, for every $M \in \mathbf{mod}(\Sigma)$, $M \models_\Sigma p$ implies $M \models_\Sigma p$.

PROOF

The proof that a π -institution is obtained in this way offers no difficulties and is left as an exercise.

Another way of presenting the closure operator or consequence relation of a π -institution is through the notion of proof, choosing an inference system for the logic. See [87] for such proof-theoretic presentations and their integration in a more comprehensive categorical account of logical systems that the author calls "General Logics".

6.5.7 REMARK – models defined via a split (co)fibration

The model functor of an institution is defined in 6.5.5 directly as an indexed category, signatures providing the indexes. As seen in 6.1.20, indexed categories can be presented as split (co)fibrations, which is how, in many cases, the model-theory of some institutions is more naturally defined. That is to say, it is often more intuitive, not to say "practical", to provide directly either a split fibration **sign**: $\mathbf{MODL} \rightarrow \mathbf{SIGN}$ or a split cofibration **sign**: $\mathbf{MODL} \rightarrow \mathbf{SIGN}^p$ for some category **MODL** of models and take **ind(sign)** as the model functor of the required institution. This happens when we already have a notion of model that corresponds to the domain of interpretation intended for a given specification formalism, and we want to use it as the "semantics functor" of the institution.

An example can be given through linear temporal logic.

6.5.8 EXAMPLE – linear temporal logic over PROC

Temporal signatures were interpreted, in 3.5.3, over infinite sequences of sets of atomic propositions corresponding to synchronisation sets of actions. Yet, from the point of view of modelling concurrent systems, such infinite sequences do not represent abstractions of full behaviour; they usually represent one single behaviour among many that may also be observed on a given system. Hence, from the point of view of providing a domain of interpretation for a formalism supporting concurrent system specification, sets of such trajectories are more meaningful in the sense that they can be taken to represent *full* behaviours. Put in another way, if we take single sequences as models of full behaviour, we are restricting the expressive power of the formalism to a much smaller class of systems: those that are deterministic and run in a deterministic environment.

We have already seen in section 6.3 how such a trace-based model of process behaviour can be organised in a category **PROC** defined as being **spa(tra)** where **tra**: $\mathbf{SET}_\square \rightarrow \mathbf{SET}$ is the functor defined by **tra**(A_\square) = $\{\square : \square \sqsubseteq A\}$ on pointed sets and by **tra**($f : A_\square \rightarrow B_\square$)(\square) = $\square ; f$ on morphisms of pointed sets. Hence, it seems intuitive that we investigate the use of this category for providing the models of an institution of linear temporal logic. However, the split cofibration defined by **spa(tra)** is over \mathbf{SET}_\square , not over \mathbf{SET}^p as required for the temporal signatures. Indeed, processes were defined

over arbitrary alphabets of actions whereas, in the case of temporal logic, propositions are interpreted over traces of synchronisation sets. Hence, some adaptation is required. There are two equivalent ways in which this adaptation can be performed.

The first approach is to work with the subcategory of **spa(tra)** that involves only alphabets of synchronisation sets. We have seen that powersets can be regarded as pointed sets, the empty set providing the distinguished element. More precisely, we saw in 3.3.2 that, by choosing the morphisms $2^B \rightarrow 2^A$ to be the inverses of the functions $A \rightarrow B$, we define a subcategory of **SET** that we called **POWER**. Therefore, we can particularise the **spa** construction that yields processes to the functor **ptr**: **POWER** \rightarrow **SET** that yields the set of infinite traces taken over sets of actions. Finally, there is a straightforward (and, as we shall see in the next chapter, powerful) way of relating **SET**^{op} to **POWER**: through the contravariant functor 2^- that maps every set A to its powerset 2^A and every function $A \rightarrow B$ to the map $2^B \rightarrow 2^A$ that computes inverse images according to that function. Hence, we can particularise **tra** to **ftra**: **SET**^{op} \rightarrow **SET** = 2^- ; **tra** and use the split cofibration that **spa(ftra)** defines over **SET**^{op} to define, as detailed in 6.1.20(2), the model functor **ind(spa(ftra)): SET**^{op} \rightarrow **CAT** of an institution of linear temporal logic: the one that associates every temporal signature (set of actions) with the category of processes whose alphabet consists of the subsets of the signature (synchronisation sets of actions).

An alternative approach is to restrict not the (co)fibration that generates the indexed category but the indexed category generated by the original (co)fibration, i.e. to work with the composition 2^- ; **ind(spa(tra))**. This is simpler because it does not require the definition of a new (co)fibration. For the same reasons, this approach is less "satisfying" because it is less "structural" in the sense that it does not characterise explicitly the class of models that are chosen. In the case of linear temporal logic, both approaches lead to the same indexed category. Indeed, taking the indexed category **SET**_{ind} \rightarrow **CAT** generated by **spa(tra)**, we can compose it with the contravariant powerset functor **SET**^{op} \rightarrow **SET** to define the required model functor **SET**^{op} \rightarrow **CAT**.

It remains to define the satisfaction relation of this institution. Given that models over a signature Σ consist of sets of traces $\Sigma^*(2^\Sigma)^\omega$, satisfaction of a temporal proposition by a process is defined as satisfaction by all the traces of the process as defined in 3.5.3. The satisfaction condition results from the properties proved in 3.5.9. Indeed, the reader should check that the notion of reduct defined therein coincides with the functor between fibres defined by the cofibration (6.1.16)

It is easy to see that the institution defined over this indexed category yields the same π -institution as the one that can be defined directly by applying definition 6.5.2 to the functor defined in 6.5.1 (which is based on a semantic interpretation over single sequences as given in 3.5.3). This is because the **PROC**-based institution and the one that uses single trajectories as in 3.5.3 give rise to the same consequence relation. However, we shall see that they satisfy different structural properties, which shows that, "forgetting" the model-theory to retain just the consequence relations, π -institutions are, indeed, more abstract than institutions..

6.5.9 REMARK – when a split (co)fibration is not available

The availability of a split cofibration for the definition of the model functor is very useful because it allows us to work with reduct functors that operate on the chosen semantic models and, therefore, compare the structures that models and theories provide for specification. However, it may happen that we have a notion of semantic

model in mind that can still be captured by a functor **sign**: **MODL** \rightarrow **SIGN**^{op} but one that is not a cofibration. The problem here is the definition of a suitable translation between models.

This problem can be solved by working not directly over the fibres defined by the functor but with what we could call "generalised" models: those that are provided by structured morphisms as defined in 6.2.2. That is to say, the idea is, for every signature Σ , to work with models of the form $\langle \Sigma: \Sigma \text{ sign}(M), M \rangle$ so that **mod**(Σ) is the category $\Sigma \text{ sign}$, making **mod** be the functor $_ \text{sign}: \mathbf{SIGN}^{\text{op}} \rightarrow \mathbf{CAT}$ defined on morphisms $\Sigma: \Sigma \rightarrow \Sigma'$ by the functor $\Sigma' \text{ sign} \rightarrow \Sigma \text{ sign}$ that maps $\langle \Sigma': \Sigma' \text{ sign}(M'), M' \rangle$ to $\langle \Sigma: \Sigma \text{ sign}(M'), M' \rangle$. (The reader is encouraged to fill-in the way the functor works on morphisms between models.) In a way, what this construction does is compensate for the lack of (co)cartesian morphisms by providing an explicit "adaptor" that maps the signature to the language of the model. As a result, instead of translating directly between models, reducts operate "syntactically" at the level of these adaptors.

The reader may be wondering whether this is not just a contrived way of bringing structured morphisms into the discussion and a good example of "abstract nonsense"... Well, it turns out that these generalised models are quite common in Modal Logic (e.g. [66])! Models for a modal logic, normally called frames or *Kripke structures*, can be regarded as concrete categories over a base category of possible worlds. The adaptors that we defined correspond to interpretation functions that, for each atomic proposition, return the set of possible worlds in which the proposition is true. A generalised model, consisting of a Kripke frame and an interpretation function, is what in Modal Logic is usually called a model.

In the case of linear temporal logic, this means that interpretation structures are no longer traces $\Sigma \text{ sign}(2^\Sigma)$ as defined in 3.5.3 – where we mentioned that they are *canonical* Kripke structures for linear, discrete, propositional logic [113] – but triples $\langle W, \Sigma \text{ sign} W, \nu: \Sigma \rightarrow 2^W \rangle$ that correspond to the more traditional notion of models: W is a set of possible worlds (events), the pair $\langle W, \Sigma \text{ sign} W \rangle$ defines a linear frame (Kripke structure), and the function ν returns the set of worlds (events) in which each atomic proposition (action) holds (occurs).

In the case that interests us, **MODL** is the original **spa(tra)** but **sign** is not the original cofibration but its composition with the powerset functor **SET**₀ \rightarrow **SET**^{op} (in fact, as we shall see in 7.3.6, the left adjoint of the powerset functor used in 6.5.8). This means that reducts are not operated by the cofibration and, hence, we loose some of the structural properties of processes. More details on these constructions can be found in [32,33].

In Logic, models are considered to be the "duals" of theories. In institutions, this duality can be explored and made explicit in several ways. We can start by realising that the theories of the underlying closure systems can be organised in a category that provides itself a split (co)fibration over the category of signatures. Indeed, we have already proved in 6.1 that the category **THEO**_{LTL} of linear temporal theories, as defined in 3.5.4, is both a split fibration and a split cofibration over **SET**. The extension to any π -institution is trivial:

6.5.10 PROPOSITION – the (co)indexed-category of theories

1. Every π -institution $\langle \mathbf{SIGN}, \mathbf{clos} \rangle$ defines an extension **theo**: $\mathbf{SIGN} \rightarrow \mathbf{CAT}$ of **clos** by mapping every signature Σ to the category $\mathbf{THEO}_{\mathbf{clos}(\Sigma)}$ of the theories over the closure system $\mathbf{clos}(\Sigma)$, and every signature morphism $\Sigma \rightarrow \Sigma'$ to the functor $\mathbf{theo}(\Sigma): \mathbf{THEO}_{\mathbf{clos}(\Sigma)} \rightarrow \mathbf{THEO}_{\mathbf{clos}(\Sigma')}$ defined by $\mathbf{theo}(\Sigma)(\Gamma) = c'(\mathbf{clos}(\Sigma)(\Gamma))$.
2. Every π -institution $\langle \mathbf{SIGN}, \mathbf{clos} \rangle$ defines a functor $\mathbf{theo}^{-1}: \mathbf{SIGN}^{\text{op}} \rightarrow \mathbf{CAT}$ by mapping every signature Σ to the category $\mathbf{THEO}_{\mathbf{clos}(\Sigma)}$ of the theories over the closure system $\mathbf{clos}(\Sigma)$, and every signature morphism $\Sigma \rightarrow \Sigma'$ to the functor $\mathbf{theo}^{-1}(\Sigma): \mathbf{THEO}_{\mathbf{clos}(\Sigma)} \rightarrow \mathbf{THEO}_{\mathbf{clos}(\Sigma')}$ defined by $\mathbf{theo}^{-1}(\Sigma)(\Gamma) = \mathbf{clos}(\Sigma)^{-1}(\Gamma)$.

PROOF

Notice that, because the category of theories over a closure system is, in fact, a pre-order, the functors **theo**(Σ) and **theo**⁻¹(Σ) do not have to be defined on morphisms. However, we are required to prove:

1. Given theories $\Gamma \leq \Gamma'$ in $\mathbf{THEO}_{\mathbf{clos}(\Sigma)}$, $\mathbf{theo}(\Sigma)(\Gamma) \leq \mathbf{theo}(\Sigma)(\Gamma')$. This holds because **clos** is a functor. More precisely, given $\Gamma \leq \Gamma'$ in $\mathbf{THEO}_{\mathbf{clos}(\Sigma)}$, we have $\mathbf{clos}(\Sigma)(\Gamma) \leq \mathbf{clos}(\Sigma)(\Gamma')$ because **clos**(Σ) is a normal function between the languages of the two closure systems, which implies $c'(\mathbf{clos}(\Sigma)(\Gamma)) \leq c'(\mathbf{clos}(\Sigma)(\Gamma'))$ because closure operators are monotonic with respect to set inclusion.
2. Given theories $\Gamma \leq \Gamma'$ in $\mathbf{THEO}_{\mathbf{clos}(\Sigma)}$, $\mathbf{theo}^{-1}(\Sigma)(\Gamma) \leq \mathbf{theo}^{-1}(\Sigma)(\Gamma')$. Given $\Gamma \leq \Gamma'$ in $\mathbf{THEO}_{\mathbf{clos}(\Sigma)}$, we have $\mathbf{clos}(\Sigma)^{-1}(\Gamma) \leq \mathbf{clos}(\Sigma)^{-1}(\Gamma')$ because **clos**(Σ) is a normal function between the languages of the two closure systems and the inverse image of a closed set is itself closed.

The reader is invited to complete the proof as an exercise.

The (co)flattening of **theo** as a (co)indexed-category provides us with a split (co)fibration $\mathbf{THEO}_{\langle \mathbf{SIGN}, \mathbf{clos} \rangle} \rightarrow \mathbf{SIGN}$ where $\mathbf{THEO}_{\langle \mathbf{SIGN}, \mathbf{clos} \rangle}$, the flattened category, consists of pairs $\langle \Sigma, \Gamma \rangle$ where Σ is a signature and Γ is a closed set of sentences of **gram**(Σ), i.e. Γ is a theory of **clos**(Σ). A morphism $\Sigma \rightarrow \Sigma'$ is a signature morphism $\Sigma \rightarrow \Sigma'$ such that $c_{\Sigma'}(\mathbf{clos}(\Sigma)(\Gamma)) \leq \Gamma'$. This is the way theories have been originally defined in (π)-institutions. Notice that it is indifferent to flatten the indexed or the co-indexed category: the categories obtained are dual. This is because $c_{\Sigma'}(\mathbf{clos}(\Sigma)(\Gamma)) \leq \Gamma'$ holds iff $\mathbf{clos}(\Sigma')^{-1}(\Gamma') \leq \mathbf{clos}(\Sigma)^{-1}(\Gamma)$.

This construction can be generalised to presentations and strict presentations:

6.5.11 DEFINITION – theories and presentations in a π -institution

Consider a π -institution $\langle \mathbf{SIGN}, \mathbf{clos} \rangle$. We define the following categories:

1. **THEO** _{$\langle \mathbf{SIGN}, \mathbf{clos} \rangle$} , the category of theories, has for objects the pairs $\langle \Sigma, \Gamma \rangle$ where Σ is a signature and Γ is a closed set of sentences of **gram**(Σ). A morphism $\Sigma \rightarrow \Sigma'$ is a signature morphism $\Sigma \rightarrow \Sigma'$ such that $c_{\Sigma'}(\mathbf{clos}(\Sigma)(\Gamma)) \leq \Gamma'$.
2. **PRES** _{$\langle \mathbf{SIGN}, \mathbf{clos} \rangle$} , the category of theory presentations, has for objects the pairs $\langle \Sigma, \Gamma \rangle$ where Σ is a signature and Γ is a presentation of **gram**(Σ). A morphism $\Sigma \rightarrow \Sigma'$ is a signature morphism $\Sigma \rightarrow \Sigma'$ such that $\mathbf{clos}(\Sigma')^{-1}(\Gamma') \leq \mathbf{clos}(\Sigma)^{-1}(\Gamma)$.

3. **SPRES**_{<SIGN,clos>}, the category of strict theory presentations, has for objects the pairs $\langle \Sigma, \Gamma \rangle$ where Σ is a signature and $\Gamma \models \mathbf{gram}(\Sigma)$. A morphism $\Sigma: \langle \Sigma, \Gamma \rangle \rightarrow \langle \Sigma', \Gamma' \rangle$ is a signature morphism $\Sigma: \Sigma \rightarrow \Sigma'$ such that $\Gamma(\Sigma) \models \Gamma'$.

These categories satisfy the properties that we have already proved for their linear temporal logic instantiations:

6.5.12 PROPOSITION

Consider a π -institution $\langle \mathbf{SIGN}, \mathbf{clos} \rangle$.

1. The categories **PRES**_{<SIGN,clos>}, **SPRES**_{<SIGN,clos>} and **THEO**_{<SIGN,clos>} define split fibrations and split cofibrations through the functor **sign** that projects their objects and morphisms to the corresponding signature components. Given a signature morphism $\Sigma: \Sigma \rightarrow \Sigma'$,

	cartesian morphism for $\langle \Sigma', \Gamma' \rangle$	cocartesian morphism for $\langle \Sigma, \Gamma \rangle$
PRES	$\Sigma: \langle \Sigma, \Sigma^{-1}(c(\Gamma)) \rangle \rightarrow \langle \Sigma', \Gamma' \rangle$	$\Sigma: \langle \Sigma, \Gamma \rangle \rightarrow \langle \Sigma', \Sigma(\Gamma) \rangle$
SPRES	$\Sigma: \langle \Sigma, \Sigma^{-1}(\Gamma) \rangle \rightarrow \langle \Sigma', \Gamma' \rangle$	$\Sigma: \langle \Sigma, \Gamma \rangle \rightarrow \langle \Sigma', \Sigma(\Gamma) \rangle$
THEO	$\Sigma: \langle \Sigma, \Sigma^{-1}(\Gamma) \rangle \rightarrow \langle \Sigma', \Gamma' \rangle$	$\Sigma: \langle \Sigma, \Gamma \rangle \rightarrow \langle \Sigma', c(\Sigma(\Gamma)) \rangle$

2. All these (co)fibrations are fibre-(co)complete. The following procedure calculates limits and colimits of a diagram Γ with $\Gamma = \langle \Sigma_i, \Gamma_i \rangle$ in these categories:
- Calculate the limit $\Sigma: \Sigma \rightarrow \Sigma$ or colimit $\Sigma: \Sigma \rightarrow \Sigma$ of the underlying diagram of signatures.
 - Lift the result by computing the **SIGN**-component according to the following rules:

	limit	colimit
PRES	$\Sigma: \langle \Sigma, \prod_{i \in I} \Sigma_i^{-1}(c(\Gamma_i)) \rangle \rightarrow \Sigma$	$\Sigma: \Sigma \rightarrow \langle \Sigma, \prod_{i \in I} \Sigma_i(\Gamma_i) \rangle$
SPRES	$\Sigma: \langle \Sigma, \prod_{i \in I} \Sigma_i^{-1}(\Gamma_i) \rangle \rightarrow \Sigma$	$\Sigma: \Sigma \rightarrow \langle \Sigma, \prod_{i \in I} \Sigma_i(\Gamma_i) \rangle$
THEO	$\Sigma: \langle \Sigma, \prod_{i \in I} \Sigma_i^{-1}(\Gamma_i) \rangle \rightarrow \Sigma$	$\Sigma: \Sigma \rightarrow \langle \Sigma, c(\prod_{i \in I} \Sigma_i(\Gamma_i)) \rangle$

3. If the category **SIGN** is (co)complete, so are **PRES**_{<SIGN,clos>}, **SPRES**_{<SIGN,clos>} and **THEO**_{<SIGN,clos>}.

This symmetry between theories and models, as captured through indexed-categories, can be used to provide every π -institution with a canonical notion of model that makes it an institution:

6.5.13 PROPOSITION – institution presented by a π -institution

Every π -institution $\langle \mathbf{SIGN}, \mathbf{gram}, \vdash \rangle$ defines the institution $\langle \mathbf{SIGN}, \mathbf{gram}, \mathbf{theo}^{-1}, \models \rangle$ where, for every signature Σ , $p \models \mathbf{gram}(\Sigma)$ and $\Sigma \models \mathbf{theo}^{-1}(\Gamma)$, $\Sigma \models p$ iff $p \models \Sigma$.

PROOF

We just have to prove that the satisfaction condition holds. Let $\square: \mathbf{SIGN}, \models_{\square}: \mathbf{mod}(\square) \rightarrow \mathbf{gram}(\square)$ satisfies for every morphism $\square: \square \rightarrow \square'$, $p \in \mathbf{gram}(\square)$ and $\square \models_{\square} \mathbf{theo}^{-1}(\square')$, $\mathbf{theo}^{-1}(\square)(\square') \models_{\square} p$ iff $\square^{-1}(\square') \models_{\square} p$ iff $p \in \square^{-1}(\square')$ iff $\square(p) \in \square'$ iff $\square' \models_{\square} \square(p)$.

What this results says is that, whereas having a model-theory that corresponds to an intended domain of interpretation for the language of a logic may be a convenient way of defining a π -institution, once we are given a π -institution there is no much point in searching for a model-theory for it: its theories are up to the job at no conceptual expense.

Notice that the consequence relation induced by this notion of satisfaction is $\square \vdash_{\square} p$ iff, for every $\square \models_{\square} \mathbf{theo}^{-1}(\square)$, $\square \models_{\square} p$ implies $p \in \square$, which is equivalent to $p \in c(\square)$. That is to say, we recover the π -institution from which we started. Hence, institutions do give us a more concrete level of abstraction at which properties of specifications can be discussed.

Duality between models and theories can also be explored by lifting some of the signature-based constructions to theory-based ones:

6.5.14 PROPOSITION

Let $\langle \mathbf{SIGN}, \mathbf{gram}, \mathbf{mod}, \models \rangle$ be an institution.

1. The model functor $\mathbf{mod}: \mathbf{SIGN}^p \rightarrow \mathbf{CAT}$ can be extended to $\mathbf{tmod}: \mathbf{THEO}^p \rightarrow \mathbf{CAT}$ by assigning to every theory $\langle \square, \square' \rangle$ the full subcategory of $\mathbf{mod}(\square)$ that consists of the models that satisfy \square .
2. When the model functor is generated by a cofibration $\mathbf{sign}: \mathbf{MODL} \rightarrow \mathbf{SIGN}^p$, we can lift it to $\mathbf{theo}: \mathbf{MODL} \rightarrow \mathbf{THEO}^p$ by associating with every model the theory that consists of all the sentences that are true for that model.

PROOF

We do have to prove that a functor is, indeed, defined in 1. This is because, given a theory morphism $\square: \langle \square, \square' \rangle \rightarrow \langle \square', \square'' \rangle$ and a model M' of $\langle \square', \square'' \rangle$, $M'|_{\square}$ satisfies \square . The reader is invited to workout the proof of the second property.

6.5.15 REMARK – theories over generalised models

We saw that functors $\mathbf{sign}: \mathbf{MODL} \rightarrow \mathbf{SIGN}^p$ give rise to model functors of the form $_ \mathbf{sign}: \mathbf{SIGN}^p \rightarrow \mathbf{CAT}$ that are typical of Modal Logic. In an institution defined over such a model functor, the notions of theory and the constructions that we have presented around them for arbitrary (π -)institutions still apply. For instance, $_ \mathbf{sign}: \mathbf{SIGN}^p \rightarrow \mathbf{CAT}$ still extends to $\mathbf{tmod}: \mathbf{THEO}^p \rightarrow \mathbf{CAT}$ as defined in 6.5.14. However, in this particular case, it is interesting to check if, or when, $\mathbf{sign}: \mathbf{MODL} \rightarrow \mathbf{SIGN}^p$ can lift to $\mathbf{theo}: \mathbf{MODL} \rightarrow \mathbf{THEO}^p$ so that \mathbf{tmod} is, in fact, $_ \mathbf{theo}$.

The intuition for this interest is that we should be able to associate with every model M a canonical specification $\mathbf{theo}(M)$ so that every morphism $T \models \mathbf{theo}(M)$ identifies T as

a specification of M and, dually, M as a realisation of T . In this case, the models of T correspond to all possible refinements of T into "programs", something that we already discussed in 6.2.1. Identifying **MODL** with (the behaviours of) programs (or processes), this corresponds to the idea that programs can themselves be regarded as specifications, and that the models of a specification can be identified with the programs which, in some sense, "complete" the specification.

Although it is always possible to define a mapping that associates a theory $\mathbf{theo}(M)$ with every model M – the obvious candidate assigns to $\mathbf{theo}(M)$ the signature $\mathbf{sign}(M)$ and all the sentences that are true in it $\{p \mid \mathbf{gram}(\mathbf{sign}(M)) : \langle id, M \rangle \models p\}$ – it is not always possible to extend it to a functor that lifts **sign**, i.e. such that $\mathbf{theo}(h : M \rightarrow M') = \mathbf{sign}(h)$. In order to motivate why this is so, assume that we do have a functor $\mathbf{theo} : \mathbf{MODL} \rightarrow \mathbf{THEO}^p$ and consider a morphism $h : M \rightarrow M'$. Because \mathbf{theo} is a functor, we have $\mathbf{theo}(h) : \mathbf{theo}(M) \rightarrow \mathbf{theo}(M')$. Hence, if M is a model of a theory T , i.e. if we have a morphism $\square : T \rightarrow \mathbf{theo}(M)$, then M' is also a model of T through the morphism $\square ; \mathbf{theo}(h)$. In particular this implies that, for every signature \square and $p \mid \mathbf{gram}(\square)$, if $\langle \square, M \rangle \models p$ then $\langle \square ; \mathbf{sign}(h), M' \rangle \models p$. This property, however, is not universal: only some institutions (logics) satisfy it.

There are two important aspects about this property. The first is that, as proved in [33], it guarantees that $\mathbf{sign} : \mathbf{MODL} \rightarrow \mathbf{SIGN}^p$ lifts to $\mathbf{theo} : \mathbf{MODL} \rightarrow \mathbf{THEO}^p$ as defined, and that \mathbf{tmod} is, in fact, $_ \mathbf{theo}$. The second is that, once again, it is not an artificial "fabrication" (abstract nonsense): it captures what, in Modal Logic, is known as "the p-property" [66]. That is to say, there are well known principles of Modal Logic that can be captured in the categorical framework that we have defined, meaning that we are, indeed, addressing structural properties of Logic as we have known them. In other words, what we are presenting is not just a formal exercise in Category Theory, or what has become known as "abstract nonsense". Once again, a more detailed discussion of the p-property and its impact in specification can be found in [32,33].

6.5.16 EXERCISE

Check that the institution of linear temporal logic as defined in 6.5.9 over general Kripke structures satisfies the p-property.

The reader will have noticed that, in an institution, the model functor is defined over **CAT** instead of **SET**. Indeed, many times, models come equipped with a non-trivial notion of morphism and, as we have argued in 6.5.9, it is not always possible to reflect in the logic the structure that they induce on models.

However, so far, the constructions that we have discussed do not depend on the notion of morphism between models and, hence, do not reflect the structure of the interpretation domain that has been chosen for the specification formalisms. We are now going to illustrate a situation in which such structures are of interest.

6.5.17 DEFINITION – initial/terminal semantics

We say that an institution $\langle \mathbf{SIGN}, \mathbf{gram}, \mathbf{mod}, \models \rangle$ has *initial* (resp. *terminal*) *semantics* provided that, for every theory $\langle \square, \square \rangle$, the category $\mathbf{tmod}(\langle \square, \square \rangle)$ has an initial (resp. terminal) object.

A non-trivial (but simple) example of an institution with terminal semantics is linear temporal logic.

6.5.18 EXAMPLE – terminal semantics of linear temporal logic

The institution of linear temporal logic as defined in 6.5.8 over the cofibration **spaftra** has terminal semantics. This is because, given any theory $\langle \Sigma, \Gamma \rangle$, the category **tmod**($\langle \Sigma, \Gamma \rangle$) consists of sets of traces ordered by inclusion and is closed under intersection.

Notice that linear temporal logic interpreted over single sequences does not have terminal semantics: the (discrete) category that represents the set of all the sequences that satisfy a given theory does not have a sequence that represents the whole set unless, of course, the set is singular.

6.5.19 EXERCISE

Check that the institution of linear temporal logic as defined in 6.5.9 over general Kripke structures has terminal semantics. Does it coincide with the one obtained in 6.5.18?

Properties such as having an initial or terminal semantics can be used for differentiating between different institutions that present the same π -institution. It is interesting to note that, what we have called the "canonical" institution for a given π -institution, the one that uses theories as models, has both initial and terminal semantics: the initial model of every theory is itself and the terminal model is the inconsistent theory, i.e. the full language. That is to say, we do not extract much information from the structure of models, which was only to be expected because we do not extract any information from the model-theory itself.

This example can also be used to transmit a certain "moral" message, namely that the pursuit for an institution with "good" properties like the ones above is, most of the times, a trivial one in the sense that it is usually possible to engineer a model-theory that satisfies one's requirements. As already said, the real purpose of these properties is to measure the relationship that the institution provides between specifications and the domain as abstracted through the models. Hence, one needs to determine the nature of the models relative to the domain of interpretation and not to the properties. Category Theory has often been accused of being "Abstract Non-sense" because it makes it so much easier to "engineer" chimeras. But, like with any good engineering tool, the non-sense can only be found in the way the tool is used, not in the tool itself.

