

## 3 BUILDING CATEGORIES

There are many ways in which new categories can be built from existing ones. One of the advantages of building new categories by "inheriting" from old ones is that it is easier to prove that the construction yields, indeed, a category. In this chapter, we will present some elementary constructions that illustrate the point, together with examples of categories that relate to software development. In later chapters, will study some more elaborate constructions that we have found useful for our day-to-day.

### 3.1 Some elementary operations

There are a number of elementary operations for constructing categories. The first example reflects the fact that the choice for the direction of the morphisms has no essential significance in the sense that, if we reverse the direction of all the morphisms, we obtain a category that has the same structural properties of the original one.

This fact does not mean, however, that the direction of the morphisms should not be carefully chosen when defining a category. It is often the case that there is a "natural" direction for morphisms, either given by the mathematical nature of the morphisms or their use in practice. If reversed, the direction of the morphisms may make it more difficult or less immediate to understand the intended meaning and the particular constructions one may want to define on objects and morphisms. Hence, if you are going to become a "practising category theorist", be prepared to answer questions like "but aren't the arrows going the wrong way?" which, many times, just means "you do not seem to belong to my club!"...

#### 3.1.1 DEFINITION – dual of a category

For any category  $\mathbf{C}$  we can construct its *dual* or *opposite*  $\mathbf{C}^{op}$ :

- $\mathbf{C}^{op}$  has the same objects and arrows as  $\mathbf{C}$ ;
- arrows of  $\mathbf{C}^{op}$  go in the reverse direction: if  $f:A \rightrightarrows B$  in  $\mathbf{C}$  then  $f:B \rightrightarrows A$  in  $\mathbf{C}^{op}$
- arrows of  $\mathbf{C}^{op}$  compose in the reverse direction:  $f;g$  in  $\mathbf{C}^{op}$  is  $g;f$  in  $\mathbf{C}$

#### 3.1.2 EXAMPLE

Following [88], the dual of the category **ancestor**(G) can be named **descendant**(G).

The fact that every category and its opposite have the same structural properties reflects a general *duality principle* that applies to all definitions and results. Every con-

cept in Category Theory has a dual, which is the concept that is obtained by reversing the direction of the arrows, i.e. the concept that holds in the dual category. Every result in Category Theory has a dual which holds for the dual concepts. We shall have plenty of occasions to illustrate how the duality principle is applied in practice.

Another elementary construction consists in taking the product of two categories:

#### 3.1.3 DEFINITION/PROPOSITION – product category

Given categories  $\mathbf{C}$  and  $\mathbf{D}$ , their *product*  $\mathbf{C} \times \mathbf{D}$  is such that its objects are the pairs  $\langle c, d \rangle$  where  $c \in \mathbf{C}$  and  $d \in \mathbf{D}$ , and the morphisms  $\langle c, d \rangle \rightarrow \langle c', d' \rangle$  are all the pairs  $\langle f, g \rangle$  where  $f: c \rightarrow c'$  in  $\mathbf{C}$  and  $g: d \rightarrow d'$  in  $\mathbf{D}$ . Composition of morphisms is defined componentwise, and the identity for an object  $\langle c, d \rangle$  is the pair  $\langle id_c, id_d \rangle$ .

PROOF

The properties of composition and the identities are trivially inherited from the corresponding properties of the components.

There are also ways of extracting categories from other mathematical structures. For instance, we proved in 2.2.5 that every pre-order defines a category. Yet a simpler construction is the one that converts sets to categories:

#### 3.1.4 DEFINITION/PROPOSITION – discrete categories

Every set  $S$  determines a category whose objects are the elements of  $S$  and whose Hom-sets are all empty except for  $Hom(s, s) = \{id_s\}$  for every  $s \in S$ . Such categories are said to be *discrete*.

PROOF

Immediate. Note that, if we take  $S$  to be the set of nodes of a graph with no arrows, then this construction is a special case of 2.2.8.

The category defined by a set may have many objects, as many as the elements of the set, but has only the minimum number of morphisms – the identities. The next example illustrates the other extreme: as many morphisms as we want, but only one object... These categories correspond to monoids:

#### 3.1.5 DEFINITION/PROPOSITION – monoid as a category

A monoid is a triple  $\langle M, *, 1 \rangle$  where  $M$  is a set,  $*$  is an associative binary operation on  $M$ , and  $1$  is an identity for  $*$ , i.e.  $m * 1 = 1 * m = m$  for every  $m \in M$ . Any monoid defines a category that consists of only one object, which we can denote by  $\bullet$ , and whose morphisms  $\bullet \rightarrow \bullet$  are the elements of  $M$ . Composition of morphism is given by the operation  $*$  and the identity morphism is  $1$ .

PROOF

The properties of composition and the identities are trivially inherited from the corresponding properties of the monoid.

### 3.1.6 EXERCISE

Characterise the duals of the categories defined by monoids. Are these categories their own duals?

## 3.2 "Adding structure"

The most typical way of building a new category is, perhaps, by adding "structure" to the objects of a given category (or a subset thereof). The expression "adding structure" has, of course, a broad meaning that the reader will only fully apprehend after building a few categories... The morphisms of the new category are then the morphisms of the old category that "preserve" the additional structure.

The following example is as typical as any other and will be used throughout the book for applications related to systems modelling.

### 3.2.1 EXAMPLE – pointed sets

The category **SET**<sub>□</sub> (of pointed sets) is defined as follows. Its objects are the pairs  $\langle A, \square_A \rangle$  where  $A$  is a set and  $\square_A$  is an element of  $A$  called the designated element. The morphisms between two pointed sets  $\langle A, \square_A \rangle$  and  $\langle B, \square_B \rangle$  are the total functions  $f: A \rightarrow B$  such that  $f(\square_A) = \square_B$ .

The category over which **SET**<sub>□</sub> is being built is, obviously, **SET**, the category of sets and total functions. The additional structure that is being added to the objects of the base category, sets, is the designation of an element (not the element itself because it is already in the set). The morphisms of the new category are all the morphisms of the base category that preserve the additional structure, which in this case means mapping the designated element of the source to the designated element of the target.

This way of building new categories makes it easier to conduct proofs as illustrated next.

PROOF

The attentive reader will have noticed that the definition of **SET**<sub>□</sub> omitted two fundamental ingredients: the definition of the composition law and of the identity map. This is because they are, by default, inherited from the base category. Because the laws of composition and identity are satisfied in the base category, they do not need to be checked again for the new category. Hence, all that needs to be checked is that (1) the composition law is closed for **SET**<sub>□</sub>, i.e. that the composition of two functions that are morphisms in **SET**<sub>□</sub> is also a morphism of **SET**<sub>□</sub>, and (2) that the identities are, indeed, morphisms in **SET**<sub>□</sub>.

1. Given morphisms  $f: \langle A, \sqsubseteq_A \rangle \rightarrow \langle B, \sqsubseteq_B \rangle$  and  $g: \langle B, \sqsubseteq_B \rangle \rightarrow \langle C, \sqsubseteq_C \rangle$ , we have to check that  $(f;g)(\sqsubseteq_A) = \sqsubseteq_C$ . But  $f(\sqsubseteq_A) = \sqsubseteq_B$  because  $f$  is a morphism of pointed sets. Hence,  $(f;g)(\sqsubseteq_A) = g(f(\sqsubseteq_A)) = g(\sqsubseteq_B)$ . On the other hand, because  $g$  is also a morphism of pointed sets,  $g(\sqsubseteq_B) = \sqsubseteq_C$ .
2. Given a pointed set  $\langle A, \sqsubseteq_A \rangle$ , we now have to prove that the identity map  $id_A$  for  $A$  as a set is a morphism of pointed sets, i.e. is such that  $id_A(\sqsubseteq_A) = \sqsubseteq_A$ . But this is true because  $id_A$  is precisely the identity function on  $A$ .

Notice the similarities between the structure of this proof and the proof outline given for **AUTOM** (2.2.10). This style of proof is typical of categories built from other categories by "adding structure".

The particular use of pointed sets that we have in mind is for modelling the behaviour of objects at their interfaces (methods). A very abstract and general model of object behaviour is one in which the events that can potentially occur during the lifetime of an object consist of all possible subsets of the set of its methods. This means that we work in a model in which every object can potentially handle concurrent method calls. We shall see later on how we can model restrictions to the degree of concurrency that an object can impose on its methods. The empty set represents an event in which the object is not involved. This explicit, but abstract, representation of environment events has been shown to be important for modelling the behaviour of concurrent and distributed systems [13].

The fact that the events that we are using for modelling object behaviour consist of sets of method calls can be abstracted away to recognise a more general notion of process alphabet as used in Concurrency. The application of categorical techniques in Concurrency Theory is particularly rich and revealing of the power of Category Theory for systematising constructions and establishing relationships between different models. The application to object behaviour that we will use for illustration purposes throughout the book touches some of these aspects but the reader interested in a more complete picture of the breadth of the field should consult [21,24,98,109]. Specific applications of these categorical techniques to object-oriented modelling can be found in [21,24].

In this more general model of process behaviour, a process alphabet is a pointed set. Each element of the set represents an event whose occurrence may be witnessed during the lifetime of the process. The designated element of the set represents an event of the environment, i.e. an event in which the process is not involved. The notion of morphism in this model captures the relationship that exists between systems and components of systems. More precisely, every morphism identifies the way in which the target is embedded, as a component, in the source. The fact that the morphism is a function between the alphabets is pretty intuitive: for every event  $a$  occurring in the lifetime of the system, we have to know how the component participates in that event; if the component is not involved in  $a$ , then the morphism should map  $a$  to the designated element of the alphabet of the component, i.e. to the element that has been designated to represent the events in which the component does not participate. On the other hand, any event occurring in the environment without the participation of the system cannot involve the component either; hence, every morphism needs to preserve the designated element.

For instance, consider a producer-consumer system. We can assign the methods *produce* and *store* to the producer and *consume* and *retrieve* to the consumer. An event of the system during which both *produce* and *consume* take place concurrently is mapped to the singleton  $\{\text{produce}\}$  by the morphism  $\text{prod}: \text{system} \rightarrow \text{producer}$  that identifies the producer as a component of the system. On the other hand, an event

of the system during which the consumer executes *retrieve* and the producer remains idle is mapped by *prod* to  $\emptyset$ , i.e. to the designated element of the alphabet of the producer.

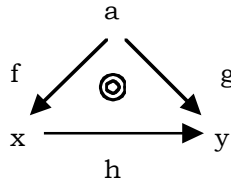
Our last example of constructing new categories from base ones is also very typical. It provides so-called *comma-categories*. Several different notations can be found in the literature for comma-categories. We shall use the same notation as [79]. The particular form of comma-categories that we are about to define are also called over/under-cone categories in [22] and (co)slice-categories in [48].

### 3.2.2 EXAMPLE – comma-categories

Given a category  $\mathbf{C}$  and an object  $a:\mathbf{C}$ , we define the category of objects under  $a$  –  $a\downarrow\mathbf{C}$  – as follows. Its objects are all the pairs  $\langle f, x \rangle$  where  $f$  is a morphism of the form  $f:a\downarrow x$  in  $\mathbf{C}$ . The morphisms between  $f:a\downarrow x$  and  $g:a\downarrow y$  are all the  $\mathbf{C}$ -morphisms  $h:x\rightarrow y$  such that  $f;h=g$ .

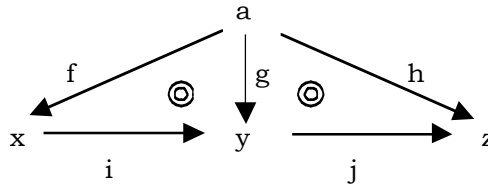
PROOF:

This is also another example of a construction of a new category by adding structure to a base one. The additional structure that is being added to every object  $x:\mathbf{C}$  is a  $\mathbf{C}$ -morphism  $f:a\downarrow x$ . Hence, again, we assume that the composition law and the identity map are inherited from the base category and prove only the closure properties. This is also a good example to illustrate how reasoning in Category Theory is often done at a graphical level. Because our objects now are morphisms of the base category, it is more convenient to represent a morphism  $h$  between  $f:a\downarrow x$  and  $g:a\downarrow y$  by the diagram:



As already mentioned, the symbol  $\odot$  is used to indicate that the diagram commutes, i.e. all compositions of morphisms in the diagram along paths that have the same source and target are equal. In the case above, commutativity of the diagram expresses the equation  $f;h=g$ .

- Given morphisms  $i$  and  $j$



we have to prove that the composition  $(i;j)$  in  $\mathbf{C}$  is a morphism in  $a\downarrow\mathbf{C}$ , i.e.  $f;(i;j)=h$ . There is another way of asserting the property that needs to be proved which makes appeal to the kind of "diagrammatic reasoning" that we have claimed to be typical of Category Theory: we have to prove that, for any

diagram of the form above, the outer triangle commutes if the inner triangles also commute. This can be done as follows:

- a.  $f; i = g$  because  $i$  is a morphism in  $a \square \mathbf{C}$
  - b.  $(f; i); j = g; j$  from (a)
  - c.  $g; j = h$  because  $j$  is a morphism in  $a \square \mathbf{C}$
  - d.  $(f; i); j = h$  from (b) and (c)
  - e.  $f; (i; j) = (f; i); j$  from the associativity of the composition law in  $\mathbf{C}$
  - f.  $f; (i; j) = h$  from (d) and (e)
- The fact that the identity morphism  $id_x$  is an identity for  $f: a \square x$  results from the property  $f; id_x = id_x$ .

For instance, when applied to a category of the form **ancestor**( $G$ ), this construction returns, for every class  $c$ , the classes that inherit from  $c$ , themselves organised as a hierarchy. When applied to the category **PROOF** it returns, for every sentence, the inferences that can be made from that sentence.

### 3.2.3 EXERCISE

Characterise the dual category of  $a \square \mathbf{C}$  – the category of objects over  $a$ , usually denoted by  $\mathbf{C} \square a$

Generalisations of the notion of comma category will be presented in section 6.1.

Together with example 2.2.10, we hope that we have provided enough insight into the way categories are often defined: by "adding structure" to the objects of another category. Because there is no abstract characterisation of this technique, we have relied on the examples to make apparent that there is a systematic procedure for checking the definition of a category built by "adding structure".

## 3.3 Subcategories

There is another intuitive, and often useful, way of building new categories from old: by forgetting some of the objects and some of the morphisms to create a **subcategory** of the original one. The only proof burden associated with this method is in making sure that we do not throw away too much (or too little...).

### 3.3.1 DEFINITION – subcategory

Given categories  $\mathbf{C} = \langle G_{\mathbf{C}}, \cdot_{\mathbf{C}}, id_{\mathbf{C}} \rangle$  and  $\mathbf{D} = \langle G_{\mathbf{D}}, \cdot_{\mathbf{D}}, id_{\mathbf{D}} \rangle$ , we say that  $\mathbf{D}$  is a *subcategory* of  $\mathbf{C}$  iff

- $|\mathbf{D}| \subseteq |\mathbf{C}|$ , i.e. every object of  $\mathbf{D}$  is an object of  $\mathbf{C}$
- For any objects  $x$  and  $y$  of  $\mathbf{D}$ ,  $Hom_{\mathbf{D}}(x, y) \subseteq Hom_{\mathbf{C}}(x, y)$ , i.e. the morphisms in  $\mathbf{D}$  are also morphisms in  $\mathbf{C}$
- The composition laws and the identity maps in the two categories agree, i.e. given composable morphisms  $f$  and  $g$  of  $\mathbf{D}$ , their composition  $f;_{\mathbf{D}} g$  in  $\mathbf{D}$  is the

same as their composition  $f \circ g$  in  $\mathbf{C}$ , and for every object  $x$  of  $\mathbf{D}$ , its identity  $id_{\mathbf{D}}(x)$  is the same as its identity  $id_{\mathbf{C}}(x)$  in  $\mathbf{C}$ .

Notice that, for  $\mathbf{D}$  to be a subcategory of  $\mathbf{C}$ , it is not enough to have inclusions between the sets of objects and of morphisms: the additional structure given by the identities and the composition law has to be preserved.

Examples of subcategories will be given as we go along, most of the time as exercises.

### 3.3.2 EXAMPLES

1. By keeping just the sets that are finite and all the total functions between them, we define a subcategory **fSET** of **SET**.
2. By keeping all sets but just the functions that are injective, we define a subcategory **INJ** of **SET**. This is because identities are injective and the composition of injective functions is still injective.
3. Given any category, we can forget all its morphisms except for the identities, and obtain a subcategory. Categories whose only morphisms are the identities are called *discrete*.
4. We can extend the morphisms of **SET** to include all partial functions between sets. Because the identity function is trivially partial and partial functions compose in the same way as total functions, we obtain a category of which **SET** is a subcategory. The category of partial functions will be called **PAR**.
5. During the discussion above around pointed sets, we have motivated the notion of alphabet of object behaviour by identifying events with sets of methods. That is, we have worked with pointed sets that consist of powersets, the empty set being the designated element of each powerset. We can show that we define a subcategory of **SET**<sub>0</sub> by choosing the morphisms  $2^B \rightarrow 2^A$  to compute inverse images for a given function  $A \rightarrow B$ , i.e.  $g: 2^B \rightarrow 2^A$  is such that, for some  $f: A \rightarrow B$ ,  $g(B') = f^{-1}(B')$  for every  $B' \subseteq B$ . We call this category **POWER**.

The cases in which we throw away some of the objects but keep all the morphisms between those that remain, like we did for finite sets, deserve a special designation:

### 3.3.3 DEFINITION – full subcategory

Given categories  $\mathbf{C}$  and  $\mathbf{D}$  we say that  $\mathbf{D}$  is a *full* subcategory of  $\mathbf{C}$  iff  $\mathbf{D}$  is a subcategory of  $\mathbf{C}$  and, for any objects  $x$  and  $y$  of  $\mathbf{D}$ ,  $Hom_{\mathbf{D}}(x, y) = Hom_{\mathbf{C}}(x, y)$ .

### 3.3.4 EXERCISE

Check which of the subcategories defined in 3.3.2 are full.

We have mentioned several times that, intensionally, a category captures a certain notion of structure, what we have called a "social life" for its objects. It is only natural to expect that, when selecting a subcategory, we obtain a different but, nevertheless related, notion of structure. For instance, if  $\mathbf{D}$  is a subcategory of  $\mathbf{C}$ , two ob-

jects that are isomorphic in  $\mathbf{C}$  are not necessarily isomorphic in  $\mathbf{D}$ . A trivial illustration of this fact can be obtained by realising that the discrete subcategory of  $\mathbf{C}$  that is obtained by forgetting all the morphisms except the identities (see 3.3.2) is such that any object is only isomorphic to itself (it has no social life...). Indeed, the more structure a category provides through its morphisms, the more observational power we have over its objects and, hence, the more isomorphisms we are able to establish. The reverse property, however, holds and is left as an exercise.

### 3.3.5 EXERCISE

Let  $\mathbf{D}$  be a subcategory of  $\mathbf{C}$ . Show that any two objects that are isomorphic in  $\mathbf{D}$  are necessarily isomorphic in  $\mathbf{C}$ .

The relationship between isomorphisms and subcategories is, in fact, very revealing of the way categories can be used to capture notions of structure. Recall that we classified a subcategory as being full iff it is obtained by selecting a subclass of the objects but leaving the morphisms between the selected objects unchanged. Typically, this happens when one wants to select just the objects that satisfy a certain property, e.g. sets that are finite as in example 3.3.2. In this case, the converse of the property proved in the exercise above also holds because, by not interfering with the morphisms, we are keeping the structure of the objects:

### 3.3.6 EXERCISE

Let  $\mathbf{D}$  be a full subcategory of  $\mathbf{C}$ . Show that any two  $\mathbf{D}$ -objects that are isomorphic in  $\mathbf{C}$  are also isomorphic in  $\mathbf{D}$ .

In the cases where, like finiteness of sets, the discriminating property does not distinguish between isomorphic objects, we obtain a subcategory that is not only full but contains all objects that are isomorphic, i.e. share the same substructure.

### 3.3.7 DEFINITION – isomorphism-closed full subcategory

A full subcategory  $\mathbf{D}$  of a category  $\mathbf{C}$  is said to be *isomorphism-closed* iff every  $\mathbf{C}$ -object that is isomorphic to a  $\mathbf{D}$ -object is also a  $\mathbf{D}$ -object.

So far, we have looked at the formation of a subcategory as the result of a selection of special objects and morphisms from a given category. In certain circumstances, this selection is, actually, the result of a process of "abstraction", i.e. certain objects are selected because they are "canonical" with respect to a set of objects that share a given property.

As an example, consider automata as defined in 2.2.10. Intuitively, only the states that are reachable, in the sense that they can be reached through the transition function from the initial state and some input sequence, are important for determining the "social life" of automata in terms of their ability to simulate other automata. Consider, then, the full subcategory **REACH** of **AUTOM** that consists of all reachable automata. The idea that, for the purposes of simulating other automata, every automaton can be represented by the automaton that is obtained by removing all non-reachable states can be formalised as follows:



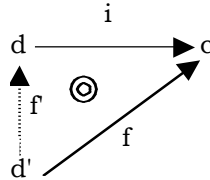
- every automaton  $A$  is related to a "canonical" reachable automaton  $R$  through a morphism  $c: R \sqsupseteq A$ . More concretely, if  $A = \langle X, S, Y, s_0, f, g \rangle$ , then  $R = \langle X, S_R, Y, s_0, f_R, g_R \rangle$  where  $X_R$  is the subset of  $X$  consisting of the states that are reachable, and  $f_R$  (resp.  $g_R$ ) is the restriction of  $f$  (resp.  $g$ ) to  $S_R$ . Notice that  $R$  is well-defined because  $f$  returns reachable states when applied to reachable states. The morphism  $c$  consists of the inclusion of  $S_R$  into  $S$  and the identities on  $X$  and  $Y$ . Notice that the inclusion satisfies the three properties of simulations in a trivial way.
- every inclusion  $c: R \sqsupseteq A$  as defined above satisfies the following property: given any reachable automata  $R'$  and simulation  $h: R' \sqsupseteq A$ , there is a unique morphism of reachable automata  $h': R' \sqsupseteq R$  such that  $h = h'c$ . In other words,  $R$  is the reachable automaton that is "closest" to  $A$  in the sense that any other reachable automata that  $A$  can simulate can also be simulated by  $R$ . Indeed, if  $R' = \langle X', S', Y', s'_0, f', g' \rangle$  and  $h = \langle h_x, h_s, h_y \rangle$ , then the equation  $h = h'c$  can only be satisfied if  $h'_s: S' \sqsupseteq S_R$  coincides with  $h_s$  on the whole domain  $S'$ , i.e. if  $h_s$  only returns reachable states. But this is the case because  $R'$  is itself reachable. On the other hand, the equation fully determines the morphism, thus establishing the uniqueness of  $h'$ .

This relationship between automata and reachable automata is an instance of what, in Category Theory, is called a co-reflective subcategory:

### 3.3.8 DEFINITION – co-reflective subcategory

Let  $\mathbf{D}$  be a subcategory of a category  $\mathbf{C}$ .

1. Let  $c$  be a  $\mathbf{C}$ -object. A  $\mathbf{D}$ -co-reflection for  $c$  is a  $\mathbf{C}$ -morphism  $i: d \sqsupseteq c$  for some  $\mathbf{D}$ -object  $d$  such that, for any  $\mathbf{C}$ -morphism  $f: d' \sqsupseteq c$  where  $d'$  is a  $\mathbf{D}$ -object, there is a unique  $\mathbf{D}$ -morphism  $f': d' \sqsupseteq d$  such that  $f = f'i$ , i.e. the following diagram commutes:



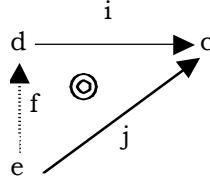
2. The category  $\mathbf{D}$  is said to be a *co-reflective subcategory* of  $\mathbf{C}$  iff every  $\mathbf{C}$ -object admits a  $\mathbf{D}$ -co-reflection.

Co-reflections are essentially "unique" in the following sense:

### 3.3.9 PROPOSITION

Let  $\mathbf{D}$  be a subcategory of a category  $\mathbf{C}$ .

1. Let  $c$  be a  $\mathbf{C}$ -object. If  $i: d \sqsupseteq c$  and  $j: e \sqsupseteq c$  are both  $\mathbf{D}$ -co-reflections for  $c$ , then there is a  $\mathbf{D}$ -isomorphism  $f: e \sqsupseteq d$  such that  $j = fi$ , i.e. the following diagram commutes:

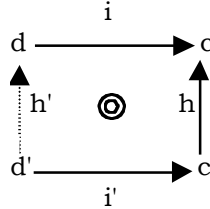


2. Let  $c$  be a  $\mathbf{C}$ -object. If  $i:d \rightarrow c$  is a  $\mathbf{D}$ -co-reflection for  $c$  and  $f:e \rightarrow d$  is a  $\mathbf{D}$ -isomorphism then  $f;i:e \rightarrow c$  is also a  $\mathbf{D}$ -co-reflection for  $c$ .

PROOF

1. The existence of  $f$  satisfying the equation results directly from the definition. The fact that  $f$  is an isomorphism can be proved as follows:
  - the existence of  $g:d \rightarrow e$  such that  $i=g;j$  can be concluded for the same reasons;
  - the composition  $g;j$  is such that  $(g;j);i=g;(f;i)=g;j=i=id_d;i$ . We can conclude that  $g;j=id_d$  by applying the uniqueness requirement of the definition of co-reflection to  $id_d$ .
  - a similar line of reasoning will allow us to conclude that  $f;i=id_e$ .
2. This is trivially proved.

Intuitively, a co-reflector for an object is a "secretary" that manages its social life, i.e. through which all communication must go. Indeed, consider two objects  $c$  and  $c'$  together with their secretaries  $d$  and  $d'$ , respectively. Consider an interaction  $h:c \rightarrow c'$ .



The composition  $i';h$  is in the same circumstances as the morphism  $f$  in the definition above. Hence, we conclude that there is an interaction  $h':d' \rightarrow d$  that factorises (intercepts)  $h$  in the sense that  $i';h=h';i$ .

Given that "secretaries" are also objects of the bigger category, a question arises naturally: who are their secretaries? Do they talk to each other directly or do they need their own secretaries and so on? It seems intuitive to expect that all secretaries communicate directly between them, i.e. that the co-reflection of a  $\mathbf{D}$ -object (secretary) is the identity (itself) up to isomorphism. However, this can only be guaranteed if (and only if)  $\mathbf{D}$  is a full subcategory:

### 3.3.10 PROPOSITION

Let  $\mathbf{D}$  be a co-reflective subcategory of a category  $\mathbf{C}$ . Then,  $\mathbf{D}$  is a full subcategory of  $\mathbf{C}$  iff, for every  $\mathbf{D}$ -object  $d$ ,  $id_d$  is a  $\mathbf{D}$ -co-reflection for  $d$ .

PROOF

1. Let  $\mathbf{D}$  be a full subcategory of  $\mathbf{C}$  and  $d$  a  $\mathbf{D}$ -object. Given any  $\mathbf{C}$ -morphism  $f:d' \rightarrow d$  where  $d'$  is also a  $\mathbf{D}$ -object, the equation  $f=f;id_d$  establishes  $f=f$  uniquely. So, the only way for the identity  $id_d$  not to be a co-reflector for  $d$  is that  $f$  is not a  $\mathbf{D}$ -morphism as required by the definition. Naturally, if the category is full, this cannot happen.
2. Consider now an arbitrary  $\mathbf{C}$ -morphism  $f:d' \rightarrow d$  where both  $d$  and  $d'$  are  $\mathbf{D}$ -objects. If  $id_d$  is a  $\mathbf{D}$ -co-reflection for  $d$  then, by definition, there exists a  $\mathbf{D}$ -morphism  $f':d' \rightarrow d$  such that  $f=f';id_d$ , i.e.  $f=f'$ . Hence,  $f$  is also a  $\mathbf{D}$ -morphism.

Indeed, if the subcategory is not full, the secretaries, when playing their roles as secretaries, may have more restricted means of interaction and, therefore, may not be able to interact as they would do as "normal" objects. This point is illustrated below with a very simple subcategory that is co-reflective but not full.

The "social motivation" that we have been using is, in fact, somewhat biased towards objects, i.e. it leads us to consider co-reflective subcategories that are full like that of reachable automata. However, any categorical property is, ultimately, determined by the morphisms. The following example shows that there are co-reflective subcategories that are not full.

### 3.3.11 EXAMPLE – SET as a non-full co-reflective subcategory of PAR

We proved in 3.3.2 that **SET** is a subcategory of **PAR**, the category of partial functions. Both categories share the same objects (sets), but **SET** retains only the functions that are total. Hence, **SET** is not a full subcategory of **PAR**. However, it is easy to prove that every set  $A$  admits as a co-reflector the partial inclusion  $A^\perp \rightarrow A$  where  $A^\perp$ , also called the "elevation of  $A$ ", is the set obtained from  $A$  by adding an additional element  $\perp$  called "bottom" or "undefined". The partial inclusion is the extension of the identity on  $A$  that is undefined on  $\perp$ . Given now an arbitrary partial function  $f:B \rightarrow A$  there is a unique elevation of  $f$  into a total function  $f^\perp:B \rightarrow A^\perp$  by mapping to  $\perp$  all the elements of  $B$  on which  $f$  is undefined. This is the standard technique that mathematicians use to deal with partiality in the context of total functions.

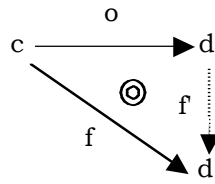
Hence, in this case, the "secretary" is also responsible for transforming from the "partial" mode of communication into the "total" mode. This is because, by not being full, the subcategory defines a more specialised mode of interaction. The co-reflector operates a transformation from the more general to the more specific mode.

Because Category Theory distinguishes between incoming and outgoing communication, co-reflectors manage, in fact, the incoming communication only. The dual notion, i.e. the notion that is obtained in the dual category, is called reflector, and manages the outgoing communication:

### 3.3.12 DEFINITION – reflective subcategory

Let  $\mathbf{D}$  be a subcategory of a category  $\mathbf{C}$ .

1. Let  $c$  be a  $\mathbf{C}$ -object. A  $\mathbf{D}$ -reflection for  $c$  is a  $\mathbf{C}$ -morphism  $o:c \rightarrow d$  for some  $\mathbf{D}$ -object  $d$  such that, for any  $\mathbf{C}$ -morphism  $f:c \rightarrow d'$  where  $d'$  is a  $\mathbf{D}$ -object, there is a unique  $\mathbf{D}$ -morphism  $f':d \rightarrow d'$  such that  $f=o;f'$  i.e. the following diagram commutes:



2. The category  $\mathbf{D}$  is said to be a *reflective subcategory* of  $\mathbf{C}$  iff every  $\mathbf{C}$ -object admits a  $\mathbf{D}$ -reflection.

Being the dual concept of co-reflections, reflections satisfy all the dual properties of co-reflections. In particular, reflections are essentially "unique" and identities are reflectors iff the subcategory is full. For consistency, in informal discussions, we shall call co-reflectors "insecretaries" or "cosecretaries", and reflectors "outsecretaries" or "secretaries". Examples of reflections are given in 3.6.4.

### 3.4 Eiffel class specifications

In order to illustrate some of the ways in which Category Theory can be used to capture the semantics of software engineering practice, we introduce a "practical" example related to the Eiffel language. This example will be used in later chapters as well.

As stated in [88], "Eiffel software texts – classes and their routines – may be equipped with elements of formal specification, called assertions, expressing correctness conditions". We shall not detail the language in which these assertions can be written: it can be found in [88].

For our purposes, an Eiffel class specification  $e$  consists of a triple  $\langle \square, P, I \rangle$  where

- $\square$  is the class signature consisting of a set (of features). Each feature has its own signature which consists of a pair  $\langle \text{arg}, \text{res} \rangle$  of sequences of types. Features can be attributes, functions or routines, the sets of which are denoted by  $\text{att}(\square)$ ,  $\text{fun}(\square)$  and  $\text{rou}(\square)$ , respectively.
- $P$  provides, for every  $r \in \text{rou}(\square)$ , a pair  $\langle \text{pre}_r, \text{pos}_r \rangle$  of Boolean expressions (the pre- and the post-condition of the routine, respectively).
- $I$  is a Boolean expression (the invariant of the class).

The semantics intended for such a specification is the one popularised under the designation "design by contract". Every client of the class when calling a routine  $r$  should make sure that the pre-condition for  $r$  holds, in which case the post-condition is assured to hold after the call is completed. The class invariant is a condition that is guaranteed by every creation procedure of the class and maintained by any routine.

For an example of a class specification consider bank accounts:

```
deferred class account
  attributes balance: int, vip: boolean;
```

### 3.4 Eiffel class specifications

```
routines deposit(i:nat)
    require true
    ensure balance = old balance + i
withdrawal(i:nat)
    require balance ≥ i
    ensure balance = old balance - i
invariant vip    balance ≥ 1000
```

Each bank account is capable of making deposits and withdrawals. These update the balance, as captured through an attribute, as specified in the *ensures* clauses. The routine that performs the withdrawals requires the client who makes the call to check that the balance is greater than the amount requested. Accounts that are considered to be *vip* have a balance greater than 1000.

The idea is that specifications are used to indicate *what* software components do, or are required to do, rather than *how* they do it. Changes to the implementations of the specified features are allowed as long as they do not violate the specifications.

Specifications have a "social life" that results from the need to adapt features. Feature adaptation [88] typically comes about when the class is inherited. That is to say, we return to our view to the social life of object classes that we have already explored earlier in the book, except that, now, we have other aspects to consider.

To be consistent with [88], the notion of morphism to be proposed for capturing the social life of class specifications has to account for the following operations:

- rename a feature;
- merge one or more features; this is called the *join mechanism* and applies to features inherited as deferred (i.e. without a chosen implementation);
- redefine a feature, changing the original signature or pre/post-conditions;
- add conditions to the invariant.

There are other circumstances in which features can be changed, like assigning an implementation to a feature that so far was "deferred". Because they relate only to implementations, and do not involve specifications except for discussing the correctness of the changes, i.e. because specifications are not socially active during these operations, we will not consider them in our formalisation.

The redefinition of a feature is subject to several constraints:

- at the level of its signature, the number of arguments and results cannot be changed; changes to their types can be performed subject to a number of conformance rules that, for simplicity, we will ignore;
- functions can be redeclared as attributes but not vice-versa; routines cannot be redeclared as attributes or functions, and attributes and functions cannot be redeclared as routines;
- pre-conditions can be weakened but not strengthened;
- pos-conditions can be strengthened but not weakened

The constraints that apply to pre/post-conditions have in mind the preservation of the contract: any client of a redefined routine has the right to expect a service that complies with the original specification; hence, it cannot be required to test for more conditions before calling the feature and, upon termination, it must get at least what he would normally get from the original feature.

All these constraints lead to the following definition:

A morphism  $F: e = \langle \square, P, I \rangle \rightarrow e' = \langle \square', P', I' \rangle$  of Eiffel class specifications consists of a total function between the class features such that

- for every feature  $f \in \square$ ,  $\arg_{F(f)} = \arg_f$  and  $\text{res}_{F(f)} = \text{res}_f$ ;
- for every attribute  $a \in \text{att}(\square)$ ,  $F(a) \in \text{att}(\square')$ ;
- for every function  $f \in \text{fun}(\square)$ ,  $F(f) \in \text{att}(\square') \cap \text{fun}(\square')$ ;
- for every routine  $r \in \text{rou}(\square)$ ,  $F(r) \in \text{rou}(\square')$ ;
- for every  $r \in \text{rou}(\square)$ ,  $F(\text{pre}_r) \vdash \text{pre}'_{F(r)}$  and  $\text{pos}'_{F(r)} \vdash F(\text{pos}_r)$ ;
- $I' \vdash F(I)$

As an example of a morphism of Eiffel class specifications, consider the following class specification obtained by inheriting from the previous example:

```
deferred_class flexible account
inherit account
  attributes credit: nat
  redefine withdrawal(i:nat)
    require else balance+credit ≥ i
  invariant vip balance ≥ 10000
```

That is to say, a flexible account extends the account with an attribute credit whose value can be added to the balance for satisfying a withdrawal request. As a counterpart to the added flexibility, the minimum balance for the account to be considered "vip" is now 10000.

The syntax of Eiffel is already such that some of the constraints are automatically met:

- the redeclaration of routines is such that pre-conditions are changed with a *require else condition* clause whose semantics is to add the specified condition to the inherited pre-condition as a disjunct (hence the weakening);
- the post-conditions are changed with a *ensure then condition* clause whose semantics is to add the specified condition to the inherited post-condition as a conjunct (hence the strengthening);
- the invariant clause is added as a conjunct to the invariant inherited from the parent class (hence the strengthening).

It is easy to prove that class specifications and their morphisms constitute a category **CLASS\_SPEC**. Later on in the book we will see how other properties of specifications, like the "join semantics rule" can be accounted for through so-called universal constructions.

## 3.5 Temporal specifications

In this section, we develop a category whose objects are specifications of process behaviour in Temporal Logic. This category will be used throughout the book for illustration purposes. Whereas our purpose with Eiffel class specifications was to show how Category Theory can apply to "real-life" modelling techniques, with temporal specifications we will try to illustrate, in the simplest way we know, typical properties of Specification Theory as applied to Concurrent Systems.

This example will also serve two other purposes. On the one hand, it will be used to show how relationships between different domains can be developed once they have been formalised in Category Theory. This will be done by relating temporal specifications with a very simple process model that we started building in 3.2.1. The other purpose is to show that there is a degree of "universality" in the kinds of constructions that are typically used across different domains for system modelling, and that this "universality" is very easily made evident, and formally characterised, through the use of Category Theory. Before proceeding, we should make clear that much of this particular topic was jointly developed with Félix Costa. The essential part of the technical details can be found in [32,33].

Specifications are often identified with theories (or theory presentations) in a given logic, an idea that has been around for quite a long time [17], although only more recently explored from the point of view of the temporal logic approach to reactive system specification [38]. In fact, process specifications are usually given as theory *presentations* rather than theories. By a theory presentation, we mean a pair consisting of a signature and a set of sentences – the non-logical axioms of the specification. The signature identifies the vocabulary symbols that are proper to the object being identified and the sentences provide a description of the properties that are being specified about that object.

We shall be modelling the behaviour of concurrent processes at the level of the actions that are provided by their interfaces. Hence, every signature identifies a set of actions in which a process can engage itself. An example of a signature is that of the specification of a vending machine able to accept coins and deliver cakes and cigars –  $\{coin, cake, cigar\}$ .

#### 3.5.1 DEFINITION – signatures of linear temporal logic

A signature of linear temporal logic is a set, the elements of which will be called action symbols.

The language in which the sentences that specify the behaviour of the process are written is that of linear temporal logic. The action symbols provide atomic propositions in the definition of the language associated with a signature:

#### 3.5.2 DEFINITION – language of linear temporal logic

The set of *temporal propositions*  $\mathbf{prop}(\Sigma)$  for a signature  $\Sigma$  is inductively defined as follows:

- every action symbol is a temporal proposition
- **beg** is a temporal proposition (denoting the initial state)
- if  $\Box$  is a temporal proposition so is  $(\neg\Box)$
- if  $\Box_1$  and  $\Box_2$  are temporal propositions so are  $(\Box_1 \Box_2)$  and  $(\Box_1 \mathbf{U}\Box_2)$ .

The temporal operator is **U** (until). Its semantics is defined below; informally,  $(\Box_1 \mathbf{U}\Box_2)$  is intended to hold whenever, from tomorrow,  $\Box_2$  will eventually hold and until then, but not necessarily then,  $\Box_1$  will hold. Other temporal operators such as **X** (next or tomorrow), **F** (eventually) and **G** (always) can be defined as abbreviations [68]. We will often make use of the operator **W** (weak until):  $(\Box_1 \mathbf{W}\Box_2)$  will hold whenever, from tomorrow, either  $\Box_2$  will eventually hold and until then, but not necessarily then,  $\Box_1$  will hold, or  $\Box_2$  will forever be false and  $\Box_1$  true.

### 3.5.3 DEFINITION – semantics of linear temporal logic

The language of linear temporal logic is interpreted over infinite sequences of sets of actions. That is, an interpretation structure for a signature  $\Sigma$  is a sequence  $\langle \Sigma(2^{\Sigma}) \rangle$ . These are canonical Kripke structures for linear, discrete, propositional logic [113]. Each infinite sequence represents a possible behaviour for the process being specified. The sets of actions represent the events that take place during the lifetime of the process. As already explained in 3.2.1, the event that consists of the empty set of actions represents a transition performed by the environment without the participation of the process.

A  $\Sigma$ -proposition  $\varphi$  is said to be true for  $\langle \Sigma(2^{\Sigma}) \rangle$  at state  $i$ , which we write  $\langle \Sigma(2^{\Sigma}) \rangle \models_i^i \varphi$  iff:

- if  $\langle \Sigma(2^{\Sigma}) \rangle, \langle \Sigma(2^{\Sigma}) \rangle \models_i^i \varphi$  iff  $\langle \Sigma(2^{\Sigma}) \rangle(i)$ ,
- $\langle \Sigma(2^{\Sigma}) \rangle \models_i^i \text{beg}$  iff  $i=0$ ,
- $\langle \Sigma(2^{\Sigma}) \rangle \models_i^i (\neg \varphi)$  iff it is not the case that  $\langle \Sigma(2^{\Sigma}) \rangle \models_i^i \varphi$
- $\langle \Sigma(2^{\Sigma}) \rangle \models_i^i (\varphi_1 \ \varphi_2)$  iff  $\langle \Sigma(2^{\Sigma}) \rangle \models_i^i \varphi_1$  implies  $\langle \Sigma(2^{\Sigma}) \rangle \models_i^i \varphi_2$ ,
- $\langle \Sigma(2^{\Sigma}) \rangle \models_i^i (\varphi_1 \text{U} \varphi_2)$  iff, for some  $j>i$ ,  $\langle \Sigma(2^{\Sigma}) \rangle \models_j^j \varphi_2$  and, for every  $i<k<j$ ,  $\langle \Sigma(2^{\Sigma}) \rangle \models_k^k \varphi_1$ .

The proposition  $\varphi$  is said to be *true* in  $\langle \Sigma(2^{\Sigma}) \rangle$ , written  $\langle \Sigma(2^{\Sigma}) \rangle \models \varphi$ , if and only if  $\langle \Sigma(2^{\Sigma}) \rangle \models_i^i \varphi$  at every state  $i$ . We also write  $\langle \Sigma(2^{\Sigma}) \rangle \models \varphi$  for a collection of propositions  $\varphi$  meaning that each proposition of  $\varphi$  is true in  $\langle \Sigma(2^{\Sigma}) \rangle$ , and  $\langle \Sigma(2^{\Sigma}) \rangle \models \varphi$  for a collection  $\varphi$  of sequences meaning that  $\varphi$  is true in every  $\langle \Sigma(2^{\Sigma}) \rangle$ .

Finally, for every set  $\varphi$  of  $\Sigma$ -propositions and every  $\Sigma$ -proposition  $\psi$ ,  $\varphi$  is a consequence of  $\psi$  – ( $\varphi \vdash_{\Sigma} \psi$ ) – if and only if  $\psi$  is true in every sequence that makes all the propositions in  $\varphi$  true.

The corresponding notion of theory is given as usual for the closure system induced by the consequence relation:

### 3.5.4 DEFINITION – temporal theories and their presentations

1. Let  $\Sigma$  be a signature. A subset  $\varphi$  of **prop**( $\Sigma$ ) is said to be *closed* if and only if, for every  $\langle \Sigma(2^{\Sigma}) \rangle \in \mathbf{prop}(\Sigma)$ ,  $\langle \Sigma(2^{\Sigma}) \rangle \vdash_{\Sigma} \varphi$  implies  $\langle \Sigma(2^{\Sigma}) \rangle \models \varphi$ . By  $c_{\Sigma}(\varphi)$  we denote the least closed set that contains  $\varphi$ .
2. A *temporal theory* is a pair  $\langle \Sigma, \varphi \rangle$  where  $\Sigma$  is a signature and  $\varphi$  is a closed set of  $\Sigma$ -propositions.
3. A *theory presentation* is a pair  $\langle \Sigma, \varphi \rangle$  where  $\Sigma$  is a signature and  $\varphi$  is a set of  $\Sigma$ -propositions. The presented theory is  $\langle \Sigma, c_{\Sigma}(\varphi) \rangle$ .

That is to say, a theory consists of a set of sentences that is closed for consequence: it contains all the theorems that can be derived from its sentences. A presentation is not necessarily closed under consequence: it provides a more "economical" way of specifying the intended behaviour of a system. A presentation consists only of a selected set of properties (also called the "axioms" of the presentation) that are required of the system, leaving the computation of the properties that can be derived from this selected set (its "theorems") to the proof theory of the logic. Hence, specifications are usually identified with presentations, not with theories.



### 3.5.5 EXERCISE

Prove that the operators  $c_\square$  defined above satisfy the following properties:

- reflexivity: for every  $\square\square\mathbf{prop}(\square)$ ,  $\square\square c_\square(\square)$
- monotonicity: for every  $\square, \square\square\mathbf{prop}(\square)$ ,  $\square\square\square$  implies  $c_\square(\square)\square c_\square(\square)$
- idempotence: for every  $\square\square\mathbf{prop}(\square)$ ,  $c_\square(c_\square(\square))\square c_\square(\square)$

### 3.5.6 EXAMPLE – a vending machine

As an example, consider the following specification of a vending machine:

```

specification vending machine is
signature      coin, cake, cigar
axioms        beg    ( $\neg$ cake $\square$  $\neg$ cigar)  $\square$  (coin ( $\neg$ cake $\square$  $\neg$ cigar)Wcoin)
                  coin  ( $\neg$ coin)W(cake cigar)
                  (cake cigar) ( $\neg$ cake $\square$  $\neg$ cigar)Wcoin
                  cake   ( $\neg$ cigar)

```

This machine is able to accept coins, deliver cakes and deliver cigars. The machine is initialised so as to accept only coins (first axiom). Once it accepts a coin it can deliver either a cake or a cigar (second axiom), but not both (fourth axiom). After delivering a cake or a cigar it is ready to accept more coins (third axiom).

As already mentioned, the model of process behaviour that we are adopting reflects an (abstract) synchronous, multi-processor architecture in which, at each transition, several actions may be executed concurrently. We are going to show that this synchronous flavour is captured through the notion of specification morphism (interpretation between theories) as a mathematical model of the relationship between systems and their components.

An interpretation between theories is typically defined as a mapping between their signatures that preserves theorems. Notice, once again, the idea of structure preservation presiding to the definition of morphisms. The structure being preserved in this case is the one given by the properties of the processes involved as captured through the theorems of the specifications.

### 3.5.7 DEFINITION – interpretation between temporal theories

1. Let  $\square$  and  $\square'$  be signatures and  $f:\square\square\square'$  a total function. The *translation map*  $f:\mathbf{prop}(\square)\rightarrow\mathbf{prop}(\square')$  induced by  $f$  is inductively defined as follows:
  - $f(\mathbf{beg})=\mathbf{beg}$
  - if  $a\square\square$  then  $f(a)=f(a)$
  - $f(\neg\square)=\neg f(\square)$
  - $f(\square_1\square\square_2)=f(\square_1)\square f(\square_2)$
  - $f(\square_1\mathbf{U}\square_2)=f(\square_1)\mathbf{U}f(\square_2)$
2. An *interpretation* between two theories (or theory morphism)  $\langle\square_1,\square_1\rangle$  and  $\langle\square_2,\square_2\rangle$  is a map  $f:\square_1\square\square_2$  such that  $f(\square_1)\square\square_2$ .

3. A *morphism* between two presentations  $\langle \Sigma_1, \Pi_1 \rangle$  and  $\langle \Sigma_2, \Pi_2 \rangle$  is a map  $f: \Sigma_1 \rightarrow \Sigma_2$  such that  $f(c_{\Pi_1}(\Pi_1)) \subseteq c_{\Pi_2}(\Pi_2)$ .

### 3.5.8 PROPOSITION – temporal theories and their presentations

1. Temporal theories and interpretations between theories define a category called ***THEO<sub>LT</sub>***.
2. Presentations of temporal theories and their morphisms define a category called ***PRES<sub>LT</sub>***. Furthermore, ***THEO<sub>LT</sub>*** is a subcategory of ***PRES<sub>LT</sub>***.

PROOF

1. Although we have not made it explicit, the composition law and identity map that we have in mind for theories are the ones inherited from signatures as sets. Hence, this is another example of constructing a category by adding structure: the underlying category is that of sets and the structure being added is the collection of theorems that constitute the specification. Therefore, the properties of the composition law and identity map are inherited from ***SET***; we only have to prove the closure properties, i.e. that the composition of theory morphisms as functions is still a theory morphism and that the identity function is a theory morphism. The latter is, of course, trivial because the translation map induced by the identity on signatures is itself the identity. The former can be proved as follows:
  - first of all, given  $f: \Sigma \rightarrow \Sigma'$  and  $g: \Sigma' \rightarrow \Sigma''$ , we have to prove that  $f \circ g = f;g$ . That is, the translation map induced by a composite function is the composition of the translation maps induced by the components. This can be proved by structural induction and is left as an exercise.
  - given theories  $\langle \Sigma, \Pi \rangle$ ,  $\langle \Sigma', \Pi' \rangle$  and  $\langle \Sigma'', \Pi'' \rangle$ , and theory morphisms  $f: \langle \Sigma, \Pi \rangle \rightarrow \langle \Sigma', \Pi' \rangle$  and  $g: \langle \Sigma', \Pi' \rangle \rightarrow \langle \Sigma'', \Pi'' \rangle$ ,  $(f;g)(\Pi) = g(f(\Pi))$  as proved above. Because  $f$  is a theory morphism,  $f(\Pi) \subseteq \Pi'$ , which implies  $(f;g)(\Pi) \subseteq g(\Pi')$ . Because  $g$  is a theory morphism,  $g(\Pi') \subseteq \Pi''$ , which implies  $(f;g)(\Pi) \subseteq \Pi''$ . Hence,  $(f;g)$  is a theory morphism.
2. The proof that theory presentations and their morphisms define a category can proceed exactly in the same way as for theories. The fact that ***THEO<sub>LT</sub>*** is a subcategory of the resulting category has a very trivial proof:
  - clearly, every theory is a presentation: it just happens to be closed.
  - given a theory morphism  $f: \langle \Sigma, \Pi \rangle \rightarrow \langle \Sigma', \Pi' \rangle$  we have to prove that it defines a presentation morphism, i.e. that  $f(c_{\Pi}(\Pi)) \subseteq c_{\Pi'}(\Pi')$ . Because  $\Pi$  and  $\Pi'$  are closed,  $c_{\Pi}(\Pi) = \Pi$  and  $c_{\Pi'}(\Pi') = \Pi'$ . Because  $f$  is a theory morphism,  $f(\Pi) \subseteq \Pi'$ .
  - the properties of the composition law and the identity map are trivially proved.

Checking that a signature map defines a morphism between two presentations by checking that every theorem of the source is translated to a theorem of the target is not "practical". Typically, one would prefer to check that only the axioms of the presentation are translated into theorems, from what it should follow that the theorems are preserved. Although this is not true for an arbitrary logic, it is a property of the temporal logic that we defined above as proved below.

In order to do so, we need to be able to relate the models of the signatures involved:

### 3.5.9 DEFINITION/PROPOSITION – reducts; the satisfaction condition

1. Let  $\Sigma$  and  $\Sigma'$  be signatures and  $f: \Sigma \rightarrow \Sigma'$  a total function. Given an interpretation structure  $\mathcal{I}(\Sigma)(2^\Sigma)^\mathcal{I}$  for  $\Sigma'$ , we define its *reduct*  $\mathcal{I}|_f$  as the interpretation structure for  $\Sigma$  defined by  $\mathcal{I}|_f(i) = f^{-1}(\mathcal{I}'(i))$ . That is, the reduct of a sequence  $\mathcal{I}'$  of  $\Sigma'$ -actions is the sequence that consists, at each point  $i$ , of the  $\Sigma$ -actions that are translated through  $f$  into  $\mathcal{I}'(i)$ .
2. Let  $\Sigma$  and  $\Sigma'$  be signatures,  $f: \Sigma \rightarrow \Sigma'$  a total function,  $\mathcal{I}(\Sigma)(2^\Sigma)^\mathcal{I}$  an interpretation structure for  $\Sigma'$ , and  $\phi$  a  $\Sigma$ -proposition. Then, for every  $i$ ,

$$\mathcal{I}(\Sigma)(2^\Sigma)^\mathcal{I} \models_i \phi \text{ iff } \mathcal{I}|_f \models_i \phi.$$

This property is usually called "the satisfaction condition", the reason for which will be disclosed later on in the book.

PROOF

The proof of 2 is by induction on the structure of  $\phi$ . The base case (action symbols) results directly from the definition of reduct:  $f(\Sigma) \models_i \phi$  iff  $\mathcal{I}(\Sigma)(2^\Sigma)^\mathcal{I} \models_i \phi$  ( $= \mathcal{I}|_f(i)$ ). The induction step presents no difficulties.

### 3.5.10 PROPOSITION – presentation lemma

1. Given a signature morphism  $f: \Sigma_1 \rightarrow \Sigma_2$  and  $\Sigma_1 \text{prop}(\Sigma_1)$ ,  $f(c_{\Sigma_1}(\Sigma_1)) \models c_{\Sigma_2}(f(\Sigma_1))$ .
2. Given presentations  $\langle \Sigma_1, \Sigma_1 \rangle$  and  $\langle \Sigma_2, \Sigma_2 \rangle$ , a map  $f: \Sigma_1 \rightarrow \Sigma_2$  is a morphism between them iff  $f(\Sigma_1) \models c_{\Sigma_2}(\Sigma_2)$ . This result is usually called "the presentation lemma".

PROOF

1. Let  $\Sigma_1 \models c_{\Sigma_1}(\Sigma_1)$ . We have to prove that  $f(\Sigma_1) \models c_{\Sigma_2}(f(\Sigma_1))$ . For that purpose, consider an arbitrary interpretation structure  $\mathcal{I}(\Sigma_1)(2^{\Sigma_1})^\mathcal{I}$  in which  $f(\Sigma_1)$  is true. From what was just proved, all the propositions in  $\Sigma_1$  are true in  $\mathcal{I}|_f$ . But, then, so is  $\Sigma_1$  because  $\Sigma_1 \models c_{\Sigma_1}(\Sigma_1)$ . Applying the same result, but in the other direction, we know that  $f(\Sigma_1)$  is true in  $\mathcal{I}$ .
2. All that needs to be proved is that  $f(\Sigma_1) \models c_{\Sigma_2}(\Sigma_2)$  iff  $f(\Sigma_1) \models c_{\Sigma_2}(\Sigma_2)$ .
  - The forward implication is an immediate consequence of the reflexivity of closure, which allows us to derive that  $\Sigma_1 \models c_{\Sigma_1}(\Sigma_1)$ .
  - Consider now the reverse implication and assume that  $f(\Sigma_1) \models c_{\Sigma_2}(\Sigma_2)$ . Because closure operators are monotone, we can infer  $c_{\Sigma_2}(f(\Sigma_1)) \models c_{\Sigma_2}(c_{\Sigma_2}(\Sigma_2))$ . Because closure operators are idempotent,  $c_{\Sigma_2}(c_{\Sigma_2}(\Sigma_2)) = c_{\Sigma_2}(\Sigma_2)$ . Hence,  $c_{\Sigma_2}(f(\Sigma_1)) \models c_{\Sigma_2}(\Sigma_2)$ . Using now the result proved in 1, we obtain by transitivity  $f(\Sigma_1) \models c_{\Sigma_2}(\Sigma_2)$ .

### 3.5.11 EXAMPLE – a regulated vending machine

As an example of the use of morphisms for system modelling, consider the following specification:

### 3. BUILDING CATEGORIES

```

specification regulated vending machine is
signature      coin, cake, cigar, token
axioms        beg    ( $\neg$ cake $\square$  $\neg$ cigar $\square$  $\neg$ token)  $\square$ 
                  (coin  $\square$  ( $\neg$ cake $\square$  $\neg$ cigar) $\mathbf{W}$ coin)
                  coin  $\square$  ( $\neg$ coin) $\mathbf{W}$ (cake cigar)
                  coin  $\square$  ( $\neg$ cigar) $\mathbf{W}$ token
                  (cake cigar)  $\square$  ( $\neg$ cake $\square$  $\neg$ cigar) $\mathbf{W}$ coin
                  cake  $\square$  ( $\neg$ cigar)
                  token  $\square$  ( $\neg$ cake $\square$  $\neg$ cigar $\square$  $\neg$ coin)

```

That is to say, the machine is now extended to be able to accept tokens, and is regulated in such a way that it will only deliver a cigar if, after having accepted a coin, it receives a token. The last axiom says that coins, cakes and cigars cannot be used as tokens...

We can see this specification as resulting from the previous vending machine through the superposition of some mechanism for controlling the sales of cigars. Later on in the book (6.1.24), we shall see how a regulator can be independently specified and connected to the vending machine in order to achieve the required superposition. That is to say, we shall see how the regulated vending machine can be defined as a configuration of which the original vending machine, as well as the regulator, are components.

In the meanwhile, we shall analyse how theory morphisms can be used to identify components of systems. For instance, in the example above, we can establish a morphism between the specifications of the vending machine and the regulated vending machine. The signature morphism maps the actions of the vending machine to those of the regulated vending machine that have the same names. To prove that the signature morphism defines an interpretation between the two specifications, we have to check if every axiom of the specification of the vending machine (3.5.6) is translated to a theorem of the regulated vending machine:

- the initialisation condition is preserved: it is strengthened with the condition  $\neg$ token;
- the second axiom is translated directly into the second axiom of the regulated vending machine;
- the third axiom is translated directly into the fourth axiom of the regulated vending machine;
- and the fourth axiom is translated directly into the fifth axiom of the regulated vending machine.

The fact that morphisms identify components of systems is captured by the following result:

#### 3.5.12 PROPOSITION

Let  $\langle \square_1, \square_1 \rangle$  and  $\langle \square_2, \square_2 \rangle$  be theories and  $f: \square_1 \square \square_2$  a signature morphism. Then,  $f$  defines a theory morphism iff, for every model  $\square'$  of  $\langle \square_2, \square_2 \rangle$ ,  $\square' \upharpoonright_f$  is a model of  $\langle \square_1, \square_1 \rangle$ .

PROOF

1. Assume that  $f$  defines a theory morphism and let  $\square'$  be a model of  $\langle \square_2, \square_2 \rangle$ . To prove that  $\square' \upharpoonright_f$  is a model of  $\langle \square_1, \square_1 \rangle$ , let  $\square \square \square_1$ . Because  $f$  is a theory morphism,

$f(\Box)\Box\Box_2$  and, hence,  $f(\Box)$  is true in  $\Box'$ . But, by the fundamental property of reducts,  $\Box$  is true in  $\Box'|_f$ . Hence,  $\Box'|_f$  is a model of  $\langle\Box_1, \Box_1\rangle$ .

2. Assume now that, for every model  $\Box'$  of  $\langle\Box_2, \Box_2\rangle$ ,  $\Box'|_f$  is a model of  $\langle\Box_1, \Box_1\rangle$ . Let  $\Box\Box\Box_1$ . We have to prove that  $f(\Box)\Box\Box_2$ . Let  $\Box'$  be a model of  $\langle\Box_2, \Box_2\rangle$ . Because  $\Box'|_f$  is a model of  $\langle\Box_1, \Box_1\rangle$ ,  $\Box$  is true in  $\Box'|_f$ . By the fundamental property of reducts (opposite direction),  $f(\Box)$  is true in  $\Box'$ .

That is to say, given a morphism between two temporal specifications, every behaviour of the target is projected back to a model of the source. Because the reduct identifies the actions of the source that occur at each point in the execution of the target, we can say that an interpretation between theories identifies in every behaviour that is according to the target specification, a behaviour that is according to the source specification. Hence, the morphism recognises in the source a component of the target.

Notice that it is not necessary that every behaviour of the source (component) be recovered through the reduct. Indeed, as a component of a larger system, some behaviours of the component may be lost because of interactions with other components. This is the case above. The behaviours in which a coin is followed immediately by a cigar are not part of any model of the regulated vending machine because a token is required after the coin before the cigar can be delivered.

## 3.6 Closure systems

In the previous section, we presented several categories related to the specification of systems on the basis of properties stated in the language of temporal logic. We will show in the next chapters that these categories satisfy a number of properties that make them useful to modularise specifications. However, these properties are in no way particular to the temporal logic that was chosen to express specifications. They are shared by a number of structures that generalise what, sometimes, are called "closure systems". This section presents the initial step in a sequence of constructions that will end up in the definition of institutions [61],  $\pi$ -institutions [45], and general logics [87]. These are all categorical generalisations of the notions of Logic and associated concepts like theories that bring out the structural properties that make them useful for specification.

### 3.6.1 DEFINITION – closure system

We call a closure system a pair  $\langle L, c \rangle$  where  $L$  is a set and  $c: 2^L \rightarrow 2^L$  is a total function satisfying the following properties:

- reflexivity: for every  $\Box \subseteq L$ ,  $\Box \subseteq c(\Box)$
- monotonicity: for every  $\Box, \Box' \subseteq L$ ,  $\Box \subseteq \Box'$  implies  $c(\Box) \subseteq c(\Box')$
- idempotence: for every  $\Box \subseteq L$ ,  $c(c(\Box)) \subseteq c(\Box)$

### 3.6.2 DEFINITION/PROPOSITION – category of closure systems

We define the category **CLOS** of closure systems by defining a morphism  $f: \langle L, c \rangle \rightarrow \langle L', c' \rangle$  to be a map  $f: L \rightarrow L'$  satisfying, for every  $\Box \subseteq L$ ,  $f(c(\Box)) \subseteq c'(f(\Box))$ .

PROOF

Again an example of adding structure to **SET**.

- the identity function on sets is trivially a morphism of closure systems;
- consider now two morphisms  $f: \langle L, c \rangle \rightarrow \langle L', c' \rangle$  and  $g: \langle L', c' \rangle \rightarrow \langle L'', c'' \rangle$  and  $\square \square L$ . Because  $f$  is a morphism,  $f(c(\square)) \subseteq c'(f(\square))$ . This also implies that  $g(f(c(\square))) \subseteq g(c'(f(\square)))$ . On the other hand, because  $g$  is a morphism,  $g(c'(f(\square))) \subseteq c''(g(f(\square)))$ . Hence,  $(f;g)(c(\square)) \subseteq c''((f;g)(\square))$ .

As an example of closure systems we have, for every temporal signature  $\square$ , the pair  $\text{clos}(\square)$  formed by **prop**( $\square$ ) as defined in 3.5.2 together with the closure operator defined in 3.5.4. Notice that the result proved in 3.5.10 shows that every signature morphism induces a morphism between the corresponding closure systems.

### 3.6.3 DEFINITION/PROPOSITION – theories in closure systems

Consider a closure system  $\langle L, c \rangle$ .

1. We say that  $\square \square L$  is closed iff  $\square = c(\square)$ .
2. We define the category **THEO** $_{\langle L, c \rangle}$  whose objects are the closed subsets of  $L$  and morphisms are given by inclusions.
3. We define the category **PRES** $_{\langle L, c \rangle}$  whose objects are the subsets of  $L$  and morphisms given by the pre-order  $\square \leq \square$  iff  $c(\square) \subseteq c(\square)$ .
4. We define the category **SPRES** $_{\langle L, c \rangle}$  whose objects are the subsets of  $L$  ordered by inclusion.

PROOF

These are trivial examples of categories that are given through pre-ordered sets.

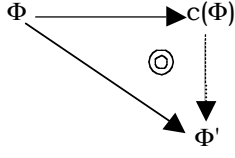
We call the objects of **SPRES** $_{\langle L, c \rangle}$  "strict presentations" because the morphisms require that the axioms be preserved. The more attentive reader will have noticed that these notions do not coincide exactly with the definitions used in section 3.5 because they do not contemplate changes of language. We shall generalise these notions in later sections to account for relationships between theories in different languages.

The following proposition defines several relationships between these categories. For simplicity, we omit the reference to the underlying closure system.

### 3.6.4 PROPOSITION

1. **THEO** is a full subcategory of **PRES** (and of **SPRES**), and **SPRES** is a subcategory of **PRES**.
2. **THEO** is a reflective subcategory of **PRES** (and of **SPRES**), but **SPRES** is not.
3. **THEO** is a co-reflective subcategory of **PRES** (but not of **SPRES**) and so is **SPRES**.

PROOF

1. The proof that **THEO** is a (full) subcategory of **SPRES** and **PRES** is trivial. The result that **SPRES** is a subcategory of **PRES** is an immediate consequence of the monotonicity of the closure operator. This subcategory is not full because theorems may be preserved even if the axioms are not.
2. The first results are about closure as a reflector. Indeed, we are going to prove that, given an arbitrary presentation  $\square$ , its reflector is its closure  $c(\square)$ . There are two parts in the proofs: (1), the very existence of the reflector as a morphism; (2), its universal property. All the reasoning evolves around the following "diagram":
 

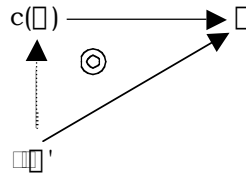
In the case of **THEO** as a subcategory of **PRES**, part 1 translates to the inclusion  $c(\square) \sqsubseteq c(c(\square))$ , which is a corollary of the reflexivity of the closure relation. Part 2 translates to  $c(\square) \sqsubseteq c(\square')$  implies  $c(\square) \sqsubseteq \square'$  because the morphism on the right is in **THEO**. This holds because, being a theory,  $c(\square') = \square'$ .

In the case of **THEO** as a subcategory of **SPRES**, part 1 translates to the inclusion  $\square \sqsubseteq c(\square)$  because the morphism is now strict. Again, this is a corollary of the reflexivity of the closure relation. Part 2 translates to  $\square \sqsubseteq \square'$  implies  $c(\square) \sqsubseteq \square'$  because the morphism on the right is in **THEO** and the one on the left is strict. This is a consequence of the monotonicity of the closure operator and the fact that, being a theory,  $c(\square') = \square'$ .

The fact that **SPRES** is not a reflective subcategory of **PRES** can be inferred from part 2:  $c(\square) \sqsubseteq c(\square')$  implies  $c(\square) \sqsubseteq \square'$  does not necessarily hold because, being a presentation,  $\square'$  is not necessarily closed.

Summarising, on the side of reflections, these results capture the fact that, for the purposes of "outward communication", i.e. for being interpreted, presentations can delegate on theories but not on strict presentations because, whereas the former will use the full theory for the interpretation, the latter will only look at relationships between axioms.

2. The second set of results is about the co-reflective properties of closure. The relevant "diagram" now is:



In the case of **THEO** as a subcategory of **PRES**, part 1 translates to the inclusion  $c(c(\square)) \sqsubseteq c(\square)$ , which is a corollary of the idempotence of the closure operator. Part

### 3. BUILDING CATEGORIES

2 translates to  $c(\Box') \Box c(\Box)$  *implies*  $\Box' \Box c(\Box)$  because the morphism on the left is in **THEO**. This holds because of reflexivity.

The fact that **THEO** is not a co-reflective subcategory of **SPRES** can be inferred from part 1: the inclusion  $c(\Box) \Box \Box$  does not necessarily hold because, being a presentation,  $\Box$  is not necessarily closed.

In the case of **SPRES** as a subcategory of **PRES**, part 1 translates again to the inclusion  $c(c(\Box)) \Box c(\Box)$ . Part 2 translates also to  $c(\Box') \Box c(\Box)$  *implies*  $\Box' \Box c(\Box)$  because the morphism on the left is strict. This holds because of reflexivity.

Summarising, on the side of co-reflections, the existence of a co-reflection from strict into loose interpretations reflects (or co-reflects...) the fact that in order to interpret another theory presentation, a given theory presentation just needs to consider the axioms of the source. On the other hand, because strict presentations are not expressive enough to interpret theories, they do not admit theories as co-reflectors, i.e. they cannot allow theories to handle their in-communication.

Globally, this result shows that theories and theory presentations with the looser notion of morphism define, essentially, the same structure. The same does not happen with strict presentations. Strict presentations can co-reflect presentations and be reflected by theories but not the other way around because they are not expressive enough to capture closure.