



**Laboratoire d'Arithmétique, de Calcul formel et d'Optimisation**  
ESA - CNRS 6090

---

## Towards Soft Typing in Computer Algebra

**Dominique Duval & Christian Lair**

Rapport de recherche n° 1998-07

---

Université de Limoges, 123 avenue Albert Thomas, 87060 Limoges Cedex  
Tél. 05 55 45 73 23 - Fax. 05 55 45 73 22 - [laco@unilim.fr](mailto:laco@unilim.fr)

<http://www.unilim.fr/laco/>



# Towards Soft Typing in Computer Algebra

Dominique DUVAL and Christian LAIR

*October 15, 1998*

This paper is a revised version of the notes for the tutorial of D. Duval at ISSAC'98. We are indebted to the participants of the working group on *sketches and computer algebra* for their questions and suggestions, and to C.N.R.S. for its support.

This document was processed with L<sup>A</sup>T<sub>E</sub>X and X<sub>Y</sub>-pic.

Dominique DUVAL  
LACO: Laboratoire d'Arithmétique, Calcul Formel et Optimisation  
Université de Limoges  
123, avenue Albert Thomas, 87000 LIMOGES (FRANCE)  
dominique.duval@unilim.fr - <http://www.unilim.fr/laco/>

Christian LAIR  
U.F.R. de Mathématiques  
Université Denis Diderot - Paris 7  
2, place Jussieu, 75251 PARIS cedex 05 (FRANCE)  
lair@ufrp7.math.jussieu.fr

## Introduction

In Computer Algebra, the use of *types*, in the sense of *abstract data types* or *algebraic specifications*, is often considered as both useful and troublesome: types can be helpful for genericity and reuse of code, or simply in order to fix bugs easier; but the precise definition of types is usually tedious, and often quite difficult.

This could be overcome by some kind of “approximate” typing. In this tutorial, we present a typing technique, called *soft typing*, which allows to start from an approximate type and to add information in order to get an exact type.

The word *type* is used in this tutorial, as often in computer algebra, as an alternative for *specification*: it is a syntactic feature designed both to describe *models* and to give a syntax for *programs*, in a coherent way. One may have several models in mind, or only one model, which is usually the *initial model*, or the *abstract data type*, defined by the specification. For instance in Axiom or Aldor usually a *category* has several models while a *domain* corresponds to only one [18].

Ehresmann introduced *sketches* in the 60’s [9, 10]; at the end of the 80’s it came to light that sketch theory can be used as a specification tool in computer science [22, 2]. In order to describe models, sketches are a non-trivial alternative to logical theories [14, 15]. Sketches can be combined together thanks to a constructor of *colimits*, which yields the usual tools for building “big” specifications from “small” ones by adding new features. Functional (or applicative) programs can be defined as *terms* generated in some given sketch.

Soft typing relies on a new constructor over sketches, called the *ribbon product*; it allows to build “big” specifications from “small” ones in the following way: one of the “small” specifications is an approximate specification, the other ones say how it must be corrected, or *extended*, in order to build an exact “big” specification.

Actually, sketches are not powerful enough for our purpose; we rather use *mosaics*, which are based on the generalization of sketches by Lair [20, 21]. A mosaic is made of elementary pieces; these pieces are *ambigraphs* (i.e. oriented graphs with some additional features).

Theoretical background for ribbon product will be soon submitted to pub-

lication [6]. No application has been made so far. This tutorial is the first presentation in an international meeting of the main ideas underlying our work on soft typing.

Our motivation for soft typing comes from computer algebra. However, soft typing can be used more generally for software design.

We will see that soft typing can be used to solve questions related to error handling, overloading and coercions, and to express imperative programs in a functional setting without expliciting the state of the machine. These questions are usually considered as important in computer science, and a lot of work has been done to solve them, using fairly different techniques: logic, category theory, generalized specifications, rewriting techniques, *etc.* Our reference list mentions some papers related to these subjects; this list is far from exhaustive. However, there is no time for any comparison of our method with others in this tutorial; see [6] for that.

This tutorial is elementary: no specific knowledge is assumed, though some previous experience with either a strongly typed language (*e.g.* [18]), algebraic specifications (*e.g.* [13]) or category theory (*e.g.* [23]) may be helpful.

Our aim is to present the basic ideas underlying ribbon product construction and its application to soft typing. One should look at [6] for precise definitions, theorems and proofs.

This tutorial is based on examples. These examples are elementary, however it should be clear that they could easily be generalized to a lot of situations.

The tutorial proceeds as follows (sections 4 and 5 could be skipped):

1. A short introduction to ambigraphs, mosaics and their models.
2. The notion of soft typing from a simple example of error handling.
3. The extension of an ambigraph by a ribbon product.
4. The extension of a mosaic by a ribbon product.
5. Specifications themselves must be specified. . .
6. An example of soft typing for overloading and coercions.
7. An example of soft typing for imperative programming.

– 1 –

## Typing with ambigraphs and mosaics

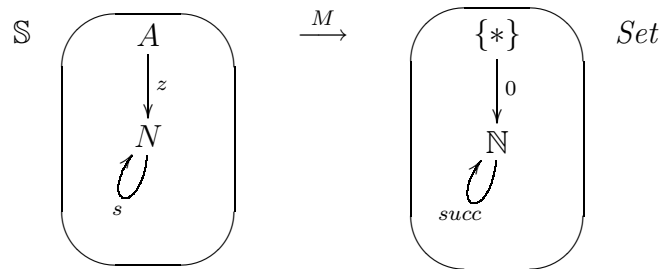
In this section we present *ambigraphs* and *mosaics* as tools for usual typing (or specification). We focus on specifications as a way to give a syntactic description for classes of models. Specifications as a frame for writing and evaluating programs will be considered only in section 7.

## Oriented graphs

The simplest mosaics are *oriented graphs*, made of *nodes* and *arrows*. For each arrow  $s : S \rightarrow T$  the nodes  $S$  and  $T$  are called respectively the *domain* and *codomain* of  $s$ , and  $S \rightarrow T$  is called the *rank* of  $s$ .

A *set-valued model*  $M$  of an oriented graph  $\mathbb{S}$  is made of an *interpretation* of each node  $S$  as a set  $M(S)$  and an *interpretation* of each arrow  $s : S \rightarrow T$  as a map  $M(s) : M(S) \rightarrow M(T)$ .

*In categorical terms, a model of  $\mathbb{S}$  is an homomorphism  $M : \mathbb{S} \rightarrow \text{Set}$  from the oriented graph  $\mathbb{S}$  towards the category *Set of sets*.*

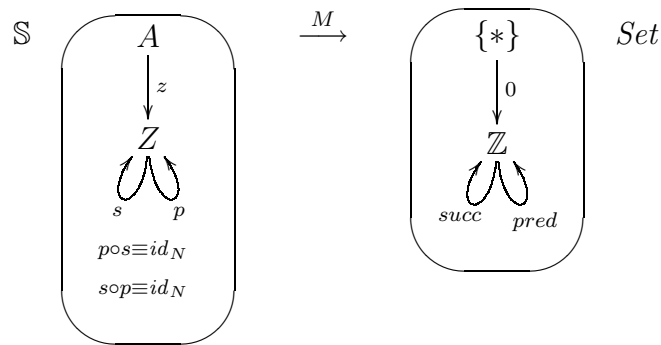


# Ambigraphs

More complex mosaics are *ambigraphs*: they are made of an oriented graph together with:

- *identity* arrows  $id_S : S \rightarrow S$  associated to some nodes  $S$ ,
- *composed* arrows  $g \circ f : S \rightarrow U$  associated to some consecutive pairs of arrows ( $f : S \rightarrow T, g : T \rightarrow U$ ),
- and *equations*, i.e. some pairs of arrows  $s_1 \equiv s_2$  where  $s_1, s_2 : S \rightarrow T$  have the same rank.

A *model*  $M$  of an ambigraph  $\mathbb{S}$  is a model of the underlying oriented graph such that identity (*resp.* composed) arrows are interpreted as identity (*resp.* composed) maps, and equations are interpreted as equalities between maps. Identities and composed arrows usually are not represented in the diagrams.

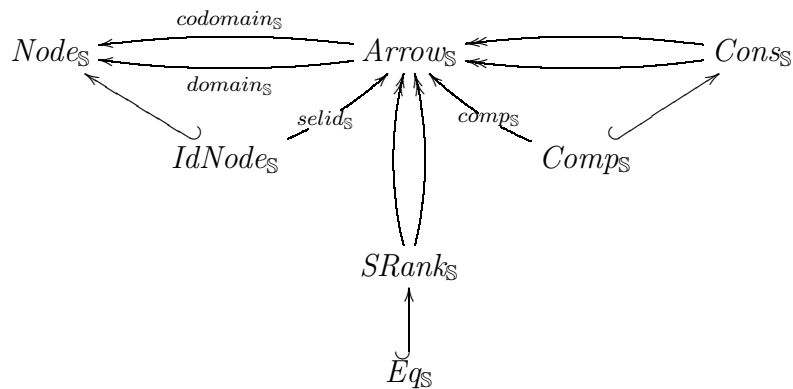


## Drawing the definition of an ambigraph

The definition of an ambigraph  $\mathbb{S}$  can be summarized by this diagram where:

- $Node_{\mathbb{S}}$  is the set of nodes of  $\mathbb{S}$ , and  $IdNode_{\mathbb{S}}$  the subset of nodes with identity;
- $Arrow_{\mathbb{S}}$  is the set of arrows of  $\mathbb{S}$ ;
- $Cons_{\mathbb{S}}$  is the set of consecutive pairs of arrows of  $\mathbb{S}$ , and  $Comp_{\mathbb{S}}$  the subset of composable pairs;
- $SRank_{\mathbb{S}}$  is the set of pairs of arrows of  $\mathbb{S}$  with the same rank, and  $Eq_{\mathbb{S}}$  the subset of equations;
- $domain_{\mathbb{S}}$  (resp.  $codomain_{\mathbb{S}}$ ) maps each arrow  $s : S \rightarrow T$  towards its domain  $S$  (resp. its codomain  $T$ );
- $selid_{\mathbb{S}}$  maps each node with identity  $S$  towards its identity arrow  $id_S$ ;
- $comp_{\mathbb{S}}$  maps each composable pair of arrows  $(f : S \rightarrow T, g : T \rightarrow U)$  towards its composed arrow  $g \circ f : S \rightarrow U$ ;
- other arrows are inclusions and projections.

We will come back on this point in section 5.



# Mosaics

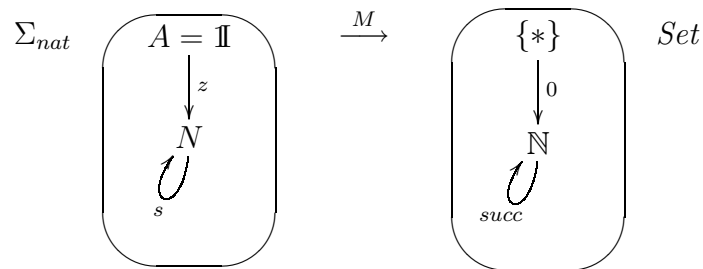
Generally, a *mosaic*  $\Sigma$  is made of an ambigraph  $\mathbb{S}$ , called the *support* of  $\Sigma$ , and additional syntactic features called *constraints*.

A *model*  $M$  of a mosaic  $\Sigma$  is a model of the support  $\mathbb{S}$  which *satisfies* the constraints.

Constraints themselves are described by ambigraphs, as will be seen in section 4. Currently, we only have to know that for any node  $S$  of  $\Sigma$ , there can be a constraint in  $\Sigma$ , which is written as “ $S = \mathbb{I}$ ”, such that a model  $M$  of the support  $\mathbb{S}$  satisfies the constraint  $S = \mathbb{I}$  if and only if the set  $M(S)$  has exactly one element. Note that one-element sets play a crucial role: arrows from  $S \rightarrow T$  where  $S = \mathbb{I}$  are interpreted as constant maps, *i.e.* as some elements of  $M(T)$ .

Here is a mosaic  $\Sigma_{nat}$  “of natural numbers”: its “minimal” model, as described below, interprets respectively the nodes  $A$  and  $N$  as the sets  $\{*\}$  (one-element set) and  $\mathbb{N}$  (set of natural numbers), and the arrows  $z$  and  $s$  as the maps  $0$  (constant map) and *succ* (for *successor*).

Mosaics may be classified according to the nature of their constraints: for instance Ehresmann’s sketches [9, 10] and Lair’s trames [20] are mosaics. Distinct natures of constraints correspond to distinct logical power [14, 15].



– 2 –

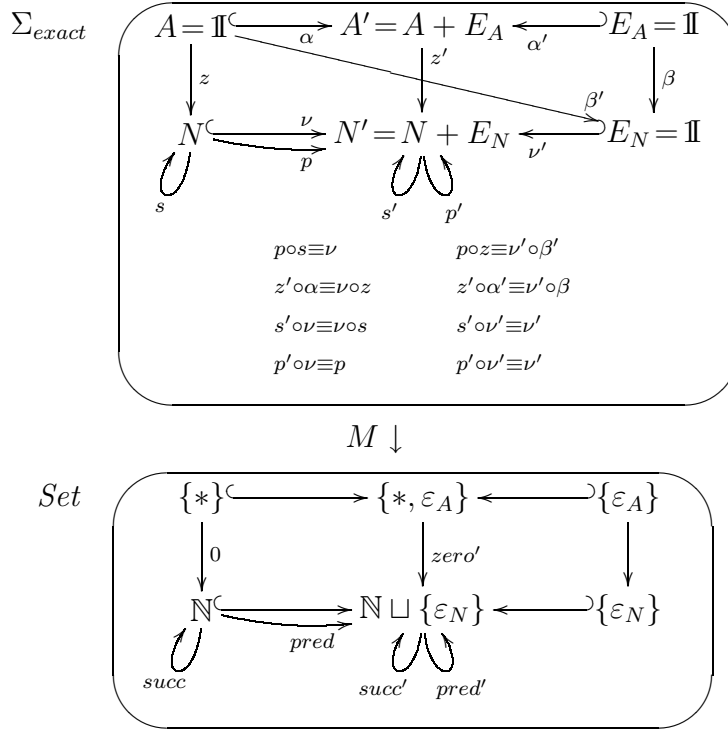
## An example about error handling

Let us enrich the previous mosaic  $\Sigma_{nat}$  in two ways: first to get a mosaic  $\Sigma_{exact}$  “of natural numbers with predecessor”, then to get an “approximation”  $\Sigma_{approx}$  (or simply  $\Sigma$ ) of  $\Sigma_{exact}$ .

## An exact mosaic for explicit error handling

The mosaic  $\Sigma_{exact}$  is a mosaic “of natural numbers with predecessor”. It has several constraints. As above, the constraints  $A = \mathbb{1}$ ,  $E_A = \mathbb{1}$  and  $E_N = \mathbb{1}$  specify one-element sets. In addition, the constraints  $A' = A + E_A$  and  $N' = N + E_N$  specify disjoint unions, which means that for any model  $M$  of  $\Sigma_{exact}$ , using “ $\sqcup$ ” for disjoint union:  $M(A') = M(A) \sqcup M(E_A)$  and  $M(N') = M(N) \sqcup M(E_N)$ . Altogether, these constraints say that  $M(A') = \{*, \varepsilon_A\}$  is a two-elements set, and that  $M(N') = M(N) \sqcup \{\varepsilon_N\}$ . Here the elements  $\varepsilon_A$  and  $\varepsilon_N$  are called the *errors*.

The “minimal” model of  $\Sigma_{exact}$  interprets  $A$ ,  $N$ ,  $z$  and  $s$  as before, hence it interprets  $A'$  and  $N'$  respectively as  $\{*, \varepsilon_A\}$ , and  $\mathbb{N} \sqcup \{\varepsilon_N\}$ ; in addition the arrow  $p$  is interpreted as the map *pred* (for *predecessor*) such that  $pred(succ(n)) = n$  and  $pred(0) = \varepsilon_N$ . The interpretations of  $z'$ ,  $s'$  and  $p'$  extend the maps  $0$ , *succ* and *pred* by *forwarding* the error:  $zero'(\varepsilon_A) = \varepsilon_N$ ,  $succ'(\varepsilon_N) = pred'(\varepsilon_N) = \varepsilon_N$ .

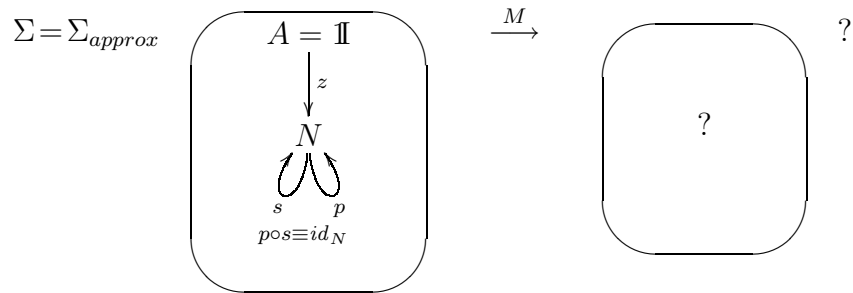


## An approximate mosaic for implicit error handling

The mosaic  $\Sigma = \Sigma_{approx}$  is made of:

*nodes* :  $A, N$   
*elementary arrows* :  $z : A \rightarrow N, s, p : N \rightarrow N$   
*identity arrow* :  $id_N : N \rightarrow N$   
*composed arrows* :  $p \circ z : A \rightarrow N, p \circ s : N \rightarrow N$   
*equation* :  $p \circ s \equiv id_N : N \rightarrow N$   
*constraint* :  $A = \mathbb{I}$

It is an *approximation* of  $\Sigma_{exact}$ : nothing is said about  $p \circ z$ . In order to say that the interpretation must contain some error elements, that maps should forward the error, and that  $pred(0)$  should be an error, we must add *comments*, which are not part of the syntax.

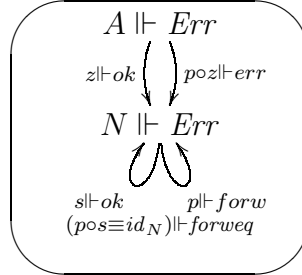


## Comments on the approximate mosaic

Comments on the support  $\mathbb{S}$  of  $\Sigma$  can be stated as follows:

- each node of  $\mathbb{S}$  must be interpreted as an *Err-set*, i.e. a set with a specific element called the *error*,
- each arrow of  $\mathbb{S}$  must be interpreted as a *forward-map*, i.e. a map which forwards the error, and, on top of that :
  - the interpretation of  $z$  and  $s$  must be an *ok-map*, i.e. a map which cannot associate an error to a non-error element,
  - the interpretation of  $p \circ z$  must be an *err-map*, i.e. a map which always returns the error,
- the equation of  $\mathbb{S}$  must be interpreted as a *forwardeq-equality*, i.e. an equality between the corresponding *forward-maps*.

$\mathbb{S} \Vdash \text{comments}$



Comment on the constraint  $A = \mathbb{1}$  is:

- $A$  must be interpreted as an *Err-set* with exactly one non-error element: hence as a two-elements set  $\{*, \varepsilon_A\}$ .

## Explicit versus implicit (informal)

Let us summarize: in order to specify natural numbers with a predecessor map  $pred$  such that  $pred(0)$  is an error, we may either choose the *explicit* point of view given by the exact mosaic  $\Sigma_{exact}$ , or the *implicit* point of view given by the approximate mosaic  $\Sigma$  together with comments.

In the explicit point of view, we are interested in the models of  $\Sigma_{exact}$  as defined above, *i.e.* its *set-valued* models. Their set is denoted:

$$Mod(\Sigma_{exact}, Set) .$$

In the implicit point of view, on the contrary, we are *not* interested in the set-valued models of  $\Sigma$ , but on some new kind of models, which we call (temporary) *the models of  $\Sigma$  according to the comments*. Their set is denoted (temporary):

$$Mod^{comments}(\Sigma) .$$

We have “proved” that:

$$\boxed{Mod(\Sigma_{exact}, Set) \simeq Mod^{comments}(\Sigma) .}$$

We will now see that:

- comments can be made “syntactic” and  $Mod^{comments}(\Sigma)$  can be given a more “syntactic” definition,
- the mosaic  $\Sigma_{exact}$  can be recovered from its approximation  $\Sigma$  and from the comments.

This will be done first for the support of  $\Sigma$  in section 3 and then for its constraint in section 4.

– 3 –

## Comments on the support

Our basic result relies on a “good” formalization of the comments. For this purpose, we consider here only the support of the approximate mosaic; constraints will be considered in section 4.

## Extensor

Let us consider the following mosaics.

The mosaic  $\mathbb{K}(Node, Err)$  has two constraints: as above, the constraint  $E = \mathbb{I}$  specifies a one-element set, and the constraint  $T = S + E$  specifies a disjoint union. Hence a model  $M$  of  $\mathbb{K}(Node, Err)$  is made of a set  $M(T) = M(S) \sqcup M(E)$  with  $M(E) = \{\varepsilon\}$ :

*models of  $\mathbb{K}(Node, Err)$  are Err-sets.*

$$\mathbb{K}(Node, Err) \quad \left( S \hookrightarrow T = S + E \longleftarrow E = \mathbb{I} \right)$$

Each “line” in the other mosaics is a copy of  $\mathbb{K}(Node, Err)$ .

A model  $M$  of  $\mathbb{K}(Arrow, forw)$  is made of a map  $M(t) : M(S_1) \sqcup \{\varepsilon_1\} \rightarrow M(S_2) \sqcup \{\varepsilon_2\}$  which forwards the error, *i.e.* such that  $M(t)(\varepsilon_1) = \varepsilon_2$ :

*models of  $\mathbb{K}(Arrow, forw)$  are forward-maps.*

$$\mathbb{K}(Arrow, forw) \quad \left( \begin{array}{ccc} S_1 \hookrightarrow T_1 = S_1 + E_1 \longleftarrow E_1 = \mathbb{I} & & \\ & \downarrow t & \equiv \quad \downarrow \\ S_2 \hookrightarrow T_2 = S_2 + E_2 \longleftarrow E_2 = \mathbb{I} & & \end{array} \right)$$

If in addition  $M(t)$  cannot associate an error to a non-error element, *i.e.* if the image of  $M(S_1)$  by  $M(t)$  is in  $M(S_2)$ :

*models of  $\mathbb{K}(Arrow, ok)$  are ok-maps.*

$$\mathbb{K}(Arrow, ok) \quad \left( \begin{array}{ccc} S_1 \hookrightarrow T_1 = S_1 + E_1 \longleftarrow E_1 = \mathbb{I} & & \\ \downarrow & \equiv & \downarrow t \quad \equiv \quad \downarrow \\ S_2 \hookrightarrow T_2 = S_2 + E_2 \longleftarrow E_2 = \mathbb{I} & & \end{array} \right)$$

If  $M(t)$  always returns the error, *i.e.* if the image of every element of  $M(S_1)$  by  $M(t)$  is  $\varepsilon_2$ :

*models of  $\mathbb{K}(Arrow, err)$  are err-maps.*

$$\mathbb{K}(Arrow, err) \quad \left( \begin{array}{ccccc} S_1 & \hookrightarrow & T_1 = S_1 + E_1 & \longleftarrow & E_1 = \mathbb{I} \\ & & \downarrow t & \searrow \equiv & \downarrow \\ S_2 & \hookrightarrow & T_2 = S_2 + E_2 & \longleftarrow & E_2 = \mathbb{I} \end{array} \right)$$

A model  $M$  of  $\mathbb{K}(Eq, forweq)$  is made of two *forward*-maps  $M(t_1), M(t_2) : M(S_1) \sqcup \{\varepsilon_1\} \rightarrow M(S_2) \sqcup \{\varepsilon_2\}$  such that  $M(t_1) = M(t_2)$ :

*models of  $\mathbb{K}(Eq, forweq)$  are equalities between forward-maps.*

$$\mathbb{K}(Eq, forweq) \quad \left( \begin{array}{ccccc} S_1 & \hookrightarrow & T_1 = S_1 + E_1 & \longleftarrow & E_1 = \mathbb{I} \\ & & \downarrow t_1 \left( \equiv \right) t_2 & \searrow \equiv & \downarrow \left( \equiv \right) \\ S_2 & \hookrightarrow & T_2 = S_2 + E_2 & \longleftarrow & E_2 = \mathbb{I} \end{array} \right)$$

The “family”  $\mathbb{K}$  of these mosaics  $\mathbb{K}(\dots)$  is called an *extensor*.

It should be noted that  $\mathbb{K}(Arrow, ok)$  (*resp.*  $\mathbb{K}(Arrow, err)$ ) can be embedded into  $\mathbb{K}(Arrow, forw)$ .

## Indexation of an ambigraph

Let us come back to the support  $\mathbb{S}$  of the mosaic  $\Sigma$ .

Thanks to the extensor  $\mathbb{K}$ , comments associated to  $\mathbb{S}$  can now be stated as follows:

- the nodes of  $\mathbb{S}$  must be interpreted as models of  $\mathbb{K}(Node, Err)$ ,
- the arrows of  $\mathbb{S}$  must be interpreted as models of  $\mathbb{K}(Arrow, forw)$  and, in addition:
- $z$  and  $s$  must be interpreted as models of  $\mathbb{K}(Arrow, ok)$ ,
- $p \circ z$  must be interpreted as a model of  $\mathbb{K}(Arrow, err)$ ,
- the equation of  $\mathbb{S}$  must be interpreted as a model of  $\mathbb{K}(Eq, forweq)$ .

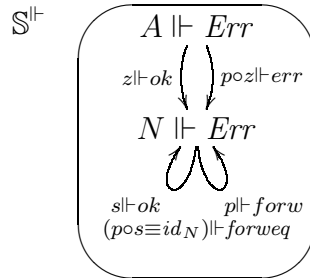
Let  $\Vdash$  denote the congruence relation on  $\mathbb{S}$  generated by:

- nodes* :  $A, N \Vdash Err$
- arrows* :  $p \Vdash forw$  ,  $z, s \Vdash ok$  ,  $p \circ z \Vdash err$
- equation* :  $(p \circ s \equiv id_N) \Vdash forweq$

This congruence relation  $\Vdash$  is called an *indexation*. Since each *ok*-map and each *err*-map is a *forward*-map, since identities are *ok*-maps, and since the composition of an *ok*-map and a *forward*-map is a *forward*-map, the congruence  $\Vdash$  is such that:

- arrows* :  $z, s, p, p \circ z, p \circ s, id_N \Vdash forw$  ,  $z, s, id_N \Vdash ok$  ,  $p \circ z \Vdash err$ .

The *indexed ambigraph*  $\mathbb{S}^{\Vdash}$  is made of the ambigraph  $\mathbb{S}$  together with the indexation  $\Vdash$ .



## Explicit versus implicit (for support)

We have seen in section 2 that:

$$Mod(\Sigma_{exact}, Set) \simeq Mod^{comments}(\Sigma) .$$

Similarly, if  $\Sigma_{exact}^0$  denote the mosaic obtained by forgetting about the constraint  $A = \mathbb{1}$  in  $\Sigma_{exact}$ :

$$Mod(\Sigma_{exact}^0, Set) \simeq Mod^{comments}(\mathbb{S}) .$$

We have just seen that the “models of  $\mathbb{S}$  according to the comments” can be described as follows: the interpretation of each node, arrow and equation of  $\mathbb{S}$  is a set-valued model of the corresponding mosaic  $\mathbb{K}(\dots)$  (the correspondence is described by  $\Vdash$ ). They are called the *models of  $\mathbb{S}^\Vdash$  in the set-valued models of  $\mathbb{K}$* , and their set is denoted  $Mod(\mathbb{S}^\Vdash, Mod(\mathbb{K}(\dots), Set))$ . In this way we get a more “syntactic” description of the “models of  $\mathbb{S}$  according to the comments” (see section 5 for more details):

$$Mod^{comments}(\mathbb{S}) = Mod(\mathbb{S}^\Vdash, Mod(\mathbb{K}(\dots), Set)) .$$

It follows that:

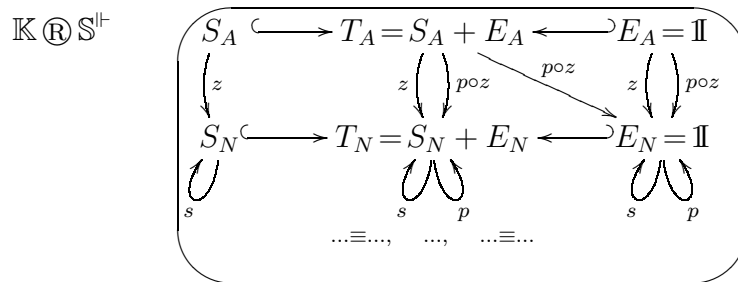
$$\boxed{Mod(\Sigma_{exact}^0, Set) \simeq Mod(\mathbb{S}^\Vdash, Mod(\mathbb{K}(\dots), Set)) .}$$

## Ribbon product (for support)

On the other hand, we can now forget about  $\Sigma_{exact}^0$  and make the following construction from  $\mathbb{S}^{\uparrow}$  and  $\mathbb{K}$ :

1. for each node, arrow and equation of  $\mathbb{S}$ , take a copy of the corresponding mosaic  $\mathbb{K}(\dots)$  (the correspondence is described by the indexation  $\uparrow$ ),
2. then glue them together, according to the way nodes, arrows and equations of  $\mathbb{S}$  are glued together.

The resulting mosaic is called the *ribbon product of  $\mathbb{S}^{\uparrow}$  by  $\mathbb{K}$*  and is denoted  $\mathbb{K} \circledast \mathbb{S}^{\uparrow}$ .



## Main result (for support)

It is easy to see that  $\mathbb{K} \textcircled{R} \mathbb{S}^{\text{!}}$  is *equivalent* to  $\Sigma_{exact}^0$ , in the sense that they have the same models. Hence we have “proven”:

$$\mathbb{K} \textcircled{R} \mathbb{S}^{\text{!}} \simeq \Sigma_{exact}^0 .$$

This result says that the explicit specification  $\Sigma_{exact}^0$  can be recovered from the approximate specification  $\mathbb{S}$  (without constraint), thanks to an appropriate formalization of the comments.

As a consequence, it is useless to build the explicit specification  $\Sigma_{exact}^0$ ! In addition, this result brings out the fact that  $\Sigma_{exact}^0$  has a structure: it says which part of  $\Sigma_{exact}^0$  is used for dealing with natural numbers, and which part is devoted to error handling. This can be compared with an equality like  $2^4 3^2 = 144$ : when this number is written as  $2^4 3^2$  we have some information on its structure which is not so easy to recover from 144.

Finally, we get immediately:

**Theorem.**

$$\boxed{Mod(\mathbb{K} \textcircled{R} \mathbb{S}^{\text{!}}, Set) \simeq Mod(\mathbb{S}^{\text{!}}, Mod(\mathbb{K}(\dots), Set)) .}$$

This result says that the set-valued models of the ribbon product  $\mathbb{K} \textcircled{R} \mathbb{S}^{\text{!}}$  are the models of  $\mathbb{S}^{\text{!}}$  in the set-valued models of  $\mathbb{K}$ , i.e. they are the “models of  $\mathbb{S}$  according to the comments”.

Note that this result is an analog of the fundamental property of the *tensor product* in linear algebra, or rather of *scalar extension* in commutative algebra.

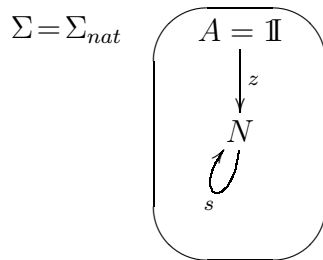
## Comments on the constraints

We now extend the basic result obtained in section 3 from  $\mathbb{S}$  to  $\Sigma$ , by considering the constraint of the approximate mosaic  $\Sigma$ . For this purpose, let us first define a constraint in term of ambigraphs.

From section 1, we know that a mosaic  $\Sigma$  is made of a support  $\mathbb{S}$  (which is an ambigraph) and some *constraints*, and that a model of  $\Sigma$  is a model of  $\mathbb{S}$  which *satisfies* the constraints.

Actually, constraints themselves are made of ambigraphs and *functors* between ambigraphs: a functor  $F : \mathbb{S} \rightarrow \mathbb{S}'$  associates to each node, arrow, equation of  $\mathbb{S}$  a node, arrow, equation of  $\mathbb{S}'$  in a “coherent way”.

The constraint “ $A = \mathbb{I}$ ” in our mosaic  $\Sigma_{nat}$ , though very simple, will give us a good idea of the general definition of constraints.



## Terminal sets

The constraint “ $A = \mathbb{1}$ ” in  $\Sigma_{nat}$  says that we are interested in the models of  $\mathbb{S}$  such that:

- the interpretation of  $A$  is a one-element set.

Actually, one-element sets can easily be characterized by a more “abstract” property: they are the *terminal* sets, *i.e.* the interpretation  $M(A)$  must satisfy: there is exactly one map from any set  $X$  to  $M(A)$ . Or, more formally:

*for all set  $X$  there exists only one map  $f : X \rightarrow M(A)$ .*

Hence, we are interested in the models of  $\mathbb{S}$  such that:

- the interpretation of  $A$  is a terminal set.

$$\text{Set} \quad \left( \begin{array}{c} \forall X \\ \exists! \downarrow f \\ M(A) \end{array} \right)$$

## Constraint for a terminal node

Formally, the constraint  $\Delta$  which says that the interpretation of  $A$  must be a terminal set is made of three ambigraphs  $\mathbb{C}$ ,  $\mathbb{D}$  and  $\mathbb{U}$ , a functor  $pick : \mathbb{C} \rightarrow \mathbb{S}$  and a pair of functors:  $forall : \mathbb{C} \rightarrow \mathbb{D}$ ,  $exists : \mathbb{D} \rightarrow \mathbb{U}$ .

The aim of  $pick : \mathbb{C} \rightarrow \mathbb{S}$  is to pick out the part of  $\mathbb{S}$  which is subject to the constraint: here it is the node  $A$ , so that the ambigraph  $\mathbb{C}$  is made of one node  $C$  (nothing else) and  $pick$  is such that  $pick(C) = A$ . Each model  $M$  of  $\mathbb{S}$  defines a model  $M \circ pick$  of  $\mathbb{C}$ , such that  $M \circ pick(C) = M(A)$ .

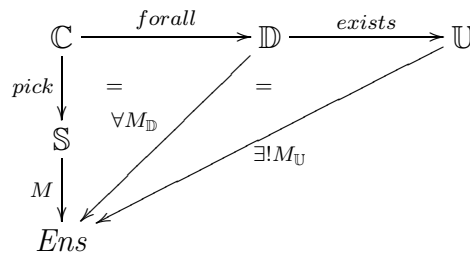
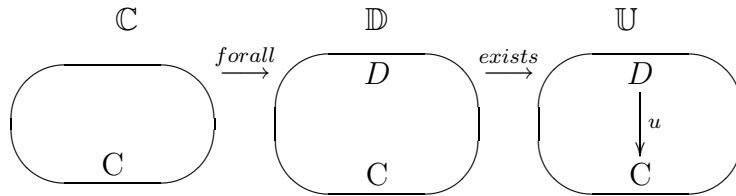
The ambigraph  $\mathbb{D}$  is made of two nodes  $C$  and  $D$ , the ambigraph  $\mathbb{U}$  is made of two nodes  $C$  and  $D$  together with an arrow  $u : D \rightarrow C$ , and the functors  $forall : \mathbb{C} \rightarrow \mathbb{D}$  and  $exists : \mathbb{D} \rightarrow \mathbb{U}$  are the obvious inclusions.

Then, for each model  $M$  of  $\mathbb{S}$ , the terminality property of the set  $M(A)$ :

*for all set  $X$  there exists only one map  $f : X \rightarrow M(A)$ .*

can be re-stated as follows:

*for all model  $M_{\mathbb{D}}$  of  $\mathbb{D}$  which extends  $M \circ pick$   
there exists only one model  $M_{\mathbb{U}}$  of  $\mathbb{U}$  which extends  $M_{\mathbb{D}}$ .*



## Indexation of a constraint

Let us now come back to the approximate mosaic  $\Sigma = \Sigma_{approx}$  of section 2. It has one constraint : “ $A = \mathbb{I}$ ”, and the comment associated to this constraint is:

- the interpretation of  $A$  is an *Err*-set with exactly one non-error element (hence it is a two-elements set).

It is equivalent to the following property: there is exactly one *ok*-map from any *Err*-set  $X' = X \sqcup \{\varepsilon\}$  to  $M(A)$ . Which can be stated as:

*for all Err-set  $X'$  there exists only one ok-map  $f : X' \rightarrow M(A)$ .*

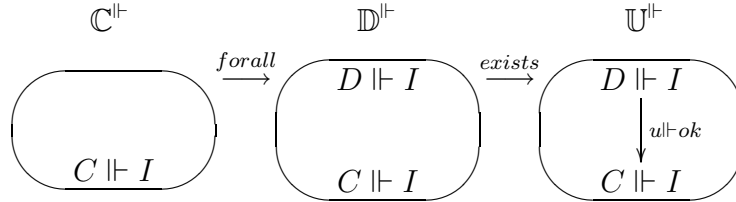
Or as:

- the interpretation of  $A$  is a terminal *Err*-set with respect to *ok*-maps.

Hence it is “natural” to define an indexation  $\Vdash_{Delta}$  (or  $\Vdash$ ) on the constraint  $\Delta$ , i.e. on the three ambigraphs occuring in the constraint, in the following way:  $C, D \Vdash_{Delta} Err$  ,  $u \Vdash_{Delta} ok$ .

The *indexation* relation on  $\Sigma$  is defined by  $\Vdash$  on the support  $\mathbb{S}$  and  $\Vdash_{Delta}$  on the constraint  $\Delta$ ; it is still denoted  $\Vdash$ .

The *indexed mosaic*  $\Sigma^{\Vdash}$  is made of the mosaic  $\Sigma$  with this indexation  $\Vdash$ .



## Explicit versus implicit (for mosaic)

We have seen in section 2 that:

$$Mod(\Sigma_{exact}, Set) \simeq Mod^{comments}(\Sigma) .$$

We have just seen that the “models of  $\Sigma$  according to the comments” are the models of  $\mathbb{S}^{\perp}$  in the models of  $\mathbb{K}$  such that the interpretation of  $A$  is a terminal *Err*-set with respect to *ok*-maps.

They are called the *models of  $\Sigma^{\perp}$  in the models of  $\mathbb{K}$* , and their set is denoted  $Mod(\Sigma^{\perp}, Mod(\mathbb{K}(\dots), Set))$ , so that we get a more “syntactic” description of the “models of  $\Sigma$  according to the comments”:

$$Mod^{comments}(\Sigma) = Mod(\Sigma^{\perp}, Mod(\mathbb{K}(\dots), Set)) .$$

It follows that:

$$\boxed{Mod(\Sigma_{exact}, Set) \simeq Mod(\Sigma^{\perp}, Mod(\mathbb{K}(\dots), Set)) .}$$

## Ribbon product (for mosaic)

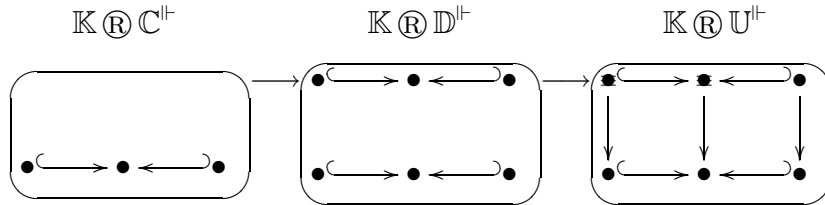
Let us now forget about  $\Sigma_{exact}$ .

In section 3 was defined the ribbon product  $\mathbb{K} \otimes \mathbb{S}^{\text{lt}}$ . Similarly we can define the ribbon products  $\mathbb{K} \otimes \mathbb{C}^{\text{lt}}$ ,  $\mathbb{K} \otimes \mathbb{D}^{\text{lt}}$  and  $\mathbb{K} \otimes \mathbb{U}^{\text{lt}}$ . Since functors *pick*, *forall* and *exists* preserve the indexation, they “naturally” give rise to functors:  $\mathbb{K} \otimes \textit{pick} : \mathbb{K} \otimes \mathbb{C} \rightarrow \mathbb{K} \otimes \mathbb{S}$ ,  $\mathbb{K} \otimes \textit{forall} : \mathbb{K} \otimes \mathbb{C} \rightarrow \mathbb{K} \otimes \mathbb{D}$  and  $\mathbb{K} \otimes \textit{exists} : \mathbb{K} \otimes \mathbb{D} \rightarrow \mathbb{K} \otimes \mathbb{U}$ .

This defines a constraint  $\mathbb{K} \otimes \Delta^{\text{lt}}$  on  $\mathbb{K} \otimes \mathbb{S}^{\text{lt}}$ . Its meaning is precisely the right one:

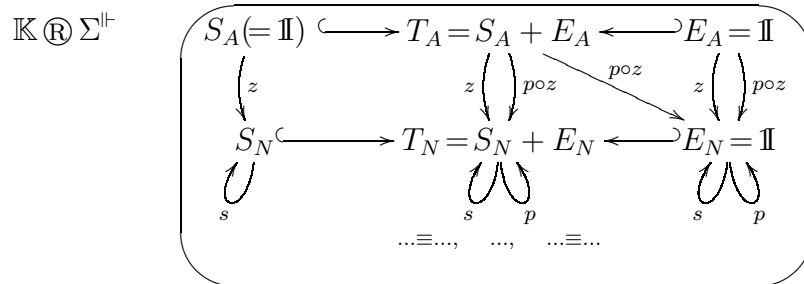
*for all Err-set  $X'$  there exists only one ok-map  $f : X' \rightarrow M(A)$ .*

*In categorical terms, the definition of  $\mathbb{K} \otimes \Delta^{\text{lt}}$  comes from the functoriality of the ribbon product of ambigraphs by  $\mathbb{K}$ .*



The mosaic  $\mathbb{K} \otimes \mathbb{S}^{\text{lt}}$  together with the constraint  $\mathbb{K} \otimes \Delta^{\text{lt}}$  is a mosaic called the *ribbon product of  $\Sigma^{\text{lt}}$  by  $\mathbb{K}$*  and denoted  $\mathbb{K} \otimes \Sigma^{\text{lt}}$ .

In addition, it is easy to see that the constraint  $\mathbb{K} \otimes \Delta^{\text{lt}}$  on  $(S_A \rightarrow T_A \leftarrow E_A)$  is equivalent to the constraint  $S_A = \mathbb{I}$ .



## Main result (for mosaic)

It is easy to see that  $\mathbb{K} \circledast \Sigma^{\text{lf}}$  is *equivalent* to  $\Sigma_{\text{exact}}$ , in the sense that they have the same models. Hence we have “proven”, for this example:

$$\mathbb{K} \circledast \Sigma^{\text{lf}} \simeq \Sigma_{\text{exact}} .$$

And we get immediately our main result:

**Theorem.**

$$\boxed{Mod(\mathbb{K} \circledast \Sigma^{\text{lf}}, Set) \simeq Mod(\Sigma^{\text{lf}}, Mod(\mathbb{K}(\dots), Set)) .}$$

This result says that the set-valued models of the ribbon product  $\mathbb{K} \circledast \Sigma^{\text{lf}}$  are the models of  $\Sigma^{\text{lf}}$  in the set-valued models of  $\mathbb{K}$ , *i.e.* the “models of  $\Sigma$  according to the comments”.

## Mosaics for the specification of ambigraphs

We have seen in section 4 that once everything is defined and proved for the support of the approximate specification, definitions and proofs for its constraints follow “by functoriality of the ribbon product”.

Hence, let us now come back to the main result on supports, as stated at the end of section 3:

$$\text{Mod}(\mathbb{K} \otimes \mathbb{S}^{\text{tr}}, \text{Set}) \simeq \text{Mod}(\mathbb{S}^{\text{tr}}, \text{Mod}(\mathbb{K}(\dots), \text{Set})) .$$

In order to understand the meaning of this result, and especially of its right handsize, it is necessary to “specify the ambigraphs”...

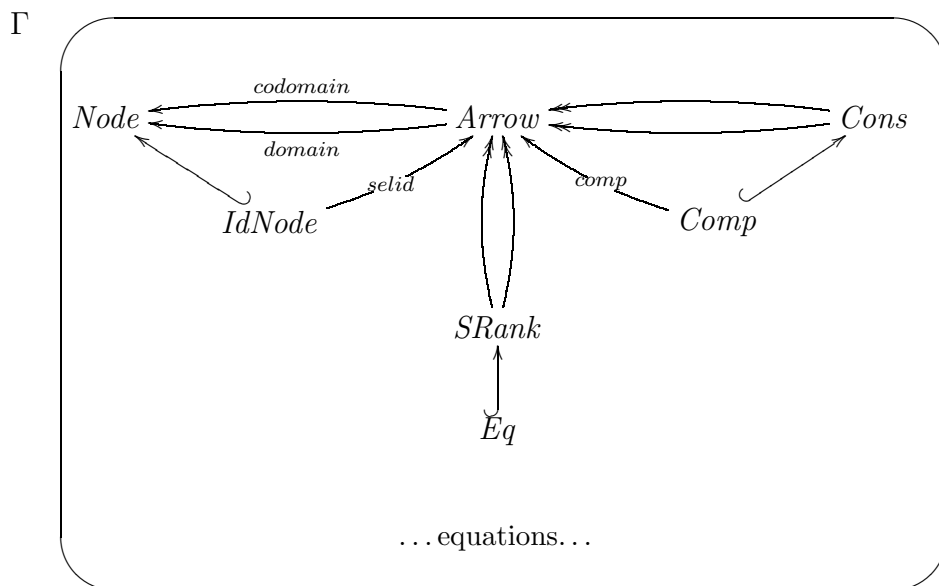
## The mosaic $\Gamma$ of ambigraphs

Ambigraphs, as defined in section 1, are themselves the set-valued models of a mosaic which we call  $\Gamma$ :

*nodes* :  $Node, Arrow, Comp, Eq, \dots$   
*arrows* :  $domain, codomain : Arrow \rightarrow Node, \dots$   
*equations* :  $\dots$   
*constraints* :  $\dots$

A “proof” of this fact is given by the comparison between the diagram for  $\Gamma$  and the diagram for the definition of an ambigraph in section 1, where  $Nodes_{\mathbb{S}}$  stands for  $\mathbb{S}(Node)$ , etc.

We are now interested in the ambigraph  $\mathbb{S}$  as a model of  $\Gamma$ .



## The indexor $\mathbb{I}$

Quite a lot of specifications have been used to build the ribbon product  $\mathbb{K} \otimes \mathbb{S}^{\text{!}}$ : namely the approximate ambigraph  $\mathbb{S}^{\text{!}}$  and the mosaics in the extensor  $\mathbb{K}$ .

Actually one is still missing: it is the *indexor* ambigraph  $\mathbb{I}$ , made of the “names of the comments”. Precisely here, the ambigraph  $\mathbb{I}$  is:

*node* :  $Err$

*arrows* :  $forw, ok, err$

It has one identity arrow  $id_{Err} = ok$ , and each pair of arrows can be composed:

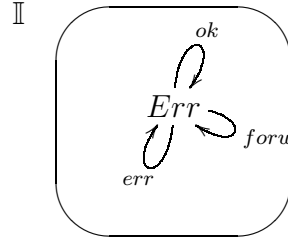
$i \circ ok = ok \circ i = i$  for each arrow  $i$  of course,

$err \circ i = i \circ err = err$  for each arrow  $i$ ,

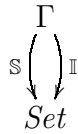
and  $forw \circ forw = forw$ .

This composition law translates the properties of maps: for instance

$forw = forw \circ ok$  because if  $f$  is an *ok*-map and  $g$  is a *forward*-map then  $g \circ f$  is a *forward*-map.



The indexor  $\mathbb{I}$  is another set-valued model of  $\Gamma$ , and the indexation  $\Vdash$  is a relation from  $\mathbb{S}$  to  $\mathbb{I}$ .



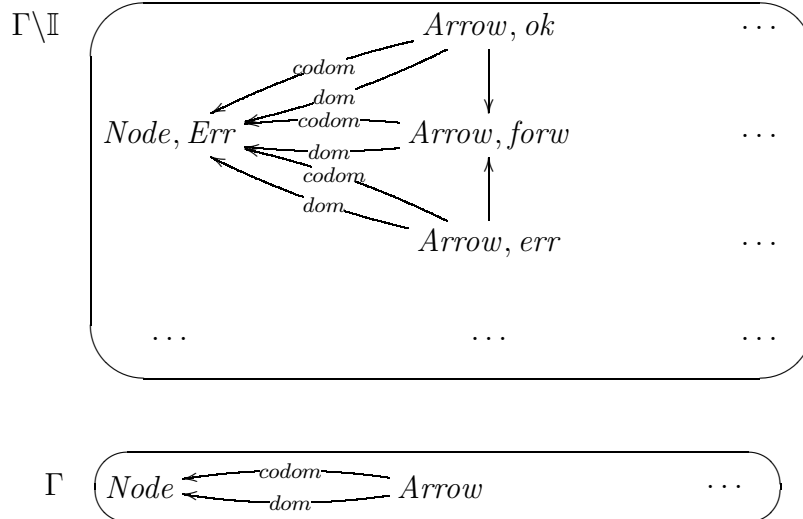
In addition,  $\mathbb{I}$  is endowed with an order relation  $\Rightarrow$ , generated by:  $ok \Rightarrow forw$ ,  $err \Rightarrow forw$ . This relation translates the following property of maps: each *ok*-map and each *err*-map is a *forward*-map.

## The mosaic $\Gamma \setminus \mathbb{I}$ of indexed ambigraphs

Now let us build a mosaic  $\Gamma \setminus \mathbb{I}$  by “counting each node  $G$  in  $\Gamma$  once for each element in  $\mathbb{I}(G)$ ”, and taking into account the order relation over  $\mathbb{I}$ :

*nodes* :  $(Node, Err)$ ,  
 $(Arrow, forw)$ ,  $(Arrow, ok)$ ,  $(Arrow, err)$ ,  
 $(Equation, forweq)$ ,  
 $\dots$   
*arrows* :  $(domain, forw) : (Arrow, forw) \rightarrow (Node, Err)$ ,  
 $(codomain, forw) : (Arrow, forw) \rightarrow (Node, Err)$ ,  
 $\dots$   
 $(ok \Rightarrow forw) : (Arrow, ok) \rightarrow (Arrow, forw)$ ,  
 $(err \Rightarrow forw) : (Arrow, err) \rightarrow (Arrow, forw)$ ,  
 $\dots$   
*equations* :  $\dots$   
*constraints* :  $\dots$

This mosaic  $\Gamma \setminus \mathbb{I}$  is called the *blow-up of  $\Gamma$  with respect to  $\mathbb{I}$* .



## $\mathbb{S}^{\text{!}}$ is a model of $\Gamma \setminus \mathbb{I}$

The blow-up  $\Gamma \setminus \mathbb{I}$  of  $\Gamma$  with respect to  $\mathbb{I}$  has the following property:

**Proposition.** *Indexed ambigraphs can be identified with models of  $\Gamma \setminus \mathbb{I}$ .*

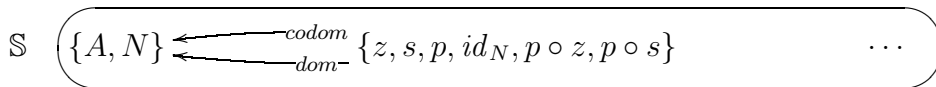
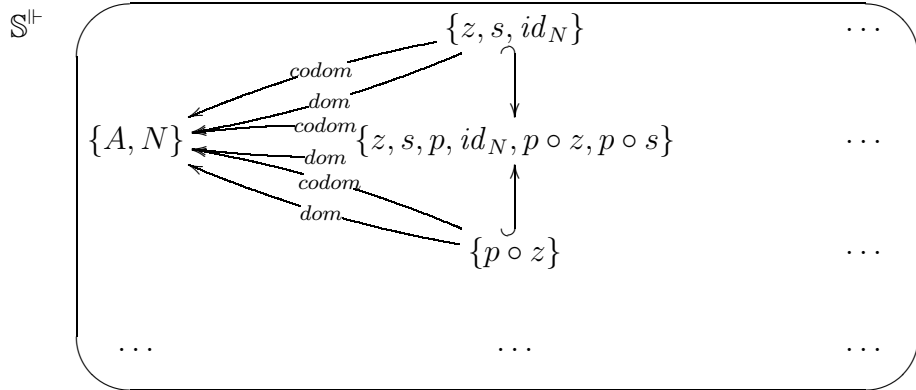
The interpretation of the node  $(Node, Err)$  by the model  $\mathbb{S}^{\text{!}}$  of  $\Gamma \setminus \mathbb{I}$  is the subset  $\mathbb{S}^{\text{!}}(Node, Err)$  of  $\mathbb{S}(Node)$  made of the nodes  $S$  in  $\mathbb{S}$  such that  $S \Vdash Err$ :  
 $\mathbb{S}^{\text{!}}(Node, Err) = \mathbb{S}(Node) = \{A, N\}$ .

For  $i$  among *forw*, *ok* and *err*, the interpretation of the node  $(Arrow, i)$  is the subset  $\mathbb{S}^{\text{!}}(Arrow, i)$  of  $\mathbb{S}(Arrow)$  made of the arrows  $s$  in  $\mathbb{S}$  such that  $s \Vdash i$ :

$$\mathbb{S}^{\text{!}}(Arrow, forw) = \mathbb{S}(Arrow) = \{z, s, p, id_N, p \circ z, p \circ s\},$$

$$\mathbb{S}^{\text{!}}(Arrow, ok) = \{z, s, id_N\},$$

$$\mathbb{S}^{\text{!}}(Arrow, err) = \{p \circ z\}.$$



## $\mathbb{K}$ is a model of $\Gamma \setminus \mathbb{I}$

Actually, the nodes of  $\Gamma \setminus \mathbb{I}$  were used in section 3 as arguments to the mosaics  $\mathbb{K}(\dots)$  in the extensor  $\mathbb{K}$ .

Indeed,  $\mathbb{K}$  can be considered as a “model” of  $\Gamma \setminus \mathbb{I}$ . Not a set-valued model, but a *mosaic-valued model*, i.e. a model which takes its values in mosaics instead of sets.

A model of  $\Gamma \setminus \mathbb{I}$  must interpret its arrows *etc.* as well as its nodes. It is easy to see that indeed to each arrow in  $\Gamma \setminus \mathbb{I}$  can be associated an homomorphism between the corresponding mosaics  $\mathbb{K}(\dots)$ , except that the direction is changed. For instance there is in  $\Gamma \setminus \mathbb{I}$  an arrow:

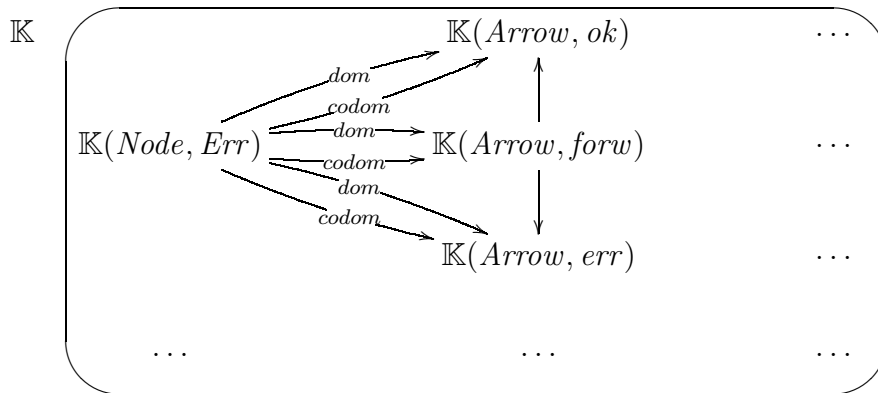
$$(domain, forw) : (Arrow, forw) \rightarrow (Node, Err)$$

and there is an homomorphism in the opposite direction:

$$\mathbb{K}(domain, forw) : \mathbb{K}(Node, Err) \rightarrow \mathbb{K}(Arrow, forw)$$

which associates to the mosaic  $\mathbb{K}(Node, Err)$  the “top-line” in  $\mathbb{K}(Arrow, forw)$ .

**Proposition.**  $\mathbb{K}$  is a contravariant *mosaic-valued model* of  $\Gamma \setminus \mathbb{I}$ .



## $Mod(\mathbb{K}(\dots), Set)$ is a model of $\Gamma \setminus \mathbb{I}$

In order to interpret  $\mathbb{S}$  in section 3, we considered the set-valued models of  $\mathbb{K}$ . Now “ $Mod(\mathbb{K}(\dots), Set)$ ” can be defined precisely: it is composed of  $\mathbb{K}$ , from  $\Gamma \setminus \mathbb{I}$  to mosaics, followed by  $Mod(\dots, Set)$ , from mosaics to sets. Note that both  $\mathbb{K}$  and  $Mod(\dots, Set)$  turn the arrows over, so that their composite preserves the direction of arrows!

$Mod(\mathbb{K}(\dots), Set)$  is made of the sets:

$$Mod(\mathbb{K}(Node, Err), Set)$$

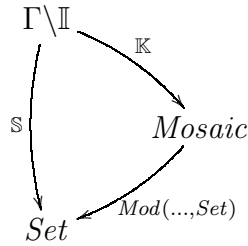
and so on, with the maps:

$$Mod(\mathbb{K})(domain, forw) : Mod(\mathbb{K})(Arrow, forw) \rightarrow Mod(\mathbb{K})(Node, Err)$$

and so on, which are now in the “right” direction.

**Proposition.**  $Mod(\mathbb{K}(\dots), Set)$  is a set-valued model of  $\Gamma \setminus \mathbb{I}$ .

Now both  $\mathbb{S}^{\dagger}$  and  $Mod(\mathbb{K}(\dots), Set)$  have the same structure: they are set-valued models of  $\Gamma \setminus \mathbb{I}$ . It follows that *models* of  $\mathbb{S}^{\dagger}$  in  $Mod(\mathbb{K}(\dots), Set)$  can be defined as *morphisms*, i.e. as structure-preserving maps.



Then, the definition of “models” of  $\Sigma^{\dagger}$  in  $Mod(\mathbb{K}(\dots), Set)$  takes care of the constraint of  $\Sigma$ .

## Overloading and coercions

The following example deals with some operations on integers and rational numbers. These operations are chosen unary so as to avoid constraints in the mosaics, however it would be easy to generalize this example to constant or binary operations. We present an indexor  $\mathbb{I}$  and an extensor  $\mathbb{K}$  which can easily be adapted to deal with general overloading and coercions.

Let us consider the following operations on “numbers”: *mul* for multiplication by 2 and *div* for division by 2. Let our “numbers” be either integers (elements of  $\mathbb{Z}$ ) or rational numbers (elements of  $\mathbb{Q}$ ).

We consider that *mul* is both an operation from  $\mathbb{Z}$  to  $\mathbb{Z}$  and from  $\mathbb{Q}$  to  $\mathbb{Q}$ : we say that *mul* is *overloaded* since one name (*mul*) represents several (here: two) operations. Similarly, *div* is overloaded: it is both an operation from  $\mathbb{Z}$  to  $\mathbb{Q}$  and from  $\mathbb{Q}$  to  $\mathbb{Q}$ .

In addition, we will use a *coercion* in order to take into account the inclusion of  $\mathbb{Z}$  in  $\mathbb{Q}$ , and the fact that the overloaded operations act “coherently” on  $\mathbb{Z}$  and  $\mathbb{Q}$ .

## An exact mosaic and an approximate mosaic

The exact mosaic  $\Sigma_{exact}$  is made of:

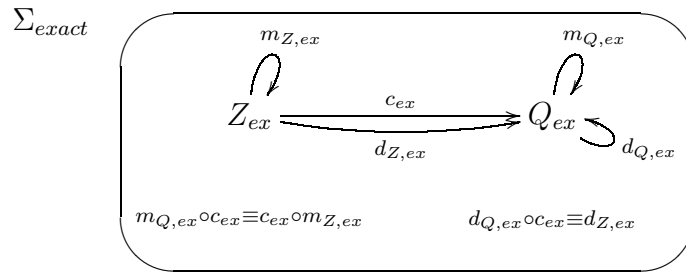
nodes :  $Z_{ex}, Q_{ex}$

arrows :  $c_{ex} : Z_{ex} \rightarrow Q_{ex}$ ,

$m_{Z,ex} : Z_{ex} \rightarrow Z_{ex}, m_{Q,ex} : Q_{ex} \rightarrow Q_{ex}$ ,

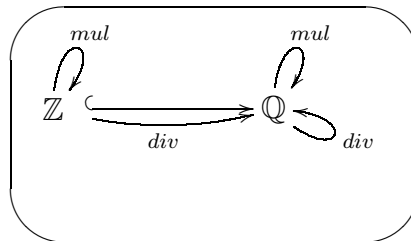
$d_{Z,ex} : Z_{ex} \rightarrow Q_{ex}, d_{Q,ex} : Q_{ex} \rightarrow Q_{ex}$

equations :  $m_{Q,ex} \circ c_{ex} \equiv c_{ex} \circ m_{Z,ex}, d_{Q,ex} \circ c_{ex} \equiv d_{Z,ex}$



$M \downarrow$

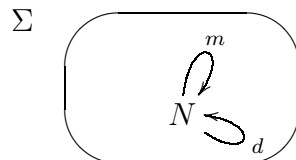
Set



The approximate mosaic  $\Sigma$  is made of:

node :  $N$  (representing “a set of Numbers”)

arrows :  $m, d : N \rightarrow N$



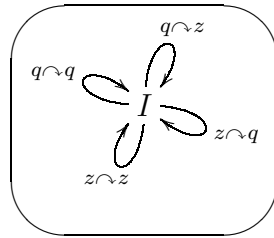
## An indexor $\mathbb{I}$

Comments on the arrows of  $\Sigma$  are the ranks for their interpretations; since there are four possible ranks, there are four arrows in the indexor  $\mathbb{I}$ .

As in our first example, each comment about an arrow of  $\Sigma$  has a “name”, which is identified to an arrow  $i$  in the indexor  $\mathbb{I}$ , and a “text”, which is “translated” by the mosaic  $\mathbb{K}(Arrow, i)$  in the extensor  $\mathbb{K}$ . The indexation  $\Vdash$  is a relation from  $\Sigma$  to  $\mathbb{I}$ .

Here the indexor  $\mathbb{I}$  is the ambigraph made of (the name of the arrows are chosen according to mnemotechnic reasons):

*node* :  $I$   
*arrows* :  $z \curvearrowright z, z \curvearrowright q, q \curvearrowright z, q \curvearrowright q : I \rightarrow I$



The order relation on  $\mathbb{I}$  is generated by:

$q \curvearrowright z \Rightarrow z \curvearrowright z, q \curvearrowright z \Rightarrow q \curvearrowright q$  and  $q \curvearrowright q \Rightarrow z \curvearrowright q$ .

The indexation  $\Vdash$  from  $\Sigma$  to  $\mathbb{I}$  is defined by:

*node* :  $N \Vdash I$   
*arrows* :  $m \Vdash z \curvearrowright z, m \Vdash q \curvearrowright q, d \Vdash z \curvearrowright q, d \Vdash q \curvearrowright q$

It follows from the order relation that, also:  $m \Vdash z \curvearrowright q$ .

## An extensor $\mathbb{K}$

The mosaic  $\mathbb{K}(\text{Node}, I)$  says how the node  $N$  of  $\Sigma$  must be interpreted. Hence  $\mathbb{K}(\text{Node}, I)$  is made of:

*nodes* :  $Z, Q$

*arrow* :  $c : Z \rightarrow Q$

where  $c$  is called a *coercion*.

$$\mathbb{K}(\text{Node}, I) \quad \left( Z \xrightarrow{c} Q \right)$$

The mosaic  $\mathbb{K}(\text{Arrow}, z \curvearrowright z)$  translates the comment called “ $z \curvearrowright z$ ”:

*nodes* :  $Z_1, Q_1, Z_2, Q_2$

*arrows* :  $c_1 : Z_1 \rightarrow Q_1, c_2 : Z_2 \rightarrow Q_2, f_{z \curvearrowright z} : Z_1 \rightarrow Z_2$

and so on for the mosaics  $\mathbb{K}(\text{Arrow}, \dots)$ .

$$\begin{array}{cc} \mathbb{K}(\text{Arrow}, z \curvearrowright z) & \mathbb{K}(\text{Arrow}, z \curvearrowright q) \\ \left( \begin{array}{ccc} Z_1 & \xrightarrow{c_1} & Q_1 \\ \downarrow f_{z \curvearrowright z} & & \\ Z_2 & \xrightarrow{c_2} & Q_2 \end{array} \right) & \left( \begin{array}{ccc} Z_1 & \xrightarrow{c_1} & Q_1 \\ & \searrow f_{z \curvearrowright q} & \\ Z_2 & \xrightarrow{c_2} & Q_2 \end{array} \right) \\ \\ \mathbb{K}(\text{Arrow}, q \curvearrowright z) & \mathbb{K}(\text{Arrow}, q \curvearrowright q) \\ \left( \begin{array}{ccc} Z_1 & \xrightarrow{c_1} & Q_1 \\ & \swarrow f_{q \curvearrowright z} & \\ Z_2 & \xrightarrow{c_2} & Q_2 \end{array} \right) & \left( \begin{array}{ccc} Z_1 & \xrightarrow{c_1} & Q_1 \\ & & \downarrow f_{q \curvearrowright q} \\ Z_2 & \xrightarrow{c_2} & Q_2 \end{array} \right) \end{array}$$

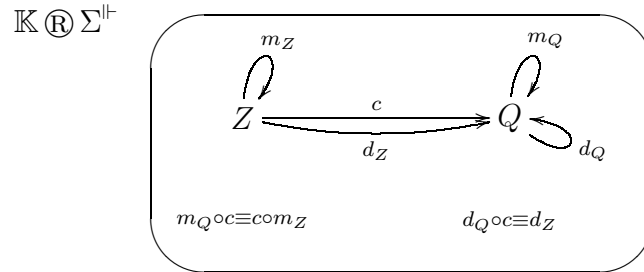
The relation  $q \curvearrowright z \Rightarrow z \curvearrowright z$  corresponds to the fact that  $\mathbb{K}(\text{Arrow}, z \curvearrowright z)$  can be embedded into  $\mathbb{K}(\text{Arrow}, q \curvearrowright z)$ : the arrow  $f_{z \curvearrowright z}$  corresponds to  $f_{q \curvearrowright z} \circ c_1$ .

## The ribbon product $\mathbb{K} \textcircled{R} \Sigma^{\text{lf}}$

The ribbon product  $\mathbb{K} \textcircled{R} \Sigma^{\text{lf}}$  is equivalent to the exact mosaic  $\Sigma_{\text{exact}}$ . The equations  $m_Q \circ c \equiv c \circ m_Z$  and  $d_Q \circ c \equiv d_Z$  come from the fact that in the ribbon product  $\mathbb{K} \textcircled{R} \Sigma^{\text{lf}}$ , the “gluing” step of section 3 takes into account the homomorphisms which correspond to the relation  $\Rightarrow$ .

In  $\Sigma_{\text{exact}}$ , the arrow  $c_{ex} : Z_{ex} \rightarrow Q_{ex}$  has exactly the same status as other arrows. On the contrary, in  $\mathbb{K} \textcircled{R} \Sigma^{\text{lf}}$  the coercion comes from  $\mathbb{K}(\text{Node}, I)$  while other arrows come from the approximate mosaic  $\Sigma$ : thanks to ribbon product, we are able to say that coercions are different from other arrows.

Hence, here too, the fact that  $\Sigma_{\text{exact}}$  can be considered as  $\mathbb{K} \textcircled{R} \Sigma^{\text{lf}}$  gives additional information on its structure.



## Imperative programming

Let us now consider a “toy” imperative programming language, with one variable  $x$  of type “natural”. We choose a very poor mosaic for the specification of naturals: the mosaic  $\mathbb{S}$  seen in section 1:

*nodes* :  $A, N$   
*arrows* :  $z : A \rightarrow N, s : N \rightarrow N$   
*constraint* :  $A = \mathbb{I}$

For the user, a *variable* in an imperative language has a *value* which can be *read* and *updated*. Hence, our variable  $x$  “of type natural” corresponds to two operations: the first one *read* returns a natural (the value of  $x$ ), the second one *update* needs a natural argument (and updates the value of  $x$  to this argument). If one updates the value of  $x$  to some natural  $n$ , and then reads this value, then the result should be  $n$ .

In addition, there is an *initialisation* map which (say) updates the value of  $x$  to 0.

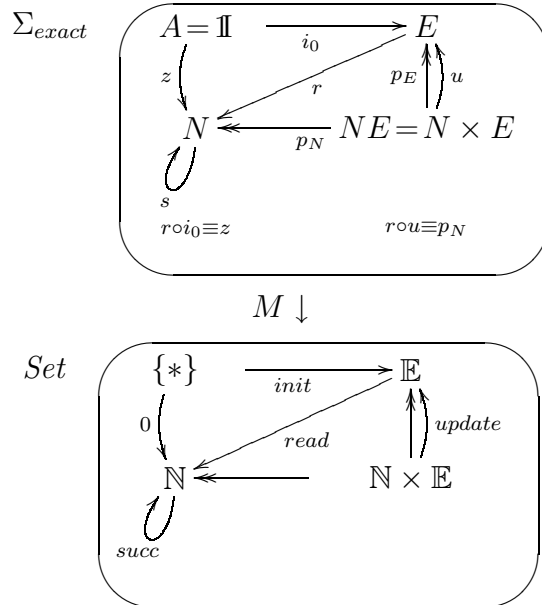
## An exact mosaic $\Sigma_{exact}$

From an explicit point of view, there is a set of *states*  $\mathbb{E}$ , and the mosaic  $\Sigma_{exact}$  is made of:

*sub-specification* :  $\mathbb{S}_{nat}$   
*nodes* :  $E, NE$   
*arrows* :  $i_0 : A \rightarrow E, r : E \rightarrow N, u : NE \rightarrow E,$   
 $p_N : NE \rightarrow N, p_E : NE \rightarrow E$   
*constraint* :  $NE = N \times E$  (with projections  $p_N, p_E$ )  
*equations* :  $r \circ i_0 \equiv z : A \rightarrow N, r \circ u \equiv p_N : NE \rightarrow N$

The constraint says that in each model  $M$ , the node  $M(NE)$  must be interpreted as the cartesian product  $M(N) \times M(E)$ , and the projections are  $M(p_N)$  and  $M(p_E)$ .

The equations say that the value of  $x$  after initialization is 0, and after updating  $x$  to some integer  $n$  it is precisely  $n$ .



## An approximate mosaic $\Sigma$

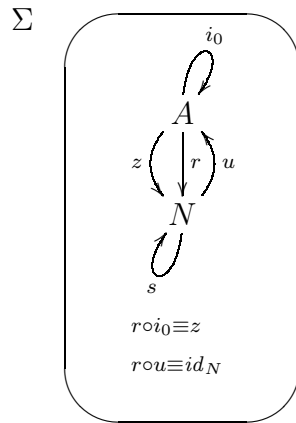
The approximate mosaic  $\Sigma$  is made of:

*sub-specification* :  $\mathbb{S}_{nat}$

*arrows* :  $i_0 : A \rightarrow A, r : A \rightarrow N, u : N \rightarrow A$

*equations* :  $r \circ i_0 \equiv z : A \rightarrow N, r \circ u \equiv id_N : N \rightarrow N$

Comments on the equations will play an important role.



## About imperative programs

Any kind of specification should be able to describe “good models” as well as “a good frame for programming and computing”. Until now, we have focused on models, though the study of programs would have been interesting too. Here with this example of an imperative language, programs become essential.

The mosaic  $\Sigma_{exact}$  corresponds to the point of view of *explicit state*, which is well known, as are its drawbacks. In a functional (or applicative) language, a program can be considered as a *term*, which is “composed”, according to some rules, from the arrows in the specification. However, imperative programs do not correspond to terms of  $\Sigma_{exact}$ , at least for two reasons.

The first reason is that the current state is always *implicit* in an imperative program: it does never appear as an argument or a value.

On the contrary, since the state is implicit in the approximate mosaic  $\Sigma$  too, its terms stick to imperative programs: for instance the instruction  $x := s(x)$  corresponds to the term  $u \circ s \circ r : A \rightarrow A$ :

$$A \xrightarrow{r} N \xrightarrow{s} N \xrightarrow{u} A .$$

The second reason is that terms of  $\Sigma_{exact}$  may use new nodes like  $E \times E$  which should be interpreted as the product  $\mathbb{E} \times \mathbb{E}$  of two copies of the state, which is usually prohibited.

We will see below how the ribbon product point of view avoids this problem.

## An indexor $\mathbb{I}$

A comment on an arrow says whether its interpretation is allowed to use (resp. to modify) the current state, or if it is an input or output arrow. Hence  $\mathbb{I}$  is made of (composition of arrows is an easy exercise):

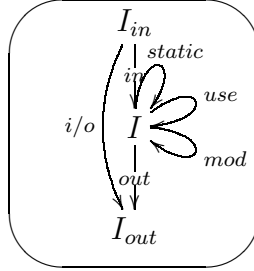
*nodes* :  $I_{in}, I, I_{out}$

*arrows* :  $static, use, mod : I \rightarrow I, in : I_{in} \rightarrow I, out : I \rightarrow I_{out}, i/o : I_{in} \rightarrow I_{out}$

*equations* :  $mod \equiv mod, out \equiv out$

The order relation on  $\mathbb{I}$  is generated by:  $static \Rightarrow use \Rightarrow mod ;$

$I \Rightarrow I_{out} ; mod \Rightarrow out ; (mod \equiv mod) \Rightarrow (out \equiv out) .$



The meaning of these indexes is the following:

- *static*: the interpretation does neither use nor modify the current state;
- *use*: the interpretation can *use* the current state, but it cannot modify it;
- *mod*: the interpretation can both use and *modify* the current state;
- $static \Rightarrow use \Rightarrow mod$  and  $mod \equiv mod$ : the interpretations of two arrows of  $\mathbb{S}$  satisfying either *static* or *use* or *mod* are equal if and only if they are equal as *mod*-maps;
- $I \Rightarrow I_{out}$  and  $mod \Rightarrow out$ : any “current” operation can be considered as an “output” operation;
- $out \equiv out$ : this equation is used as an index for equations of  $\mathbb{S}$  which are only *observation* equations: both members can have a different action on the current state;
- $(mod \equiv mod) \Rightarrow (out \equiv out)$ : if two maps agree “everywhere” (i.e. on outputs and state) then they agree on outputs.

The indexation  $\Vdash$  from  $\Sigma$  to  $\mathbb{I}$  is defined by:

*nodes* :  $A, N \Vdash I$

*arrows* :  $z, s \Vdash static, i_0 \Vdash in, r \Vdash use, u \Vdash mod$

*equations* :  $(r \circ i_0 \equiv z), (r \circ u \equiv id_N) \Vdash (out \equiv out)$

## An extensor $\mathbb{K}$

The mosaics  $\mathbb{K}(\text{Node}, I_{in})$  and  $\mathbb{K}(\text{Node}, I_{out})$  are very simple:

*node* :  $S$

$$\mathbb{K}(\text{Node}, I_{in}) = \mathbb{K}(\text{Node}, I_{out}) \quad \boxed{S}$$

The mosaic  $\mathbb{K}(\text{Node}, I)$  says that “there is a state node”:

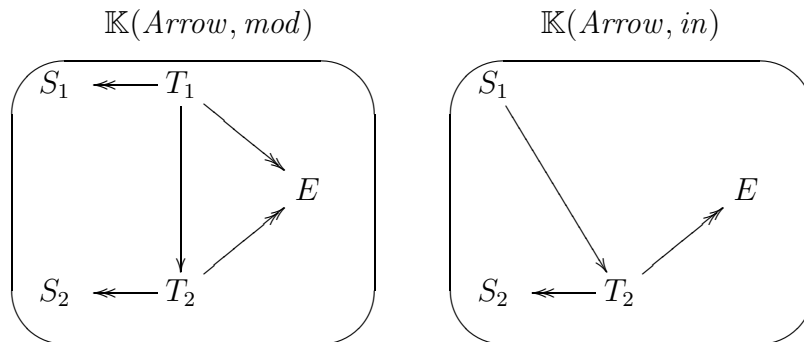
*nodes* :  $S, T, E$

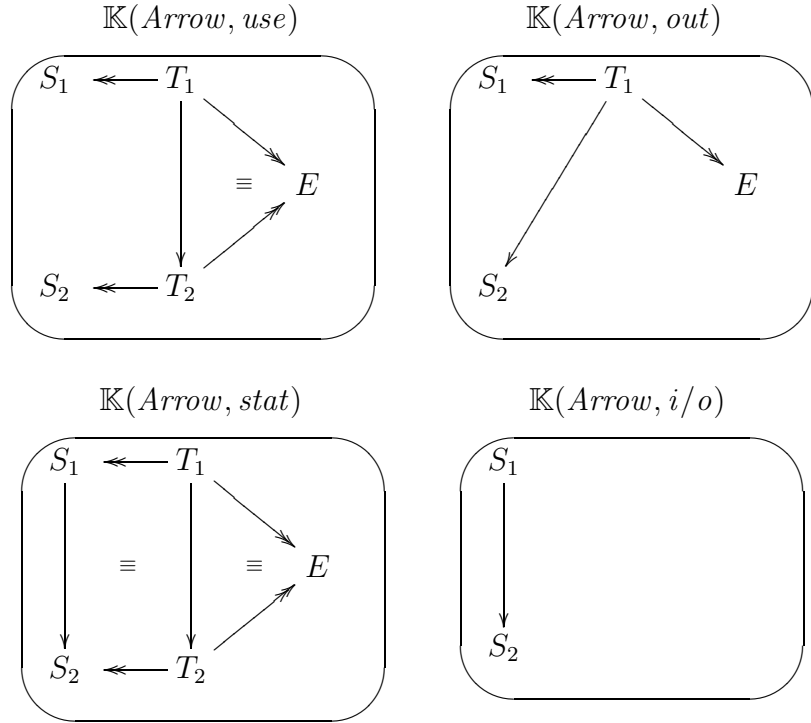
*arrows* :  $\pi_S : T \rightarrow S, \pi_E : T \rightarrow E$

*constraint* :  $T = S \times E$  (with projections  $\pi_S, \pi_E$ )

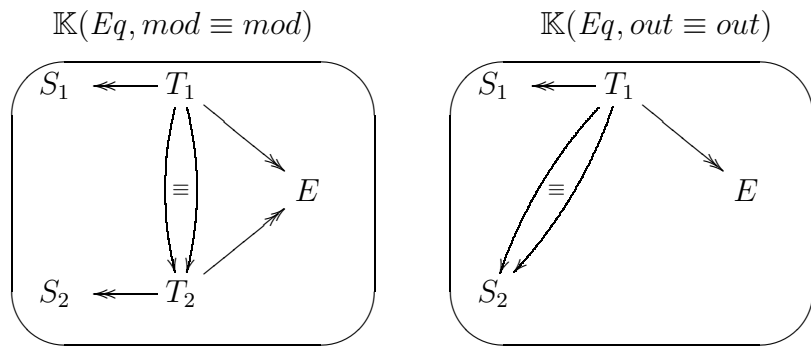
$$\mathbb{K}(\text{Node}, I) \quad \boxed{S \xleftarrow{\pi_S} T \xrightarrow{\pi_E} E}$$

The mosaics  $\mathbb{K}(\text{Arrow}, \dots)$  are easily built:





The mosaics  $\mathbb{K}(Eq, mod \equiv mod)$  and  $\mathbb{K}(Eq, out \equiv out)$  express the meaning of both equations in  $\mathbb{I}$ . The first one is essentially made of an equation between arrows of rank  $T_1 \rightarrow T_2$ , while for the second one it is an equation between arrows of rank  $T_1 \rightarrow S_2$ .



## The ribbon product $\mathbb{K} \textcircled{\mathbb{R}} \Sigma^{\text{!}}$

The ribbon product  $\mathbb{K} \textcircled{\mathbb{R}} \Sigma^{\text{!}}$  is equivalent to the exact mosaic  $\Sigma_{exact}$ .

An *instruction* can now be defined as a composed arrow  $p : A \rightarrow A$  in the approximate mosaic  $\Sigma$  such that  $p \Vdash mod$ , for instance  $m \circ s \circ r : A \rightarrow A$  for:

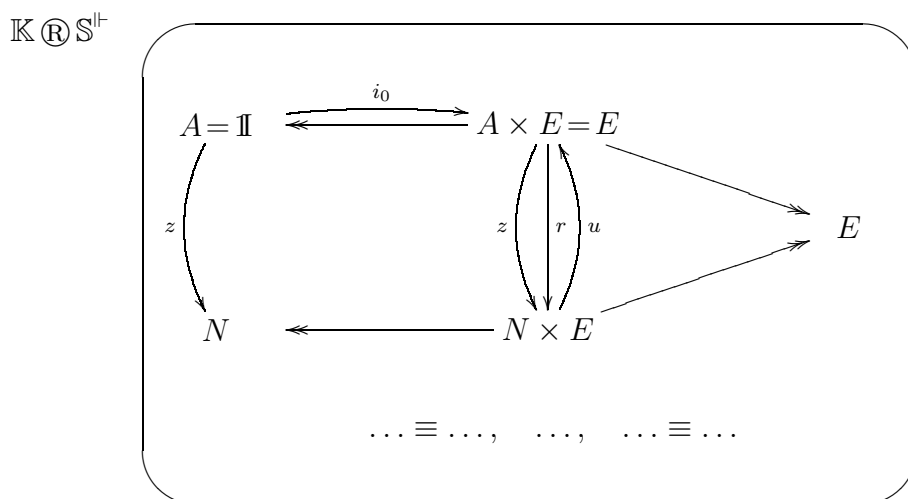
$$x := s(x)$$

A *program* is a composed arrow  $p$  in the approximate mosaic  $\Sigma$  such that  $p \Vdash i/o$ . For instance  $r \circ u \circ s \circ r \circ i_0 : A \rightarrow N$  for:

$$init ; \quad x := s(x) ; \quad x .$$

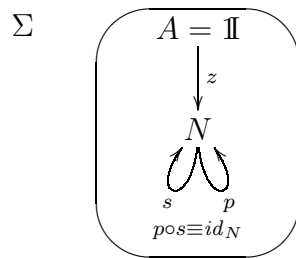
Hence the input and output of a program do not mention the state. The structure of  $\mathbb{I}$  and of the  $\mathbb{K}(\dots)$ 's shows that such a program will never use more than one copy of the state.

By looking at  $\Sigma_{exact}$  as the ribbon product  $\mathbb{K} \textcircled{\mathbb{R}} \Sigma^{\text{!}}$ , we are able to recognize “good” terms among all the terms of  $\Sigma_{exact}$ : indeed the “good” terms are precisely those which we are able to reach from an arrow  $p$  in  $\Sigma$  such that  $p \Vdash i/o$ .



## An exercise

Finally, let us suggest an exercise about the approximate mosaic  $\Sigma$  of section 2:



Compare three extensions, corresponding to three different ways of dealing with the fact that “the predecessor of 0 does not make sense”:

1. error handling as in section 2;
2. extend the node  $N$  in two nodes  $P$  and  $N$  with a coercion  $c : P \rightarrow N$  as in section 6, where  $P$  represents the positive integers, and define the extension of  $p$  as an arrow from  $P$  to  $N$ ;
3. let  $\text{pred}(0) = 0$ , together with some “boolean-valued state” ( $E = \mathbb{I} + \mathbb{I}$ ) as in section 7 used as a “flag” which sets to *false* as soon as  $\text{pred}(0)$  is executed.

This exercise shows that, thanks to soft typing, it is possible to define only one type for natural numbers with predecessor, allowing several ways of dealing with the predecessor of 0, according to the choice of the extensor.

## Conclusion

In this tutorial we have given some hints on the ribbon product constructor for soft typing, by looking at various examples.

It proves that, with some care about the comments, a specification can be approximate without leading to any error.

It proves much more than that: actually, an exact specification may hold more information when it is described by an approximation and comments.

One domain of interest for applications of soft typing might be computer algebra.

## References

- [1] E. ASTESIANO, E. ZUCCA. D-oids: model for dynamic data types, *Mathematical Structures in Computer Science* 5 (1995) 257-282.
- [2] M. BARR, C. WELLS. *Categories theory for computing science*, Prentice Hall (1990).
- [3] L. COPPEY, C. LAIR. Leçons de théorie des esquisses (I), *Diagrammes* 12 (1984).
- [4] L. COPPEY, C. LAIR. Leçons de théorie des esquisses (II), *Diagrammes* 19 (1988).
- [5] P. DAUCHY, M.-C. GAUDEL. Algebraic specifications with implicit state, Rapport de recherche, Université de Paris-Sud (1994).
- [6] D. DUVAL, C. LAIR. Forthcoming papers on ribbon product and its applications.
- [7] D. DUVAL, J.-C. REYNAUD. Sketches and computation (Part I): Basic Definitions and Static Evaluation, *Mathematical Structures in Computer Science* 4 (1994) 185-238.
- [8] D. DUVAL, J.-C. REYNAUD. Sketches and computation (Part II): Dynamic Evaluation and Applications, *Mathematical Structures in Computer Science* 4 (1994) 239-271.
- [9] C. EHRESMANN. Introduction to the theory of structured categories, Technical Report 10, University of Kansas, Lawrence (1966).
- [10] C. EHRESMANN. Esquisses et types de structures algébriques, *Bulletin de l'Institut Polytechnique, Iasi* 14 (1968).
- [11] H. EHRIG, F. OREJAS. Integration paradigm for data type and process specification technique, *Bulletin of the European Association for Theoretical Computer Science* 65 (1998) 90-97.
- [12] J.A. GOGUEN. Abstract errors for abstract data types, *Formal Description of Programming Concepts*, E.J. Neuhold ed., North-Holland (1978).
- [13] J.A. GOGUEN, J.W. THATCHER, E.G. WAGNER. An initial algebra approach to the specification, correctness, and implementation of abstract data types, *Current Trends in Programming Methodology, Vol. IV: Data Structuring*, R.T. Yeh ed., Prentice-Hall (1978) 80-149.

- [14] R. GUITART, C. LAIR. Calcul syntaxique des modèles et calcul des formules internes, *Diagrammes* 4 (1980).
- [15] R. GUITART, C. LAIR. Limites et colimites pour représenter les formules, *Diagrammes* 7 (1982).
- [16] Y. GUREVITCH. Evolving Algebras, A Tutorial Introduction, *Bulletin of the European Association for Theoretical Computer Science* 43 (1991) 264-284.
- [17] C. HINTERMEIER, C. KIRCHNER, H. KIRCHNER. Dynamically Typed Computations for Order-sorted Equational Presentations, *Journal of Symbolic Computation* 25 (1998) 455-526.
- [18] R.D. JENKS, R.S. SUTOR. *Axiom. The Scientific Computation System*, NAG, Springer-Verlag (1992).
- [19] Y. LAFONT. The linear abstract machine, *Theoretical Computer Science* 59 (1988) 157-180.
- [20] C. LAIR. Trames et sémantiques catégoriques des systèmes de trames, *Diagrammes* 18 (1987).
- [21] C. LAIR. Éléments de théorie des patchworks, *Diagrammes* 29 (1993).
- [22] S.K. LELLAHI. Categorical abstract data types, *Diagrammes* 21 (1989).
- [23] S. MAC LANE. *Categories for the working mathematician*, Springer (1971).
- [24] E. MOGGI. Notions of computation and monads, *Information and Computation* 93 (1991) 55-92.
- [25] P.D. MOSSES. The use of Sorts in Algebraic Specifications, *8th Workshop on Abstract Data Types* (1991).
- [26] P. WADLER. Linear types can change the world!, *Programming Concepts and Methods*, M. Broy and C.B. Jones ed., North Holland (1990).
- [27] P. WADLER. Monads for functional programming, *Proceedings of the Båstad Spring School*, J. Jeuring and E. Meijer ed., Springer Verlag Lecture Notes in Computer Science 925 (1995).
- [28] E. WAGNER. On the role of memory in object-based and object-oriented languages, *Theoretical Computer Science* 140 (1995) 179-199.