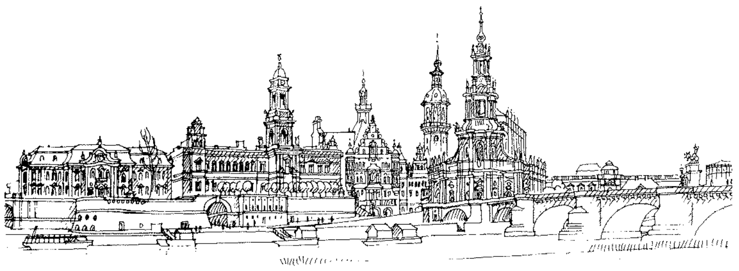
ISSN 1430-211X

TUD-FI01-10 — November 2001

Claus Jürgensen
Heiko Vogler
Institut für Theoretische Informatik

Syntactic composition of top-down tree transducers is short cut fusion



Syntactic composition of top-down tree transducers is short cut fusion

Claus Jürgensen[*] [†] and Heiko Vogler[‡]
Faculty of Computer Science
Dresden University of Technology
D-01062 Dresden, Germany

Abstract

We compare two deforestation techniques: short cut fusion formalized in category theory and the syntactic composition of tree transducers. The former strongly depends on types and uses the parametricity property or free theorem whereas the latter makes no use of types at all and allows more general compositions. We introduce the notion of a categorical transducer which is a generalization of a catamorphism and show a respective fusion result which is a generalization of the 'acid rain theorem'. We prove the following main theorems: (i) The class of all categorical transducers builds a category where composition is fusion. (ii) The semantics of categorical transducers is a functor. (iii) The subclass of top-down categorical transducers is a subcategory. (iv) Syntactic composition of top-down tree transducers is equivalent to the fusion of top-down categorical transducers.

Contents

[*]Email: `Claus.Juergensen@Inf.TU-Dresden.DE`

[†]Supported by the postgraduate program 'Specification of discrete processes and systems of processes by operational models and logics' (GRK 334/2) of the German Research Community (DFG)

[‡]Email: `Vogler@TCS.Inf.TU-Dresden.DE`

1 Introduction

A transformation of functional programs to eliminate intermediate data structures is called *deforestation*. The name 'deforestation' has been introduced by [Wad90] inspired by the fact that the values of algebraic data structures are trees.

A particular kind of deforestation is the following: for three given data structures A, B, C and programs $A \xleftarrow{f} B$ and $B \xleftarrow{g} C$ consider the program $A \xleftarrow{f \cdot g} C$, where $f \cdot g$ denotes the program which computes the value of g from the input and computes the value of f from the value of g. The deforestation transformation is the construction of a new program $A \xleftarrow{h} C$ such that the semantics of h is the composition of the semantics of f and the semantics of g, and the construction of h is independent from the explicit data structure B. Thus, the intermediate transfer from g to f of values of the data structure B is eliminated. This deforestation technique is called *fusion*. It is of practical significance, because it can be used to deforest modularly constructed functional programs. Since memory space and time for the allocation of the eliminated data structures is saved, this can improve the runtime performance [PTH01].

In this paper we will compare two specific fusion techniques in a theoretical way: *short cut fusion* and the *syntactic composition of top-down tree transducers*. In fact, we will compare these two fusion techniques not only w.r.t. their behavior, but rather we will show that the techniques themselves are structurally equivalent.

Let us first briefly recall these two techniques and then indicate how we compare them.

Short cut fusion is a fusion technique which uses a single, local transformation rule called the `cata/build`-rule [GLP93, Gil96]. Consider the Haskell program:

```
data Tree = Alpha | Sigma(Tree, Tree)
data List = N | A List | B List

zig :: Tree -> List
zag :: Tree -> List
zig Alpha           = N
zag Alpha           = N
zig (Sigma(x1, x2)) = A(zag x1)
zag (Sigma(x1, x2)) = B(zag x2)

bin :: List -> Tree
bin N      = Alpha
bin (A x) = Sigma(bin x, bin x)
bin (B x) = Sigma(bin x, bin x)

bin_zig :: Tree -> Tree
bin_zig = bin . zig

bin_zag :: Tree -> Tree
bin_zag = bin . zag
```

We will deforest the function `bin_zig`: In order to use the `cata/build`-rule to fuse the functions `bin` and `zig` we have to express `zig` as a `build` and `bin` as a `cata` (which is a shorthand for catamorphism). The `build` for the `List` data structure is defined by

```
build :: (forall c. c -> (c -> c) -> (c -> c) -> d -> c) -> (d -> List)
build g = g N A B
```

where the semantics of `build` is to apply its argument to the `List`-constructors. Alternatively, we may view the term `g` as the result of abstracting in the term `build g` from the `List`-constructor-symbols `N`, `A`, and `B`, *i.e.* `g` $=$ `\N A B -> build g`.[1] Then we can write

```
zig = build zig'
zag = build zag'

zig' :: c -> (c -> c) -> (c -> c) -> Tree -> c
zag' :: c -> (c -> c) -> (c -> c) -> Tree -> c
zig' n a b Alpha           = n
zag' n a b Alpha           = n
zig' n a b (Sigma(x1, x2)) = a(zag' n a b x1)
zag' n a b (Sigma(x1, x2)) = b(zig' n a b x2)
```

where `zig'` and `zag'` are `zig` and `zag`, respectively, in which we have abstracted from the `List`-constructors. The second ingredient — the catamorphism for the `List` data structure — is given by

```
cata n a b N     = n
cata n a b (A x) = a(cata n a b x)
cata n a b (B x) = b(cata n a b x)
```

where the semantics of `cata` is to substitute the `List`-constructors `N`, `A`, and `B` by the functions `n`, `a`, and `b`, respectively, *e.g.* `cata n a b (B(A(B N)))` $=$ `b(a(b n))`. It is easy to see that we can express the function `bin` as a `cata` in the following way:

```
bin = cata Alpha (\x -> Sigma x x) (\x -> Sigma x x)
```

Now we can apply the `cata/build`-rule

$$\frac{\texttt{g :: c -> (c -> c) -> (c -> c) -> d -> c}}{\texttt{cata n a b . build g = g n a b}}$$

which leads to

[1]'`\x -> t`' is the Haskell-notation for 'λ`x`. `t`', *i.e.* the λ-abstraction from the variable `x` in the term `t`.

```
bin_zig = bin . zig
        = cata Alpha (\x -> Sigma x x) (\x -> Sigma x x) . build zig'
        = zig' Alpha (\x -> Sigma x x) (\x -> Sigma x x)
```

and similarly for `bin_zag`. Finally, by applying `bin_zig` (and `bin_zag`) to every possible input pattern, we obtain the program

```
bin_zig Alpha           = Alpha
bin_zag Alpha           = Alpha
bin_zig (Sigma(x1, x2)) = Sigma((bin_zag x1), (bin_zag x1))
bin_zag (Sigma(x1, x2)) = Sigma((bin_zig x2), (bin_zig x2))
```

Notice that the `List` data structure has been eliminated.

Originally, short cut fusion and the `cata/build`-rule were defined only for the list data structure. This restricted transformation has also been implemented in the GHC (Glasgow Haskell Compiler) [PTH01]. For arbitrary algebraic data structures of the polymorphic λ-calculus 'PolyFix' (but not for Haskell) a proof for the correctness of short cut fusion is given in [Joh01]. It is also possible to prove an abstract version in terms of category theory of the `cata/build`-rule which is known as the 'acid rain theorem' [TM95]:

$$\frac{\mathsf{H} : (\boldsymbol{\mathcal{A}lg_{\mathcal{C}}}\mathsf{F}, |\centerdot|_{\mathsf{F}}) \leftarrow (\boldsymbol{\mathcal{A}lg_{\mathcal{C}}}\mathsf{G}, |\centerdot|_{\mathsf{G}})}{(\!|\varphi|\!)_{\mathsf{G}} \cdot (\!|\mathsf{H}\mathit{in}_{\mathsf{G}}|\!)_{\mathsf{F}} = (\!|\mathsf{H}\varphi|\!)_{\mathsf{F}}}.$$

where $(\!|\varphi|\!)_{\mathsf{G}}$ is the category theory notation for the catamorphism, which is the unique solution of the equation $(\!|\varphi|\!)_{\mathsf{G}} \cdot \mathit{in}_{\mathsf{G}} = \varphi \cdot \mathsf{G}(\!|\varphi|\!)_{\mathsf{G}}$, G is an endofunctor, φ is a G-algebra, and in_{G} is an initial G-algebra. The rôle of the `build` is taken by a concrete functor H. Other generalizations of short cut fusion can *e.g.* be found in [LS95] and [HIT96].

Syntactic composition of top-down tree transducers is the other fusion technique, we are interested in. The concept of top-down tree transducers has been introduced by [Rou68, Rou70] and [Tha70]. Roughly speaking, such a transducer $T = (Q, \Sigma, \Delta, q_0, R)$ is a deterministic finite-state top-down tree automaton (with state set Q) which reads a given input tree (over some ranked alphabet Σ) starting from the root, stepping towards the leaves, and thereby producing an output tree (over some ranked alphabet Δ). The behavior of the transducer is determined by a finite set R of term rewrite rules, as *e.g.*

$$\begin{array}{lll} \mathit{zig}\,\alpha & \to & N \\ \mathit{zag}\,\alpha & \to & N \\ \mathit{zig}(\sigma(x_1, x_2)) & \to & A(\mathit{zag}\,x_1) \\ \mathit{zag}(\sigma(x_1, x_2)) & \to & B(\mathit{zig}\,x_2) \end{array}$$

where zig and zag are states of rank 1, σ and α are input symbols of rank 2 and 0, respectively, A, B, and N are output symbols of rank 1, 1, and 0, respectively, and x_1 and x_2 are term rewrite variables. By applying the usual term rewrite semantics, for every input tree t, the unique normal form of $\mathit{zig}(t)$ is a monadic tree $A(B(A \ldots N \ldots))$ which shows the zig-zag path through t. Thus, in general, the semantics of a tree transducer is a tree transformation, *i.e.* a function mapping trees onto trees.

A top-down tree transducer can be viewed as a functional program by turning states into functions and rewrite rules into defining equations (*cf.* the functional program from above). Clearly, only particular functional programs are related to top-down tree transducers. Roughly speaking, a top-down tree transducer is a primitive-recursion scheme with mutual recursion (*cf.* [EV91, FHVV93, NV01] for the computational power of tree transducers).

Let us denote the fact that a top-down tree transducer T has input alphabet Σ and output alphabet Δ by $\Delta \xleftarrow{T} \Sigma$. The semantics of T is a tree transformation $T_\Delta \xleftarrow{\tau T} T_\Sigma$ where T_Σ and T_Δ are the sets of trees over Σ and Δ, respectively. For two given top-down tree transducers $\Gamma \xleftarrow{T_2} \Delta \xleftarrow{T_1} \Sigma$ with semantics $T_\Gamma \xleftarrow{\tau T_2} T_\Delta \xleftarrow{\tau T_1} T_\Sigma$ we can construct a new top-down tree transducer $\Gamma \xleftarrow{T_2 \cdot T_1} \Sigma$ such that the following composition result holds:

$$\tau T_2 \cdot \tau T_1 = \tau(T_2 \cdot T_1) \qquad (*)$$

and thus the intermediate Δ-trees are eliminated. The basic idea for the construction of $T_2 \cdot T_1$ is to run a slightly modified version of T_2 on the right hand sides of the rules of T_1 to obtain the right hand sides of the rules of $T_2 \cdot T_1$. Then $T_2 \cdot T_1$ is called the *syntactic composition* of T_1 and T_2. In order to illustrate this fusion technique let us consider the top-down tree transducer T_1 as shown above and the following top-down tree transducer T_2:

$$\begin{array}{lcl} bin\,N & \rightarrow & \alpha \\ bin(A\,x_1) & \rightarrow & \sigma(bin\,x_1, bin\,x_1) \\ bin(B\,x_1) & \rightarrow & \sigma(bin\,x_1, bin\,x_1) \end{array}$$

which also corresponds to a part of our example Haskell-program. If we apply the syntactic composition to these two tree transducers, then we obtain the top-down tree transducer $T_2 \cdot T_1$:

$$\begin{array}{lcl} (bin, zig)\alpha & \rightarrow & \alpha \\ (bin, zag)\alpha & \rightarrow & \alpha \\ (bin, zig)(\sigma(x_1, x_2)) & \rightarrow & \sigma((bin, zag)x_1, (bin, zag)x_1) \\ (bin, zag)(\sigma(x_1, x_2)) & \rightarrow & \sigma((bin, zig)x_2, (bin, zig)x_2) \end{array}$$

And this corresponds to the equations of the functional program that we have calculated for `bin_zig` (and `bin_zag`) using short cut fusion.

The syntactic composition of top-down tree transducers has been introduced and thoroughly studied in [Eng75, Eng77, Bak79, Eng82]. Further investigations of composition of semantically larger classes of transducers can be found in: [Fül81, Gie88, CDPR97b, CDPR97a] for attributed tree transducers, [Eng80, CF82, EV85b] for macro tree transducers (or: primitive-recursive program schemes with parameters), and [EV88] for high-level tree transducers (also cf. the survey articles and monographs [GS84, GS97, FV98]). In [KV01, VK01] the composition of tree transducers has been compared with the deforestation method for functional programs [Wad90] in a syntactical framework.

This finishes the short review on short cut fusion and syntactic composition. Since both fusion techniques eliminate intermediate data structures and produce equivalent results in our example, the obvious question is: what is the relationship between these two fusion techniques? In order to compare them we will introduce the notion of a *categorical transducer* which is a generalization of a catamorphism, and we will show a respective fusion result which is a generalization of the 'acid rain theorem'. Then — described in the language of category theory — both fusion techniques are instances of this generalization. The following diagram gives a rough illustration of this generalization process:

$\boxed{\text{tree transducer}}$	?	short cut fusion
syntactic composition	$\longleftrightarrow$	`cata/build`-rule
universal algebra		*polymorphic λ-calculi*
$\downarrow$ generalization		$\downarrow$ generalization
$\boxed{\text{categorical transducer}}$		functorial short cut fusion
composition	$\subseteq$	generalized 'acid rain theorem'
category theory	$\longleftarrow$	*category theory*

Let us explain now a bit more the notion of a categorical transducer and the consequences of the generalized 'acid rain theorem'. (We assure those readers who are not familiar with notions from category theory that we will develop later all the needed techniques in quite some detail.) The main idea is to reinvent tree transducers on an the abstract level of category theory, where we have the following intuition how the ingredients of a top-down categorical transducer are related to those of a top-down tree transducer:

top-down tree transducer		$\longleftrightarrow$	top-down categorical transducer over $\boldsymbol{\mathcal{S}et}$	
$T = (Q, \Sigma, \Delta, q_0, R)$		$\longleftrightarrow$	$C = (\mathsf{H}, \mathsf{U}, \pi) : \mathsf{G} \leftarrow \mathsf{F}$	
finite set of states	Q	$\longleftrightarrow$	$\mathsf{U} : \boldsymbol{\mathcal{S}et} \leftarrow \boldsymbol{\mathcal{S}et}$	faithful endofunctor
ranked input-alphabet	Σ	$\longleftrightarrow$	$\mathsf{F} : \boldsymbol{\mathcal{S}et} \leftarrow \boldsymbol{\mathcal{S}et}$	endofunctor
ranked output-alphabet	Δ	$\longleftrightarrow$	$\mathsf{G} : \boldsymbol{\mathcal{S}et} \leftarrow \boldsymbol{\mathcal{S}et}$	endofunctor
initial state	$q_0 \in Q$	$\longleftrightarrow$	$\pi : \mathsf{Id} \dot{\leftarrow} \mathsf{U}$	natural transformation
finite set of rules	R	$\longleftrightarrow$	$\mathsf{H} in_\mathsf{G}$	where H is a concrete functor
			$\mathsf{H} : (\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{F}, \lvert\centerdot\rvert_\mathsf{F}) \leftarrow (\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{G}, \mathsf{U} \cdot \lvert\centerdot\rvert_\mathsf{G})$	
set of Σ-trees	T_Σ	$\longleftrightarrow$	$\mu\mathsf{F}$	least fixpoint of F
set of Δ-trees	T_Δ	$\longleftrightarrow$	$\mu\mathsf{G}$	least fixpoint of G
tree transformation	$\tau T : T_\Delta \leftarrow T_\Sigma$	$\longleftrightarrow$	$\mathsf{S}\, C = \pi \cdot (\![\mathsf{H}\, in_\mathsf{G}]\!)_\mathsf{F} : \mu\mathsf{G} \leftarrow \mu\mathsf{F}$	

We will also formalize this relationship and call the top-down tree transducer T and the top-down categorical transducer C *related* and write $T \approx C$. Moreover we will define a function R which maps a given top-down tree transducer T to a related top-down categorical transducer $T \approx \mathsf{R}\, T$.

The concrete functor H plays a particular rôle in our formalization. Intuitively H describes a 'rule-pattern' in which the parameters can be substituted by particular 'output-functions', such that we obtain the 'rules' of C if we substitute the parameters by the 'output-symbols' of C. The initial G-algebra in_G stands for the 'output-symbols' of C and thus $\mathsf{H}\, in_\mathsf{G}$ describes the 'rules' of C. Finally, the catamorphism $(\![\mathsf{H}\, in_\mathsf{G}]\!)_\mathsf{F}$ yields the fixpoint of the 'rules' by induction and the natural transformation π selects the value of the 'initial state'. We show that all categorical transducers over some category $\boldsymbol{\mathcal{C}}$ are the morphisms of a category (denoted by $\boldsymbol{cat\mathcal{T}_\mathcal{C}}$), where the composition is defined as follows:

$$(\mathsf{H}_2, \mathsf{U}_2, \pi_2) \cdot (\mathsf{H}_1, \mathsf{U}_1, \pi_1) = (\mathsf{H}_1 \cdot \mathsf{H}_2, \mathsf{U}_1 \cdot \mathsf{U}_2, \pi_1 * \pi_2),$$

and the category $td\text{-}\boldsymbol{cat\mathcal{T}_\mathcal{C}}$ of all top-down categorical transducers is its subcategory. The crucial point is the composition $\mathsf{H}_1 \cdot \mathsf{H}_2$ where the 'rules' of the second categorical transducer $(\mathsf{H}_2, \mathsf{U}_2, \pi_2)$ are used as 'output-symbols' for the first categorical transducer $(\mathsf{H}_1, \mathsf{U}_1, \pi_1)$. The reader should compare this idea with the `cata/build`-rule on page 3.

The amazing coincidence is that this composition which comes natural with the definition of a categorical transducer turns out to be a generalization of short cut fusion in the following sense: With the functorial 'acid rain theorem' (Definition and Corollary 3.2.6) we can prove that the semantics S of categorical transducers over $\boldsymbol{\mathcal{C}}$ is a functor $\mathsf{S} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{cat\mathcal{T}_\mathcal{C}}$, *i.e.*

$$\mathsf{S}(\mathsf{H}_2, \mathsf{U}_2, \pi_2) \cdot \mathsf{S}(\mathsf{H}_1, \mathsf{U}_1, \pi_1) = \mathsf{S}(\mathsf{H}_1 \cdot \mathsf{H}_2, \mathsf{U}_1 \cdot \mathsf{U}_2, \pi_1 * \pi_2).$$

On the other hand we prove that top-down tree transducers (modulo isomorphism) with syntactic composition form a category $td\text{-}tree\boldsymbol{\mathcal{T}}$ and that $\mathsf{R} : td\text{-}\boldsymbol{cat\mathcal{T}_{\mathcal{S}et}} \leftarrow td\text{-}tree\boldsymbol{\mathcal{T}}$ is an embedding functor (Theorem 6.3.5). This functor respects the semantics, *i.e.* $\tau = \mathsf{R} \cdot \mathsf{S}$ (Corollary 6.2.11). Then the syntactic composition of top-down tree transducers (*cf.* statement $(*)$: $\tau T_2 \cdot \tau T_1 = \tau(T_2 \cdot T_1)$ on page 5) is short cut fusion of the corresponding categorical transducers (Theorem 6.3.5):

$$\mathsf{R}\, T_2 \cdot \mathsf{R}\, T_1 = \mathsf{R}(T_2 \cdot T_1).$$

Note that this equation compares the result of the syntactic composition $T_2 \cdot T_1$ of top-down tree transducers T_1 and T_2 with the short cut fusion $\mathsf{R}\, T_2 \cdot \mathsf{R}\, T_1$ of the corresponding categorical transducers $\mathsf{R}\, T_1$ and $\mathsf{R}\, T_2$ on a *syntactic level*, *i.e.*, the two fusion techniques are structurally the same. Clearly, as a consequence on the semantical level, we obtain

$$\mathsf{S}(\mathsf{R}\, T_2) \cdot \mathsf{S}(\mathsf{R}\, T_1) = \mathsf{S}(\mathsf{R}(T_2 \cdot T_1))$$

i.e.

$$\tau T_2 \cdot \tau T_1 = \tau(T_2 \cdot T_1).$$

Let us now compare our approach with other well-known approaches of embedding finite-state machines into the framework of category theory. The straightforward embedding would assume that, for two tree transducers $\Delta \xleftarrow{T_2, T_1} \Sigma$ we define a homomorphism $h : T_2 \Leftarrow T_1$ as a rule-preserving function between the state sets. Then the objects of the desired category are the tree transducers and the morphisms are the tree transducer homomorphisms. Clearly, then the composition of morphisms is the composition of tree transducer homomorphisms as shown in the following figure:

$$\begin{tikzcd}
T_1 \arrow[d, Rightarrow, "h_1"'] \arrow[dd, Rightarrow, bend left, "h_2 \cdot h_1"] \\
T_2 \arrow[d, Rightarrow, "h_2"'] \\
T_3
\end{tikzcd}$$

In fact, this approach of embedding finite-state machines into category theory has been successfully applied in in [EP72, Ehr74, BH75]. However, in the present paper we are not interested in the composition of homomorphisms between tree transducers, but rather in the composition of the tree transducers themselves. Thus our figure should look as follows:

$$\begin{tikzcd}
\Gamma & \Delta \arrow[l, "T_2"'] & \Sigma \arrow[l, "T_1"'] \arrow[ll, bend left, "T_2 \cdot T_1"]
\end{tikzcd}$$

Consequently, as objects we will use ranked alphabets and as morphisms we will use tree transducers (modulo isomorphism). Hence, our point of view is orthogonal to the classical approach of the cited literature. We note that the two approaches can be integrated by considering 2-categories (*cf.* Definition 7.1.1 of [Bor94]) where the ranked alphabets (like Σ and Δ) are 0-cells, the tree transducers (like T_1 and T_2) are 1-cells, and the tree transducer homomorphisms (like $h : T_2 \Leftarrow T_1$) are 2-cells:

$$\begin{tikzcd}
\Delta & & \Sigma \arrow[ll, bend right=50, "T_1"', ""{name=U, below}] \arrow[ll, bend left=50, "T_2", ""{name=D, above}] \arrow[from=U, to=D, Rightarrow, "h"]
\end{tikzcd}$$

Finally let us summarize the main results of this paper and outline its structure:

- The class of all categorical transducers builds a category (Definition 5.1.1) where composition is fusion (Theorem 5.2.3).
- The semantics of categorical transducers is a functor (Theorem 5.2.3).
- The subclass of top-down categorical transducers is a subcategory (Theorem 5.4.2).
- For every top-down tree transducer there exists a top-down categorical transducer which has the same semantics (Lemma 6.2.9 and Theorem 6.2.7). We use this construction to define a function R (Definition 6.2.10).
- Syntactic composition of top-down tree transducers is equivalent to the fusion of top-down categorical transducers (Theorem 6.3.2).
- The class of all top-down tree transducers modulo isomorphism is a category and the function R is a functor from this category to the category of top-down categorical transducers (Theorem 6.3.5).

The structure of this paper is as follows: In Section 2 we collect the needed notions and notation of universal algebra and category theory. The reader who is unfamiliar with either topic may have a look at Appendix B or C, respectively. In Subsection 2.4 we define concrete categories and functors, which we will use in our generalized formulation of the 'acid rain theorem' in Section 3 (*cf.* Definition and Corollary 3.2.6). Then we give a little introduction into the theory of top-down tree transducers and the syntactic composition of these in Section 4. In Section 5 we generalize the notion of a tree transducer in terms of category theory and define the categorical transducer. We also define a top-down categorical transducer and show a respective composition result. Finally in Section 6 we establish a relation between top-down tree transducers and categorical transducers such that we are able to compare the two fusion techniques. We close with a brief outlook over some future work we plan to do.

2 Preliminaries

2.1 General notions

Functions and arrows

We denote the fact that a function f maps to a set A from a set B by $B = \operatorname{dom} f$ and $A = \operatorname{cod} f$ or by the relation $f : A \leftarrow B$. We will use this notation for a morphism f to an object A from an object B as well, because a function is nothing else than a morphism in the category $\boldsymbol{\mathcal{S}et}$. In order to avoid parentheses we will use the conventions $fx = f(x)$ and $\mathsf{F}fx = (\mathsf{F}f)x$ for function applications. The composition $f \cdot g : A \leftarrow C$ of two functions $f : A \leftarrow B$ and $g : B \leftarrow C$ is defined by $\forall\, x \in C :: \ (f \cdot g)x = f(gx)$. This is the reason why the arrows point to the left[2]:

$$\begin{tikzcd} A & B \arrow[l, "f"'] & C \arrow[l, "g"'] \arrow[ll, bend left, "f \cdot g"] \end{tikzcd}$$

Besides the composition operator $\cdot$ we will later define the binary operators $+$ and $\times$ on functions and, more general, on morphisms. The function application may also be viewed as an invisible binary operator. We declare the following operator precedences:

$$\xleftarrow[\text{binds weaker}]{\text{lower precedence}} \quad + \quad \times \quad \cdot \quad \text{'function application'} \quad \xrightarrow[\text{binds stronger}]{\text{higher precedence}},$$

e.g. $\mathsf{F}f \cdot g + h \times i = \big((\mathsf{F}f) \cdot g\big) + (h \times i)$. Partial functions are denoted by $\leftarrow\!\cdots$. We denote the identity function on a set A by id_A. For every two sets F and G of functions we define

$$F \cdot G = \big\{ f \cdot g \ \big| \ f \in F \ \wedge \ g \in G \ \wedge \ \operatorname{dom} f = \operatorname{cod} g \big\}.$$

We denote the cardinality of a set M by $\#M$.

Inference rules

Sometimes we will write an implication

$$\forall\, x_1 \cdots \forall\, x_k :: \ A_1 \ \wedge \ \cdots \ \wedge \ A_m \ \Longrightarrow \ B_1 \ \wedge \ \cdots \ \wedge \ B_n$$

in the form of an inference rule

$$\frac{A_1 \quad \cdots \quad A_m}{B_1 \quad \cdots \quad B_n}$$

where free variables in the latter should be considered as universally quantified.

[2]Arrows pointing to the right are consistent with the commuted composition $g \,;\, f = f \cdot g$.

2.2 Basic universal algebra

We will need the notions: ranked alphabet, algebra (homomorphism), free (term) algebra, and (2nd-order) substitution. For a ranked alphabet Σ and a set A we define $\Sigma A = \{\sigma(a_1, \dots, a_{\mathrm{rank}_\Sigma \sigma}) \mid a \in A\}$. The free Σ-term algebra over a set X is denoted by $T_\Sigma X$. The elements of $T_\Sigma X$ are called terms or trees. The substitution of the variables $x_1, \dots, x_n$ by the terms $s_1, \dots, s_n \in T_\Sigma$, respectively, in the term $t \in T_\Sigma$ is denoted by $[s_1/x_1, \dots, s_n/x_n]t$. The reader can find a more detailed description of the universal algebra needed for our belongings in Appendix B or in [Ihr88, Wec92].

2.3 Basic category theory

The reader who is unfamiliar with category theory can find a brief introduction in Appendix C. We will need the notions: (quasi-)(pre-)category, (full) subcategory, object, morphism, (faithful) functor, embedding, (horizontal and vertical composition of a) natural transformation, unique mediating morphism ((co-)mediator), initial/final object, (co-)product (functor), projection, injection, initial F-algebra, least fixed point of a functor, and catamorphism. In Subsection 2.4 we will introduce the notions of concrete categories and concrete functors.

For improving the readability we use various fonts: we usually write categories $\mathcal{C}, \mathcal{D}, \mathcal{E}, \dots$; objects $A, B, C, \dots$; morphisms $f, g, h, \dots$; functors $\mathsf{F}, \mathsf{G}, \mathsf{H}, \dots$; and natural transformations $\varrho, \sigma, \tau, \dots$.

We denote the final object by 1 and the initial object by 0 where we write $1 \xleftarrow{!_A} A$ and $A \xleftarrow{¡_A} 0$ for the unique mediating morphisms. The product of the I-indexed family $(A_i)_{i\in I}$ with projections $(\pi_i)_{i\in I}$ is denoted by $A_j \xleftarrow{\pi_j} \prod_{i\in I} A_i$. For the unique mediating morphism to $(\pi_i)_{i\in I}$ from $A_i \xleftarrow{f_i} B$ we write $\prod_{i\in I} A_i \xleftarrow{\langle f_i \rangle_{i\in I}} B$ and call it pairing. Dually we use the notations $\coprod$, ι, and $[\,.\,]$ for the coproduct, and we call the unique mediating morphism copairing. For every endofunctor F we write $\mu\mathsf{F} \xleftarrow{in_\mathsf{F}} \mathsf{F}(\mu\mathsf{F})$ for the initial F-algebra where $\mu\mathsf{F}$ denotes the least fixed point of F. The unique mediating morphism to $A \xleftarrow{\varphi} \mathsf{F}A$ from in_F is denoted by $A \xleftarrow{(\!|\varphi|\!)_\mathsf{F}} \mu\mathsf{F}$ and we call it catamorphism.

Our notation follows [AHS90] and [BdM97]. In particular, the arrows point to the left (*cf.* Subsection 2.1), *i.e.*

$$\forall A, B \in \mathrm{Ob}\,\mathcal{C} :: \; f : A \underset{\mathcal{C}}{\leftarrow} B \iff A \underset{\mathcal{C}}{\xleftarrow{f}} B \iff f \in \mathcal{C}(A, B).$$

Notice that $\mathcal{C}(A, B)$ is the hom-class of morphisms with *codomain* A and *domain* B.

2.4 Concrete categories

Many familiar examples for categories are constructs (*i.e.* categories of structured sets and structure-preserving functions between them). *E.g.* the category Σ-$\mathcal{Alg}$ of Σ-algebras is a construct. It turns out that many of the interesting properties of constructs result from the construction upon the base category $\mathcal{Set}$. If we view a construct as an *abstract* category we will loose the information of this construction. In order to keep the information we can describe the construction by a faithful functor $\mathsf{U} : \mathcal{Set} \leftarrow \Sigma\text{-}\mathcal{Alg}$, where U forgets the additional structure of a Σ-algebra, *i.e.* it maps Σ-algebras onto their carrier sets and Σ-algebra homomorphisms onto their underlying functions. The pair $(\Sigma\text{-}\mathcal{Alg}, \mathsf{U})$ (which we will call a *construct*) encodes the information of the construction. The concept of *concrete* categories is a generalization thereof, where the base category may differ from $\mathcal{Set}$.

For some category $\mathcal{C}$ we will use the concrete category $(\mathcal{Alg}_\mathcal{C}\mathsf{F}, |\,.\,|_\mathsf{F})$ built upon $\mathcal{C}$ in order to derive a more general notion of a ‘type functor’ (*cf.* [BdM97] Section 2.7), which we will need in our generalization of the ‘acid rain theorem’ (Definition and Corollary 3.2.6).

2.4.1 Definition (embedding, faithful functor). Let $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{D}$ be a functor.

(i) F is called an **embedding** provided that F is injective on morphisms and

(ii) F is called **faithful** provided that it is injective on hom-classes, *i.e.* $\forall A, B \in \mathcal{D} :: \forall f, g \in \mathcal{D}(A, B) :: \mathsf{F}f = \mathsf{F}g \implies f = g$. $*$

2.4.2 Note. A functor is an embedding iff it is injective on objects and faithful. $*$

2.4.3 Definition (concrete category, forgetful functor [AHS90]**).** Let $\mathcal{C}$ and $\mathcal{C}'$ be categories and $\mathsf{U} : \mathcal{C} \leftarrow \mathcal{C}'$ be a faithful functor. The pair $(\mathcal{C}', \mathsf{U})$ is called a **concrete category built upon $\mathcal{C}$** with **forgetful functor** U. We call $\mathcal{C}$ the **base category** of $(\mathcal{C}', \mathsf{U})$. We also say '$\mathcal{C}'$ is built upon $\mathcal{C}$', if there exists a functor U such that $(\mathcal{C}', \mathsf{U})$ is a concrete category built upon $\mathcal{C}$. For every $\mathcal{C}'$-object A we call $\mathsf{U}A$ the **underlying $\mathcal{C}$-object** (or **carrier**) of A and for every $\mathcal{C}'$-morphism f we call $\mathsf{U}f$ the **underlying $\mathcal{C}$-morphism** of f. A concrete category built upon $\mathcal{S}et$ is called a **construct**. $*$

2.4.4 Example (concrete category, forgetful functor). The category Σ-$\mathcal{A}lg$ is built upon $\mathcal{S}et$. The forgetful functor maps a Σ-algebra onto its carrier set. More generally: Let $\mathcal{C}$ be a category and $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{C}$ an endofunctor. The category $\mathcal{A}lg_{\mathcal{C}}\mathsf{F}$ is built upon $\mathcal{C}$. The forgetful functor $|\,.\,|_{\mathsf{F}}$ maps an F-algebra onto its carrier. $*$

2.4.5 Definition (concrete functor [AHS90]**).** Let $\mathcal{C}$ be a category and $(\mathcal{D}, \mathsf{U})$ and $(\mathcal{D}', \mathsf{U}')$ be concrete categories built upon $\mathcal{C}$. A functor $\mathsf{F} : \mathcal{D} \leftarrow \mathcal{D}'$ is called a **concrete functor** to $(\mathcal{D}, \mathsf{U})$ from $(\mathcal{D}', \mathsf{U}')$ provided that

$$\mathsf{U} \cdot \mathsf{F} = \mathsf{U}'.$$

In this case we write

$$\mathsf{F} : (\mathcal{D}, \mathsf{U}) \leftarrow (\mathcal{D}', \mathsf{U}').$$

$*$

2.4.6 Lemma ([AHS90]**).** A concrete functor is completely determined by its values on objects.

Proof. Let $\mathcal{C}$ be a category and $(\mathcal{D}, \mathsf{U})$ and $(\mathcal{D}', \mathsf{U}')$ be concrete categories built upon $\mathcal{C}$. Let $\mathsf{F}, \mathsf{G} : (\mathcal{D}, \mathsf{U}) \leftarrow (\mathcal{D}', \mathsf{U}')$ be two concrete functors with $\forall A \in \mathrm{Ob}\,\mathcal{D}' :: \mathsf{F}A = \mathsf{G}A$. We have to show that $\mathsf{F} = \mathsf{G}$. Let $f : A \underset{\mathcal{D}'}{\longleftarrow} B$. Then

$$\mathsf{F}f, \mathsf{G}f : \mathsf{F}A = \mathsf{G}A \underset{\mathcal{D}}{\longleftarrow} \mathsf{F}B = \mathsf{G}B,$$

i.e. $\mathsf{F}f$ and $\mathsf{G}f$ are morphisms in the same hom-class. Since F and G are concrete, it holds

$$\mathsf{U}(\mathsf{F}f) = \mathsf{U}'f = \mathsf{U}(\mathsf{G}f)$$

and thus $\mathsf{F}f = \mathsf{G}f$ because U is faithful. ∎

2.4.7 Definition and Lemma (quasi-categories of concrete categories). Let $\mathcal{C}$ be a category. It is easy to see that identity functors and the compositions of concrete functors are concrete. Thus the conglomerate of all concrete categories built upon $\mathcal{C}$ is the object class of a quasi-pre-category with all concrete functors as morphisms. We denote the according quasi-category (see Note C.1.4) of all concrete categories built upon $\mathcal{C}$ by $c\mathcal{CAT}\ \mathcal{C}$. The quasi-category $c\mathcal{CAT}\ \mathcal{C}$ itself is built upon the quasi-category $\mathcal{CAT}$ by the forgetful functor $|\,.\,|$ which maps a $(c\mathcal{CAT}\ \mathcal{C})$-morphism to its underlying $\mathcal{CAT}$-morphism, *i.e.* functor. $*$

2.4.8 Note. Let $\mathcal{C}$ be a category and $(c\mathcal{CAT}\ \mathcal{C}, |\,.\,|)$ be the concrete quasi-category built upon $\mathcal{CAT}$ from Definition and Lemma 2.4.7. Let $(\mathcal{D}, \mathsf{U}), (\mathcal{D}', \mathsf{U}') \in \mathrm{Ob}(c\mathcal{CAT}\ \mathcal{C})$ be two concrete categories and $\mathsf{F} : (\mathcal{D}, \mathsf{U}) \leftarrow (\mathcal{D}', \mathsf{U}')$ be a concrete functor. We may view F out of three different perspectives:

(i) as the functor $\mathsf{F} : \mathcal{D} \leftarrow \mathcal{D}'$ (*i.e.* as a $\mathcal{CAT}$-morphism) with the property $\mathsf{U} \cdot \mathsf{F} = \mathsf{U}'$,

(ii) as the concrete functor $\mathsf{F} : (\mathcal{D}, \mathsf{U}) \leftarrow (\mathcal{D}', \mathsf{U}')$, *i.e.* as a morphism in the quasi-pre-category from Definition and Lemma 2.4.7, or

(iii) as the $c\mathcal{CAT}\ \mathcal{C}$-morphism $\big((\mathcal{D}, \mathsf{U}), \mathsf{F}, (\mathcal{D}', \mathsf{U}')\big) : (\mathcal{D}, \mathsf{U}) \leftarrow (\mathcal{D}', \mathsf{U}')$.

The connection

- from (i) to (ii) is the Definition 2.4.5 (and Definition and Lemma 2.4.7),
- from (ii) to (iii) is the construction from Note C.1.4, and
- from (iii) to (i) is the forgetful functor: $|\big((\boldsymbol{\mathcal{D}}, \mathsf{U}), \mathsf{F}, (\boldsymbol{\mathcal{D}}', \mathsf{U}')\big)| = \mathsf{F}$. $*$

2.4.9 Definition and Lemma ('forgetting more' is a concrete functor). Let $\boldsymbol{\mathcal{C}}$ and $\boldsymbol{\mathcal{C}}'$ be categories and $(\boldsymbol{\mathcal{D}}_1, \mathsf{U}_1)$ and $(\boldsymbol{\mathcal{D}}_2, \mathsf{U}_2)$ be concrete categories built upon $\boldsymbol{\mathcal{C}}'$ and let $\mathsf{U} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}'$ be a faithful functor. Then $(\boldsymbol{\mathcal{D}}_1, \mathsf{U} \cdot \mathsf{U}_1)$ and $(\boldsymbol{\mathcal{D}}_2, \mathsf{U} \cdot \mathsf{U}_2)$ are concrete categories built upon $\boldsymbol{\mathcal{C}}$ and if $\mathsf{H} : (\boldsymbol{\mathcal{D}}_1, \mathsf{U}_1) \leftarrow (\boldsymbol{\mathcal{D}}_2, \mathsf{U}_2)$ is a concrete functor, so is $\mathsf{H} : (\boldsymbol{\mathcal{D}}_1, \mathsf{U} \cdot \mathsf{U}_1) \leftarrow (\boldsymbol{\mathcal{D}}_2, \mathsf{U} \cdot \mathsf{U}_2)$. This motivates the definition of the function ${}_{\mathsf{U}}(\,\centerdot\,)$ by

$$ {}_{\mathsf{U}}\big((\boldsymbol{\mathcal{D}}_1, \mathsf{U}_1), \mathsf{H}, (\boldsymbol{\mathcal{D}}_2, \mathsf{U}_2)\big) = \big((\boldsymbol{\mathcal{D}}_1, \mathsf{U} \cdot \mathsf{U}_1), \mathsf{H}, (\boldsymbol{\mathcal{D}}_2, \mathsf{U} \cdot \mathsf{U}_2)\big). $$

This function is a concrete functor

$$ {}_{\mathsf{U}}(\,\centerdot\,) : (\boldsymbol{cCAT}\ \boldsymbol{\mathcal{C}}, |\,\centerdot\,|) \leftarrow (\boldsymbol{cCAT}\ \boldsymbol{\mathcal{C}}', |\,\centerdot\,|). $$

Proof. With the operation on objects ${}_{\mathsf{U}}(\boldsymbol{\mathcal{D}}_1, \mathsf{U}_1) = (\boldsymbol{\mathcal{D}}_1, \mathsf{U} \cdot \mathsf{U}_1)$ the typing axiom is obvious. The function ${}_{\mathsf{U}}(\,\centerdot\,)$ is also multiplicative and preserves identities, because it operates trivially on morphisms, *i.e.* it only changes domain and codomain. The concreteness property of ${}_{\mathsf{U}}(\,\centerdot\,)$ follows from $|{}_{\mathsf{U}}\big((\boldsymbol{\mathcal{D}}_1, \mathsf{U}_1), \mathsf{H}, (\boldsymbol{\mathcal{D}}_2, \mathsf{U}_2)\big)| = |\big((\boldsymbol{\mathcal{D}}_1, \mathsf{U} \cdot \mathsf{U}_1), \mathsf{H}, (\boldsymbol{\mathcal{D}}_2, \mathsf{U} \cdot \mathsf{U}_2)\big)| = \mathsf{H} = |\big((\boldsymbol{\mathcal{D}}_1, \mathsf{U}_1), \mathsf{H}, (\boldsymbol{\mathcal{D}}_2, \mathsf{U}_2)\big)|$. ■

$*$

3 Calculating with initial and final objects

In this section we will develop our tools to calculate with the morphisms of a category. The basic idea is to use the laws for initial and final objects (*cf.* Tables 4 and 5). We consider such a concrete category that the underlying category has an initial/final object and investigate the laws in the base category. This is a generalization of the well known constructions for (co-)product functors (Definitions and Corollaries C.5.1 and C.5.2).

The following Theorem 3.1.1 enables us to construct functors for arbitrary limits[3]: We can conveniently translate the laws (reflection, fusion and cancelation) for (co-)mediators into laws for the according functors. In the case of (co-)products, the functors derived by Theorem 3.1.1 are the well known (co-)product functors. However the use of a quasi-category of concrete functors enables us to derive more general functors: n the case of initial algebras this directly leads to our version of the 'acid rain theorem' or to generalizations thereof.

3.1 Deriving functors by initiality or finality

3.1.1 Theorem (deriving natural transformations and functors from initiality (*cf.* Theorem 4.1.1 of [Jür00])**).** Let $\boldsymbol{\mathcal{C}}$ and $\boldsymbol{\mathcal{D}}$ be categories and

$$ \boldsymbol{CAT} \xleftarrow{|\,\centerdot\,|} \boldsymbol{cCAT}\ \boldsymbol{\mathcal{C}} \xleftarrow{\mathsf{E}} \boldsymbol{\mathcal{D}} $$

where $(\boldsymbol{cCAT}\ \boldsymbol{\mathcal{C}}, |\,\centerdot\,|)$ is the concrete quasi-category from Definition and Lemma 2.4.7 and $\mathsf{E} : \boldsymbol{cCAT}\ \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ is a functor. For every $D \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}}$ let the category $|\mathsf{E}D|$ have an initial object $0_D \in \mathrm{Ob}\,|\mathsf{E}D|$ where for every $A \in \mathrm{Ob}\,|\mathsf{E}D|$

$$ \mathfrak{i}^D_A : A \underset{|\mathsf{E}D|}{\longleftarrow} 0_D $$

is the unique mediating morphism. Let $\mathsf{U}_D : \boldsymbol{\mathcal{C}} \leftarrow |\mathsf{E}D|$ be the forgetful functor of the concrete category $\mathsf{E}D$, *i.e.* $\mathsf{E}D = (|\mathsf{E}D|, \mathsf{U}_D)$. Now we define two functions:

[3]A *limit* is a general notion for universal constructions like (co-)products, initial/final algebras, *etc.*.

(a) the function τ on $\boldsymbol{\mathcal{D}}$-objects by

$$\forall\, D \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}} ::\ \tau^D = (\tau^D_A)_{A \in \mathrm{Ob}\,|\mathsf{E}D|} \text{ where } \forall\, A \in \mathrm{Ob}\,|\mathsf{E}D| ::\ \tau^D_A = \mathsf{U}_D \text{¡}^D_A$$

(b) and the function F on $\boldsymbol{\mathcal{D}}$-objects by

$$\forall\, D \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}} ::\ \mathsf{F}D = \mathsf{U}_D 0_D$$

and on $\boldsymbol{\mathcal{D}}$-morphisms by

$$\forall\, D, D' \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}} ::\ \forall\, d \in \boldsymbol{\mathcal{D}}(D, D') ::\ \mathsf{F}d = \tau^D_{|\mathsf{E}d|0_{D'}}.$$

Then the following holds:

(i) for every $D \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}}$, the function τ^D is a natural transformation $\tau^D : \mathsf{U}_D \overset{\cdot}{\leftarrow} \mathsf{K}_{\mathsf{F}D}$,

(ii) for every $D, D' \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}}$, every $d \in \boldsymbol{\mathcal{D}}(D, D')$, and $A \in \mathrm{Ob}\,|\mathsf{E}D'|$ we have $\tau^{D'}_A \cdot \mathsf{F}d = \tau^D_{|\mathsf{E}d|A}$, and

(iii) $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}^{\mathrm{op}}$ is a functor.

Proof. Let $D, D' \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}}$ and $d \in \boldsymbol{\mathcal{D}}(D, D')$.

(i) Due to Corollary C.3.5 the comediator is a natural transformation $\text{¡}^D : \mathsf{Id} \overset{\cdot}{\leftarrow} \mathsf{K}_{0_D}$. Thus with Definition and Lemma C.2.6 (i) $\tau^D = \mathsf{U}_D \text{¡}^D$ is a natural transformation $\tau^D : \mathsf{U}_D \overset{\cdot}{\leftarrow} \mathsf{U}_D \cdot \mathsf{K}_{0_D}$. From (b) and Corollary C.1.9 we obtain $\mathsf{U}_D \cdot \mathsf{K}_{0_D} = \mathsf{K}_{\mathsf{U}_D 0_D} = \mathsf{K}_{\mathsf{F}D}$.

(ii) Let $A \in \mathrm{Ob}\,|\mathsf{E}D'|$. Starting with (i) we infer

$$\begin{aligned}
& \tau^D : \mathsf{U}_D \overset{\cdot}{\leftarrow} \mathsf{K}_{\mathsf{F}D} \\
\Longrightarrow\quad & \{ \text{ Definition Lemma C.2.6 (ii) and Corollary C.1.9 } \} \\
& \tau^D|\mathsf{E}d| : \mathsf{U}_D \cdot |\mathsf{E}d| \overset{\cdot}{\leftarrow} \mathsf{K}_{\mathsf{F}D} \\
\Longrightarrow\quad & \{ \ |\mathsf{E}d| \text{ is concrete, } \textit{i.e.}\ \mathsf{U}_D \cdot |\mathsf{E}d| = \mathsf{U}_{D'} \ \} \\
& \tau^D|\mathsf{E}d| : \mathsf{U}_{D'} \overset{\cdot}{\leftarrow} \mathsf{K}_{\mathsf{F}D} \\
\Longrightarrow\quad & \{ \text{ naturalness condition for } \tau^D|\mathsf{E}d| \text{ from Definition C.2.1 applied to } \text{¡}^{D'}_A \ \} \\
& \mathsf{U}_{D'} \text{¡}^{D'}_A \cdot (\tau^D|\mathsf{E}d|)_{0_{D'}} = (\tau^D|\mathsf{E}d|)_A \cdot \mathsf{K}_{\mathsf{F}D} \text{¡}^{D'}_A \\
\Longrightarrow\quad & \{ \text{ Definition and Lemma C.2.6 (ii) and definition of } \mathsf{K} \text{ in Definition C.1.8 } \} \\
& \mathsf{U}_{D'} \text{¡}^{D'}_A \cdot \tau^D_{|\mathsf{E}d|0_{D'}} = \tau^D_{|\mathsf{E}d|A} \cdot id_{\mathsf{F}D} \\
\Longrightarrow\quad & \{ \text{ definition of } \tau \text{ in (a) and of } \mathsf{F} \text{ in (b) } \} \\
& \tau^{D'}_A \cdot \mathsf{F}d = \tau^D_{|\mathsf{E}d|A}.
\end{aligned}$$

(iii) We have to verify the functor axioms of Definition C.1.8 for F:

- To check the typing axiom, we calculate the domain of $\mathsf{F}d$ using (a) and (b) and the typing axiom of the functor U_D

  $$\mathrm{dom}(\mathsf{F}d) = \mathrm{dom}(\tau^D_{|\mathsf{E}d|0_{D'}}) = \mathrm{dom}(\mathsf{U}_D \text{¡}^D_{|\mathsf{E}d|0_{D'}}) = \mathsf{U}_D(\mathrm{dom}\,\text{¡}^D_{|\mathsf{E}d|0_{D'}}) = \mathsf{U}_D 0_D = \mathsf{F}D$$

  and the codomain of $\mathsf{F}d$ using (ii) and the typing axiom of the functor $\mathsf{U}_{D'}$

  $$\mathrm{cod}(\mathsf{F}d) = \mathrm{cod}\,\tau^D_{|\mathsf{E}d|0_{D'}}\ \mathrm{cod}\,\tau^{D'}_{0_{D'}} = \mathrm{cod}(\mathsf{U}_{D'} \text{¡}^{D'}_{0_{D'}}) = \mathsf{U}_{D'}(\mathrm{cod}\,\text{¡}^{D'}_{0_{D'}}) = \mathsf{U}_{D'} 0_{D'} = \mathsf{F}D'.$$

- Using the reflection law of Table 4 for ¡^D and the identity axioms of the functors $|\,.\,|$, E, and U_D we calculate

  $$\mathsf{F}id_D = \tau^D_{|\mathsf{E}id_D|0_D} = \tau^D_{0_D} = \mathsf{U}_D \text{¡}^D_{0_D} = \mathsf{U}_D id_{0_D} = id_{\mathsf{U}_D 0_D} = id_{\mathsf{F}D}.$$

  Thus the identity axiom is satisfied for F.

- It remains to show that F satisfies the multiplicativity axiom: Let $D'' \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}}$, $e \in \boldsymbol{\mathcal{D}}(D', D'')$, and $A = |\mathsf{E}e|0_{D''}$. Starting with (ii) we infer

$$
\begin{aligned}
& \tau_A^{D'} \cdot \mathsf{F}d = \tau^D_{|\mathsf{E}d|A} \\
\Longrightarrow \quad & \{ \text{ definition of } A \text{ and of } \mathsf{F} \text{ in (b) } \} \\
& \mathsf{F}e \ \cdot \ \mathsf{F}d = \tau^D_{|\mathsf{E}d| \cdot |\mathsf{E}e|0_{D''}} \\
\Longrightarrow \quad & \{ \text{ multiplicativity of the functors } \mathsf{E} \text{ and } |\,\boldsymbol{.}\,|\text{; definition of } \mathsf{F} \text{ in (b) } \} \\
& \mathsf{F}e \ \cdot \ \mathsf{F}d = \mathsf{F}(d \cdot_{\boldsymbol{\mathcal{D}}} e) = \mathsf{F}(e \cdot_{\boldsymbol{\mathcal{D}}^{\mathrm{op}}} d).
\end{aligned}
$$

The dual proposition (for finality and $\boldsymbol{\mathcal{C}}^{\mathrm{op}}$ and $\boldsymbol{\mathcal{D}}^{\mathrm{op}}$) is also true. ∎

3.1.2 Note. The functor F in Theorem 3.1.1 is only determined uniquely up to isomorphism. It depends on the choice of the initial objects $(0_D)_{D \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}}}$, which are only determined uniquely up to isomorphism themselves. $*$

3.1.3 Corollary. With the preconditions and notations from Theorem 3.1.1 we obtain the laws in Table 1.

Laws for F	
reflection	$\mathsf{F}id_D = id_{\mathsf{F}D}$
fusion (i)	$\tau_A^{D'} \ \cdot \ \mathsf{F}d = \tau^D_{|\mathsf{E}d|A}$
fusion (ii)	$\mathsf{F}e \ \cdot \ \mathsf{F}d = \mathsf{F}(e \cdot d)$
where $D, D' \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}}$, $A \in |\mathsf{E}D'|$, and $e, d \in \mathrm{Mor}\,\boldsymbol{\mathcal{D}}^{\mathrm{op}}$	

Table 1: Laws for F

Proof. The fusion (i) law follows from Theorem 3.1.1 (ii). The reflection and fusion (ii) laws are functor axioms of F and follow from Theorem 3.1.1 (iii). ∎

3.2 Generalized ‘acid rain theorems’

The ‘acid rain theorem’ is a generalization of the cata/build-rule. We can generalize it even more:

3.2.1 Corollary (generalized ‘short cut fusion’ or ‘acid rain theorem’ (*cf.* Corollary 4.3.1 of [Jür00]**)).** Let $\boldsymbol{\mathcal{C}}$ be a category and $A \in \mathrm{Ob}\boldsymbol{\mathcal{C}}$. Let (D, U_D) and $(D', \mathsf{U}_{D'})$ be concrete categories built upon $\boldsymbol{\mathcal{C}}$, such that the categories D and D' have initial objects. With the preconditions and notations from Theorem 3.1.1 we get

$$
\frac{\mathsf{H} : (D, \mathsf{U}_D) \leftarrow (D', \mathsf{U}_{D'})}{\tau_A^{D'} \ \cdot \ \tau^D_{\mathsf{H}0_{D'}} = \tau^D_{\mathsf{H}A}}
$$

Proof. This is an instance of Theorem 3.1.1 (ii) where $\boldsymbol{\mathcal{D}} \subseteq \boldsymbol{cCAT}\ \boldsymbol{\mathcal{C}}$ is the full subcategory with $\mathrm{Ob}\,\boldsymbol{\mathcal{D}} = \{(D, \mathsf{U}_D), (D', \mathsf{U}_{D'})\}$ and embedding functor $\mathsf{E} : \boldsymbol{cCAT}\ \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$. With $|d| = \mathsf{H}$ it follows that $\mathsf{F}d = \tau^D_{\mathsf{H}0_{D'}}$. ∎

The well known ‘acid rain theorem’ is just an instance of this:

3.2.2 Corollary (mutual ‘acid rain theorem’). Let $\boldsymbol{\mathcal{C}}$ be a category and $\mathsf{F}, \mathsf{G}, \mathsf{U} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$ be endofunctors such that the categories $\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\mathsf{F}$ and $\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\mathsf{G}$ have initial objects and U is faithful. Let $\varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\mathsf{F})$.

$$
\frac{\mathsf{H} : (\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\mathsf{F}, |\,\boldsymbol{.}\,|_{\mathsf{F}}) \leftarrow (\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\mathsf{G}, \mathsf{U} \cdot |\,\boldsymbol{.}\,|_{\mathsf{G}})}{\mathsf{U}(\![\varphi]\!)_{\mathsf{G}} \cdot (\![\mathsf{H}in_{\mathsf{G}}]\!)_{\mathsf{F}} = (\![\mathsf{H}\varphi]\!)_{\mathsf{F}}}.
$$

Proof. This is an instance of Corollary 3.2.1 with $D = \mathcal{A}lg_{\mathcal{C}}\mathsf{F}$, $\mathsf{U}_D = |\bullet|_{\mathsf{F}}$, $D' = \mathcal{A}lg_{\mathcal{C}}\mathsf{G}$, and $\mathsf{U}'_D = \mathsf{U}\cdot|\bullet|_{\mathsf{G}}$. Note that, from the definition of catamorphisms (Definition and Lemma C.6.1) and the definition of τ in Theorem 3.1.1 (a), we obtain

$$\mathsf{U}(\!|\varphi|\!)_{\mathsf{G}} = \mathsf{U}|\text{¡}^{D'}_{\varphi}|_{\mathsf{G}} = \mathsf{U}_{D'}\text{¡}^{D'}_{\varphi} = \tau^{D'}_{\varphi}.$$

■

3.2.3 Note. In the above Corollary 3.2.2 we use the concrete functor H in order to map G-algebras onto F-algebras such that the U is applied on the carrier. If $\mathsf{U} = \mathsf{Id}$ this means that H maps G-algebras onto F-algebras with the same carrier. A function on algebras (and more general on dialgebras) with this properties is called an *algebra transformer* in [Fok92b].

3.2.4 Corollary ('short cut fusion' or 'acid rain theorem' (*cf.* Theorem 3.2 of [TM95] and Corollary 4.3.2 of [Jür00])**).** Let $\mathcal{C}$ be a category and $\mathsf{F}, \mathsf{G} : \mathcal{C} \leftarrow \mathcal{C}$ endofunctors such that the categories $\mathcal{A}lg_{\mathcal{C}}\mathsf{F}$ and $\mathcal{A}lg_{\mathcal{C}}\mathsf{G}$ have initial objects. Let $\varphi \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\mathsf{F})$.

$$\frac{\mathsf{H} : (\mathcal{A}lg_{\mathcal{C}}\mathsf{F}, |\bullet|_{\mathsf{F}}) \leftarrow (\mathcal{A}lg_{\mathcal{C}}\mathsf{G}, |\bullet|_{\mathsf{G}})}{(\!|\varphi|\!)_{\mathsf{G}} \cdot (\!|\mathsf{H}\,in_{\mathsf{G}}|\!)_{\mathsf{F}} = (\!|\mathsf{H}\varphi|\!)_{\mathsf{F}}}.$$

Proof. This is just an instance of Corollary 3.2.2 where $\mathsf{U} = \mathsf{Id}$. ■

3.2.5 Note. It is worth stressing that Corollary 3.2.4 is just a corollary of Theorem 3.1.1. In contrast to this, the classical 'acid rain theorem' (*cf.* Theorem 3.2 of [TM95]) requires a naturalness precondition. In its turn, this needs the **parametricity** property or **free theorem**, *i.e.* polymorphic functions of the functional programming language will be mapped onto *natural* transformations by the categorical semantics (*cf.* [Wad89, dB89]). Our new approach is entirely different, because neither language nor semantics concepts occured — it is pure category theory. $*$

Theorem 3.1.1 supplies not only a natural transformation but also a functor. We will use this functor to give a new version of the 'acid rain theorem' which is symmetrical, *i.e.* the form of either of the two morphisms we fuse is similar:

3.2.6 Definition and Corollary (functorial 'short cut fusion' or 'acid rain theorem' (*cf.* Definition and Corollary 4.3.4 of [Jür00])**).** Let $\mathcal{C}$ be a category. We define $\mathcal{D}$ to be the full subcategory of $\boldsymbol{cCAT}\ \mathcal{C}$ with

$$\mathrm{Ob}(\mathcal{D}) = \left\{(\mathcal{A}lg_{\mathcal{C}}\mathsf{F}, \mathsf{U}_{\mathsf{F}}\cdot|\bullet|_{\mathsf{F}}) \;\middle|\; \mathsf{F} : \mathcal{C} \leftarrow \mathcal{C}, \mathcal{A}lg_{\mathcal{C}}\mathsf{F} \text{ has an initial object}, \mathsf{U}_{\mathsf{F}} : \mathcal{C} \leftarrow \mathcal{C} \text{ faithful}\right\}$$

and the function

$$\forall\,\mathsf{H} \in \mathcal{D}\big((\mathcal{A}lg_{\mathcal{C}}\mathsf{F}, \mathsf{U}_{\mathsf{F}}\cdot|\bullet|_{\mathsf{F}}), (\mathcal{A}lg_{\mathcal{C}}\mathsf{G}, \mathsf{U}_{\mathsf{G}}\cdot|\bullet|_{\mathsf{G}})\big) :: \ \mathsf{M}\,\mathsf{H} = \mathsf{U}_{\mathsf{F}}(\!|\,|\mathsf{H}|in_{\mathsf{G}}|\!)_{\mathsf{F}}.$$

The latter is a functor

$$\mathsf{M} : \mathcal{C} \leftarrow \mathcal{D}^{\mathrm{op}}$$

and in particular

$$\frac{(\mathcal{A}lg_{\mathcal{C}}\mathsf{F}_3, \mathsf{U}_{\mathsf{F}_3}\cdot|\bullet|_{\mathsf{F}_3}) \xleftarrow[\mathcal{D}]{\mathsf{H}_2} (\mathcal{A}lg_{\mathcal{C}}\mathsf{F}_2, \mathsf{U}_{\mathsf{F}_2}\cdot|\bullet|_{\mathsf{F}_2}) \xleftarrow[\mathcal{D}]{\mathsf{H}_1} (\mathcal{A}lg_{\mathcal{C}}\mathsf{F}_1, \mathsf{U}_{\mathsf{F}_1}\cdot|\bullet|_{\mathsf{F}_1})}{\mathsf{U}_{\mathsf{F}_2}(\!|\,|\mathsf{H}_1|in_{\mathsf{F}_1}|\!)_{\mathsf{F}_2}\cdot\mathsf{U}_{\mathsf{F}_3}(\!|\,|\mathsf{H}_2|in_{\mathsf{F}_2}|\!)_{\mathsf{F}_3} = \mathsf{U}_{\mathsf{F}_3}(\!|\,|\mathsf{H}_2\cdot\mathsf{H}_1|in_{\mathsf{F}_1}|\!)_{\mathsf{F}_3}}.$$

Proof. We use Theorem 3.1.1 (ii) with $\mathsf{E} = \mathsf{Id}$. With $in_{\mathsf{F}_i} = 0_{(\mathcal{A}lg_{\mathcal{C}}\mathsf{F}_i, \mathsf{U}_{\mathsf{F}_i}\cdot|\bullet|_{\mathsf{F}_i})}$ and the definition of F in Theorem 3.1.1 (b) and the fact that $\mathsf{U}_{\mathsf{F}_i}(\!|\varphi|\!)_{\mathsf{F}_i} = \tau_{\varphi}^{(\mathcal{A}lg_{\mathcal{C}}\mathsf{F}_i, \mathsf{U}_{\mathsf{F}_i}\cdot|\bullet|_{\mathsf{F}_i})}$ (*cf.* the proof of Corollary 3.2.4) it is easy to see that $\mathsf{M} = \mathsf{F}$, because

$$\mathsf{F}\,\mathsf{H}_1 = \tau_{|\mathsf{H}|in_{\mathsf{F}_1}}^{(\mathcal{A}lg_{\mathcal{C}}\mathsf{F}_2, \mathsf{U}_{\mathsf{F}_2}\cdot|\bullet|_{\mathsf{F}_2})} = \mathsf{U}_{\mathsf{F}_2}(\!|\,|\mathsf{H}_1|in_{\mathsf{F}_1}|\!)_{\mathsf{F}_2} = \mathsf{M}\,\mathsf{H}_1.$$

∎

The functor M is a generalization of the 'type functor' μ : $(\mu\mathsf{F} \xleftarrow{(\![in_\mathsf{F} \cdot \tau]\!)_\mathsf{G}} \mu\mathsf{G}) \leftarrow\!\shortmid (\mathsf{F} \xleftarrow{\tau} \mathsf{G})$ (*cf.* [BdM97] Section 2.7), *i.e.* $\mathsf{M\,H} = \mu$ where $\forall\,\varphi ::\ \mathsf{H}\varphi = \varphi\tau$ (Later in Corollary 3.3.2 (ii) we will see that this indeed defines a concrete functor H).

3.2.7 Lemma. Let $\mathcal{C}$ be a category and $\mathsf{U} : \mathcal{C} \leftarrow \mathcal{C}$ be faithful. Then

$$\mathsf{M} \cdot {}_\mathsf{U}(\,\centerdot\,) = \mathsf{U} \cdot \mathsf{M}$$

where ${}_\mathsf{U}(\,\centerdot\,) : \mathcal{D} \leftarrow \mathcal{D}$ is the functor defined in Definition and Definition and Lemma 2.4.9 restricted to $\mathcal{D}$ defined in Definition and Corollary 3.2.6.

Proof. For every

$$(\mathcal{A}lg_\mathcal{C}\mathsf{F}, \mathsf{U_F} \cdot |\,\centerdot\,|_\mathsf{F}) \xleftarrow[\mathcal{D}]{\mathsf{H}} (\mathcal{A}lg_\mathcal{C}\mathsf{G}, \mathsf{U_G} \cdot |\,\centerdot\,|_\mathsf{G})$$

we calculate

$$\mathsf{M}({}_\mathsf{U}\mathsf{H}) = (\mathsf{U} \cdot \mathsf{U_F})(\![|\mathsf{H}|in_\mathsf{G}]\!)_\mathsf{F} = \mathsf{U}\big(\mathsf{U_F}(\![|\mathsf{H}|in_\mathsf{G}]\!)_\mathsf{F}\big) = \mathsf{U}(\mathsf{M\,H}). \tag{3.2.1}$$

∎

3.3 Characterization of concrete functors between categories of algebras

3.3.1 Lemma (characterization of concrete functors on categories of algebras (*cf.* Lemma 4.4.1 of [Jür00])**).** Let $\mathcal{C}$ be a category, $\mathsf{F}, \mathsf{G}, \mathsf{U} : \mathcal{C} \leftarrow \mathcal{C}$ be endofunctors where U is faithful and $\mathsf{H} : \mathrm{Ob}(\mathcal{A}lg_\mathcal{C}\mathsf{F}) \leftarrow \mathrm{Ob}(\mathcal{A}lg_\mathcal{C}\mathsf{G})$ a function. The following two statements are equivalent:

(i) The function H can be uniquely extended on $(\mathcal{A}lg_\mathcal{C}\mathsf{G})$-morphisms to a concrete functor

$$\mathsf{H} : (\mathcal{A}lg_\mathcal{C}\mathsf{F}, |\,\centerdot\,|_\mathsf{F}) \leftarrow (\mathcal{A}lg_\mathcal{C}\mathsf{G}, \mathsf{U} \cdot |\,\centerdot\,|_\mathsf{G}).$$

(ii) The function H satisfies the following condition: for every $\varphi, \varphi' \in \mathrm{Ob}(\mathcal{A}lg_\mathcal{C}\mathsf{G})$ and every $f : |\varphi|_\mathsf{G} \xleftarrow[\mathcal{C}]{} |\varphi'|_\mathsf{G}$:

$$\frac{\varphi \cdot \mathsf{G}f = f \cdot \varphi'}{\mathsf{H}\varphi \cdot \mathsf{F}(\mathsf{U}f) = \mathsf{U}f \cdot \mathsf{H}\varphi'}. \tag{$*$}$$

Notice the different usage of functors F, G on morphisms and H on objects in the above equations.

Proof. '(ii) $\Longrightarrow$ (i)': We extend H on morphisms by

$$\forall\,\varphi, \varphi' \in \mathrm{Ob}(\mathcal{A}lg_\mathcal{C}\mathsf{G}) ::\ \forall\,(\varphi, f, \varphi') \in \mathcal{A}lg_\mathcal{C}\mathsf{G}(\varphi, \varphi') ::\ \mathsf{H}(\varphi, f, \varphi') = (\mathsf{H}\varphi, \mathsf{U}f, \mathsf{H}\varphi').$$

If this is a functor $\mathsf{H} : \mathcal{A}lg_\mathcal{C}\mathsf{F} \leftarrow \mathcal{A}lg_\mathcal{C}\mathsf{G}$, then it is obviously concrete, *i.e.*: $\mathsf{H} : (\mathcal{A}lg_\mathcal{C}\mathsf{F}, |\,\centerdot\,|_\mathsf{F}) \leftarrow (\mathcal{A}lg_\mathcal{C}\mathsf{G}, \mathsf{U} \cdot |\,\centerdot\,|_\mathsf{G})$ and thus uniquely determined due to Lemma 2.4.6. The function H satisfies the functor axioms by construction, because U is a functor. The only property that we have to verify is $\mathsf{H} : \mathrm{Mor}(\mathcal{A}lg_\mathcal{C}\mathsf{F}) \leftarrow \mathrm{Mor}(\mathcal{A}lg_\mathcal{C}\mathsf{G})$, *i.e.* H maps G-algebra-morphisms onto F-algebra-morphisms. This is equivalent to the condition ($*$) which is easy to see using the definition of H on morphisms and Definition and Lemma C.6.1.

'(i) $\Longrightarrow$ (ii)': If H can be extended uniquely to a concrete functor $\mathsf{H} : (\mathcal{A}lg_\mathcal{C}\mathsf{F}, |\,\centerdot\,|_\mathsf{F}) \leftarrow (\mathcal{A}lg_\mathcal{C}\mathsf{G}, \mathsf{U} \cdot |\,\centerdot\,|_\mathsf{G})$, then in particular $\mathsf{H} : \mathrm{Mor}(\mathcal{A}lg_\mathcal{C}\mathsf{F}) \leftarrow \mathrm{Mor}(\mathcal{A}lg_\mathcal{C}\mathsf{G})$. Let $\varphi, \varphi' \in \mathrm{Ob}(\mathcal{A}lg_\mathcal{C}\mathsf{F})$ and $f : \varphi \leftarrow \varphi'$, which is equivalent to the precondition of ($*$) for $|f|$. We have $\mathsf{H}f : \mathsf{H}\varphi \leftarrow \mathsf{H}\varphi'$ and thus $\mathsf{H}\varphi \cdot \mathsf{F}|\mathsf{H}f|_\mathsf{F} = |\mathsf{H}f|_\mathsf{F} \cdot \mathsf{H}\varphi'$. Using the concreteness of H, *i.e.* $|\mathsf{H}f|_\mathsf{F} = \mathsf{U}|f|_\mathsf{F}$, the latter is equivalent to the proposition of ($*$). ∎

The following corollary shows how to construct concrete functors on categories of algebras using the condition from Lemma 3.3.1.

3.3.2 Corollary (Construction of concrete functors (*cf.* Corollary 4.4.3 of [Jür00])**).** With the preconditions from Lemma 3.3.1, each of the following definitions yields an $\mathsf{H} : (\mathcal{Alg}_{\mathcal{C}}\mathsf{G}, |\,.\,|_{\mathsf{G}}) \leftarrow (\mathcal{Alg}_{\mathcal{C}}\mathsf{F}, |\,.\,|_{\mathsf{F}})$. For every $\varphi \in \mathrm{Ob}(\mathcal{Alg}_{\mathcal{C}}\mathsf{F})$:

(i) $\mathsf{H}\varphi = \varphi \cdot \mathsf{F}\varphi$ where $\mathsf{G} = \mathsf{F} \cdot \mathsf{F}$,

(ii) $\mathsf{H}\varphi = \varphi \cdot \tau$ where $\tau : \mathsf{F} \dot{\leftarrow} \mathsf{G}$,

(iii) $\mathsf{H}\varphi = \tau$ where $\tau : \mathsf{Id} \dot{\leftarrow} \mathsf{G}$,

(iv) If $\mathcal{C}$ has coproducts: $\mathsf{H}\varphi = [\mathsf{H}_1\varphi, \mathsf{H}_2\varphi]$, where $\mathsf{H}_1 : (\mathcal{Alg}_{\mathcal{C}}\mathsf{G}_1, |\,.\,|_{\mathsf{G}_1}) \leftarrow (\mathcal{Alg}_{\mathcal{C}}\mathsf{F}, |\,.\,|_{\mathsf{F}})$ and $\mathsf{H}_2 : (\mathcal{Alg}_{\mathcal{C}}\mathsf{G}_2, |\,.\,|_{\mathsf{G}_2}) \leftarrow (\mathcal{Alg}_{\mathcal{C}}\mathsf{F}, |\,.\,|_{\mathsf{F}})$ and $\mathsf{G} = \mathsf{G}_1 + \mathsf{G}_2$,

Proof. All cases are instances of Lemma 3.3.1 with $\mathsf{U}_{\mathsf{G}} = \mathsf{Id}$. ■

4 Syntax and semantics of top-down tree transducers

The concept of top-down tree transducers was introduced by [Rou68, Rou70] and [Tha70]. A top-down tree transducer can be regarded as a restricted functional program that computes a tree transformation, *i.e.* a function which maps trees onto trees.

4.1 Syntax of top-down tree transducers

4.1.1 Definition (variables). Let $X = \{x_1, x_2 \dots\}$ be a countable infinite set of variables. We will use this set X and for every $k \in \mathbb{N}_0$ the set $X_k = \{x_1, \dots, x_k\}$ ($X_0 = \emptyset$) throughout the paper. $*$

4.1.2 Definition (top-down tree transducer). A **top-down tree transducer** $T = (Q, \Sigma, \Delta, q_0, R)$ consists of a unary ranked alphabet Q of so called **states**, ranked alphabets Σ and Δ called the **input** and **output alphabet**, respectively, an element $q_0 \in Q$ called the **initial state**, and a relation $R \subseteq \bigcup_{k\in\mathbb{N}_0} Q(\Sigma X_k) \times T_\Delta(QX_k)$ such that

$$\forall\, q \in Q :: \ \forall\, k \in \mathbb{N}_0 :: \ \forall\, \sigma \in \Sigma^{(k)} :: \ \exists!\ \mathrm{rhs}_{R,\sigma}\, q \in T_\Delta(QX_k) :: \ \big(q(\sigma(x_1, \dots, x_k)), \mathrm{rhs}_{R,\sigma}\, q\big) \in R$$

and no other elements are in R. The elements of R are called **rules** and we will write them as $q(\sigma(x_1, \dots, x_k)) \to \mathrm{rhs}_{R,\sigma}\, q$ rather than $\big(q(\sigma(x_1, \dots, x_k)), \mathrm{rhs}_{R,\sigma}\, q\big)$. Notice that for every $\sigma \in \Sigma$ we have $\mathrm{rhs}_{R,\sigma} : T_\Delta(QX_{\mathrm{rank}_\Sigma\, \sigma}) \leftarrow Q$ is a function. We denote the class of all top-down tree transducers by $td\text{-}tree\mathrm{T}$ and the subclass of all top-down tree transducers with output alphabet Δ and input alphabet Σ by $td\text{-}tree\mathrm{T}(\Delta, \Sigma)$. $*$

4.1.3 Example (top-down tree transducer). We define the top-down tree transducer $T_{\text{zigzag}} = (Q, \Sigma, \Delta, zig, R)$ where $Q = \{zig, zag\}$; $\Sigma = \{\alpha^{(0)}, \sigma^{(2)}\}$; $\Delta = \{N^{(0)}, A^{(1)}, B^{(1)}\}$ and

$$R = \{ \begin{array}{lll} zig\,\alpha & \to & N, \\ zag\,\alpha & \to & N, \\ zig(\sigma(x_1, x_2)) & \to & A(zag\, x_1), \\ zag(\sigma(x_1, x_2)) & \to & B(zig\, x_2) \quad \}. \end{array}$$

$*$

4.2 Semantics of top-down tree transducers

4.2.1 Definition (computed tree transformation). Let $T = (Q, \Sigma, \Delta, q_0, R)$ be a top-down tree transducer. The **tree transformation**

$$\tau T : T_\Delta \leftarrow T_\Sigma$$

computed by T is defined by $\tau T = \tau_{q_0} T$ where

$$\forall\, q \in Q :: \ \forall\, k \in \mathbb{N}_0 :: \ \forall\, \sigma \in \Sigma^{(k)} :: \ \forall\, t_1, \dots, t_k \in T_\Sigma ::$$
$$\tau_q T(\sigma(t_1, \dots, t_k)) = [\tau_p T\, t_j / p x_j]_{\substack{p \in Q \\ x_j \in X_k}} (\mathrm{rhs}_{R,\sigma}\, q).$$

Since the number of symbols in the term $\sigma(t_1, \ldots, t_k)$ is finite and for every $p \in Q$ the function $\tau_p T$ on the right hand side is applied on the proper subexpressions $t_1, \ldots, t_k$ of $\sigma(t_1, \ldots, t_k)$, for every $q \in Q$ the function $\tau_q T$ is well defined. We call the function $\tau : T_\Delta^{T_\Sigma} \leftarrow \mathit{td}\text{-}\mathit{tree}\mathrm{T}(\Delta, \Sigma)$ the **semantics** of a top-down tree transducer. $*$

4.2.2 Definition (top-down tree transformation). We denote the image class of the semantics function τ by TOP_{tree}, *i.e.* $TOP_{tree} = \{\tau T \mid T \in \mathit{td}\text{-}\mathit{tree}\mathrm{T}\}$. The elements of TOP_{tree} are called **top-down tree transformations**. $*$

4.2.3 Example (computed tree transformation). The tree transformation computed by the top-down tree transducer T_{zigzag} from Example 4.1.3 is a function, which reads an input tree by traversing the σ's in a zig-zag-shape until an α is reached. It outputs a monadic tree of alternating A's and B's where the number of A's and B's together is the same as the number of traversed σ's, *e.g.* for arbitrary terms $t_1, t_2 \in T_\Sigma$:

$$\begin{aligned}
&\tau T_{\text{zigzag}}(\sigma(\sigma(t_1, \alpha), t_2)) \\
&= \tau_{zig} T_{\text{zigzag}}(\sigma(\sigma(t_1, \alpha), t_2)) \\
&= A(\tau_{zag} T_{\text{zigzag}}(\sigma(t_1, \alpha))) \\
&= A(B(\tau_{zig} T_{\text{zigzag}} \alpha)) \\
&= A(BN).
\end{aligned}$$

$*$

4.2.4 Definition (top-down tree transducer homomorphism). Let $T = (Q, \Sigma, \Delta, q_0, R)$ and $T' = (Q', \Sigma, \Delta, q_0', R')$ be top-down tree transducers. A function

(i) $h : Q \leftarrow Q'$ with

(ii) $q_0 = h q_0'$ and

(iii) $\forall\, \sigma \in \Sigma :: \ [hq'\, x / q' x]_{\substack{q' \in Q' \\ x \in X_{\mathrm{rank}_\Sigma \sigma}}} \cdot \mathrm{rhs}_{R', \sigma} = \mathrm{rhs}_{R, \sigma} \cdot h,$

is called a **top-down tree transducer homomorphism** and we write

$$h : T \leftarrow T'.$$

If h is bijective then we call it a **top-down tree transducer isomorphism**. If a top-down tree transducer isomorphism to T from T' exists then we call T and T' **isomorphic** and write $T \cong T'$. $*$

4.2.5 Lemma (homomorphisms preserve semantics of top-down tree transducers). Let $T, T' \in \mathit{td}\text{-}\mathit{tree}\mathrm{T}(\Delta, \Sigma)$ be top-down tree transducers.

$$\frac{\exists\, h : T \leftarrow T'}{\tau T = \tau T'}$$

Proof. Let $T = (Q, \Sigma, \Delta, q_0, R)$ and $T' = (Q', \Sigma, \Delta, q_0', R')$ and $h : T \leftarrow T'$. We will show:

$$\forall\, t \in T_\Sigma :: \ \forall\, q' \in Q' :: \ \tau_{hq'} T\, t = \tau_{q'} T'\, t$$

by induction on t. Let $k \in \mathbb{N}_0$, $\sigma \in \Sigma^{(k)}$, and $t_1, \ldots, t_k \in T_\Sigma$ and assume that

$$\forall\, j \in \{1, \ldots, k\} :: \ \forall\, p \in Q' :: \ \tau_{hp} T t_j = \tau_p T' t_j.$$

Then we calculate:

$$\begin{aligned}
&\tau_{hq'}T(\sigma(t_1,\dots,t_k))\\
=\;&[\tau_p T t_j/px_j]_{\substack{p\in Q\\ x_j\in X_k}}(\mathrm{rhs}_{R,\sigma}(hq'))\\
=\;&[\tau_p T t_j/px_j]_{\substack{p\in Q\\ x_j\in X_k}}\big([hp\,x/px]_{\substack{p\in Q'\\ x\in X_k}}(\mathrm{rhs}_{R',\sigma}\,q')\big)\\
=\;&[\tau_{hp} T t_j/px_j]_{\substack{p\in Q'\\ x_j\in X_k}}(\mathrm{rhs}_{R',\sigma}\,q')\\
=\;&[\tau_p T' t_j/px_j]_{\substack{p\in Q'\\ x_j\in X_k}}(\mathrm{rhs}_{R',\sigma}\,q')\\
=\;&\tau_{q'}T(\sigma(t_1,\dots,t_k)).
\end{aligned}$$

And thus: $\tau T = \tau_{q_0}T = \tau_{hq'_0}T = \tau_{q'_0}T' = \tau T'$. ■

4.3 Syntactic composition of top-down tree transducers

4.3.1 Definition (syntactic composition of top-down tree transducers). (*cf.* Theorem 2 of [Rou70] and p. 195 of [Bak79]) Let $T_1 = (P,\Sigma,\Delta,p_0,R_1)$ and $T_2 = (Q,\Delta,\Gamma,q_0,R_2)$ be top-down tree transducers. We modify the top-down tree transducer T_2 so that it can operate on the right hand sides of rules of T_1:

$$\begin{aligned}
T'_2 &= \big(Q, \Delta \uplus \{(px)^{(0)}\}_{\substack{p\in P\\ x\in X_r}}, \Gamma \uplus \{((q,p)x)^{(0)}\}_{\substack{q\in Q\\ p\in P\\ x\in X_r}}, q_0, R'_2\big) \quad \text{where}\\
r &= \max_{\sigma\in\Sigma}(\mathrm{rank}_\Sigma\,\sigma) \quad \text{and}\\
R'_2 &= R_2 \uplus \big\{q(px) \to (q,p)x\big\}_{\substack{q\in Q\\ p\in P\\ x\in X_r}}.
\end{aligned}$$

The **syntactic composition** $T_2 \cdot T_1$ of T_2 and T_1 is the top-down tree transducer defined by

$$\begin{aligned}
T_2\cdot T_1 &= \big(Q\times P,\Sigma,\Gamma,(q_0,p_0),R\big) \quad \text{where}\\
R &= \big\{(q,p)(\sigma(x_1,\dots,x_{\mathrm{rank}_\Sigma\,\sigma})) \to \tau_q T'_2(\mathrm{rhs}_{R_1,\sigma}\,p)\big\}_{\substack{q\in Q\\ p\in P\\ \sigma\in\Sigma}}.
\end{aligned}$$

Notice that the expressions px and $(q,p)x$ are viewed from two different perspectives: for T'_2 they are symbols of rank 0. For T_1 and $T_2\cdot T_1$ they are composite terms built out of unary symbols (p or (p,q)) and a variable x. $*$

4.3.2 Theorem (syntactic composition preserves semantics). Let Σ, Δ, and Γ be ranked alphabets. Then

$$\frac{T_2 \in \textit{td-tree}\mathrm{T}(\Gamma,\Delta) \qquad T_1 \in \textit{td-tree}\mathrm{T}(\Delta,\Sigma)}{\tau T_2\cdot\tau T_1 = \tau(T_2\cdot T_1)}$$

Proof. See Theorem 2 of [Rou70] and Theorem 3.39 of [FV98]. ■

4.3.3 Example (syntactic composition). Consider the top-down tree transducer T_{zigzag} from Example 4.1.3 and the top-down tree transducer $T_{\text{bin}} = (Q',\Delta,\Sigma,\textit{bin},R')$ where $Q' = \{\textit{bin}\}$, Σ and Δ are defined as in Example 4.1.3, and

$$\begin{aligned}
R' = \{\;& \textit{bin}\,N && \to \alpha,\\
& \textit{bin}(A\,x_1) && \to \sigma(\textit{bin}\,x_1,\textit{bin}\,x_1),\\
& \textit{bin}(B\,x_1) && \to \sigma(\textit{bin}\,x_1,\textit{bin}\,x_1) \quad \}.
\end{aligned}$$

The tree transformation computed by T_{bin} constructs the full binary tree the height of which is equal to the height of the input tree, *e.g.* $\tau T_{\text{bin}}\big(A(BN)\big) = \sigma(\sigma(\alpha,\alpha),\sigma(\alpha,\alpha))$. We may construct the following syntactic composition

$$T_{\text{bin}}\cdot T_{\text{zigzag}} = \big(Q'\times Q,\Sigma,\Sigma,(\textit{bin},\textit{zig}),R_1\big)$$

where

$$\begin{array}{rlcl} R_1 = \{ & (bin, zig)\alpha & \to & \alpha, \\ & (bin, zag)\alpha & \to & \alpha, \\ & (bin, zig)(\sigma(x_1, x_2)) & \to & \sigma((bin, zag)x_1, (bin, zag)x_1), \\ & (bin, zag)(\sigma(x_1, x_2)) & \to & \sigma((bin, zig)x_2, (bin, zig)x_2) \quad \}. \end{array}$$

Let $t_1, t_2 \in T_\Sigma$ be arbitrary terms and $t = \sigma(\sigma(t_1, \alpha), t_2)$. The tree transformation computed by the top-down tree transducer $T_{\text{bin}} \cdot T_{\text{zigzag}}$ constructs a full binary tree, where the height is determined by the length of the 'zig-zag-path' of its argument tree (*cf.* Example 4.2.3), thus we have

$$\tau(T_{\text{bin}} \cdot T_{\text{zigzag}})t = \sigma(\sigma(\alpha, \alpha), \sigma(\alpha, \alpha)).$$

In Example 4.2.3 we saw that $\tau T_{\text{zigzag}} t = A(B(N))$. Together with the example for T_{bin} from above we get

$$(\tau T_{\text{bin}} \cdot \tau T_{\text{zigzag}})t = \tau T_{\text{bin}}(\tau T_{\text{zigzag}} t) = \sigma(\sigma(\alpha, \alpha), \sigma(\alpha, \alpha)).$$

We can also compose the transducers T_{zigzag} and T_{bin} in the other order:

$$T_{\text{zigzag}} \cdot T_{\text{bin}} = \big(Q \times Q', \Delta, \Delta, (zig, bin), R_2\big)$$

where

$$\begin{array}{rlcl} R_2 = \{ & (zig, bin)(N) & \to & N, \\ & (zag, bin)(N) & \to & N, \\ & (zig, bin)(A\,x_1) & \to & A((zag, bin)x_1), \\ & (zag, bin)(A\,x_1) & \to & A((zig, bin)x_1), \\ & (zig, bin)(B\,x_1) & \to & B((zag, bin)x_1), \\ & (zag, bin)(B\,x_1) & \to & B((zig, bin)x_1) \quad \}. \end{array}$$

$*$

4.3.4 Lemma. For every ranked alphabet Σ:

$$id_{T_\Sigma} \in TOP_{tree}.$$

Proof. It is easy to see that the tree transformation computed by the top-down tree transducer

$$\begin{array}{l} T_{id} = (\{q^{(1)}\}, \Sigma, \Sigma, q, R) \quad \text{where} \\ \;R = \{q(\sigma(x_1, \dots, x_{\text{rank}_\Sigma\,\sigma})) \to \sigma(qx_1, \dots, qx_{\text{rank}_\Sigma\,\sigma})\}_{\sigma \in \Sigma} \end{array}$$

is the identity function, *i.e.* $\tau T_{id} = id_{T_\Sigma}$. ■

4.3.5 Corollary. From Theorem 4.3.2 and Lemma 4.3.4 we obtain

$$TOP_{tree} \cdot TOP_{tree} = TOP_{tree}$$

$*$

5 Syntax and semantics of categorical transducers

In the previous section we have seen top-down tree transducers and the syntactic composition of these. We want to compare this syntactic composition with short cut fusion formalized in category theory from Definition and Corollary 3.2.6. In order to do so we will need a category theory model of a top-down tree transducer. This will be the top-down *categorical transducer*.

5.1 Syntax of categorical transducers

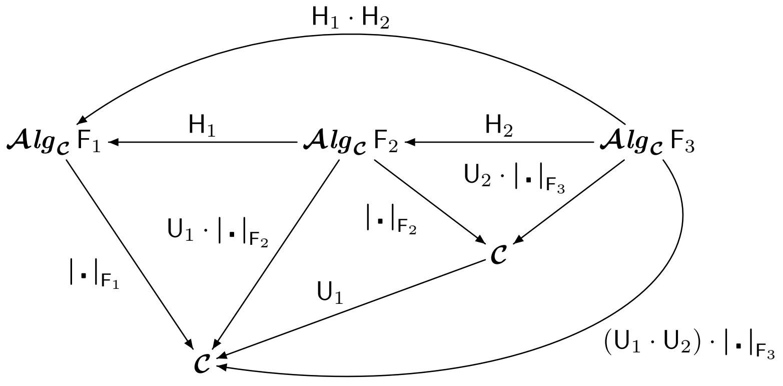
5.1.1 Definition (categorical transducer). Let $\mathcal{C}$ be a category. We define the category $\boldsymbol{cat\mathcal{T}}_{\mathcal{C}}$ by

$$\begin{aligned}
\mathrm{Ob}(\boldsymbol{cat\mathcal{T}}_{\mathcal{C}}) &= \big\{\mathsf{F} \;\big|\; \mathsf{F} : \mathcal{C} \leftarrow \mathcal{C} \text{ such that } \boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\mathsf{F} \text{ has an initial object}\big\},\\
\boldsymbol{cat\mathcal{T}}_{\mathcal{C}}(\mathsf{G},\mathsf{F}) &= \big\{(\mathsf{H},\mathsf{U},\pi) \;\big|\; \mathsf{H} : (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\mathsf{F}, |\centerdot|_{\mathsf{F}}) \leftarrow (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\mathsf{G}, \mathsf{U}\cdot|\centerdot|_{\mathsf{G}})\\
&\qquad\qquad \wedge\; \mathsf{U} : \mathcal{C} \leftarrow \mathcal{C} \text{ faithful}\\
&\qquad\qquad \wedge\; \pi : \mathsf{Id} \dot{\leftarrow} \mathsf{U} \quad \big\}
\end{aligned}$$

where

$$\begin{aligned}
id_{\mathsf{F}} &= (\mathsf{Id}_{(\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\mathsf{F},|\centerdot|_{\mathsf{F}})^{(\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\mathsf{F},|\centerdot|_{\mathsf{F}})}}, \mathsf{Id}_{\mathcal{C}}, id),\\
(\mathsf{H}_2,\mathsf{U}_2,\pi_2)\cdot(\mathsf{H}_1,\mathsf{U}_1,\pi_1) &= (\mathsf{H}_1\cdot\mathsf{H}_2, \mathsf{U}_1\cdot\mathsf{U}_2, \pi_1 * \pi_2)
\end{aligned}$$

and $\pi_1 * \pi_2 = \pi_1 \cdot \mathsf{U}_1\pi_2 = \pi_2 \cdot \pi_1\mathsf{U}_2$ is the vertical composition of the natural transformations π_1 and π_2 from Definition and Lemma C.2.7. This is a category, because the composition is obviously associative in the first and second component and also in the third component, because the vertical composition of natural transformations is associative (Lemma C.2.9). A morphism of the category $\boldsymbol{cat\mathcal{T}}_{\mathcal{C}}$ is called a **categorical transducer** over $\mathcal{C}$. Notice that $\mathsf{H}_1\cdot\mathsf{H}_2 : (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\mathsf{F}_1, |\centerdot|_{\mathsf{F}_1}) \leftarrow (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\mathsf{F}_3, \mathsf{U}_1\cdot\mathsf{U}_2\cdot|\centerdot|_{\mathsf{F}_3})$ as follows from the diagram:



$$\begin{tikzcd}
\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\mathsf{F}_1 \arrow[ddr, "|\centerdot|_{\mathsf{F}_1}"'] & & \boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\mathsf{F}_2 \arrow[ll, "\mathsf{H}_1"'] \arrow[ddl, "\mathsf{U}_1\cdot|\centerdot|_{\mathsf{F}_2}"'] \arrow[dr, "|\centerdot|_{\mathsf{F}_2}"] & & \boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\mathsf{F}_3 \arrow[ll, "\mathsf{H}_2"'] \arrow[dl, "\mathsf{U}_2\cdot|\centerdot|_{\mathsf{F}_3}"'] \arrow[llll, bend right, "\mathsf{H}_1\cdot\mathsf{H}_2"'] \arrow[ddlll, bend left=60, "(\mathsf{U}_1\cdot\mathsf{U}_2)\cdot|\centerdot|_{\mathsf{F}_3}"] \\
& & & \mathcal{C} \arrow[dll, "\mathsf{U}_1"'] & \\
& \mathcal{C} & & &
\end{tikzcd}$$

$*$

5.2 Semantics of categorical transducers

5.2.1 Definition (semantics of categorical transducers). Let $\mathcal{C}$ be a category. For every $(\mathsf{H},\mathsf{U},\pi) \in \boldsymbol{cat\mathcal{T}}_{\mathcal{C}}(\mathsf{G},\mathsf{F})$ we define

$$\mathsf{S}(\mathsf{H},\mathsf{U},\pi) = \pi \cdot (\![\mathsf{H}in_{\mathsf{G}}]\!)_{\mathsf{F}} : \mu\mathsf{G} \leftarrow \mu\mathsf{F}$$

where in the right hand side expression the functor U is hidden in the codomain of the concrete functor $\mathsf{H} : (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\mathsf{F}, |\centerdot|_{\mathsf{F}}) \leftarrow (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\mathsf{G}, \mathsf{U}\cdot|\centerdot|_{\mathsf{G}})$. We call $\mathsf{S} : \mathrm{Mor}\,\mathcal{C} \leftarrow \mathrm{Mor}\,\boldsymbol{cat\mathcal{T}}_{\mathcal{C}}$ the **semantics** of categorical transducers. Notice that S depends on the choice of the initial algebras of the functors in $\mathrm{Ob}\,\boldsymbol{cat\mathcal{T}}_{\mathcal{C}}$. In Definition 5.1.1 we demanded only the existence of initial algebras. From Lemma C.3.3 we know that initial algebras are uniquely determined up to algebra isomorphism, thus we may choose an initial algebra out of every isomorphism class. $*$

5.2.2 Example (categorical transducer). Let $\mathcal{C}$ be a category which has finite products and coproducts. Let $\mathsf{F} = \mathsf{K}_1 + \mathsf{Id}\times\mathsf{Id} : \mathcal{C} \leftarrow \mathcal{C}$ and $\mathsf{G} = \mathsf{K}_1 + \mathsf{Id} + \mathsf{Id} : \mathcal{C} \leftarrow \mathcal{C}$ be functors which have initial algebras, where $in_{\mathsf{G}} = [N, A, B] : \mu\mathsf{G} \leftarrow \mathsf{G}(\mu\mathsf{G})$. For every $C \in \mathrm{Ob}\,\mathcal{C}$ and every $[\varphi_1,\varphi_2,\varphi_3] : C \leftarrow \mathsf{G}C$ we define

$$\mathsf{H}[\varphi_1,\varphi_2,\varphi_3] = [\langle\varphi_1,\varphi_1\rangle, \langle\varphi_2\cdot\pi_2\cdot\pi_1, \varphi_3\cdot\pi_1\cdot\pi_2\rangle].$$

which obviously satisfies $\mathsf{H}[\varphi_1, \varphi_2, \varphi_3] : C \times C \leftarrow \mathsf{F}(C \times C)$. We use Lemma 3.3.1 to prove that H can be uniquely extended to a concrete functor

$$\mathsf{H} : (\mathcal{Alg}_{\mathcal{C}}\mathsf{F}, |\,.\,|_{\mathsf{F}}) \leftarrow (\mathcal{Alg}_{\mathcal{C}}\mathsf{G}, \textstyle\prod^2 \cdot |\,.\,|_{\mathsf{G}})$$

More precisely we show that for every $\varphi = [\varphi_1, \varphi_2, \varphi_3], \varphi' = [\varphi_1', \varphi_2', \varphi_3'] \in \mathrm{Ob}(\mathcal{Alg}_{\mathcal{C}}\mathsf{G})$ and every $f : |\varphi|_{\mathsf{G}} \xleftarrow[\mathcal{C}]{} |\varphi'|_{\mathsf{G}}$ the condition

$$\frac{\varphi \cdot \mathsf{G}f = f \cdot \varphi'}{\mathsf{H}\varphi \cdot \mathsf{F}(f \times f) = (f \times f) \cdot \mathsf{H}\varphi'} \qquad (*)$$

is satisfied: With the definition of G the precondition of $(*)$ can be restated as $\varphi \cdot (id_1 + f + f) = f \cdot \varphi'$, *i.e.*

$$\varphi_1 = f\varphi_1' \qquad \varphi_2 \cdot f = f \cdot \varphi_2' \qquad \varphi_3 \cdot f = f \cdot \varphi_3'.$$

Using this we calculate

$$\langle \varphi_1, \varphi_1 \rangle = \langle f \cdot \varphi_1', f \cdot \varphi_1' \rangle = (f \times f) \cdot \langle \varphi_1', \varphi_1' \rangle$$

where we used the fusion (i) law (Table 8). Moreover we calculate:

$$\begin{array}{cl} & \langle \varphi_2 \cdot \pi_2 \cdot \pi_1, \varphi_3 \cdot \pi_1 \cdot \pi_2 \rangle \cdot ((f \times f) \times (f \times f)) \\ = & \quad \{ \text{ fusion (Table 6) and four times cancelation (Table 8) } \} \\ & \langle \varphi_2 \cdot f \cdot \pi_2 \cdot \pi_1, \varphi_3 \cdot f \cdot \pi_1 \cdot \pi_2 \rangle \\ = & \quad \{ \text{ precondition of } (*) \ \} \\ & \langle f \cdot \varphi_2' \cdot \pi_2 \cdot \pi_1, f \cdot \varphi_3' \cdot \pi_1 \cdot \pi_2 \rangle \\ = & \quad \{ \text{ fusion (i) (Table 8) } \} \\ & (f \times f) \cdot \langle \varphi_2' \cdot \pi_2 \cdot \pi_1, \varphi_3' \cdot \pi_1 \cdot \pi_2 \rangle. \end{array}$$

Using the fusion (i) law (Table 9) we get

$$[\langle \varphi_1, \varphi_1 \rangle, \langle \varphi_2 \cdot \pi_2 \cdot \pi_1, \varphi_3 \cdot \pi_1 \cdot \pi_2 \rangle] \cdot (id_1 + ((f \times f) \times (f \times f))) = (f \times f) \cdot [\langle \varphi_1', \varphi_1' \rangle, \langle \varphi_2' \cdot \pi_2 \cdot \pi_1, \varphi_3' \cdot \pi_1 \cdot \pi_2 \rangle]$$

which is nothing else than the conclusion of $(*)$. Thus we have

$$\mathsf{H} : (\mathcal{Alg}_{\mathcal{C}}\mathsf{F}, |\,.\,|_{\mathsf{F}}) \leftarrow (\mathcal{Alg}_{\mathcal{C}}\mathsf{G}, \textstyle\prod^2 \cdot |\,.\,|_{\mathsf{G}})$$

by means of Lemma 3.3.1. Obviously we have a natural transformation $\pi_1 : \mathsf{Id} \mathrel{\dot\leftarrow} \mathsf{Id} \times \mathsf{Id}$. Finally we obtain that

$$(\mathsf{H}, \mathsf{Id} \times \mathsf{Id}, \pi_1) : \mathsf{G} \leftarrow \mathsf{F}$$

is a categorical transducer over $\mathcal{C}$. $*$

5.2.3 Theorem (semantics functor). Let $\mathcal{C}$ be a category. The semantics of categorical transducers over $\mathcal{C}$ is a functor

$$\mathsf{S} : \mathcal{C} \leftarrow \boldsymbol{cat\mathcal{T}_{\mathcal{C}}}$$

and thus in particular (*cf.* Definition and Corollary 3.2.6 and Theorem 4.3.2):

$$\frac{\mathsf{F}_3 \xleftarrow{(\mathsf{H}_2, \mathsf{U}_2, \pi_2)} \mathsf{F}_2 \xleftarrow{(\mathsf{H}_1, \mathsf{U}_1, \pi_1)} \mathsf{F}_1}{\mathsf{S}(\mathsf{H}_2, \mathsf{U}_2, \pi_2) \cdot \mathsf{S}(\mathsf{H}_1, \mathsf{U}_1, \pi_1) = \mathsf{S}(\mathsf{H}_1 \cdot \mathsf{H}_2, \mathsf{U}_1 \cdot \mathsf{U}_2, \pi_1 * \pi_2)}.$$

Proof. S satisfies the typing axiom by construction where $\forall\, \mathsf{F} \in \mathrm{Ob}\, \boldsymbol{cat\mathcal{T}_{\mathcal{C}}} :: \mathsf{SF} = \mu\mathsf{F}$. In order to prove the other functor axioms, we use the functor M from Definition and Corollary 3.2.6 to express the function S: For every $\mathsf{H} \in \boldsymbol{cCAT}\ \boldsymbol{\mathcal{C}}\big((\boldsymbol{\mathcal{A}lg_{\mathcal{C}}}\mathsf{F}, |\centerdot|_{\mathsf{F}}), (\boldsymbol{\mathcal{A}lg_{\mathcal{C}}}\mathsf{G}, \mathsf{U}\cdot|\centerdot|_{\mathsf{G}})\big)$, every faithful $\mathsf{U} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$ and every $\pi : \mathsf{Id} \dot{\leftarrow} \mathsf{U}$ we know that $(|\mathsf{H}|, \mathsf{U}, \pi) \in \boldsymbol{cat\mathcal{T}_{\mathcal{C}}}(\mathsf{G}, \mathsf{F})$, and we obtain

$$\mathsf{S}(|\mathsf{H}|, \mathsf{U}, \pi) = \pi \cdot (\![|\mathsf{H}|in_{\mathsf{G}}]\!)_{\mathsf{F}} = \pi \cdot \mathsf{M\,H}.$$

Thus $\mathsf{S}(\mathsf{Id}, \mathsf{Id}, id) = id \cdot \mathsf{M\,Id} = id$ and hence S satisfies the identity axiom. Finally we show the multiplicativity axiom: For

$$\mathsf{F}_3 \xleftarrow{(|\mathsf{H}_2|, \mathsf{U}_2, \pi_2)} \mathsf{F}_2 \xleftarrow{(|\mathsf{H}_1|, \mathsf{U}_1, \pi_1)} \mathsf{F}_1$$

we have (Definition 5.1.1 and Note 2.4.8)

$$\begin{aligned}\mathsf{H}_1 &= \big((\boldsymbol{\mathcal{A}lg_{\mathcal{C}}}\mathsf{F}_1, |\centerdot|_{\mathsf{F}_1}), |\mathsf{H}_1|, (\boldsymbol{\mathcal{A}lg_{\mathcal{C}}}\mathsf{F}_2, \mathsf{U}_1\cdot|\centerdot|_{\mathsf{F}_2})\big),\\ \mathsf{H}_2 &= \big((\boldsymbol{\mathcal{A}lg_{\mathcal{C}}}\mathsf{F}_2, |\centerdot|_{\mathsf{F}_2}), |\mathsf{H}_2|, (\boldsymbol{\mathcal{A}lg_{\mathcal{C}}}\mathsf{F}_3, \mathsf{U}_2\cdot|\centerdot|_{\mathsf{F}_3})\big)\end{aligned}$$

and calculate

$$\begin{aligned}
& \mathsf{S}\big((|\mathsf{H}_2|, \mathsf{U}_2, \pi_2)\cdot(|\mathsf{H}_1|, \mathsf{U}_1, \pi_1)\big)\\
= \quad & \{ \text{ Definition 5.1.1 } \}\\
& \mathsf{S}(|\mathsf{H}_1|\cdot|\mathsf{H}_2|, \mathsf{U}_1\cdot\mathsf{U}_2, \pi_1 * \pi_2)\\
= \quad & \{ \text{ Definition and Lemma 2.4.9: } |_{\mathsf{U}_1}\mathsf{H}_2| = |\mathsf{H}_2| \text{ and } |\centerdot| \text{ is a functor } \}\\
& \mathsf{S}(|\mathsf{H}_1\cdot{}_{\mathsf{U}_1}\mathsf{H}_2|, \mathsf{U}_1\cdot\mathsf{U}_2, \pi_1 * \pi_2)\\
= \quad & \{ \text{ see above } \}\\
& (\pi_1 * \pi_2)\cdot\mathsf{M}(\mathsf{H}_1\cdot{}_{\mathsf{U}_1}\mathsf{H}_2)\\
= \quad & \{ \ \mathsf{M} \text{ is a contravariant functor (short cut fusion) } \}\\
& (\pi_1 * \pi_2)\cdot\mathsf{M}({}_{\mathsf{U}_1}\mathsf{H}_2)\cdot\mathsf{M\,H}_1\\
= \quad & \{ \text{ Definition and Lemma C.2.7 of } * \text{ and Lemma 3.2.7 } \}\\
& \pi_1\cdot\mathsf{U}_1\pi_2\cdot\mathsf{U}_1(\mathsf{M\,H}_2)\cdot\mathsf{M\,H}_1\\
= \quad & \{ \ \mathsf{U}_1 \text{ is a functor } \}\\
& \pi_1\cdot\mathsf{U}_1(\pi_2\cdot\mathsf{M\,H}_2)\cdot\mathsf{M\,H}_1\\
= \quad & \{ \ \pi_1 : \mathsf{Id} \dot{\leftarrow} \mathsf{U}_1 \ \}\\
& \pi_2\cdot\mathsf{M\,H}_2 \ \cdot\ \pi_1\cdot\mathsf{M\,H}_1\\
= \quad & \{ \text{ see above } \}\\
& \mathsf{S}(|\mathsf{H}_2|, \mathsf{U}_2, \pi_2) \ \cdot\ \mathsf{S}(|\mathsf{H}_1|, \mathsf{U}_1, \pi_1).
\end{aligned}$$

■

5.3 Categorical transducer homomorphisms

In this subsection we lift the notion of homomorphisms between tree automata (*cf.* Definition 6.1 of [GS84]) to the abstract level of categorical transducers. Then we can prove that homomorphic categorical transducers have equal semantics and that isomorphism is a congruence w.r.t. to fusion, *i.e.* the composition of categorical transducers.

5.3.1 Definition (categorical transducer homomorphism). (*cf.* Definition 4.2.4) Let $\boldsymbol{\mathcal{C}}$ be a category and $C = (\mathsf{H}, \mathsf{U}, \pi) : \mathsf{G} \leftarrow \mathsf{F}$ and $C' = (\mathsf{H}', \mathsf{U}', \pi') : \mathsf{G} \leftarrow \mathsf{F}$ categorical transducers over $\boldsymbol{\mathcal{C}}$. A natural transformation

(i) $\eta : \mathsf{U} \dot{\leftarrow} \mathsf{U}'$ such that

(ii) $\exists\, \tilde{\eta} : \mathsf{H} \dot{\leftarrow} \mathsf{H}'$ with $|\tilde{\eta}|_{\mathsf{F}} = \eta_{|\,\centerdot\,|_{\mathsf{G}}}$ and

(iii) $\pi \cdot \eta = \pi'$

is called a **categorical transducer homomorphism** to C from C' and we write

$$\eta : C \leftarrow C'.$$

If η is a natural isomorphism, then we call it a **categorical transducer isomorphism**. If a categorical transducer isomorphism to C from C' exists, then we call C and C' **isomorphic** and write $C \cong C'$. $*$

The condition (ii) from Definition 5.3.1 seems to be peculiar. The following Lemma 5.3.2 shows an equivalent statement which is closer to Definition 4.2.4 (ii):

5.3.2 Lemma. Let $\mathcal{C}$ be a category, $\mathsf{F}, \mathsf{G}, \mathsf{U} : \mathcal{C} \leftarrow \mathcal{C}$ endofunctors where U is faithful, and let $\mathsf{H}, \mathsf{H}' : (\mathcal{A}lg_{\mathcal{C}}\mathsf{F}, |\,\centerdot\,|_{\mathsf{F}}) \leftarrow (\mathcal{A}lg_{\mathcal{C}}\mathsf{G}, \mathsf{U} \cdot |\,\centerdot\,|_{\mathsf{G}})$ be concrete functors.

(i) If there exists a natural transformation $\tilde{\eta} : \mathsf{H} \dot{\leftarrow} \mathsf{H}'$ with $|\tilde{\eta}|_{\mathsf{F}} = \eta_{|\,\centerdot\,|_{\mathsf{G}}}$ then it is uniquely determined by $\forall\, \varphi \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\mathsf{G}) :: \ \tilde{\eta}_\varphi = (\mathsf{H}\varphi, \eta_{|\varphi|_{\mathsf{G}}}, \mathsf{H}'\varphi)$ and

(ii) for $\tilde{\eta}$ given by $\forall\, \varphi \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\mathsf{G}) :: \ \tilde{\eta}_\varphi = (\mathsf{H}\varphi, \eta_{|\varphi|_{\mathsf{G}}}, \mathsf{H}'\varphi)$, we have that

$$\tilde{\eta} : \mathsf{H} \dot{\leftarrow} \mathsf{H}' \quad \Longleftrightarrow \quad \forall\, \varphi \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\mathsf{G}) :: \ \eta \cdot \mathsf{H}'\varphi = \mathsf{H}\varphi \cdot \mathsf{F}\eta.$$

Proof. Let $\varphi, \psi \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\mathsf{G})$.

(i) From the definition of $|\,\centerdot\,|_{\mathsf{F}}$ on morphisms and the preconditions we obtain $|(\mathsf{H}\varphi, \eta_{|\varphi|_{\mathsf{G}}}, \mathsf{H}'\varphi)|_{\mathsf{F}} = \eta_{|\varphi|_{\mathsf{G}}} = |\tilde{\eta}_\varphi|_{\mathsf{F}}$. Since $|\,\centerdot\,|_{\mathsf{F}}$ is faithful, its restriction to $\mathcal{A}lg_{\mathcal{C}}\mathsf{F}(\mathsf{H}\varphi, \mathsf{H}'\varphi)$ is injective and we have $(\mathsf{H}\varphi, \eta_{|\varphi|_{\mathsf{G}}}, \mathsf{H}'\varphi) = \tilde{\eta}_\varphi$.

(ii) '$\Rightarrow$':

$$\begin{aligned}
& \tilde{\eta} : \mathsf{H} \dot{\leftarrow} \mathsf{H}' \\
\Longrightarrow \quad & \{\ \text{Definition C.2.1}\ \} \\
& \tilde{\eta}_\varphi : \mathsf{H}\varphi \xleftarrow[\mathcal{A}lg_{\mathcal{C}}\mathsf{F}]{} \mathsf{H}'\varphi \\
\Longleftrightarrow \quad & \{\ \text{Definition and Lemma C.6.1}\ \} \\
& |\tilde{\eta}_\varphi|_{\mathsf{F}} \cdot \mathsf{H}'\varphi = \mathsf{H}\varphi \cdot \mathsf{F}|\tilde{\eta}_\varphi|_{\mathsf{F}} \\
\Longleftrightarrow \quad & \{\ \text{Definition C.2.4 and definition of } \tilde{\eta}\ \} \\
& \eta \cdot \mathsf{H}'\varphi = \mathsf{H}\varphi \cdot \mathsf{F}\eta
\end{aligned}$$

'$\Leftarrow$': The last two steps in the above derivation are in fact equivalences, thus we have a transformation $\tilde{\eta} : \mathsf{H} \leftarrow \mathsf{H}'$ and have to show that it is natural:

$$\begin{aligned}
& \eta : \mathsf{U} \dot{\leftarrow} \mathsf{U}' \\
\Longrightarrow \quad & \{\ \text{naturalness of } \eta\ \} \\
& \mathsf{U}|f|_{\mathsf{G}} \cdot \eta_{|\psi|_{\mathsf{G}}} = \eta_{|\varphi|_{\mathsf{G}}} \cdot \mathsf{U}'|f|_{\mathsf{G}} \\
\Longrightarrow \quad & \{\ \text{concreteness of } \mathsf{H} \text{ and } \mathsf{H}' \text{ and definition of } \tilde{\eta}\ \} \\
& |\mathsf{H}f|_{\mathsf{F}} \cdot |\tilde{\eta}_\psi|_{\mathsf{F}} = |\tilde{\eta}_\varphi|_{\mathsf{F}} \cdot |\mathsf{H}'f|_{\mathsf{F}} \\
\Longrightarrow \quad & \{\ |\,\centerdot\,|_{\mathsf{F}} \text{ is faithful}\ \} \\
& \mathsf{H}f \cdot \tilde{\eta}_\psi = \tilde{\eta}_\varphi \cdot \mathsf{H}'f \\
\Longrightarrow \quad & \{\ \text{the latter is true for every } \varphi \text{ and } \psi\ \} \\
& \tilde{\eta} : \mathsf{H} \dot{\leftarrow} \mathsf{H}'
\end{aligned}$$

■

5.3.3 Lemma. Let $\mathcal{C}$ be a category and $(\mathsf{H},\mathsf{U},\pi),(\mathsf{H},\mathsf{U},\pi) : \mathsf{G} \leftarrow \mathsf{F}$ be categorical transducers over $\mathcal{C}$.

$$\frac{(\mathsf{H},\mathsf{U},\pi) \xleftarrow{\eta} (\mathsf{H}',\mathsf{U}',\pi')}{\forall\, \varphi \in \mathrm{Ob}(\mathcal{Alg}_{\mathcal{C}}\mathsf{G}) :: \ (\![\mathsf{H}\varphi]\!)_{\mathsf{F}} = \eta \cdot (\![\mathsf{H}'\varphi]\!)_{\mathsf{F}}}.$$

Proof.

$$\begin{array}{ll} & (\![\mathsf{H}\varphi]\!)_{\mathsf{F}} \\ = & \{ \text{ fusion Table 10 with } |\tilde{\eta}_\varphi|_{\mathsf{F}} \cdot \mathsf{H}'\varphi = \mathsf{H}\varphi \cdot \mathsf{F}|\tilde{\eta}_\varphi|_{\mathsf{F}} \ \} \\ & |\tilde{\eta}_\varphi|_{\mathsf{F}} \cdot (\![\mathsf{H}'\varphi]\!)_{\mathsf{F}} \\ = & \{ \text{ Definition 5.3.1 (ii) } \} \\ & \eta_{|\varphi|_{\mathsf{G}}} \cdot (\![\mathsf{H}'\varphi]\!)_{\mathsf{F}} \\ = & \{ \text{ Definition C.2.4 } \} \\ & \eta \cdot (\![\mathsf{H}'\varphi]\!)_{\mathsf{F}}. \end{array}$$

■

5.3.4 Theorem (homomorphisms preserve semantics of categorical transducers). (*cf.* Lemma 4.2.5) Let $\mathcal{C}$ be a category and $C, C' : \mathsf{G} \leftarrow \mathsf{F}$ categorical transducers over $\mathcal{C}$.

$$\frac{\exists\, \eta : C \leftarrow C'}{\mathsf{S}C = \mathsf{S}C'}$$

Proof. Let $C = (\mathsf{H},\mathsf{U},\pi)$ and $C' = (\mathsf{H}',\mathsf{U}',\pi')$ and $\eta : C \leftarrow C'$ be a categorical transducer homomorphism. We calculate:

$$\begin{array}{ll} & \mathsf{S}C \\ = & \{ \text{ Definition 5.2.1 } \} \\ & \pi \cdot (\![\mathsf{H}\mathit{in}_{\mathsf{G}}]\!)_{\mathsf{F}} \\ = & \{ \text{ Theorem 5.3.3 } \} \\ & \pi \cdot \eta \cdot (\![\mathsf{H}'\mathit{in}_{\mathsf{G}}]\!)_{\mathsf{F}} \\ = & \{ \text{ Definition 5.3.1 } \} \\ & \pi' \cdot (\![\mathsf{H}'\mathit{in}_{\mathsf{G}}]\!)_{\mathsf{F}} \\ = & \{ \text{ Definition 5.2.1 } \} \\ & \mathsf{S}C'. \end{array}$$

■

5.3.5 Theorem (vertical composition of categorical transducer homomorphisms). Let $\mathcal{C}$ be a category and

$$\mathsf{F}_3 \xleftarrow{C_2 = (\mathsf{H}_2,\mathsf{U}_2,\pi_2),\ C_2' = (\mathsf{H}_2',\mathsf{U}_2',\pi_2')} \mathsf{F}_2 \xleftarrow{C_1 = (\mathsf{H}_1,\mathsf{U}_1,\pi_1),\ C_1' = (\mathsf{H}_1',\mathsf{U}_1',\pi_1')} \mathsf{F}_1$$

be categorical transducers over $\mathcal{C}$. Then

$$\frac{C_2 \xleftarrow{\eta_2} C_2' \qquad C_1 \xleftarrow{\eta_1} C_1'}{C_2 \cdot C_1 \xleftarrow{\eta_1 * \eta_2} C_2' \cdot C_1'}.$$

Proof. We have to show that the vertical composition $\eta_1 * \eta_2$ of η_1 and η_2 is a categorical transducer homomorphism to $C_2 \cdot C_1 = (\mathsf{H}_1 \cdot \mathsf{H}_2, \mathsf{U}_1 \cdot \mathsf{U}_2, \pi_1 * \pi_2)$ from $C_2' \cdot C_1' = (\mathsf{H}_1' \cdot \mathsf{H}_2', \mathsf{U}_1' \cdot \mathsf{U}_2', \pi_1' * \pi_2')$. according to Definition 5.3.1:

(i) Obviously, we have $\eta_1 * \eta_2 : \mathsf{U}_1 \cdot \mathsf{U}_2 \dot{\leftarrow} \mathsf{U}'_1 \cdot \mathsf{U}'_2$.

(ii) We set $\widetilde{\eta_1 * \eta_2} = \tilde{\eta}_1 * \tilde{\eta}_2$ then we have $\widetilde{\eta_1 * \eta_2} : \mathsf{H}_1 \cdot \mathsf{H}_2 \dot{\leftarrow} \mathsf{H}'_1 \cdot \mathsf{H}'_2$. It remains to show that $|\widetilde{\eta_1 * \eta_2}|_{\mathsf{F}_1} = (\eta_1 * \eta_2)|\centerdot|_{\mathsf{F}_3}$ holds: For every $\mathcal{C}$-endofunctor F we use the abbreviation $\hat{\mathsf{F}}$ for the identical natural transformation from the forgetful functor $|\centerdot|_{\mathsf{F}}$ to itself, *i.e.* $\hat{\mathsf{F}} = id_{|\centerdot|_{\mathsf{F}}} : |\centerdot|_{\mathsf{F}} \dot{\leftarrow} |\centerdot|_{\mathsf{F}}$. Notice that from Definition and LemmaC.2.7 for every categorical transducer homomorphism between categorical transducers in $\boldsymbol{cat\mathcal{T}_{\mathcal{C}}}(\mathsf{G},\mathsf{F})$ we have $\hat{\mathsf{F}} * \tilde{\eta} = |\tilde{\eta}|_{\mathsf{F}}$ and $\eta_{|\centerdot|_{\mathsf{G}}} = \eta * \hat{\mathsf{G}}$. Thus we have got $\hat{\mathsf{F}}_1 * \tilde{\eta}_1 = \eta_1 * \hat{\mathsf{F}}_2$ and $\hat{\mathsf{F}}_2 * \tilde{\eta}_2 = \eta_2 * \hat{\mathsf{F}}_3$ and we have to show $\hat{\mathsf{F}}_1 * \widetilde{\eta_1 * \eta_2} = \eta_1 * \eta_2 * \hat{\mathsf{F}}_3$ which now is a straightforward calculation (using the associativity of $*$ according to Lemma C.2.9): $\hat{\mathsf{F}}_1 * \widetilde{\eta_1 * \eta_2} = \hat{\mathsf{F}}_1 * \tilde{\eta}_1 * \tilde{\eta}_2 = \eta_1 * \hat{\mathsf{F}}_2 * \tilde{\eta}_2 = \eta_1 * \eta_2 * \hat{\mathsf{F}}_3$.

(iii) With Lemma C.2.8 we have $(\pi_1 * \pi_2) \cdot (\eta_1 * \eta_2) = (\pi_1 \cdot \eta_1) * (\pi_2 \cdot \eta_2) = \pi'_1 * \pi'_2$.

■

5.3.6 Corollary (isomorphism is a congruence w.r.t. fusion). With the preconditions from Theorem 5.3.5 we have

$$\frac{C_2 \cong C'_2 \qquad C_1 \cong C'_1}{C_2 \cdot C_1 \cong C'_2 \cdot C'_1}.$$

Proof. We just have to show that the vertical composition of natural transformations $*$ preserves isomorphisms, which is easy to see with Lemma C.2.8 and the obvious fact that $id * id = id$. ■

5.4 Top-down categorical transducers

The concept of ‘categorical transducer’ is very general. Now we will define a subclass of so called top-down categorical transducers, which will be our model for top-down tree transducers in category theory. We derive a respective composition result for top-down categorical transducers.

5.4.1 Definition (top-down categorical transducer). Let $\mathcal{C}$ be a category which has finite products and finite coproducts. A categorical transducer $(\mathsf{H},\mathsf{U},\pi) \in \boldsymbol{cat\mathcal{T}_{\mathcal{C}}}(\mathsf{G},\mathsf{F})$ is called a **top-down categorical transducer** over $\mathcal{C}$ provided that

(i) F and G are finite coproducts of finite products of identity functors,

(ii) U is a finite product of identity functors, and

(iii) π is one of the projections of the product U.

Notice that it is part of the definition of the (top-down) categorical transducer that F and G have initial algebras. *

5.4.2 Theorem (composition of top-down categorical transducers). Let $\mathcal{C}$ be a category which has finite products and finite coproducts. The class of all top-down categorical transducers over $\mathcal{C}$ is a subcategory of $\boldsymbol{cat\mathcal{T}_{\mathcal{C}}}$ which we will denote by $td\text{-}\boldsymbol{cat\mathcal{T}_{\mathcal{C}}}$

Proof. According to Definition C.1.11 it suffices to show that $td\text{-}\boldsymbol{cat\mathcal{T}_{\mathcal{C}}}$ contains the identities and is closed under composition. Obviously, the identical categorical transducer $(\mathsf{Id},\mathsf{Id},id)$ is a top-down categorical transducer. Let $(\mathsf{H},\mathsf{U},\pi_1) : \mathsf{F}_3 \leftarrow \mathsf{F}_2$ and $(\mathsf{H}',\mathsf{U}',\pi'_1) : \mathsf{F}_2 \leftarrow \mathsf{F}_1$ be top-down categorical transducers. By Definition 5.4.1 there exist finite products

$$(\pi_i : \mathsf{Id} \dot{\leftarrow} \mathsf{U})_{i \in I} \quad \text{and} \quad (\pi'_j : \mathsf{Id} \dot{\leftarrow} \mathsf{U}')_{j \in J}.$$

where I and J are some finite sets with $1 \in I \cap J$. In order to show that

$$(\mathsf{H},\mathsf{U},\pi_1) \cdot (\mathsf{H}',\mathsf{U}',\pi'_1) = (\mathsf{H}' \cdot \mathsf{H}, \mathsf{U}' \cdot \mathsf{U}, \pi'_1 * \pi_1) : \mathsf{F}_3 \leftarrow \mathsf{F}_1$$

is a top-down categorical transducer it is sufficient to show that

$$(\pi'_j * \pi_i : \mathsf{Id} \dot{\leftarrow} \mathsf{U}' \cdot \mathsf{U})_{(i,j)\in I\times J}$$

is a finite product. We define for every $\mathsf{U}'' : \mathcal{C} \leftarrow \mathcal{C}$ and every $\tau_{ij} : \mathsf{Id} \dot{\leftarrow} \mathsf{U}''$ the pairing $\langle \tau_{ij} \rangle_{(i,j)\in I\times J} = \langle \langle \tau_{ij} \rangle_{i\in I} \rangle_{j\in J}$ and verify the UP (Table 6): Let $\sigma : \mathsf{Id} \dot{\leftarrow} \mathsf{U}''$ such that $\forall\,(i,j) \in I \times J ::\ (\pi'_j * \pi_i) \cdot \sigma = \tau_{ij}$. We have to show $\sigma = \langle \tau_{ij} \rangle_{(i,j)\in I\times J}$. First we calculate for every $(i,j) \in I \times J$:

$$\begin{array}{cl} & \pi'_j * \pi_i \\ = & \{ \text{ Definition and Lemma C.2.7 } \} \\ & \pi'_j \cdot \mathsf{U}'\pi_i \\ = & \{ \ \mathsf{U}' = \prod_{k\in J} \mathsf{Id} \ \} \\ & \pi'_j \cdot \prod_{j\in J} \pi_i \\ = & \{ \text{ fusion (i) for product functors (Table 8) } \} \\ & \pi_i \cdot \pi'_j . \end{array}$$

With this we continue:

$$\begin{array}{cl} & \forall\,(i,j) \in I \times J ::\ (\pi'_j * \pi_i) \cdot \sigma = \tau_{ij} \\ \Longleftrightarrow & \{ \text{ see above } \} \\ & \forall\,(i,j) \in I \times J ::\ \pi_i \cdot \pi'_j \cdot \sigma = \tau_{ij} \\ \Longleftrightarrow & \{ \text{ UP (Table 6) for the product } \mathsf{U} \ \} \\ & \forall\, j \in J ::\ \pi'_j \cdot \sigma = \langle \tau_{ij} \rangle_{i\in I} \\ \Longleftrightarrow & \{ \text{ UP (Table 6) for the product } \mathsf{U}' \ \} \\ & \sigma = \langle \langle \tau_{ij} \rangle_{i\in I} \rangle_{j\in J} = \langle \tau_{ij} \rangle_{(i,j)\in I\times J} . \end{array}$$

∎

5.4.3 Lemma. Let $\mathcal{C}$ be a category. The class $\mathrm{Ob}\, td\text{-}\boldsymbol{cat\mathcal{T}_{\mathcal{C}}}$ is a $\cong$-block in $\mathrm{Ob}\, \boldsymbol{cat\mathcal{T}_{\mathcal{C}}}$, *i.e.* a disjoint union of categorical transducer isomorphism classes.

Proof. Let $C = (\mathsf{H}, \mathsf{U}, \pi)$ and $C' = (\mathsf{H}', \mathsf{U}', \pi')$ be categorical transducers over $\mathcal{C}$ with $C \cong C'$. We have to show: if C is a top-down categorical transducer then C' is also a top down categorical transducer. Using Definition 5.3.1 and Definition 5.4.1 this is obvious, because we have a natural isomorphism $\eta : \mathsf{U} \dot{\leftarrow} \mathsf{U}'$ with $\pi \cdot \eta = \pi'$. ∎

5.4.4 Definition. We define the following class:

$$TOP_{cat} \ = \ \{\mathsf{S}C \ \mid \ C \in \mathrm{Mor}\, td\text{-}\boldsymbol{cat\mathcal{T}_{\mathcal{S}et}}\},$$

which the reader should compare with TOP_{tree} from Definition 4.2.2. $*$

6 Relating top-down tree transducers and categorical transducers

In this section we describe a translation (Lemma 6.2.9) of a top-down tree transducer into a top-down categorical transducer.

6.1 Category of forests

6.1.1 Definition (forest). Let Σ be a ranked alphabet. A Σ-**forest** is a tuple of Σ-trees, *i.e.* an element of T_Σ^*.

Now we will give the set of Σ-forests the structure of a category, by defining an appropriate composition. The following construction can also be found in [GTWW77] on page 74 (footnote 10) where functions are used instead of tuples. It is also possible to generalize this construction, which is then known as a Kleisli category [Kle65]. In this paper we will not focus on monads or Kleisli categories.

6.1.2 Definition (category of forests). Let Σ be a ranked alphabet. We define the category $\boldsymbol{\mathcal{T}}_\Sigma$ by

$$\operatorname{Ob}\boldsymbol{\mathcal{T}}_\Sigma = \mathbb{N}_0,$$
$$\boldsymbol{\mathcal{T}}_\Sigma(m,n) = (T_\Sigma X_n)^m$$

where $\forall\, l,m,n \in \operatorname{Ob}\boldsymbol{\mathcal{T}}_\Sigma ::\ \forall\, f = (f_i)_{i=1}^l \in \boldsymbol{\mathcal{T}}_\Sigma(l,m) ::\ \forall\, g = (g_j)_{j=1}^m \in \boldsymbol{\mathcal{T}}_\Sigma(m,n) ::$

$$f \cdot g = ([g_j/x_j]_{j=1}^m f_i)_{i=1}^l$$

and $\forall\, n \in \operatorname{Ob}\boldsymbol{\mathcal{T}}_\Sigma ::$

$$id_n = (x_i)_{i=1}^n.$$

Notice that $\boldsymbol{\mathcal{T}}_\Sigma$ is actually a pre-category, which we view as a category according to Note C.1.4. $*$

6.1.3 Lemma (finite products in the category of forests). Let Σ be a ranked alphabet. The category $\boldsymbol{\mathcal{T}}_\Sigma$ has finite products.

Proof. To simplify the notation we will only give the proof for a binary product. This may be generalized straightforwardly for arbitrary finite products. It is obvious that $0 \in \operatorname{Ob}\boldsymbol{\mathcal{T}}_\Sigma$ is a final object. Let $m,n \in \operatorname{Ob}\boldsymbol{\mathcal{T}}_\Sigma$. We define

$$\begin{aligned} m \times n &= m+n \quad \text{where } + \text{ is the usual sum of natural numbers,}\\ \pi_1 &= (x_1,\dots,x_m),\\ \pi_2 &= (x_{m+1},\dots,x_{m+n}) \end{aligned}$$

and $\forall\, l \in \operatorname{Ob}\boldsymbol{\mathcal{T}}_\Sigma ::\ \forall\, f = (f_i)_{i=1}^m \in \boldsymbol{\mathcal{T}}_\Sigma(m,l) ::\ \forall\, g = (g_j)_{j=1}^n \in \boldsymbol{\mathcal{T}}_\Sigma(n,l) ::$

$$\langle f,g\rangle = (f_1,\dots,f_m,\ g_1,\dots,g_n)$$

and can easily verify the UP of the product (Table 6). ∎

6.1.4 Note. The pairing of the product in $\boldsymbol{\mathcal{T}}_\Sigma$ from Lemma 6.1.3 is associative, because it is defined by the concatenation of tuples. Thus for the product functor in $\boldsymbol{\mathcal{T}}_\Sigma$ the following holds for every $m,n \in \mathbb{N}_0$: $\prod_{i=1}^m \cdot \prod_{j=1}^n = \prod_{i=1}^{m+n}$. This is not true in general: The pairing of the product in $\boldsymbol{\mathcal{S}et}$ from Lemma C.4.2 is not associative, because it is defined by tupling and successive tupling is not associative. However it is associative up to isomorphism. $*$

6.1.5 Lemma (embedding of the category of forests). Let Σ be a ranked alphabet. The function $\mathsf{E} : \operatorname{Mor}\boldsymbol{\mathcal{S}et} \leftarrow \operatorname{Mor}\boldsymbol{\mathcal{T}}_\Sigma$ defined by

$$\forall\, m,n \in \operatorname{Ob}\boldsymbol{\mathcal{T}}_\Sigma ::\ \forall\, f = (f_i)_{i=1}^m \in \boldsymbol{\mathcal{T}}_\Sigma(m,n) ::\ \forall\, t = (t_1,\dots,t_n) \in T_\Sigma^n ::$$
$$\mathsf{E}ft = ([t_j/x_j]_{j=1}^n f_i)_{i=1}^m$$

is an embedding functor

$$\mathsf{E} : \boldsymbol{\mathcal{S}et} \leftarrow \boldsymbol{\mathcal{T}}_\Sigma$$

which preserves finite products.

Proof. Note that, for every $l \in \operatorname{Ob}\boldsymbol{\mathcal{T}}_\Sigma$ we have $\mathsf{E}l = \boldsymbol{\mathcal{T}}_\Sigma^l$. We have to prove three statements:

(i) E is a functor. The composition in $\boldsymbol{\mathcal{T}}_\Sigma$ (*cf.* Definition 6.1.2) and the function E are both defined by means of a substitution operator. Using these definitions it is a straightforward calculation to show that E is a functor.

(ii) The functor E is an embedding. Let $f, g \in \boldsymbol{\mathcal{T}}_\Sigma(m,n)$. Then $\mathsf{E}f = \mathsf{E}g \implies f = \mathsf{E}f(x_i)_{i=1}^n = \mathsf{E}g(x_i)_{i=1}^n = g$

(iii) The functor E preserves products, *i.e.* it maps products onto products. From the definition of E we get that $\forall\, n \in \operatorname{Ob}\boldsymbol{\mathcal{T}}_\Sigma = \mathbb{N}_0 :: \ \mathsf{E}n = T_\Sigma^n$ and thus $\forall\, (m_i)_{i=1}^n \in \operatorname{Ob}\boldsymbol{\mathcal{T}}_\Sigma^n :: \ \mathsf{E}(\prod_{i=1}^n m_i) = \mathsf{E}(\sum_{i=1}^n m_i) = T_\Sigma^{\sum_{i=1}^n m_i} \cong \prod_{i=1}^n T_\Sigma^{m_i}$. ■

6.1.6 Note. Let Σ be a ranked alphabet.

(i) In order to avoid unnecessary notation, we will assume that the above embedding

$$\mathsf{E} : \boldsymbol{\mathcal{S}et} \leftarrow \boldsymbol{\mathcal{T}}_\Sigma$$

is an inclusion, or in other words, we identify a morphism $f \in \operatorname{Mor}\boldsymbol{\mathcal{T}}_\Sigma$ (*i.e.* a Σ-forest) with the set function $\mathsf{E}f$. Notice that this means that we have to identify an object $n \in \operatorname{Ob}\boldsymbol{\mathcal{T}}_\Sigma$ with the set T_Σ^n, which is not a problem, since in category theory objects are only indexes for the identities.

(ii) To make our notation even simpler, we will identify a symbol $\sigma \in \Sigma$ with the set function $\mathsf{E}\big(\sigma(x_1, \ldots, x_{\operatorname{rank}_\Sigma \sigma})\big)$ (see Definition B.2.2). Consider *e.g.* $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}\}$. We identify the forest $\big(\sigma(\alpha, x_1), \alpha, \sigma(\sigma(x_1,x_2),\alpha)\big) \in (T_\Sigma X_2)^3$ with a set function that we may write $\big\langle \langle \sigma \cdot \langle \alpha \cdot \mathbf{!}, \pi_1 \rangle, \alpha \cdot \mathbf{!}, \sigma \cdot \langle \sigma, \alpha \cdot \mathbf{!} \rangle \big\rangle : T_\Sigma^3 \leftarrow T_\Sigma^2$ $*$

6.1.7 Note. Let Σ, and Σ' be ranked alphabets with $\Sigma \subseteq \Sigma'$ such that for every $\sigma \in \Sigma$ the equality $\operatorname{rank}_{\Sigma'} \sigma = \operatorname{rank}_\Sigma \sigma$ holds. Since for every $n \in \mathbb{N}_0$: $T_\Sigma X_n \subseteq T_{\Sigma'} X_n$ we obviously have an embedding

$$\boldsymbol{\mathcal{T}}_{\Sigma'} \leftarrow \boldsymbol{\mathcal{T}}_\Sigma.$$

We will identify this embedding with the inclusion. $*$

6.2 Relating semantics of top-down tree transducers and top-down categorical transducers

6.2.1 Example (motivating example for relation). To motivate the description of the semantics of top-down tree transducers as categorical transducers over $\boldsymbol{\mathcal{S}et}$, consider the top-down tree transducer T_{zigzag} from Example 4.1.3. We view the rules of R as equations in the category $\boldsymbol{\mathcal{T}}_{Q\cup\Sigma\cup\Delta}$ which is embedded into $\boldsymbol{\mathcal{S}et}$ by E:

$$\begin{aligned} \mathsf{E}(\mathit{zig}\, x_1) \cdot \mathsf{E}(\alpha) &= \mathsf{E}(N), \\ \mathsf{E}(\mathit{zag}\, x_1) \cdot \mathsf{E}(\alpha) &= \mathsf{E}(N), \\ \mathsf{E}(\mathit{zig}\, x_1) \cdot \mathsf{E}(\sigma(x_1,x_2)) &= \mathsf{E}(A\, x_1) \cdot \mathsf{E}(\mathit{zag}\, x_1), \\ \mathsf{E}(\mathit{zag}\, x_1) \cdot \mathsf{E}(\sigma(x_1,x_2)) &= \mathsf{E}(B\, x_1) \cdot \mathsf{E}(\mathit{zig}\, x_2). \end{aligned}$$

If we identify forests with set functions according to Note 6.1.6 (i), we may describe the rules of R as a system of equations in the category $\boldsymbol{\mathcal{S}et}$ just by omitting the E. Our aim is to find solutions for zig and zag that suffice the above equations. First, let us simplify the notation according to Note 6.1.6 (ii):

$$\begin{aligned} \mathit{zig} \cdot \alpha &= N, \\ \mathit{zag} \cdot \alpha &= N, \\ \mathit{zig} \cdot \sigma &= A \cdot \mathit{zag} \cdot \pi_1, \\ \mathit{zag} \cdot \sigma &= B \cdot \mathit{zig} \cdot \pi_2. \end{aligned}$$

We use pairing to collect all the states:

$$\begin{aligned} \langle \mathit{zig} \cdot \alpha, \mathit{zag} \cdot \alpha \rangle &= \langle N, N \rangle, \\ \langle \mathit{zig} \cdot \sigma, \mathit{zag} \cdot \sigma \rangle &= \langle A \cdot \mathit{zag} \cdot \pi_1, B \cdot \mathit{zig} \cdot \pi_2 \rangle. \end{aligned}$$

On the left hand side we use the fusion law for the product (Table 6) and on the right hand side the fusion (i) law for product functors (Table 9) and cancelation law for products (Table 6) and obtain

$$\begin{aligned}&\langle zig, zag\rangle \cdot \alpha = \langle N, N\rangle \cdot id_1,\\ &\langle zig, zag\rangle \cdot \sigma = \langle A \cdot \pi_2 \cdot \pi_1, B \cdot \pi_1 \cdot \pi_2\rangle \cdot (\langle zig, zag\rangle \times \langle zig, zag\rangle).\end{aligned}$$

We use copairing to collect all input symbols:

$$[\langle zig, zag\rangle \cdot \alpha, \langle zig, zag\rangle \cdot \sigma] = [\langle N, N\rangle \cdot id_1, \langle A \cdot \pi_2 \cdot \pi_1, B \cdot \pi_1 \cdot \pi_2\rangle \cdot (\langle zig, zag\rangle \times \langle zig, zag\rangle)].$$

To the left hand side we apply the fusion law for coproducts (Table 7) and to the right hand side we apply the fusion (i) law for coproduct functors (Table 9) and obtain

$$\langle zig, zag\rangle \cdot [\alpha, \sigma] = [\langle N, N\rangle, \langle A \cdot \pi_2 \cdot \pi_1, B \cdot \pi_1 \cdot \pi_2\rangle] \cdot (id_1 + \langle zig, zag\rangle \times \langle zig, zag\rangle).$$

Since the functor $\mathsf{F} = \mathsf{K}_1 + \mathsf{Id} \times \mathsf{Id} : \mathcal{S}et \leftarrow \mathcal{S}et$ has the least fixed point $\mu\mathsf{F} = T_\Sigma$ where the initial F-algebra is $[\alpha, \sigma] = in_\mathsf{F} : \mu\mathsf{F} \leftarrow \mathsf{F}(\mu\mathsf{F})$, we can write

$$\langle zig, zag\rangle \cdot in_\mathsf{F} = [\langle N, N\rangle, \langle A \cdot \pi_2 \cdot \pi_1, B \cdot \pi_1 \cdot \pi_2\rangle] \cdot \mathsf{F}\langle zig, zag\rangle$$

Due to the UP of the catamorphism (Table 10), the above is equivalent to

$$\langle zig, zag\rangle = (\!|[\langle N, N\rangle, \langle A \cdot \pi_2 \cdot \pi_1, B \cdot \pi_1 \cdot \pi_2\rangle]|\!)_\mathsf{F}.$$

Notice that we found a *unique* solution for zig and for zag, respectively. Using the functor $\mathsf{G} = \mathsf{K}_1 + \mathsf{Id} + \mathsf{Id}$ with $in_\mathsf{G} = [N, A, B]$, we may write

$$\begin{aligned} zig &= \pi_1 \cdot (\!|[\langle N, N\rangle, \langle A \cdot \pi_2 \cdot \pi_1, B \cdot \pi_1 \cdot \pi_2\rangle]|\!)_\mathsf{F}\\ &= \pi_1 \cdot (\!|\mathsf{H}in_\mathsf{G}|\!)_\mathsf{F} \quad \text{where} \quad \mathsf{H}[\varphi_1, \varphi_2, \varphi_3] = [\langle \varphi_1, \varphi_1\rangle, \langle \varphi_2 \cdot \pi_2 \cdot \pi_1, \varphi_3 \cdot \pi_1 \cdot \pi_2\rangle]\end{aligned}$$

With Example 5.2.2 we get that

$$C_{\text{zigzag}} = (\mathsf{H}, \mathsf{Id} \times \mathsf{Id}, \pi_1) : \mathsf{G} \leftarrow \mathsf{F}$$

is a categorical transducer over $\mathcal{S}et$ with

$$zig = \pi_1 \cdot (\!|\mathsf{H}in_\mathsf{G}|\!)_\mathsf{F} = \mathsf{S}(\mathsf{H}, \mathsf{Id} \times \mathsf{Id}, \pi_1).$$

It is worth mentioning that the fact that H is a certain concrete functor is important in Theorem 5.2.3, because it is a precondition for the functorial 'acid rain theorem' 3.2.6. $*$

6.2.2 Example (top-down categorical transducer). The categorical transducer $C_{\text{zigzag}} = (\mathsf{H}, \mathsf{Id} \times \mathsf{Id}, \pi_1)$ from the Example 6.2.1 is a top-down categorical transducer over $\mathcal{S}et$. With the notations from the Example 6.2.1 we can show how a top-down categorical transducer 'works': for every $f : T^2_\Sigma \leftarrow T^0_\Sigma$:

$$\begin{aligned}
& \mathsf{S}C_{\text{zigzag}} \cdot \sigma \cdot f\\
=\;& \pi_1 \cdot (\!|\mathsf{H}in_\mathsf{G}|\!)_\mathsf{F} \cdot in_\mathsf{F} \cdot \iota_2 \cdot f\\
=\;& \quad \{\text{ UP (Table 10) }\}\\
& \pi_1 \cdot \mathsf{H}in_\mathsf{G} \cdot \mathsf{F}(\!|\mathsf{H}in_\mathsf{G}|\!)_\mathsf{F} \cdot \iota_2 \cdot f\\
=\;& \quad \{\text{ definition of H and F and cancelation for coproduct functors (Table 9) }\}\\
& \pi_1 \cdot [\langle N, N\rangle, \langle A \cdot \pi_2 \cdot \pi_1, B \cdot \pi_1 \cdot \pi_2\rangle] \cdot \iota_2 \cdot ((\!|\mathsf{H}in_\mathsf{G}|\!)_\mathsf{F} \times (\!|\mathsf{H}in_\mathsf{G}|\!)_\mathsf{F}) \cdot f\\
=\;& \quad \{\text{ cancelation (Table 7 and Table 6) }\}\\
& A \cdot \pi_2 \cdot \pi_1 \cdot ((\!|\mathsf{H}in_\mathsf{G}|\!)_\mathsf{F} \times (\!|\mathsf{H}in_\mathsf{G}|\!)_\mathsf{F}) \cdot f\\
=\;& \quad \{\text{ cancelation (Table 8) }\}\\
& A \cdot \pi_2 \cdot (\!|\mathsf{H}in_\mathsf{G}|\!)_\mathsf{F} \cdot \pi_1 \cdot f\\
=\;& A \cdot \mathsf{S}(\mathsf{H}, \mathsf{Id} \times \mathsf{Id}, \pi_2) \cdot \pi_1 \cdot f.
\end{aligned}$$

$*$

6.2.3 Note (motivation for relation). We want to generalize the construction from Example 6.2.1. Let $T \in \mathit{td\text{-}tree}\mathsf{T}(\Delta, \Sigma)$ be a top-down tree transducer and $C = (\mathsf{H}, \mathsf{U}, \pi) : \mathsf{G} \leftarrow \mathsf{F}$ be a top-down categorical transducer over $\boldsymbol{\mathcal{S}et}$. In Table 2 we list the parts of T and C which correspond to each other. The following Definition 6.2.5 will define this correspondence formally. $*$

top-down tree transducer	$\longleftrightarrow$	top-down categorical transducer
$T = (Q, \Sigma, \Delta, q_0, R)$	$\longleftrightarrow$	$C = (\mathsf{H}, \mathsf{U}, \pi) : \mathsf{G} \leftarrow \mathsf{F}$
Q	$\longleftrightarrow$	U
Σ	$\longleftrightarrow$	F
Δ	$\longleftrightarrow$	G
q_0	$\longleftrightarrow$	π
R	$\longleftrightarrow$	$\mathsf{H}in_{\mathsf{G}}$
T_Σ	$\longleftrightarrow$	$\mu\mathsf{F}$
T_Δ	$\longleftrightarrow$	$\mu\mathsf{G}$
τT	$\longleftrightarrow$	$\mathsf{S}C$

Table 2: Relation between the components of top-down tree and top-down categorical transducers

6.2.4 Lemma. Let $T = (Q, \Sigma, \Delta, q_1, R)$ be a top-down tree transducer with $Q = \{q_1, \ldots, q_l\}$, $r \in \mathbb{N}_0$, $\sigma \in \Sigma^{(r)}$ and $q \in Q$. We can write the right hand side of the rule

$$q(\sigma(x_1, \ldots, x_r)) \to \mathrm{rhs}_{R,\sigma}\, q$$

of R in the form (see Definition and Corollary C.5.1 for the definition of $\prod^r$)

$$\mathrm{rhs}_{R,\sigma}\, q = \mathrm{rhs}'_{R,\sigma}\, q \cdot \prod^{r}\big(\langle q_i\rangle_{i=1}^{l}\big) \quad \text{with} \quad \mathrm{rhs}'_{R,\sigma}\, q \in T_\Delta X_{r\cdot l}$$

where the composition is that of the category of forests and

$$\mathrm{rhs}'_{R,\sigma}\, q = [x_{(t-1)\cdot l+s}/q_s x_t]_{s=1}^{l}{}_{t=1}^{r}(\mathrm{rhs}_{R,\sigma}\, q).$$

Proof.

$$\begin{aligned}
& \mathrm{rhs}'_{R,\sigma}\, q \cdot \prod^{r}\big(\langle q_i\rangle_{i=1}^{l}\big) \\
= \quad & \{ \text{ Lemma 6.1.3 } \} \\
& \mathrm{rhs}'_{R,\sigma}\, q \cdot \prod_{t=1}^{r}\big(q_1 x_t, \ldots, q_l x_t\big) \\
= \quad & \{ \text{ Lemma 6.1.3 and Definition and Corollary C.5.1 } \} \\
& \mathrm{rhs}'_{R,\sigma}\, q \cdot (q_1x_1, \ldots, q_lx_1,\ q_1x_2, \ldots, q_lx_2,\ \ldots\ q_1x_r, \ldots, q_lx_r) \\
= \quad & \{ \text{ Definition 6.1.2 } \} \\
& [q_s x_t / x_{(t-1)\cdot l+s}]_{s=1}^{l}{}_{t=1}^{r}(\mathrm{rhs}'_{R,\sigma}\, q) \\
= \quad & \{ \text{ definition of } \mathrm{rhs}' \} \\
& \mathrm{rhs}_{R,\sigma}\, q.
\end{aligned}$$

■

6.2.5 Definition (relation). Let $T = (Q, \Sigma, \Delta, q_0, R)$ be a top-down tree transducer and $C = (\mathsf{H}, \mathsf{U}, \pi) \in \boldsymbol{cat\mathcal{T}_{\mathcal{S}et}}(\mathsf{G}, \mathsf{F})$ be a categorical transducer. We call T and C **related** and write

$$T \approx C$$

provided that

(i) $\mathsf{F} = \coprod_{\sigma\in\Sigma}(\prod^{\mathrm{rank}_\Sigma\,\sigma}\mathsf{Id})$ and $\mathsf{G} = \coprod_{\delta\in\Delta}(\prod^{\mathrm{rank}_\Delta\,\delta}\mathsf{Id})$ such that $in_\mathsf{F} = [\sigma]_{\sigma\in\Sigma}$ and $in_\mathsf{G} = [\delta]_{\delta\in\Delta}$,

(ii) U is a product $(\mathsf{Id} \xleftarrow{\pi_q} \mathsf{U})_{q\in Q}$ such that $\pi_{q_0} = \pi$, and

(iii) $\mathsf{H}in_\mathsf{G} = [\langle \mathrm{rhs}'_{R,\sigma}\, q\rangle_{q\in Q}]_{\sigma\in\Sigma}$. $*$

6.2.6 Example (relation). The top-down tree transducer T_{zigzag} from Example 4.1.3 and the categorical transducer C_{zigzag} from Example 6.2.1 are related: $T_{\text{zigzag}} \approx C_{\text{zigzag}}$. It is obvious by construction that (i) and (ii) from Definition 6.2.5 are satisfied. Let us have a look at (iii): From the definition of T_{zigzag} in Example 4.1.3 and with Lemma 6.2.4 we obtain

$$\begin{aligned}
\mathrm{rhs}'_{R,\alpha}\, zig &= N\\
\mathrm{rhs}'_{R,\alpha}\, zag &= N\\
\mathrm{rhs}'_{R,\sigma}\, zig &= A\, x_2\\
\mathrm{rhs}'_{R,\sigma}\, zag &= B\, x_3.
\end{aligned}$$

With Definition 6.1.2 and Lemma 6.1.3 we calculate:

$$\begin{aligned}
A\cdot\pi_2\cdot\pi_1 &= (A\,x_1)\cdot(x_2)\cdot(x_1,x_2) = A\,x_2,\\
B\cdot\pi_1\cdot\pi_2 &= (B\,x_1)\cdot(x_1)\cdot(x_3,x_4) = B\,x_3
\end{aligned}$$

and thus

$$\mathsf{H}in_\mathsf{G} = [\langle N,N\rangle, \langle A\cdot\pi_2\cdot\pi_1, B\cdot\pi_1\cdot\pi_2\rangle] = \big[\langle \mathrm{rhs}'_{R,\alpha}\, zig, \mathrm{rhs}'_{R,\alpha}\, zag\rangle, \langle \mathrm{rhs}'_{R,\sigma}\, zig, \mathrm{rhs}'_{R,\sigma}\, zag\rangle\big].$$

Notice that we have used the same symbols for different projections: $\pi_1,\pi_2 : T_\Delta \leftarrow T_\Delta^2$ and $\pi_1,\pi_2 : T_\Delta^2 \leftarrow T_\Delta^4$. We do this according to Definition C.2.4, because projections are natural transformations due to Definition and Corollary C.5.1. $*$

6.2.7 Theorem. Let C be a top-down categorical transducer over $\mathcal{S}et$ and T be a top-down tree transducer.

$$T \approx C \implies \tau T = \mathsf{S}C.$$

Proof. We use the notations from Definition 6.2.5 and let $Q = \{q_1,\dots,q_l\}$ with $l\in\mathbb{N}$.

$$\begin{aligned}
&(\!|\mathsf{H}in_\mathsf{G}|\!)_\mathsf{F} = \langle\tau_q T\rangle_{q\in Q}\\
\iff&\quad\{\text{ UP (Table 10) }\}\\
&\langle\tau_q T\rangle_{q\in Q}\cdot[\sigma]_{\sigma\in\Sigma} = \mathsf{H}in_\mathsf{G}\cdot\mathsf{F}\langle\tau_q T\rangle_{q\in Q}\\
\iff&\quad\{\text{ UP (Tables 6 and 7) }\}\\
&\forall\,\sigma\in\Sigma ::\ \forall\, q\in Q ::\ \tau_q T\cdot\sigma = \pi_q\cdot\mathsf{H}in_\mathsf{G}\cdot\mathsf{F}\langle\tau_q T\rangle_{q\in Q}\cdot\iota_\sigma\\
\iff&\quad\{\text{ definition of } \mathsf{F} \text{ in Definition 6.2.5; cancelation (Table 9) }\}\\
&\forall\,\sigma\in\Sigma ::\ \forall\, q\in Q ::\ \tau_q T\cdot\sigma = \pi_q\cdot\mathsf{H}in_\mathsf{G}\cdot\iota_\sigma\cdot\prod^{\mathrm{rank}_\Sigma\,\sigma}\big(\langle\tau_q T\rangle_{q\in Q}\big)\\
\iff&\quad\{\text{ Definition 6.2.5 (iii) }\}\\
&\forall\,\sigma\in\Sigma ::\ \forall\, q\in Q ::\ \tau_q T\cdot\sigma = \mathrm{rhs}'_{R,\sigma}\, q\cdot\prod^{\mathrm{rank}_\Sigma\,\sigma}\big(\langle\tau_q T\rangle_{q\in Q}\big)\\
\iff&\quad\{\text{ pointwise on terms with Lemma 6.2.4 and Definition 6.1.2 }\}\\
&\forall\,\sigma\in\Sigma ::\ \forall\, q\in Q ::\ \forall\,(t_r)_{r=1}^{\mathrm{rank}_\Sigma\,\sigma}\in T_\Sigma^{\mathrm{rank}_\Sigma\,\sigma} ::\\
&\quad\tau_q T\big(\sigma(t_1,\dots,t_{\mathrm{rank}_\Sigma\,\sigma})\big)\\
&\qquad=\big[\tau_{q_1}Tt_1/x_1,\dots,\tau_{q_l}Tt_1/x_l,\tau_{q_1}Tt_2/x_{l+1},\dots,\tau_{q_l}Tt_2/x_{2\cdot l},\dots\big](\mathrm{rhs}'_{R,\sigma}\, q)\\
&\qquad=\big[\tau_p T t_j/px_j\big]_{\substack{p\in Q\\ x_j\in X_{\mathrm{rank}_\Sigma\,\sigma}}}(\mathrm{rhs}_{R,\sigma}\, q).
\end{aligned}$$

The latter is the definition of τ in Definition 4.2.1 and thus with Definition 6.2.5 (ii):

$$\mathsf{S}C = \pi \cdot (\![\mathsf{H}in_{\mathsf{G}}]\!)_{\mathsf{F}} = \tau_{q_0} T = \tau T.$$

■

6.2.8 Lemma. Let $\Delta = \{\delta_1, \ldots, \delta_n\}$ be a ranked alphabet and $A = (|A|; \varphi_1, \ldots, \varphi_n)$ and $B = (|B|; \varphi'_1, \ldots, \varphi'_n)$ be Δ-algebras. For every Δ-algebra homomorphism $f : A \leftarrow B$, every $k \in \mathbb{N}_0$, and every Δ-term $t \in T_\Delta X_k$ the following holds:

$$[\varphi_i/\delta_i]_{i=1}^n t \cdot \prod^k |f| = |f| \cdot [\varphi'_i/\delta_i]_{i=1}^n t.$$

Proof. Let $(b_j)_{j=1}^k \in B^k$. The Δ-algebra $T_\Delta X_k$ is free over X_k. The 2nd-order substitution operators from Definition B.2.4 (ii) are defined as follows

(α) $\big([\varphi_i/\delta_i]_{i=1}^n t \cdot \prod^k |f|\big)(b_j)_{j=1}^k = [\varphi_i/\delta_i]_{i=1}^n t(|f|b_j)_{j=1}^k = |g|t$ where $g : A \leftarrow T_\Delta X_k$ is the unique Δ-algebra homomorphism with $\forall\, x_j \in X_k :: \ |g|x_j = |f|b_j$ and

(β) $[\varphi'_i/\delta_i]_{i=1}^n t(b_j)_{j=1}^k = |h|t$ where $h : B \leftarrow T_\Delta X_k$ is the unique Δ-algebra homomorphism with $\forall\, x_j \in X_k :: \ |h|x_j = b_j$.

Thus $\forall\, x_i \in X_k :: \ |f \cdot h|x_i = |f|\big(|h|x_i\big) = |f|b_i = |g|x_i$ and hence with (β): $g = f \cdot h$. ■

6.2.9 Lemma. For every top-down tree transducer $T \in td\text{-}tree\mathrm{T}$ there exists a related top-down categorical transducer $C \in \mathrm{Mor}\ td\text{-}\boldsymbol{cat\mathcal{T}}_{\boldsymbol{\mathcal{S}et}}$*, i.e.* $T \approx C$.

Proof. Let $T = (Q, \Sigma, \Delta, q_0, R)$ be a top-down tree transducer. We use the product and coproduct functor according to Lemma 6.1.3 and Lemma C.4.4 to define the functors $\mathsf{F}, \mathsf{G}, \mathsf{U} : \boldsymbol{\mathcal{S}et} \leftarrow \boldsymbol{\mathcal{S}et}$ by

$$\mathsf{F} = \coprod_{\sigma\in\Sigma} \Big(\prod^{\mathrm{rank}_\Sigma \sigma} \mathsf{Id} \Big), \qquad \mathsf{G} = \coprod_{\delta\in\Delta} \Big(\prod^{\mathrm{rank}_\Delta \delta} \mathsf{Id} \Big), \qquad \mathsf{U} = \prod^{\#Q} \mathsf{Id}.$$

According to Lemma C.6.3 we have the initial algebras

$$in_{\mathsf{F}} = [\sigma]_{\sigma\in\Sigma} \quad \text{and} \quad in_{\mathsf{G}} = [\delta]_{\delta\in\Delta}.$$

This choice of initial algebras leads to

$$\mu\mathsf{F} = T_\Sigma \quad \text{and} \quad \mu\mathsf{G} = T_\Delta.$$

Lemma C.5.5 ensures that U is faithful. We claim that the function $\mathsf{H} : \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{F}) \leftarrow \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{G})$ defined by

$$\forall\, \varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{G}) :: \ \mathsf{H}\varphi = \Big[\big\langle [\varphi \cdot \iota_\delta/\delta]_{\delta\in\Delta}(\mathrm{rhs}'_{R,\sigma}\ q)\big\rangle_{q\in Q}\Big]_{\sigma\in\Sigma}$$

can be uniquely extended to a concrete functor

$$\mathsf{H} : (\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{F}, |\centerdot|_{\mathsf{F}}) \leftarrow (\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{G}, \mathsf{U} \cdot |\centerdot|_{\mathsf{G}}).$$

To prove this we use Lemma 3.3.1, thus we have to verify that for every $\varphi, \varphi' \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{G})$ and every $f : |\varphi|_{\mathsf{G}} \leftarrow |\varphi|'_{\mathsf{G}}$ the condition

$$\frac{\varphi \cdot \mathsf{G}f = f \cdot \varphi'}{\mathsf{H}\varphi \cdot \mathsf{F}(\mathsf{U}f) = \mathsf{U}f \cdot \mathsf{H}\varphi'} \qquad (*)$$

holds. First we restate the precondition of $(*)$:

$$\begin{aligned}
& \varphi \cdot \mathsf{G}f = f \cdot \varphi' \\
\iff & \quad \{ \text{ fusion and reflection in Table 7 and definition of } \mathsf{G} \} \\
& [\varphi \cdot \iota_\delta]_{\delta\in\Delta} \cdot \coprod_{\delta\in\Delta} (\prod^{\mathrm{rank}_\Delta\,\delta} f) = f \cdot [\varphi' \cdot \iota_\delta]_{\delta\in\Delta} \\
\iff & \quad \{ \text{ fusion (i) in Table 9 } \} \\
& [\varphi \cdot \iota_\delta \cdot \prod^{\mathrm{rank}_\Delta\,\delta} f]_{\delta\in\Delta} = f \cdot [\varphi' \cdot \iota_\delta]_{\delta\in\Delta} \\
\iff & \quad \{ \text{ UP and cancelation in Table 7 } \} \\
& \forall\, \delta \in \Delta :: \ \varphi \cdot \iota_\delta \cdot \prod^{\mathrm{rank}_\Delta\,\delta} f = f \cdot \varphi' \cdot \iota_\delta \\
\iff & \quad \{ \text{ Definition B.1.3 } \} \\
& f : (|\varphi|_{\mathsf{G}}; (\varphi \cdot \iota_\delta)_{\delta\in\Delta}) \leftarrow (|\varphi'|_{\mathsf{G}}; (\varphi' \cdot \iota_\delta)_{\delta\in\Delta}) \text{ is a } \Delta\text{-algebra homomorphism.}
\end{aligned}$$

Now we show the conclusion of $(*)$:

$$\begin{aligned}
& \mathsf{H}\varphi \cdot \mathsf{F}(\mathsf{U}f) \\
= & \left[\left\langle [\varphi \cdot \iota_\delta/\delta]_{\delta\in\Delta}(\mathrm{rhs}'_{R,\sigma}\, q) \right\rangle_{q\in Q} \right]_{\sigma\in\Sigma} \cdot \coprod_{\sigma\in\Sigma} (\prod^{\mathrm{rank}_\Sigma\,\sigma} (\mathsf{U}f)) \\
= & \left[\left\langle [\varphi \cdot \iota_\delta/\delta]_{\delta\in\Delta}(\mathrm{rhs}'_{R,\sigma}\, q) \cdot \prod^{\mathrm{rank}_\Sigma\,\sigma} (\mathsf{U}f) \right\rangle_{q\in Q} \right]_{\sigma\in\Sigma} \\
= & \quad \{ \text{ Lemma 6.2.8 with } \prod^{\mathrm{rank}_\Sigma\,\sigma\cdot\#Q} = \prod^{\mathrm{rank}_\Sigma\,\sigma} \cdot\mathsf{U} \text{ and precondition of } (*) \} \\
& \left[\left\langle f \cdot [\varphi' \cdot \iota_\delta/\delta]_{\delta\in\Delta}(\mathrm{rhs}'_{R,\sigma}\, q) \right\rangle_{q\in Q} \right]_{\sigma\in\Sigma} \\
= & \quad \{ \text{ fusion (i) for product functors (Table 8), definition of } \mathsf{U}\text{, and fusion for coproducts (Table 7) } \} \\
& \mathsf{U}f \cdot \left[\left\langle [\varphi' \cdot \iota_\delta/\delta]_{\delta\in\Delta}(\mathrm{rhs}'_{R,\sigma}\, q) \right\rangle_{q\in Q} \right]_{\sigma\in\Sigma} \\
= & \ \mathsf{U}f \cdot \mathsf{H}\varphi'.
\end{aligned}$$

It is easy to see that $\mathsf{H}in_{\mathsf{G}} = \left[\left\langle (\mathrm{rhs}'_{R,\sigma}\, q) \right\rangle_{q\in Q} \right]_{\sigma\in\Sigma}$ and thus $C = (\mathsf{H}, \mathsf{U}, \pi_{q_0})$ is a categorical transducer over $\boldsymbol{\mathcal{S}et}$ with $T \approx C$. ■

6.2.10 Definition. We use the construction from the preceding Lemma 6.2.9 to define a function

$$\mathsf{R} : \mathrm{Mor}\ td\text{-}\boldsymbol{cat\mathcal{T}}_{\boldsymbol{\mathcal{S}et}} \leftarrow td\text{-}tree\mathrm{T}$$

Notice that the functors F, G, and U from Lemma 6.2.9 are only determined up to isomorphism. We make R a function just by choosing one of the representatives of the isomorphism class. Notice that we can view the category $td\text{-}\boldsymbol{cat\mathcal{T}}_{\boldsymbol{\mathcal{S}et}}$ modulo categorical transducer isomorphisms, because of Corollary 5.3.6, Lemma 5.4.3, and Theorem 5.3.4.

6.2.11 Corollary. From Lemma 6.2.9 and Theorem 6.2.7 it follows that:

(i) For every top-down tree transducer T we have: $T \approx \mathsf{R}\,T$.

(ii) $\tau = \mathsf{S} \cdot \mathsf{R}$.

$*$

6.2.12 Corollary. With Lemma 6.2.9 and Theorem 6.2.7 we get

$$TOP_{tree} \subseteq TOP_{cat}$$

6.2.13 Note. We have not yet proven the other direction, and thus equality, in Corollary 6.2.12, because there may be top-down categorical transducers, which are not in the image of R. Perhaps we would need additional preconditions on the concrete functors of the categorical transducers, to force R to be surjective.

6.3 Relating syntactic composition and fusion

6.3.1 Example (composition of top-down categorical transducers). Consider the two top-down tree transducers T_{zigzag} and T_{bin} from the Examples 4.1.3 and 4.3.3. For both we may construct a related categorical transducer over $\boldsymbol{\mathcal{S}et}$ according to Lemma 6.2.9, *i.e.* $T_{\text{zigzag}} \approx C_{\text{zigzag}}$ and $T_{\text{bin}} \approx C_{\text{bin}}$. In Example 6.2.1 we have already seen $C_{\text{zigzag}} = (\mathsf{H}, \mathsf{Id} \times \mathsf{Id}, \pi_1)$ where $\mathsf{H}[\varphi_1, \varphi_2, \varphi_3] = [\langle \varphi_1, \varphi_1 \rangle, \langle \varphi_2 \cdot \pi_2 \cdot \pi_1, \varphi_3 \cdot \pi_1 \cdot \pi_2 \rangle]$. Likewise we can construct the $C_{\text{bin}} = (\mathsf{H}', \mathsf{Id}, id)$ where $\mathsf{H}'[\varphi_1, \varphi_2] = [\varphi_1, \varphi_2 \cdot \langle id, id \rangle, \varphi_2 \cdot \langle id, id \rangle]$. We construct the composition (*cf.* Example 4.3.3):

$$\begin{aligned}
&C_{\text{bin}} \cdot C_{\text{zigzag}} \\
&= (\mathsf{H}', \mathsf{Id}, id) \cdot (\mathsf{H}, \mathsf{Id} \times \mathsf{Id}, \pi_1) \\
&= (\mathsf{H} \cdot \mathsf{H}', \mathsf{Id} \times \mathsf{Id}, \pi_1) \quad \text{where } (\mathsf{H} \cdot \mathsf{H}')[\varphi_1, \varphi_2] = [\langle \varphi_1, \varphi_1 \rangle, \varphi_2 \cdot \langle id, id \rangle \cdot \pi_2 \cdot \pi_1, \varphi_2 \cdot \langle id, id \rangle \cdot \pi_2 \cdot \pi_1]
\end{aligned}$$

In the same way we could construct the composition:

$$C_{\text{zigzag}} \cdot C_{\text{bin}} = (\mathsf{H}' \cdot \mathsf{H}, \mathsf{Id} \times \mathsf{Id}, \pi_1) \quad \text{where } (\mathsf{H}' \cdot \mathsf{H})[\varphi_1, \varphi_2, \varphi_3] = [\varphi_2 \cdot \pi_2 \cdot \pi_1, \varphi_3 \cdot \pi_1, \varphi_3 \cdot \pi_1].$$

$*$

The following theorem justifies the title of the paper: syntactic composition of top-down tree transducers is short cut fusion.

6.3.2 Theorem. Let T_1 and T_2 be top-down tree transducers such that the input alphabet of T_2 is the output alphabet of T_1. Then

$$\mathsf{R}\,T_2 \cdot \mathsf{R}\,T_1 = \mathsf{R}(T_2 \cdot T_1)$$

Proof. Let $T_1 = (P, \Sigma, \Delta, p_0, R_1)$, $T_2 = (Q, \Delta, \Gamma, q_0, R_2)$, and $T_2 \cdot T_1 = (Q \times P, \Sigma, \Gamma, (q_0, p_0), R)$ with

$$R = \big\{ (q,p)(\sigma(x_1, \dots)) \to \tau_q T_2'(\text{rhs}_{R_1,\sigma}\, p) \;\big|\; q \in Q \;\wedge\; p \in P \;\wedge\; \sigma \in \Sigma \big\}$$

where T_2' is constructed from T_2 as in the proof of Theorem 4.3.2. Let $r = \max(\text{rank}_\Sigma\, \sigma)$. Without loss of generality we can assume that Δ and Γ are disjoint. We define the top-down categorical tree transducers

$$\mathsf{F}_3 \xleftarrow{C_2 = (\mathsf{H}_2, \mathsf{U}_2, \pi_{q_0})} \mathsf{F}_2 \xleftarrow{C_1 = (\mathsf{H}_1, \mathsf{U}_1, \pi_{p_0})} \mathsf{F}_1 \quad \text{and} \quad \mathsf{F}_3 \xleftarrow{C = (\mathsf{H}, \mathsf{U}, \pi_{(q_0,p_0)})} \mathsf{F}_1$$

by $C_1 = \mathsf{R}\,T_1$, $C_2 = \mathsf{R}\,T_2$, and $C = \mathsf{R}(T_2 \cdot T_1)$ where we use the construction of Lemma 6.2.9 and thus in particular:

$$\begin{aligned}
\forall\, \varphi \in \text{Ob}(\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{F}_2) &:: \; \mathsf{H}_1\varphi = \big[\big\langle [\varphi \cdot \iota_\delta/\delta]_{\delta\in\Delta}(\text{rhs}'_{R_1,\sigma}\, p) \big\rangle_{p \in P}\big]_{\sigma\in\Sigma}, \\
\forall\, \varphi \in \text{Ob}(\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{F}_3) &:: \; \mathsf{H}_2\varphi = \big[\big\langle [\varphi \cdot \iota_\gamma/\gamma]_{\gamma\in\Gamma}(\text{rhs}'_{R_2,\delta}\, q) \big\rangle_{q \in Q}\big]_{\delta\in\Delta}, \text{ and} \\
\forall\, \varphi \in \text{Ob}(\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{F}_3) &:: \; \mathsf{H}\varphi = \big[\big\langle [\varphi \cdot \iota_\gamma/\gamma]_{\gamma\in\Gamma}(\text{rhs}'_{R,\sigma}\, p) \big\rangle_{(q,p) \in Q \times P}\big]_{\sigma\in\Sigma}.
\end{aligned}$$

We have to show that $C_2 \cdot C_1 = C$ holds. From Definition 5.1.1 we obtain $C_2 \cdot C_1 = (\mathsf{H}_1 \cdot \mathsf{H}_2, \mathsf{U}_1 \cdot \mathsf{U}_2, \pi_1 * \pi_2)$. Since the functors F_1, F_2, F_3, U_1, U_2, and U are determined only up to isomorphism, we may assume that $\mathsf{U}_1 \cdot \mathsf{U}_2 = \mathsf{U}$ and $\pi_{(q_0,p_0)} = \pi_{p_0} * \pi_{q_0}$ (*cf.* Theorem 5.4.2).

The essential statement we have to show is $\mathsf{H}_1 \cdot \mathsf{H}_2 = \mathsf{H}$: From the definition of τ in Definition 4.2.1 and with Lemma 6.2.4 we obtain

$$\forall\, \delta \in \Delta :: \ \langle \tau_q T_2' \rangle_{q\in Q} \cdot \delta = \langle \mathrm{rhs}'_{R_2,\sigma}\, q\rangle_{q\in Q} \cdot \prod^{\mathrm{rank}_\Sigma\, \sigma} \langle \tau_q T_2' \rangle_{q\in Q}$$

and $\forall\, px \in PX_r ::\ \langle \tau_q T_2' \rangle_{q\in Q}(px) = \big((q,p)x\big)_{q\in Q}$, *i.e.* $\langle \tau_q T_2' \rangle_{q\in Q}$ is a unique Δ-algebra homomorphism on the Δ-algebra $T_\Delta(PX_r)$ which is free on PX_r. With Definition B.2.4 (ii) we get

$$\begin{aligned}
&\forall\, p \in P ::\ \forall\, \sigma \in \Sigma ::\ [\langle \mathrm{rhs}_{R_2',\sigma}\, q\rangle_{q\in Q}/\delta]_{\delta\in\Delta}(\mathrm{rhs}_{R_1,\sigma}\, p) = \langle \tau_q T_2' \rangle(\mathrm{rhs}_{R_1,\sigma}\, p)\\
\Longrightarrow\quad & \{ \text{ definition of } R \ \}\\
&\forall\, p \in P ::\ \forall\, \sigma \in \Sigma ::\ [\langle \mathrm{rhs}_{R_2',\sigma}\, q\rangle_{q\in Q}/\delta]_{\delta\in\Delta}(\mathrm{rhs}_{R_1,\sigma}\, p) = \langle \mathrm{rhs}_{R,\sigma}(q,p)\rangle_{q\in Q}\\
\Longrightarrow\quad & \{ \text{ Lemma 6.2.4 } \}\\
&\forall\, p \in P ::\ \forall\, \sigma \in \Sigma ::\ [\langle \mathrm{rhs}'_{R_2,\sigma}\, q\rangle_{q\in Q}/\delta]_{\delta\in\Delta}(\mathrm{rhs}'_{R_1,\sigma}\, p) = \langle \mathrm{rhs}'_{R,\sigma}(q,p)\rangle_{q\in Q}\\
\Longrightarrow\quad & \{ \text{ apply the substitution operator } [\varphi\cdot\iota_\gamma/\gamma]_{\gamma\in\Gamma} \text{ on both sides } \}\\
&\forall\, \varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{F}_3) ::\ \forall\, p \in P ::\ \forall\, \sigma \in \Sigma ::\\
&\quad [\varphi\cdot\iota_\gamma/\gamma]_{\gamma\in\Gamma}\big([\langle \mathrm{rhs}'_{R_2,\sigma}\, q\rangle_{q\in Q}/\delta]_{\delta\in\Delta}(\mathrm{rhs}'_{R_1,\sigma}\, p)\big) = \langle [\varphi\cdot\iota_\gamma/\gamma]_{\gamma\in\Gamma}\, \mathrm{rhs}'_{R,\sigma}(q,p)\rangle_{q\in Q}\\
\Longrightarrow\quad & \{ \text{ there are no symbols from } \Gamma \text{ in the term } \mathrm{rhs}'_{R_1,\sigma p} \in T_\Delta X \ \}\\
&\forall\, \varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{F}_3) ::\ \forall\, p \in P ::\ \forall\, \sigma \in \Sigma ::\\
&\quad \big[\langle [\varphi\cdot\iota_\gamma/\gamma]_{\gamma\in\Gamma}\, \mathrm{rhs}'_{R_2,\sigma}\, q\rangle_{q\in Q}/\delta\big]_{\delta\in\Delta}(\mathrm{rhs}'_{R_1,\sigma}\, p) = \langle [\varphi\cdot\iota_\gamma/\gamma]_{\gamma\in\Gamma}\, \mathrm{rhs}'_{R,\sigma}(q,p)\rangle_{q\in Q}\\
\Longrightarrow\quad & \{ \text{ definition of } \mathsf{H}_2 \text{ and cancelation (Table 7) } \}\\
&\forall\, \varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{F}_3) ::\ \forall\, \sigma \in \Sigma ::\\
&\quad \big[\mathsf{H}_2\varphi\cdot\iota_\delta/\delta\big]_{\delta\in\Delta}(\mathrm{rhs}'_{R_1,\sigma}\, p) = \langle [\varphi\cdot\iota_\gamma/\gamma]_{\gamma\in\Gamma}\, \mathrm{rhs}'_{R,\sigma}(q,p)\rangle_{(q,p)\in Q\times P}\\
\Longrightarrow\quad & \{ \text{ definition of } \mathsf{H}_1 \text{ and } \mathsf{H} \text{ and cancelation (Table 7) } \}\\
&\forall\, \varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{S}et}}\mathsf{F}_3) ::\ \mathsf{H}_1(\mathsf{H}_2\varphi) = \mathsf{H}\varphi\\
\Longrightarrow\quad & \{ \text{ Lemma 2.4.6 } \}\\
&\mathsf{H}_1 \cdot \mathsf{H}_2 = \mathsf{H}.
\end{aligned}$$

■

6.3.3 Lemma. Let T_{id} be the top-down tree transducer from Lemma 4.3.4. Then $\mathsf{R}\,T_{id} = \mathsf{Id}$.

Proof. Let $\mathsf{R}\,T_{id} = (\mathsf{H}, \mathsf{U}, \pi)$. Since T_{id} has only one state, it is obvious that $\mathsf{U} = \mathsf{Id}$ and $\pi = id$. The construction for R in Lemma 6.2.9 yields for every φ: $\mathsf{H}\varphi = \big[[\varphi\cdot\iota_\sigma/\sigma]_{\sigma\in\Sigma}(\mathrm{rhs}'_{R,\sigma}\, q)\big]_{\sigma\in\Sigma} = \big[[\varphi\cdot\iota_\sigma/\sigma]_{\sigma\in\Sigma}\big]_{\sigma\in\Sigma} = [\varphi\cdot\iota_\sigma]_{\sigma\in\Sigma} = \varphi$. And thus we obtain with Lemma 2.4.6 that $\mathsf{H} = \mathsf{Id}$. ■

6.3.4 Lemma. Let $T_1 = (P, \Sigma, \Delta, p_0, R_1)$ and $T_2 = (Q, \Sigma, \Delta, q_0, R_2)$ be top-down tree transducers. The following holds:

$$\mathsf{R}\,T_1 = \mathsf{R}\,T_2 \quad\Longleftrightarrow\quad T_1 \cong T_2$$

where $\cong$ is the isomorphism of top-down tree transducers from Definition 4.2.4.

Proof. The direction '$\Leftarrow$' is obvious, because the construction from Lemma 6.2.9 which is the definition of R (Definition 6.2.10) depends only on the number of states rather than on the set of states. Let $C_1 = (\mathsf{H}_1, \mathsf{U}_1, \pi_{p_0}) = \mathsf{R}\,T_1$ and $C_2 = (\mathsf{H}_2, \mathsf{U}_2, \pi_{q_0}) = \mathsf{R}\,T_2$ where $C_1, C_2 : \mathsf{G} \leftarrow \mathsf{F}$. Thus we have $\mathsf{H}_1 = \mathsf{H}_2$, $\mathsf{U}_1 = \mathsf{U}_2$, and $\pi_{p_0} = \pi_{q_0}$. With the construction of R from Lemma 6.2.9 we obtain $\prod^{\#P} \mathsf{Id} = \mathsf{U}_1 =$

$\mathsf{U}_2 = \prod^{\#Q} \mathsf{Id}$ and thus $\#P = \#Q$, *i.e.* there exists a bijection between P and Q. We calculate

$$\begin{array}{ll}
 & \mathsf{R}\,T_1 = \mathsf{R}\,T_2 \\
\Longrightarrow & \quad \{ \text{ see above } \} \\
 & \mathsf{H}_1 = \mathsf{H}_2 \\
\Longrightarrow & \mathsf{H}_1\,in_{\mathsf{G}} = \mathsf{H}_2\,in_{\mathsf{G}} \\
\Longrightarrow & \quad \{ \text{ Corollary 6.2.11: } T_1 \approx C_1 \text{ and } T_2 \approx C_2 \text{ and Definition 6.2.5 } \} \\
 & \forall\,\sigma \in \Sigma{::}\ \langle \mathrm{rhs}'_{R_1,\sigma}\,p\rangle_{p\in P} = \langle \mathrm{rhs}'_{R_2,\sigma}\,q\rangle_{q\in Q} \\
\Longrightarrow & \quad \{ \text{ choose the appropriate bijection } h : P \leftarrow Q \} \\
 & \forall\,q \in Q{::}\ \forall\,\sigma \in \Sigma{::}\ \ \mathrm{rhs}'_{R_1,\sigma}(hq) = \mathrm{rhs}'_{R_2,\sigma}\,q \\
\Longrightarrow & \forall\,q \in Q{::}\ \forall\,\sigma \in \Sigma{::}\ \ \mathrm{rhs}'_{R_1,\sigma}(hq) \cdot \prod^{\mathrm{rank}_\Sigma\,\sigma} (\langle px\rangle_{p\in P}) = \mathrm{rhs}'_{R_2,\sigma}\,q \cdot \prod^{\mathrm{rank}_\Sigma\,\sigma} (\langle px\rangle_{p\in P}) \\
\Longrightarrow & \forall\,q \in Q{::}\ \forall\,\sigma \in \Sigma{::}\ \ \mathrm{rhs}'_{R_1,\sigma}(hq) \cdot \prod^{\mathrm{rank}_\Sigma\,\sigma} (\langle px\rangle_{p\in P}) = \mathrm{rhs}'_{R_2,\sigma}\,q \cdot \prod^{\mathrm{rank}_\Sigma\,\sigma} (\langle hq\,x\rangle_{q\in Q}) \\
\Longrightarrow & \quad \{ \text{ Lemma 6.2.4 } \} \\
 & \forall\,q \in Q{::}\ \forall\,\sigma \in \Sigma{::}\ \ \mathrm{rhs}_{R_1,\sigma}(hq) = [hq\,x/qx]_{\substack{q\in Q\\ x\in X}}(\mathrm{rhs}_{R_2,\sigma}\,q)
\end{array}$$

The latter is the property (iii) from Definition 4.2.4. ■

6.3.5 Theorem. (i) The class of all top-down tree transducers modulo $\cong$ is a category (denoted by $\textit{td-tree}\boldsymbol{\mathcal{T}}$) where the composition is the syntactic composition of top-down tree transducers.

(ii) The function R is an embedding functor $\mathsf{R} : \textit{td-}\boldsymbol{cat}\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{S}et}} \leftarrow \textit{td-tree}\boldsymbol{\mathcal{T}}$.

Proof. (i) We have to show that $\cong$ is a congruence relation w.r.t. syntactic composition of top-down tree transducers, and we have to show that syntactic composition is associative modulo $\cong$. Let T_1, T_2, and T_3 be top-down tree transducers (with input and output alphabets, such that the following compositions are defined): Since R maps to a category and is multiplicative (Theorem 6.3.2) we have $\mathsf{R}((T_3 \cdot T_2) \cdot T_1) = \mathsf{R}\,T_3 \cdot \mathsf{R}\,T_2 \cdot \mathsf{R}\,T_1 = \mathsf{R}(T_3 \cdot (T_2 \cdot T_1))$ and thus with Lemma 6.3.4 we obtain $(T_3 \cdot T_2) \cdot T_1 \cong T_3 \cdot (T_2 \cdot T_1)$. Similarly we show that $\cong$ is a congruence: Let T_1, T_1', T_2, and T_2' be top-down tree transducers (with input and output alphabets, such that the following compositions are defined) such that $T_1 \cong T_1'$ and $T_2 \cong T_2'$. From Lemma 6.3.4 we get that $\mathsf{R}\,T_1 = \mathsf{R}\,T_1'$ and $\mathsf{R}\,T_2 = \mathsf{R}\,T_2'$ and thus with Theorem 6.3.2: $\mathsf{R}(T_2 \cdot T_1) = \mathsf{R}\,T_2 \cdot \mathsf{R}\,T_1 = \mathsf{R}\,T_2' \cdot \mathsf{R}\,T_1' = \mathsf{R}(T_2' \cdot T_1')$. We use Lemma 6.3.4 again and have $T_2 \cdot T_1 \cong T_2' \cdot T_1'$. The latter means that $\cong$ is a congruence. Thus we obtain a category $\textit{td-tree}\boldsymbol{\mathcal{T}}$ where $\mathrm{Mor}\ \textit{td-tree}\boldsymbol{\mathcal{T}} = \textit{td-tree}\mathrm{T}/\cong$ and $\mathrm{Ob}\ \textit{td-tree}\boldsymbol{\mathcal{T}}$ is the class of all finite ranked alphabets. ■

(ii) We already know from Theorem 6.3.2 that R is multiplicative, from Lemma 6.3.3 that R preserves identities, and from Lemma 6.3.4 that R does not depend on the representative of the isomorphism class. Thus together with (i), we obtain that R is a functor $\mathsf{R} : \textit{td-}\boldsymbol{cat}\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{S}et}} \leftarrow \textit{td-tree}\boldsymbol{\mathcal{T}}$. It is obvious from Lemma 6.3.4 that R is also an embedding.

6.3.6 Note. We try to visualize the relation between top-down tree transducers and top-down categorical transducers over $\boldsymbol{\mathcal{S}et}$ in a diagram in Figure 1.

*

7 Future work

The composition of top-down tree transducers leads to a multiplication of the number states, thus it would be a good idea to *minimize* the number of states of the composition result. We want to investigate this

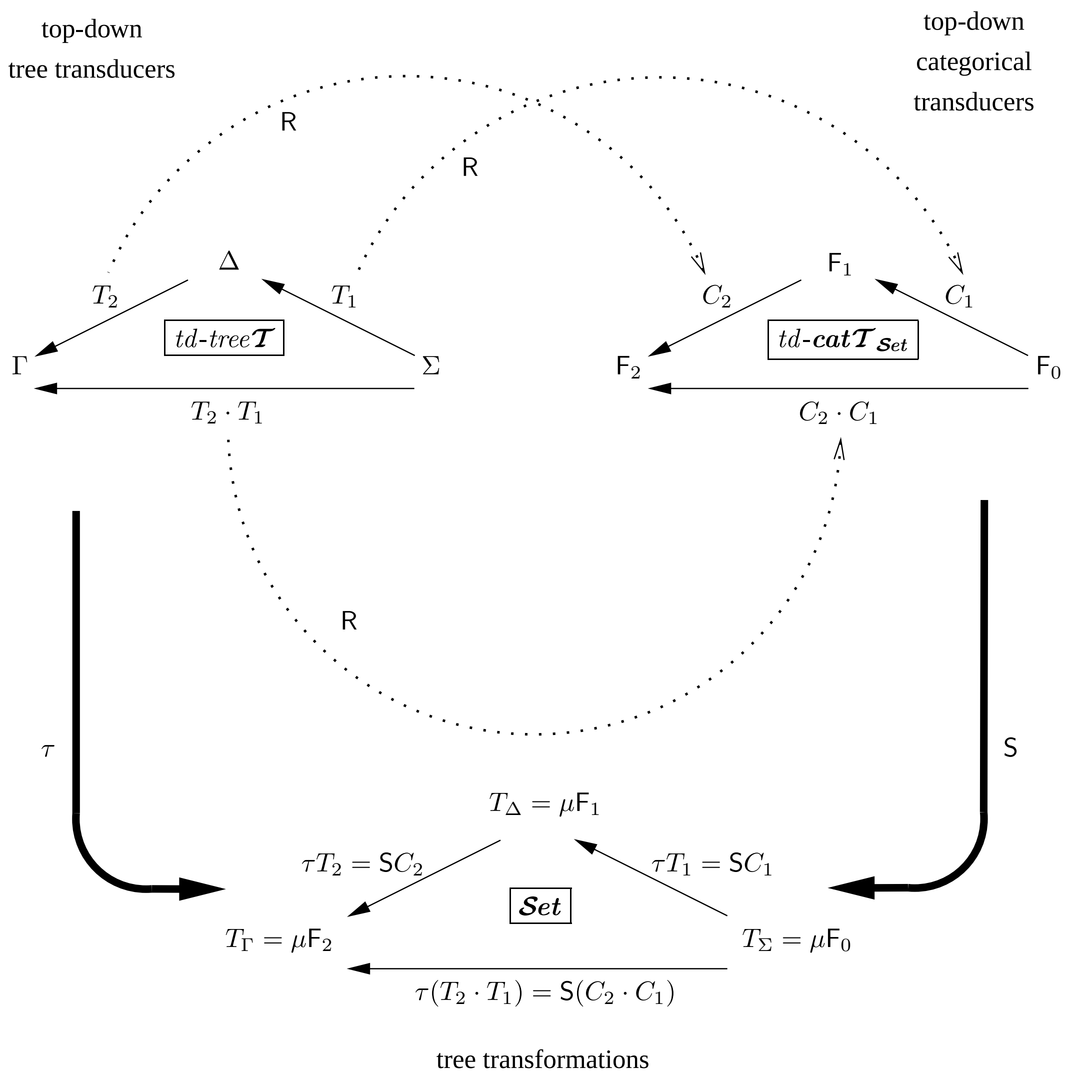

top-down tree transducers
top-down categorical transducers
R
R
Δ
T_2
T_1
td-tree$\boldsymbol{\mathcal{T}}$
Γ
Σ
$T_2 \cdot T_1$
F_1
C_2
C_1
td-$\boldsymbol{cat\mathcal{T}_{\mathcal{S}et}}$
F_2
F_0
$C_2 \cdot C_1$
R
τ
S
$T_\Delta = \mu\mathsf{F}_1$
$\tau T_2 = \mathsf{S}C_2$
$\tau T_1 = \mathsf{S}C_1$
$\boldsymbol{\mathcal{S}et}$
$T_\Gamma = \mu\mathsf{F}_2$
$T_\Sigma = \mu\mathsf{F}_0$
$\tau(T_2 \cdot T_1) = \mathsf{S}(C_2 \cdot C_1)$
tree transformations

Figure 1: Relation between $td\text{-}tree\boldsymbol{\mathcal{T}}$ and $td\text{-}\boldsymbol{cat\mathcal{T}_{\mathcal{S}et}}$

for categorical transducers, where we can use Lemma 5.3.4, *i.e.* categorical transducer homomorphisms preserve semantics:

$$\frac{\exists\,\eta : C \leftarrow C'}{\mathsf{S}C = \mathsf{S}C'}$$

and the fact that the number of states of a top-down categorical transducer can be expressed by the functor

$$\mathsf{Size} : (\mathbb{N}, \cdot, 1) \leftarrow td\text{-}\boldsymbol{cat\mathcal{T}_{\mathcal{C}}} : n \hookleftarrow (\mathsf{H}, \textstyle\prod_{i=1}^{n}\mathsf{Id}, \pi).$$

We want to describe the number of states of more complicated categorical transducers by analogous Size-functors. Then we want to investigate minimal categorical transducers like this is done for automata (*cf.* [Ehr74]).

We have shown that syntactic composition of top-down tree transducers and top-down categorical transducers over $\boldsymbol{\mathcal{S}et}$ is essentially the same. We want to extend our notion of a categorical transducer to describe tree transducers which are more complex than top-down tree transducers, *e.g. macro tree transducers* [Eng80, CF82, EV85b] where the class of all macro tree transformations is denoted by MAC.

A macro tree transducer is a generalization of a top-down tree transducer, where the states may have additional context parameters for values of the output tree. In [Eng81, EV85b] it is demonstrated that MAC is *not* closed under composition, *i.e.* the subclass of respective macro categorical transducers can not be a subcategory. However, there exist composition results for restricted macro tree transducers: using a translation into so called attributed tree transducers [KV01] or even a direct approach [VK01]. We want to know what all this means for a macro categorical transducer. Short cut fusion has some problems dealing with functions that have context parameters: either one has to introduce extra list consumption functions (which can not be guaranteed to be removed by subsequent fusions) or one can use the so called `cata/augment`-rule (*cf.* [Gil96, Joh01]), but that is only possible in certain cases. We believe that our concept of a categorical transducer leads us to a generalized fusion-rule for functions with context parameters.

Moreover, functions with context parameters may also be described using monads [Wad92] and the category of forests from Definition 6.1.1 can be generalized to a Kleisli category using some monad. We should investigate whether we can bring together these monads with our fusion results in the sense of [Fok94] and [Par00].

The attempt to close the class of macro tree transducers under composition leads to the class of *high-level tree transducers* [EV85a, EV88] where the context parameters may be functions. The types of the functions are restricted to a specific level hierarchy (also *cf.* [Dam82]). A high-level tree transducer is called n-level tree transducer if $n \in \mathbb{N}$ is the highest level occuring in this tree transducer. A 0-level or 1-level tree transducer is a top-down or macro tree transducer, respectively. The syntactic composition of an n-level and a k-level tree transducer is an $(n + k)$-level tree transducer, *i.e.* the class of high-level tree transducers is closed under composition. We want to define the class of high-level categorical transducers, such that it will be a category $hl\text{-}\boldsymbol{cat\mathcal{T}}$ where composition is fusion and the level is a functor $\mathsf{Level} : (\mathbb{N}_0, +, 0) \leftarrow hl\text{-}\boldsymbol{cat\mathcal{T}_{\mathcal{C}}}$. We hope that this leads to new insights into short cut fusion of functions with context parameters.

Acknowledgments

We would like to thank Janis Voigtländer and Dragan Masulovic for proofreading and Janis for his patience during our discussions.

A Laws

	comediators	**mediators**
UP	$f = \text{¡}_A \iff f : A \leftarrow 0$	$f = !_A \iff f : 1 \leftarrow A$
reflection	$\text{¡}_0 = id_0$	$!_1 = id_1$
fusion	$f : A \leftarrow B \implies f \cdot \text{¡}_B = \text{¡}_A$	$f : A \leftarrow B \implies !_A \cdot f = !_B$

Table 4 Table 5

	coproducts	**products**
UP	$h = [f_i]_{i\in I} \iff \forall\, i \in I :: \; h \cdot \iota_i = f_i$	$h = \langle f_i \rangle_{i\in I} \iff \forall\, i \in I :: \; \pi_i \cdot h = f_i$
reflection	$[\iota_i]_{i\in I} = id_{\coprod_{i\in I} A_i}$	$\langle \pi_i \rangle_{i\in I} = id_{\prod_{i\in I} A_i}$
fusion	$h \cdot [f_i]_{i\in I} = [h \cdot f_i]_{i\in I}$	$\langle f_i \rangle_{i\in I} \cdot h = \langle f_i \cdot h \rangle_{i\in I}$
cancelation	$\forall\, j \in I :: \; [f_i]_{i\in I} \cdot \iota_j = f_j$	$\forall\, j \in I :: \; \pi_j \cdot \langle f_i \rangle_{i\in I} = f_j$

Table 7 Table 6

	catamorphisms
UP	$f = \llparenthesis \varphi \rrparenthesis_{\mathsf{F}} \iff f \cdot in_{\mathsf{F}} = \varphi \cdot \mathsf{F} f$
reflection	$\llparenthesis in_{\mathsf{F}} \rrparenthesis_{\mathsf{F}} = id_{\mu \mathsf{F}}$
fusion	$f \cdot \varphi' = \varphi \cdot \mathsf{F} f \implies f \cdot \llparenthesis \varphi' \rrparenthesis_{\mathsf{F}} = \llparenthesis \varphi \rrparenthesis_{\mathsf{F}}$

Table 10

	coproduct functors	**product functors**
reflection	$\coprod_{i=1}^n id_{A_i} = id_{\coprod_{i=1}^n A_i}$	$\prod_{i=1}^n id_{A_i} = id_{\prod_{i=1}^n A_i}$
fusion (i)	$[f_i]_{i=1}^n \cdot \coprod_{i=1}^n g_i = [f_i \cdot g_i]_{i=1}^n$	$\prod_{i=1}^n f_i \cdot \langle g_i \rangle_{i=1}^n = \langle f_i \cdot g_i \rangle_{i=1}^n$
fusion (ii)	$\coprod_{i=1}^n g_i \cdot \coprod_{i=1}^n h_i = \coprod_{i=1}^n (g_i \cdot h_i)$	$\prod_{i=1}^n f_i \cdot \prod_{i=1}^n h_i = \prod_{i=1}^n (f_i \cdot h_i)$
cancelation	$\forall\, j :: \; \coprod_{i=1}^n g_i \cdot \iota_j = \iota_j \cdot g_j$	$\forall\, j :: \; \pi_j \cdot \prod_{i\in I}^n f_i = f_j \cdot \pi_j$

Table 9 Table 8

B Basic universal algebra

B.1 Algebras and homomorphisms

B.1.1 Definition (ranked alphabet). A finite set Σ together with a function $\mathrm{rank}_\Sigma : \mathbb{N}_0 \leftarrow \Sigma$ is called a **ranked alphabet**. The function rank_Σ is called the **rank-function** of the **alphabet** Σ. The elements of Σ are called **function-symbols** or just **symbols**. Sometimes we will use the (formally incorrect, but easy to understand) notation

$$\Sigma = \{\sigma_1^{(r_1)}, \ldots, \sigma_m^{(r_m)}\}$$

to indicate that $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ is a ranked alphabet with rank-function

$$\mathrm{rank}_\Sigma : \mathbb{N}_0 \leftarrow \Sigma : r_i \hookleftarrow \sigma_i \quad \forall\, i.$$

For every $k \in \mathbb{N}_0$ we define the following subset of the ranked alphabet Σ:

$$\Sigma^{(k)} = \big\{\sigma \in \Sigma \;\big|\; \mathrm{rank}_\Sigma\, \sigma = k\big\}.$$

A ranked alphabet Σ is called **unary** if $\Sigma = \Sigma^{(1)}$. If $\#\Sigma^{(0)} = 1$ and $\Sigma^{(k)} = \emptyset$ for every natural number $k > 2$, then Σ is called **monadic**. $*$

B.1.2 Definition (algebra). Let $\Sigma = \{\sigma_1^{(r_1)}, \ldots, \sigma_m^{(r_m)}\}$ be a ranked alphabet. A tuple $A = (|A|; f_1, \ldots, f_m)$, where $|A|$ is a set and the f_i are functions, is called a Σ**-algebra** provided that $\forall\, i :: \; f_i : |A| \leftarrow |A|^{r_i}$. We call $|A|$ the **carrier-set** of A, f_i the function belonging to the function-symbol σ_i, and r_i the **arity** of f_i. Functions with arity 1 are called **unary** and those with arity 2 are called **binary** functions. In general, a function with arity $k \in \mathbb{N}_0$ is called a k**-ary** function. $*$

B.1.3 Definition (algebra homomorphism). Let $\Sigma = \{\sigma_1^{(r_1)}, \ldots, \sigma_m^{(r_m)}\}$ be a ranked alphabet and $A = (|A|; f_1, \ldots, f_m)$ and $B = (|B|; g_1, \ldots, g_m)$ be Σ-algebras. A function $h : |A| \leftarrow |B|$ such that

$$\forall\, i :: \; \forall\, \xi_1, \ldots, \xi_{r_i} \in |B| :: \; h\big(g_i(\xi_1, \ldots, \xi_{r_i})\big) = f_i(h\xi_1, \ldots, h\xi_{r_i})$$

is called a Σ**-algebra homomorphism**. We denote this fact by

$$h : A \leftarrow B.$$

To emphasize the difference between the Σ-algebra homomorphism h and its underlying function, we sometimes denote this function by $|h| : |A| \leftarrow |B|$. $*$

B.1.4 Lemma (composition of algebra homomorphisms). The identity-functions and the composition of algebra homomorphisms are algebra homomorphisms. $*$

B.1.5 Definition (algebra isomorphism, isomorphic). Let Σ be a ranked alphabet and A and B be two Σ-algebras. A Σ-algebra homomorphism $f : A \leftarrow B$ is called a Σ**-algebra isomorphism**, provided that there exists a Σ-algebra homomorphism $g : B \leftarrow A$ such that

$$f \cdot g = id_A \quad \text{and} \quad g \cdot f = id_B.$$

We say that A and B are **isomorphic** if a Σ-algebra isomorphism to A from B exist, and we denote this fact by $A \cong B$. It is easy to see that the relation $\cong$ is an equivalence relation on the class of all Σ-algebras. $*$

B.2 Free algebras and substitutions

B.2.1 Definition (free algebra). Let Σ be a ranked alphabet, A be a Σ-algebra and $X \subseteq |A|$ a set. The Σ-algebra A is called **free** over the set X provided that for every Σ-algebra B and every function $f : |B| \leftarrow X$ there exists a unique Σ-algebra homomorphism $f' : B \leftarrow A$ such that $\forall\, x \in X :: \; |f'|x = fx$. $*$

B.2.2 Definition (term algebra, term, tree). Let $\{\boxed{(},\boxed{,},\boxed{)}\}$ be a set of three symbols which do not occur in any alphabet that we will use. Let Σ be a ranked alphabet and A a set disjoint with Σ. We define the sets

$$\Sigma A = \bigl\{\sigma\boxed{(}a_1\boxed{,}\,\ldots\,\boxed{,}a_{\mathrm{rank}_\Sigma\,\sigma}\boxed{)}\ \mid\ \sigma \in \Sigma\ \wedge\ \forall i\!::\ a_i \in A\bigr\}$$

and

$$T_\Sigma A = \bigcap\bigl\{T \subseteq (\Sigma \cup \{\boxed{(},\boxed{,},\boxed{)}\} \cup A)^*\ \mid\ A \cup \Sigma T \subseteq T\bigr\}.$$

Since $A \cup \Sigma(T_\Sigma A) \subseteq T_\Sigma A$, we obtain that $T_\Sigma A$ is the smallest subset $T \subseteq (\Sigma \cup \{\boxed{(},\boxed{,},\boxed{)}\} \cup A)^*$ such that

$$\frac{a \in A}{a \in T} \quad\text{and}\quad \frac{\sigma \in \Sigma \qquad t_1, \ldots, t_{\mathrm{rank}_\Sigma\,\sigma} \in T}{\sigma\boxed{(}t_1\boxed{,}\,\ldots\,\boxed{,}t_{\mathrm{rank}_\Sigma\,\sigma}\boxed{)} \in T}.$$

An element of $T_\Sigma A$ is called a (finite, labeled and ordered) Σ-**term** or Σ-**tree** indexed by A. (More precisely a Σ-tree indexed by A is a $(\Sigma \cup A)$-labeled graph, representing a term.) We will write just $\sigma(t_1, \ldots, t_{\mathrm{rank}_\Sigma\,\sigma})$ for $\sigma\boxed{(}t_1\boxed{,}\,\ldots\,\boxed{,}t_{\mathrm{rank}_\Sigma\,\sigma}\boxed{)}$. Furthermore, for every $\sigma \in \Sigma^{(1)}$ we will write σt_1 for $\sigma(t_1)$ and for every $\sigma \in \Sigma^{(0)}$ we will write σ for $\sigma()$. If we identify an element $\sigma \in \Sigma$ with the following function

$$\sigma : T_\Sigma A \leftarrow (T_\Sigma A)^{\mathrm{rank}_\Sigma\,\sigma} : \sigma(t_1, \ldots, t_{\mathrm{rank}_\Sigma\,\sigma}) \hookleftarrow (t_1, \ldots, t_{\mathrm{rank}_\Sigma\,\sigma}) \quad \forall\, t_1, \ldots, t_{\mathrm{rank}_\Sigma\,\sigma} \in T_\Sigma A,$$

then it is easy to see that $(T_\Sigma A; \sigma_1, \ldots, \sigma_m)$, where $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$, is a Σ-algebra, which we call the Σ-**term algebra** over A. Usually we denote it just by $T_\Sigma A$. Finally we set $T_\Sigma = T_\Sigma\emptyset$ and call it the **initial** Σ-**term algebra**. Notice that we have *identified* function-symbols of rank k with k-ary functions, in order to avoid extra notation. $*$

B.2.3 Theorem (essential uniqueness of free algebras). Let Σ be a ranked alphabet and X a set disjoint with Σ.

(i) A free Σ-algebra over X is uniquely determined up to isomorphism.

(ii) The Σ-term algebra $T_\Sigma X$ is free over X. $*$

B.2.4 Definition (substitution). Let Σ be a ranked alphabet and A be a free Σ-algebra over $X \subseteq |A|$.

(i) For every $k \in \mathbb{N}_0$ and $a_1, \ldots, a_k \in |A|$ and pairwise distinct $x_1, \ldots, x_k \in X$ we define the **substitution** operator

$$[a_j/x_j]_{j=1}^k = [a_1/x_1, \ldots, a_k/x_k] : |A| \leftarrow |A|$$

by $[a_j/x_j]_{j=1}^k = |f|$ where $f : A \leftarrow A$ is the unique Σ-algebra homomorphism with $\forall\, j \in \{1, \ldots, k\}\!::\ |f|x_j = a_j$ and $\forall\, x \in X \setminus \{x_1, \ldots, x_k\}\!::\ |f|x = x$. Notice that in the case $A = T_\Sigma X$ this is the common term substitution.

(ii) Let $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ and $X = \{x_1, \ldots, x_k\}$. For every $\varphi_1, \ldots, \varphi_n$ such that $B = (|B|; \varphi_1, \ldots, \varphi_n)$ is a Σ-algebra we define the 2**nd** **order substitution** operator

$$[\varphi_i/\sigma_i]_{i=1}^n = [\varphi_1/\sigma_1, \ldots, \varphi_n/\sigma_n] : |B|^{|B|^k} \leftarrow |A|$$

for every $a \in A$ and every $(b_j)_{j=1}^k \in |B|^k$ by $[\varphi_i/\sigma_i]_{i=1}^n a\,(b_j)_{j=1}^k = |g|a$ where $g : B \leftarrow A$ is the unique Σ-algebra homomorphism with $\forall\, j \in \{1, \ldots, k\}\!::\ |g|x_j = b_j$. $*$

C Basic category theory

We state the basic notions and notations of category theory, which will be used in this paper. Our notation follows [AHS90] and [BdM97]. The collection of calculation laws in tables (Tables 4, 5, 6, 7, and 10) is inspired by [Fok92a].

C.0.5 Note (set, class, conglomerate). The set of all sets does not exist due to Russel's Paradox. In order to handle 'large collections of things' like 'the collection of all sets' we use class-theory (*cf.* [Fel78]). A **class** is a generalization of a set. Intuitively classes are in a way bigger than sets. Now we define a set to be a class, which is an element of some other class, *i.e.*: A is a set, iff A is a class and there exists a class C such that $A \in C$. Thus every set is a class. However the collection of all sets is a class and it is not a set.

Again, due to Russel's Paradox, the collection of all classes is not a class. But we can iterate the above concept to define **conglomerates** and build the conglomerate of all classes.

In fact one could construct classes of order n for $n \in \mathbb{N}_0$ where the collection of all classes of order n is a class of order $n{+}1$ which is not a class of order n itself. In that context sets, classes and conglomerates would be classes of order 0, 1, and 2, respectively. We will not need classes of order greater than 2 in this paper. $*$

C.0.6 Example (notation of classes). We denote classes (of any order) using braces $\{\dots\}$ just like we denote sets. The class of all singleton sets $S = \{s \mid s$ is a set with one element$\}$ is *not* a set. $*$

C.1 Categories and Functors

C.1.1 Definition (category). A **category** $\mathcal{C} = (\mathrm{Ob}\,\mathcal{C}, \mathrm{Mor}\,\mathcal{C}, \mathrm{dom}, \mathrm{cod}, \cdot, id)$ consists of a class $\mathrm{Ob}\,\mathcal{C}$ of so called **objects**, a class $\mathrm{Mor}\,\mathcal{C}$ of so called **morphisms**, two functions $\mathrm{dom}, \mathrm{cod} : \mathrm{Ob}\,\mathcal{C} \leftarrow \mathrm{Mor}\,\mathcal{C}$ called **domain**- (or **source**-)function and **codomain**- (or **target**-)function, a partial function $\cdot : \mathrm{Mor}\,\mathcal{C} \leftarrow\!\cdots \mathrm{Mor}\,\mathcal{C} \times \mathrm{Mor}\,\mathcal{C}$ called **composition**, and a function $id_{\centerdot} : \mathrm{Mor}\,\mathcal{C} \leftarrow \mathrm{Ob}\,\mathcal{C}$ called **identity**. Before we introduce the axioms which a category has to satisfy, we define the **hom-classes** as the classes

$$\forall\, A, B \in \mathrm{Ob}\,\mathcal{C} :: \ \mathcal{C}(A, B) = \big\{ f \in \mathrm{Mor}\,\mathcal{C} \ \big|\ A = \mathrm{cod}\, f \ \wedge\ \mathrm{dom}\, f = B \big\}$$

and the ternary relation $(\centerdot : \centerdot \underset{\mathcal{C}}{\longleftarrow} \centerdot) \subseteq \mathrm{Mor}\,\mathcal{C} \times \mathrm{Ob}\,\mathcal{C} \times \mathrm{Ob}\,\mathcal{C}$ by

$$\forall\, A, B \in \mathrm{Ob}\,\mathcal{C} :: \ f : A \underset{\mathcal{C}}{\longleftarrow} B \iff A \overset{f}{\underset{\mathcal{C}}{\longleftarrow}} B \iff f \in \mathcal{C}(A, B)$$

which we will write just as

$$f : A \leftarrow B \qquad \text{or} \qquad A \xleftarrow{f} B$$

if the connection to the category $\mathcal{C}$ is obvious. The axioms are:

$$\frac{f, g, h \in \mathrm{Mor}\,\mathcal{C} \qquad f \cdot g = h}{\mathrm{dom}\, f = \mathrm{cod}\, g} \qquad \text{(typing)}$$

$$\frac{f : A \leftarrow B \qquad g : B \leftarrow C}{f \cdot g : A \leftarrow C} \qquad \text{(composition)}$$

$$\frac{f : A \leftarrow B \qquad g : B \leftarrow C \qquad h : C \leftarrow D}{(f \cdot g) \cdot h = f \cdot (g \cdot h)} \qquad \text{(associativity)}$$

$$\frac{A \in \mathrm{Ob}\,\mathcal{C}}{id_A : A \leftarrow A} \qquad \frac{f : A \leftarrow B}{f \cdot id_B = f = id_A \cdot f} \qquad \text{(identity)}$$

$*$

C.1.2 Lemma. For every category $\mathcal{C}$ the class $\{\mathcal{C}(A,B) \mid A,B \in \mathrm{Ob}\,\mathcal{C}\}$ is a partition of $\mathrm{Mor}\,\mathcal{C}$, *i.e.* $\mathrm{Mor}\,\mathcal{C} = \biguplus_{A,B\in\mathrm{Ob}\,\mathcal{C}} \mathcal{C}(A,B)$. $*$

C.1.3 Definition (pre-category). A pre-category is defined by the same axioms as a category except that domain and codomain are not unique, *i.e.* $\mathrm{dom}, \mathrm{cod} : \mathcal{Pot}(\mathrm{Ob}\,\mathcal{C}) \leftarrow \mathrm{Mor}\,\mathcal{C}$. We use the same notations for pre-categories as defined for categories in Definition C.1.1. For a pre-category $\mathcal{C}$ we like to mention the definition of hom-classes

$$\forall\, A,B \in \mathrm{Ob}\,\mathcal{C}(A,B) = \{f \in \mathrm{Mor}\,\mathcal{C} \mid A \in \mathrm{cod}\, f \ \wedge\ B \in \mathrm{dom}\, f\}$$

and the typing axiom

$$\frac{f,g,h \in \mathrm{Mor}\,\mathcal{C} \qquad f \cdot g = h}{\mathrm{dom}\, f \cap \mathrm{cod}\, g \neq \emptyset}.$$

$*$

C.1.4 Note. For every pre-category $\mathcal{C}$ we can construct a category $\mathcal{C}'$ by

$$\mathrm{Ob}\,\mathcal{C}' = \mathrm{Ob}\,\mathcal{C} \qquad \forall\, A,B \in \mathrm{Ob}\,\mathcal{C} :: \ \mathcal{C}'(A,B) = \{(A,f,B) \mid f : A \xleftarrow[\mathcal{C}]{} B\}$$

with the obvious composition and identities inherited from $\mathcal{C}$. Notice that the definition of the $\mathcal{C}'$-hom-classes also uniquely determines domains and codomains. $*$

C.1.5 Example (category). We show some categories in Table 3.

category	**objects**	**morphisms**
$\mathcal{Set}$	all sets	all set functions
Σ-$\mathcal{Alg}$	all Σ-algebras	all Σ-algebra homomorphisms
$\mathcal{Top}$	all topological spaces	all continuous functions

Table 3: Some categories

$*$

C.1.6 Definition and Theorem (duality principle). For every category $\mathcal{C}$ we define the **dual** (or **opposite**) category $\mathcal{C}^{\mathrm{op}}$ by

$$\begin{aligned} \mathrm{Ob}\,\mathcal{C}^{\mathrm{op}} &= \mathrm{Ob}\,\mathcal{C} \\ \mathcal{C}^{\mathrm{op}}(A,B) &= \mathcal{C}(B,A) \quad \forall\, A,B \in \mathrm{Ob}\,\mathcal{C} \\ f \cdot_{\mathcal{C}^{\mathrm{op}}} g &= g \cdot_{\mathcal{C}} f \quad \forall\, f,g \in \mathrm{Mor}\,\mathcal{C} \text{ with } \mathrm{dom}_{\mathcal{C}}\, g = \mathrm{cod}_{\mathcal{C}}\, f \\ id_{\mathcal{C}^{\mathrm{op}}} &= id_{\mathcal{C}}, \end{aligned}$$

i.e. $\mathcal{C}^{\mathrm{op}}$ has the same objects and morphisms as $\mathcal{C}$, but dom and cod are exchanged and the composition is commuted. The transformation from $\mathcal{C}$ to $\mathcal{C}^{\mathrm{op}}$ may be viewed as the reversion of all morphism arrows. If this is done twice, then we receive the original category:

$$(\mathcal{C}^{\mathrm{op}})^{\mathrm{op}} = \mathcal{C}.$$

We will use this symmetry as follows: For every predicate $A(\mathcal{C})$ about a category $\mathcal{C}$ we derive the **dual predicate** $A^{\mathrm{op}}(\mathcal{C}) \iff A(\mathcal{C}^{\mathrm{op}})$ by reversing all morphism arrows. Then the following equivalence holds:

$$\Big(\text{for every category } \mathcal{C} :: \ A(\mathcal{C})\Big) \iff \Big(\text{for every category } \mathcal{C} :: \ A^{\mathrm{op}}(\mathcal{C})\Big)$$

Proof.

$$\begin{array}{ll} & \forall \mathcal{C} :: \ A(\mathcal{C}) \\ \iff & \quad \{ \text{ since } \forall \mathcal{C} :: \ \mathcal{C} = (\mathcal{C}^{\mathrm{op}})^{\mathrm{op}} \ \} \\ & \forall \mathcal{C} :: \ A(\mathcal{C}^{\mathrm{op}}) \\ \iff & \quad \{ \text{ definition of the dual predicate } \} \\ & \forall \mathcal{C} :: \ A^{\mathrm{op}}(\mathcal{C}) \end{array}$$

■

Thus a proof for a predicate about categories is also a proof for the dual predicate. Furthermore, for every notion defined in category theory there is a **dual notion**. We only need to investigate one of them and get the properties of the dual notion by the duality principle. $*$

C.1.7 Definition (isomorphism (*cf.* Definition B.1.5)**).** Let $\mathcal{C}$ be a category and $A, B \in \mathrm{Ob}\,\mathcal{C}$. A morphism $f : A \leftarrow B$ for which

$$\exists\, g : B \leftarrow A :: \ f \cdot g = id_A \ \wedge \ g \cdot f = id_B$$

holds is called an **isomorphism**. It is easy to see that in this case g is unique. We call g the **inverse** of f and denote it by f^{-1}. Two objects $A, B \in \mathrm{Ob}\,\mathcal{C}$ such that

$$\exists \text{ isomorphism } f \in \mathrm{Mor}\,\mathcal{C} :: \ f : A \leftarrow B$$

are called **isomorphic**, and we write

$$A \cong B$$

Obviously, the relation $\cong$ is an equivalence relation on $\mathrm{Ob}\,\mathcal{C}$. $*$

C.1.8 Definition (functor). Let $\mathcal{C}$ and $\mathcal{D}$ be categories. A **(covariant) functor** F to $\mathcal{C}$ from $\mathcal{D}$ consists of two functions

$$\mathsf{F} : \mathrm{Ob}\,\mathcal{C} \leftarrow \mathrm{Ob}\,\mathcal{D} \quad \text{and} \quad \mathsf{F} : \mathrm{Mor}\,\mathcal{C} \leftarrow \mathrm{Mor}\,\mathcal{D},$$

which satisfy the following axioms:

$$\frac{f : A \underset{\mathcal{D}}{\longleftarrow} B}{\mathsf{F}f : \mathsf{F}A \underset{\mathcal{C}}{\longleftarrow} \mathsf{F}B} \qquad \text{(typing)}$$

$$\frac{A \in \mathrm{Ob}\,\mathcal{D}}{\mathsf{F}id_A = id_{\mathsf{F}A}} \qquad \text{(identity)}$$

$$\frac{f : A \underset{\mathcal{D}}{\longleftarrow} B \qquad g : B \underset{\mathcal{D}}{\longleftarrow} C}{\mathsf{F}(f \cdot g) = \mathsf{F}f \ \cdot \ \mathsf{F}g} \qquad \text{(multiplicativity)}$$

In this case we write:

$$\mathsf{F} : \mathcal{C} \leftarrow \mathcal{D}$$

It is common to denote the functor as well as the two underlying functions by the same symbol F. In fact, a functor is already determined on objects, if it is defined on morphisms, because it follows from the typing-axiom that $\forall A \in \mathrm{Ob}\,\mathcal{C} :: \ \mathsf{F}A = \mathrm{dom}(\mathsf{F}id_A)$.

For every functor $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{D}$ we define the **dual functor** $\mathsf{F}^{\mathrm{op}} : \mathcal{C}^{\mathrm{op}} \leftarrow \mathcal{D}^{\mathrm{op}}$, which is determined by the same underlying functions on objects and morphisms as F. It is easy to see that F^{op} is indeed a functor.

A functor $\mathsf{E} : \mathcal{C} \leftarrow \mathcal{C}$, which maps a category on itself, is called an **endofunctor**.

A functor $\mathsf{G} : \mathcal{C} \leftarrow \mathcal{D}^{\mathrm{op}}$, *i.e.* a covariant functor from $\mathcal{D}^{\mathrm{op}}$, is called a **contravariant functor**[4] from $\mathcal{D}$ (*not* $\mathcal{D}^{\mathrm{op}}$). We define the **identity functor**

$$\mathsf{Id} : \begin{cases} \mathcal{C} & \leftarrow & \mathcal{C} \\ A & \leftarrow\!\shortmid & A & \forall\, A \in \operatorname{Ob}\mathcal{C} \\ f & \leftarrow\!\shortmid & f & \forall\, f \in \operatorname{Mor}\mathcal{C} \end{cases}$$

and for every $A \in \operatorname{Ob}\mathcal{C}$ the **constant functor**

$$\mathsf{K}_A : \begin{cases} \mathcal{C} & \leftarrow & \mathcal{D} \\ A & \leftarrow\!\shortmid & B & \forall\, B \in \operatorname{Ob}\mathcal{D} \\ id_A & \leftarrow\!\shortmid & f & \forall\, f \in \operatorname{Mor}\mathcal{D}. \end{cases}$$

where it is easy to see that these are indeed functors.

For simplicity we avoid notations like $\mathsf{Id}_{\mathcal{C}}$ or $\mathsf{K}_A^{\mathcal{C},\mathcal{D}}$, if the connection to the categories is obvious. $*$

C.1.9 Corollary. The constant functors absorbs other functors in compositions. More precisely: Let $\mathcal{C}$ be a category and $A \in \operatorname{Ob}\mathcal{C}$. For every functors F from $\mathcal{C}$ and every functors G to $\mathcal{C}$ holds

(i) $\mathsf{F} \cdot \mathsf{K}_A = \mathsf{K}_{\mathsf{F}A}$ and

(ii) $\mathsf{K}_A \cdot \mathsf{G} = \mathsf{K}_A$.

Proof. Immediately using the definition of the constant functor and for (i) the identity functor axiom. ■

C.1.10 Lemma (functors preserve isomorphisms). Let $\mathcal{C}$ and $\mathcal{D}$ be categories, $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{D}$ a functor, and $f \in \operatorname{Mor}\mathcal{D}$ an isomorphism. Then $\mathsf{F}f$ is an isomorphism in $\mathcal{C}$.

Proof. Using the definitions it is easy to show that $\mathsf{F}(f^{-1}) = (\mathsf{F}f)^{-1}$. ■

C.1.11 Definition ((full) subcategory). Let $\mathcal{C}$ and $\mathcal{D}$ be categories. The category $\mathcal{D}$ is called a **subcategory** of $\mathcal{C}$ provided that

$$\operatorname{Ob}\mathcal{D} \subseteq \operatorname{Ob}\mathcal{C} \text{ and } \operatorname{Mor}\mathcal{D} \subseteq \operatorname{Mor}\mathcal{C}$$

holds and the $\mathcal{D}$-composition is the restriction of the $\mathcal{C}$-composition. If in addition

$$\forall\, A, B \in \operatorname{Ob}\mathcal{D} ::\ \mathcal{D}(A,B) = \mathcal{C}(A,B)$$

is true, then $\mathcal{D}$ is called a **full** subcategory of $\mathcal{C}$. The functor

$$\mathsf{E} : \begin{cases} \mathcal{C} & \leftarrow & \mathcal{D} \\ A & \leftarrow\!\shortmid & A & \forall\, A \in \operatorname{Ob}\mathcal{D} \\ f & \leftarrow\!\shortmid & f & \forall\, f \in \operatorname{Mor}\mathcal{D} \end{cases}$$

is called the **canonical embedding** of the subcategory $\mathcal{D}$ in $\mathcal{C}$. $*$

C.1.12 Definition (product-category). Let I be a set and $(\mathcal{C}_i)_{i\in I}$ be a family of categories. We define the **product-category** $\prod_{i\in I}\mathcal{C}_i$ by

$$\operatorname{Ob}\big(\prod_{i\in I}\mathcal{C}_i\big) = \Big\{(A_i)_{i\in I} \ \Big|\ \forall\, i \in I ::\ A_i \in \operatorname{Ob}\mathcal{C}_i\Big\}$$

and

$$\prod_{i\in I}\mathcal{C}_i\big((A_i)_{i\in I}, (B_i)_{i\in I}\big) = \Big\{(f_i)_{i\in I} \ \Big|\ \forall\, i \in I ::\ f_i : A_i \xleftarrow[\mathcal{C}_i]{} B_i\Big\}$$

[4]According to our definition any functor is covariant. We use the notion 'contravariant functor' to emphasize that the functor maps from the dual category of some given category. Many authors use a different definition.

with pointwise identities and composition. The functors $(\mathsf{P}_j)_{j\in I}$ defined by $\forall\, j \in I$:

$$\mathsf{P}_j : \begin{cases} \mathcal{C}_j \leftarrow \prod_{i\in I} \mathcal{C}_i \\ A_j \hookleftarrow (A_i)_{i\in I} & \forall\,(A_i)_{i\in I} \in \mathrm{Ob}(\prod_{i\in I}\mathcal{C}_i) \\ f_j \hookleftarrow (f_i)_{i\in I} & \forall\,(f_i)_{i\in I} \in \mathrm{Mor}(\prod_{i\in I}\mathcal{C}_i), \end{cases}$$

are called the **projection-functors**. If I is finite, *e.g.* $I = \{1,\dots,n\}$, we write $\mathcal{C}_1 \times \cdots \times \mathcal{C}_n = \prod_{i=1}^{n}\mathcal{C}_i = \prod_{i\in I}\mathcal{C}_i$. For a category $\mathcal{C}$ we define $\mathcal{C}^I = \prod_{i\in I}\mathcal{C}$ and if I is finite, *e.g.* $I = \{1,\dots,n\}$, we write $\mathcal{C}^n = \mathcal{C}^I$. $\ast$

C.1.13 Definition (quasi-category). A **quasi-category** $\mathcal{C}$ is defined by the same axioms as a category except that the order of all involved classes is incremented, *i.e.* the collection of objects and morphisms are conglomerates. $\ast$

C.1.14 Note. We can easily lift every notion or predicate from category-theory to quasi-category-theory just by incrementing the order of all involved classes. For every true statement on categories the lifted statement on quasi-categories is also true. For every notion N we will denote the lifted notion by **quasi-**N (*e.g.* ‘quasi-set’ = ‘class’). If the notion N does not depend on the order of any class we can obviously omit the prefix ‘quasi-’ (*e.g.* ‘quasi-object’ = ‘object’). The reason for the distinction of quasi-categories from categories is that the category of all categories (with functors as morphisms) does not exist, since the class of all classes does not exist. But using quasi-categories we can construct the following: $\ast$

C.1.15 Definition (quasi-categories of all classes and of all categories). We denote the quasi-category of all classes with all functions as morphisms by $\boldsymbol{CLASS}$ and the quasi-category of all categories with all functors as morphisms by $\boldsymbol{CAT}$. $\ast$

C.1.16 Note. The opposite notion of ‘quasi-’ is **small**, *e.g.* ‘small class = set’, ‘small conglomerate = class’, and ‘small quasi-category = category’.

C.2 Natural transformations

C.2.1 Definition ((natural) transformation). Let $\mathcal{C}$ and $\mathcal{D}$ be categories and $\mathsf{F}, \mathsf{G} : \mathcal{C} \leftarrow \mathcal{D}$ be functors. A function

$$\tau = (\tau_A)_{A\in\mathrm{Ob}\,\mathcal{D}} \in (\mathrm{Mor}\,\mathcal{C})^{\mathrm{Ob}\,\mathcal{D}}$$

with

$$\forall\, A \in \mathrm{Ob}\,\mathcal{D} ::\ \tau_A : \mathsf{F}A \underset{\mathcal{C}}{\longleftarrow} \mathsf{G}A$$

is called a **transformation** to F from G. If in addition τ satisfies the so called **naturalness condition**

$$\frac{h : A \underset{\mathcal{D}}{\longleftarrow} B}{\mathsf{F}h \cdot \tau_B = \tau_A \cdot \mathsf{G}h},$$

then it is called a **natural** transformation to F from G, and we write:

$$\tau : \mathsf{F} \stackrel{\cdot}{\leftarrow} \mathsf{G}.$$

$\ast$

C.2.2 Example (natural transformation). Let $\mathcal{C}$ be a category. The identity id is a natural transformation:

$$id = (id_A)_{A\in\mathrm{Ob}\,\mathcal{C}} : \mathsf{Id} \stackrel{\cdot}{\leftarrow} \mathsf{Id}$$

$\ast$

C.2.3 Definition and Lemma (horizontal composition of (natural) transformations). Let $\mathcal{C}$ and $\mathcal{D}$ be categories and $\mathsf{F}, \mathsf{G}, \mathsf{H} : \mathcal{C} \leftarrow \mathcal{D}$ be functors and $\sigma : \mathsf{F} \leftarrow \mathsf{G}$ and $\tau : \mathsf{G} \leftarrow \mathsf{H}$ be transformations. The composition of σ and τ is the transformation which is defined pointwise by

$$\forall\, A \in \mathrm{Ob}\,\mathcal{D} :: \ (\sigma \cdot \tau)_A = \sigma_A \cdot \tau_A$$

The composition of natural transformations is a natural transformation, *i.e.*:

$$\frac{\sigma : \mathsf{F} \dot{\leftarrow} \mathsf{G} \qquad \tau : \mathsf{G} \dot{\leftarrow} \mathsf{H}}{\sigma \cdot \tau : \mathsf{F} \dot{\leftarrow} \mathsf{H}}$$

Proof. Immediately with Definition C.2.1. ■

This composition of natural transformations is often called **horizontal composition**. We will see *vertical composition* of natural transformations in Definition and Lemma C.2.7.

C.2.4 Definition (composition of morphisms and natural transformations). Let $\mathcal{C}$ and $\mathcal{D}$ be categories and $\mathsf{F}, \mathsf{G} : \mathcal{C} \leftarrow \mathcal{D}$ be functors. For every $\mathcal{C}$-morphism f, we define the composition of $\mathsf{F}f$ with a natural transformation by

(i) $\forall\, \sigma : \mathsf{G} \dot{\leftarrow} \mathsf{F} :: \ \sigma \cdot \mathsf{F}f = \sigma_{\mathrm{cod}\, f} \cdot \mathsf{F}f$,

(ii) $\forall\, \tau : \mathsf{F} \dot{\leftarrow} \mathsf{G} :: \ \mathsf{F}f \cdot \tau = \mathsf{F}f \cdot \tau_{\mathrm{dom}\, f}$. $*$

$*$

C.2.5 Definition (functor-category). Let $\mathcal{C}$ and $\mathcal{D}$ be categories. We define the quasi-category $\mathcal{C}^{\mathcal{D}}$ by

$$\mathrm{Ob}\,\mathcal{C}^{\mathcal{D}} = \{\mathsf{F} \mid \mathsf{F} : \mathcal{C} \leftarrow \mathcal{D}\}$$

and for every $\mathsf{F}, \mathsf{G} \in \mathrm{Ob}\,\mathcal{C}^{\mathcal{D}}$:

$$\mathcal{C}^{\mathcal{D}}(\mathsf{F}, \mathsf{G}) = \{\tau \mid \tau : \mathsf{F} \dot{\leftarrow} \mathsf{G}\}$$

with the composition of natural transformations and for every $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{D}$ the identity $id_{\mathsf{F}} = (id_{\mathsf{F}A})_{A \in \mathrm{Ob}\,\mathcal{D}} : \mathsf{F} \dot{\leftarrow} \mathsf{F}$.

$*$

C.2.6 Definition and Lemma (functors preserve naturalness). Let $\mathcal{C}$, $\mathcal{D}$, and $\mathcal{E}$ be categories and $\mathsf{F}, \mathsf{G} : \mathcal{C} \leftarrow \mathcal{D}$ be functors. For every natural transformation

$$\tau : \mathsf{F} \dot{\leftarrow} \mathsf{G}$$

the following statements hold:

(i) The composition $\mathsf{H}\tau$ of a functor $\mathsf{H} : \mathcal{E} \leftarrow \mathcal{C}$ and τ defined by

$$\forall\, A \in \mathrm{Ob}\,\mathcal{D} :: \ (\mathsf{H}\tau)_A = \mathsf{H}(\tau_A)$$

is a natural transformation

$$\mathsf{H}\tau : \mathsf{H} \cdot \mathsf{F} \dot{\leftarrow} \mathsf{H} \cdot \mathsf{G}.$$

(ii) The composition $\tau\mathsf{H}$ of τ and a functor $\mathsf{H} : \mathcal{D} \leftarrow \mathcal{E}$ defined by

$$\forall\, A \in \mathrm{Ob}\,\mathcal{E} :: \ (\tau\mathsf{H})_A = \tau_{\mathsf{H}A}$$

is a natural transformation

$$\tau\mathsf{H} : \mathsf{F} \cdot \mathsf{H} \dot{\leftarrow} \mathsf{G} \cdot \mathsf{H}.$$

Proof. Immediately by the Definitions C.1.8 and C.2.1. ■

C.2.7 Definition and Lemma (vertical composition of natural transformations). Let $\mathcal{C}$, $\mathcal{D}$, and $\mathcal{E}$ be categories, and

$$\mathcal{C} \xleftarrow{\mathsf{F},\mathsf{F}'} \mathcal{D} \xleftarrow{\mathsf{G},\mathsf{G}'} \mathcal{E}$$

be functors, and $\sigma : \mathsf{F} \dot{\leftarrow} \mathsf{F}'$ and $\tau : \mathsf{G} \dot{\leftarrow} \mathsf{G}'$ be natural transformations. It holds $\sigma\mathsf{G} \cdot \mathsf{F}'\tau = \mathsf{F}\tau \cdot \sigma\mathsf{G}'$. We use this to define the **vertical composition** (or **Godement product**) $\sigma * \tau$ of σ and τ by

$$\sigma * \tau = \sigma\mathsf{G} \cdot \mathsf{F}'\tau = \mathsf{F}\tau \cdot \sigma\mathsf{G}'$$

which is a natural transformation $\sigma * \tau : \mathsf{F} \cdot \mathsf{G} \dot{\leftarrow} \mathsf{F}' \cdot \mathsf{G}'$, *i.e.* the following diagram of natural transformations commutes:

$$\begin{tikzcd}
 & \mathsf{F}' \cdot \mathsf{G} \arrow[dl, "\sigma\mathsf{G}"'] & \\
\mathsf{F} \cdot \mathsf{G} & & \mathsf{F}' \cdot \mathsf{G}' \arrow[ul, "\mathsf{F}'\tau"'] \arrow[ll, "\sigma * \tau"'] \arrow[dl, "\sigma\mathsf{G}'"] \\
 & \mathsf{F} \cdot \mathsf{G}' \arrow[ul, "\mathsf{F}\tau"] &
\end{tikzcd}$$

Notice that $\sigma * id_{\mathsf{G}} = \sigma\mathsf{G}$ and $id_{\mathsf{F}} * \tau = \mathsf{F}\tau$.

Proof. Let $A \in \mathrm{Ob}\,\mathcal{E}$.

$$\begin{array}{ll}
 & (\sigma\mathsf{G} \cdot \mathsf{F}'\tau)_A \\
= & \{ \text{Definition and Lemma C.2.3 and Definition and Lemma C.2.6} \} \\
 & \sigma_{\mathsf{G}A} \cdot \mathsf{F}'\tau_A \\
= & \{ \text{naturalness of } \sigma \} \\
 & \mathsf{F}\tau_A \cdot \sigma_{\mathsf{G}'A} \\
= & \{ \text{Definition and Lemma C.2.3 and Definition and Lemma C.2.6} \} \\
 & (\mathsf{F}\tau \cdot \sigma\mathsf{G}')_A
\end{array}$$

■

C.2.8 Lemma (vertical composition versus horizontal composition). Let $\mathcal{C}$, $\mathcal{D}$, and $\mathcal{E}$ be categories, and

$$\mathcal{C} \xleftarrow{\mathsf{F},\mathsf{F}',\mathsf{F}'} \mathcal{D} \xleftarrow{\mathsf{G},\mathsf{G}',\mathsf{G}'} \mathcal{E}$$

be functors, and

$$\mathsf{F} \xleftarrow{\sigma} \mathsf{F}' \xleftarrow{\sigma'} \mathsf{F}'' \quad \text{and} \quad \mathsf{G} \xleftarrow{\tau} \mathsf{G}' \xleftarrow{\tau'} \mathsf{G}''$$

be natural transformations. It holds:

$$(\sigma * \tau) \cdot (\sigma' * \tau') = (\sigma \cdot \sigma') * (\tau \cdot \tau').$$

Proof. Let $A \in \mathrm{Ob}\,\boldsymbol{\mathcal{E}}$.

$$\begin{aligned}
&\big((\sigma * \tau) \cdot (\sigma' * \tau')\big)_A \\
= &\quad \{ \text{Definition and Lemma C.2.7 and Definition and Lemma C.2.6} \} \\
&\sigma_{\mathsf{G}A} \cdot \mathsf{F}'\tau_A \cdot \mathsf{F}'\tau'_A \cdot \sigma'_{\mathsf{G}''A} \\
= &\quad \{ \mathsf{F}' \text{ is a functor} \} \\
&\sigma_{\mathsf{G}A} \cdot \mathsf{F}'(\tau_A \cdot \tau'_A) \cdot \sigma'_{\mathsf{G}''A} \\
= &\quad \{ \text{naturalness of } \sigma' \} \\
&\sigma_{\mathsf{G}A} \cdot \sigma'_{\mathsf{G}A} \cdot \mathsf{F}''(\tau_A \cdot \tau'_A) \\
= &\quad \{ \text{Definition and Lemma C.2.7 and Definition and Lemma C.2.6} \} \\
&\big((\sigma \cdot \sigma') * (\tau \cdot \tau')\big)_A
\end{aligned}$$

■

C.2.9 Lemma (vertical composition is associative). Let $\boldsymbol{\mathcal{C}}$, $\boldsymbol{\mathcal{D}}$, $\boldsymbol{\mathcal{E}}$,and $\boldsymbol{\mathcal{F}}$ be categories, and

$$\boldsymbol{\mathcal{C}} \xleftarrow{\mathsf{F},\mathsf{F}'} \boldsymbol{\mathcal{D}} \xleftarrow{\mathsf{G},\mathsf{G}'} \boldsymbol{\mathcal{E}} \xleftarrow{\mathsf{H},\mathsf{H}'} \boldsymbol{\mathcal{F}}$$

be functors, and

$$\mathsf{F} \xleftarrow{\sigma} \mathsf{F}' \qquad \mathsf{G} \xleftarrow{\tau} \mathsf{G}' \qquad \mathsf{H} \xleftarrow{\varrho} \mathsf{H}'$$

be natural transformations. It holds:

$$(\sigma * \tau) * \varrho = \sigma * (\tau * \varrho).$$

Proof. Let $A \in \mathrm{Ob}\,\boldsymbol{\mathcal{F}}$.

$$\begin{aligned}
&\big((\sigma * \tau) * \varrho\big)_A \\
= &\quad \{ \text{Definition and Lemma C.2.7 and Definition and Lemma C.2.6} \} \\
&(\sigma * \tau)_{\mathsf{H}A} \cdot (\mathsf{F}' \cdot \mathsf{G}')\varrho_A \\
= &\quad \{ \text{Definition and Lemma C.2.7 and Definition and Lemma C.2.6} \} \\
&\sigma_{(\mathsf{G}\cdot\mathsf{H})A} \cdot \mathsf{F}'\tau_{\mathsf{H}A} \cdot (\mathsf{F}' \cdot \mathsf{G}')\varrho_A \\
= &\quad \{ \mathsf{F}' \text{ is a functor} \} \\
&\sigma_{(\mathsf{G}\cdot\mathsf{H})A} \cdot \mathsf{F}'(\tau_{\mathsf{H}A} \cdot \mathsf{G}'\varrho_A) \\
= &\quad \{ \text{Definition and Lemma C.2.7 and Definition and Lemma C.2.6} \} \\
&\sigma_{(\mathsf{G}\cdot\mathsf{H})A} \cdot \mathsf{F}'(\tau * \varrho)_A \\
= &\quad \{ \text{Definition and Lemma C.2.7 and Definition and Lemma C.2.6} \} \\
&\big(\sigma * (\tau * \varrho)\big)_A
\end{aligned}$$

■

C.3 Initial and final objects

C.3.1 Definition (initial/final object). Let $\boldsymbol{\mathcal{C}}$ be a category. An object $0 \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$ is called an **initial object** of $\boldsymbol{\mathcal{C}}$, if for every $\boldsymbol{\mathcal{C}}$-object A there exists a unique $\boldsymbol{\mathcal{C}}$-morphism to A from 0, *i.e.*:

$$\forall\, A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}} ::\ \#\big(\boldsymbol{\mathcal{C}}(A, 0)\big) = 1.$$

or equivalently

$$\forall\, A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}} ::\ \exists!\, f \in \mathrm{Mor}\,\boldsymbol{\mathcal{C}} ::\ f : A \leftarrow 0$$

This formula has the form '$\forall \cdots \exists! \cdots$' and can thus be used to define the function

$$\text{¡}_{(\,.\,)\leftarrow 0} : \mathrm{Mor}\,\mathcal{C} \leftarrow \mathrm{Ob}\,\mathcal{C}$$

by

$$\forall\, f \in \mathrm{Mor}\,\mathcal{C} ::\ f = \text{¡}_{A\leftarrow 0} \iff f : A \leftarrow 0. \tag{UP}$$

The defining property (UP) is called the **universal property**. The function ¡ is called the **comediator** of 0. The morphism $\text{¡}_{A\leftarrow 0}$ is called the **unique mediating morphism to** A from the initial object 0. If the connection to the initial object $0 \in \mathrm{Ob}\,\mathcal{C}$ is obvious, we simply write ¡_A for $\text{¡}_{A\leftarrow 0}$.

Dually an object $1 \in \mathrm{Ob}\,\mathcal{C}$ with

$$\forall\, A \in \mathrm{Ob}\,\mathcal{C} ::\ \#\big(\mathcal{C}(1, A)\big) = 1$$

is called **final object** (and sometimes **terminal object**) of $\mathcal{C}$ and the function

$$!_{(\,.\,)\rightarrow 1} : \mathrm{Mor}\,\mathcal{C} \leftarrow \mathrm{Ob}\,\mathcal{C}$$

with

$$\forall\, f \in \mathrm{Mor}\,\mathcal{C} ::\ f = !_{A\rightarrow 1} \iff f : 1 \leftarrow A \tag{UP$^{\mathrm{op}}$}$$

is called the **mediator** of 1. The morphism $!_{A\rightarrow 1}$ is called the **unique mediating morphism from** A to the final object 1. If the connection to the final object $1 \in \mathrm{Ob}\,\mathcal{C}$ is obvious, we simply write $!_A$ for $!_{A\rightarrow 1}$.
$*$

C.3.2 Example (initial and final object). In the category $\mathcal{Set}$ the empty set $\emptyset$ is an initial object and every singleton set is a final object. $*$

C.3.3 Lemma (essential uniqueness of initial objects). Let $\mathcal{C}$ be a category. Every two initial objects of $\mathcal{C}$ are isomorphic.

Proof. Let 0 and $0'$ be initial objects with comediators $\text{¡}_{(\,.\,)\leftarrow 0}$ and $\text{¡}_{(\,.\,)\leftarrow 0'}$, respectively.

$$\begin{aligned}
\Longrightarrow\quad & \{\ \text{(composition)}\ \} \\
& \text{¡}_{0\leftarrow 0'} \cdot \text{¡}_{0'\leftarrow 0} : 0 \leftarrow 0 \\
\Longrightarrow\quad & \{\ \text{(UP), (typing) \& (identity)}\ \} \\
& \text{¡}_{0\leftarrow 0'} \cdot \text{¡}_{0'\leftarrow 0} = \text{¡}_{0\leftarrow 0} = id_0 .
\end{aligned}$$

Dually follows: $\text{¡}_{0'\leftarrow 0} \cdot \text{¡}_{0\leftarrow 0'} = id_{0'}$. Hence $\text{¡}_{0\leftarrow 0'} : 0 \leftarrow 0'$ is an isomorphism and thus $0 \cong 0'$.

Since $\#\big(\mathcal{C}(0, 0')\big) = 1$ the morphism $\text{¡}_{0\leftarrow 0'}$ is even the only isomorphism to 0 from $0'$. ∎

C.3.4 Lemma (laws for initial/final objects). Let $\mathcal{C}$ be a category with initial object $0 \in \mathrm{Ob}\,\mathcal{C}$. Then the laws in Table 4 hold.

Laws for comediators	
UP	$f = \text{¡}_A \iff f : A \leftarrow 0$
reflection	$\text{¡}_0 = id_0$
fusion	$f : A \leftarrow B \implies f \cdot \text{¡}_B = \text{¡}_A$
where $f \in \mathrm{Mor}\,\mathcal{C}$ and $A, B \in \mathrm{Ob}\,\mathcal{C}$	

Table 4: Laws for comediators

And thus the dual laws in Table 5 hold for a category $\mathcal{C}$ with a final object $1 \in \mathrm{Ob}\,\mathcal{C}$.

Proof. The reflection law follows from the first part of the proof of Lemma C.3.3. The fusion law can be proven similarly: Because of $f \cdot \text{¡}_B : A \leftarrow 0$ it is obvious from the (UP) that $f \cdot \text{¡}_B = \text{¡}_A$. ∎

Laws for mediators	
UP	$f = !_A \iff f : 1 \leftarrow A$
reflection	$!_1 = id_1$
fusion	$f : A \leftarrow B \implies !_A \cdot f = !_B$
where $f \in \mathrm{Mor}\,\mathcal{C}$ and $A, B \in \mathrm{Ob}\,\mathcal{C}$	

Table 5: Laws for mediators

C.3.5 Corollary (naturalness of (co-)mediators). Let $\mathcal{C}$ be a category with initial object $0 \in \mathrm{Ob}\,\mathcal{C}$ and $\mathsf{K}_0 : \mathcal{C} \leftarrow \mathcal{C}$ be the constant functor to 0. The comediator $¡$ is a natural transformation, *i.e.*:

$$¡ : \mathsf{Id} \stackrel{.}{\leftarrow} \mathsf{K}_0.$$

Proof. Since for every $A \in \mathrm{Ob}\,\mathcal{C}$ holds $¡_A : A \leftarrow 0$, the comediator is a transformation to Id from K_0. It is also natural, because its naturalness condition (*cf.* Definition C.2.1) is equivalent to the fusion law (Table 4). ■

C.4 Products and coproducts

C.4.1 Definition and Lemma (product). Let $\mathcal{C}$ be a category, I a set, $(A_i)_{i\in I} \in \mathrm{Ob}\,\mathcal{C}^I$, and $P \in \mathrm{Ob}\,\mathcal{C}$. An I-family $(A_i \xleftarrow{\pi_i} P)_{i\in I}$ of $\mathcal{C}$-morphisms is called *a* **product** of $(A_i)_{i\in I}$ provided that for every object $B \in \mathrm{Ob}\,\mathcal{C}$ and every I-family $(A_i \xleftarrow{f_i} B)_{i\in I}$ of $\mathcal{C}$-morphisms there exists a unique morphism

$$\langle f_i \rangle_{i\in I} : P \leftarrow B,$$

which is called **pairing** such that the diagram in Figure 2 commutes.

$$\forall j \in I :: \quad \begin{tikzcd} P \arrow[r, "\pi_j"] & A_j \\ B \arrow[u, "\langle f_i \rangle_{i\in I}"'] \arrow[ur, "f_j"'] & \end{tikzcd}$$

Figure 2: Product UP

For every $i \in I$ the morphism $\pi_i : A_i \leftarrow P$ is called the **projection** onto A_i from P. It is also common to say that the object P *itself* is a **product** of $(A_i)_{i\in I}$ with **projections** $(A_i \xleftarrow{\pi_i} P)_{i\in I}$. If there exists a product of $(A_i)_{i\in I} \in \mathrm{Ob}\,\mathcal{C}^I$ and the connection to the respective projections is obvious or unimportant, then we denote the product-object P by $\prod_{i\in I} A_i$. Notice that the object $\prod_{i\in I} A_i$ depends on the projections $(\pi_i)_{i\in I}$, which is not obvious from the notation. If I is finite, then $\prod_{i\in I} A_i$ is called a **finite product**. For finite products with *e.g.* $I = \{1, \ldots, n\}$, we write $A_1 \times \cdots \times A_n = \prod_{i\in I} A_i$ and $\langle f_1, \ldots, f_n \rangle = \langle f_i \rangle_{i\in I}$. Notice that an empty product (*i.e.* $I = \emptyset$) is a final object.

We say that $\mathcal{C}$ **has (finite) products**, if for every (finite) set I and every $(A_i)_{i\in I} \in \mathrm{Ob}\,\mathcal{C}^I$ there exists a product $\prod_{i\in I} A_i$ in $\mathcal{C}$. It is easy to see that we find the laws in Table 6 in analogy to the laws in Table 5 from Lemma C.3.4.

Laws for products	
UP	$h = \langle f_i \rangle_{i \in I} \iff \forall\, i \in I :: \ \pi_i \cdot h = f_i$
reflection	$\langle \pi_i \rangle_{i \in I} = id_{\prod_{i \in I} A_i}$
fusion	$\langle f_i \rangle_{i \in I} \cdot h = \langle f_i \cdot h \rangle_{i \in I}$
cancelation	$\forall\, j \in I :: \ \pi_j \cdot \langle f_i \rangle_{i \in I} = f_j$
where $h \in \operatorname{Mor} \mathcal{C}$ and $\forall\, j \in I :: \ f_j : A_j \leftarrow B$	

Table 6: Laws for products

Proof. The UP is equivalent to the definition of the product. The remaining laws follow in analogy to Lemma C.3.3 and Lemma C.3.4. ∎

C.4.2 Lemma (products in $\mathcal{S}et$). The category $\mathcal{S}et$ has products.

Proof. Let I be a set and for every $i \in I$ let A_i be a set. We will show that

$$\prod_{i \in I} A_i = \big\{ (a_i)_{i \in I} \ \big| \ \forall\, i \in I :: \ a_i \in A_i \big\},$$

is a product of $(A_i)_{i \in I}$ with projections

$$\forall\, j \in I :: \ \pi_j : A_j \leftarrow \prod_{i \in I} A_i : a_j \leftarrow\!\shortmid (a_i)_{i \in I}.$$

Therefore it is sufficient to show that for every set B and every $(A_i \xleftarrow{f_i} B)_{i \in I}$

$$\langle f_i \rangle_{i \in I} : \prod_{i \in I} A_i \leftarrow B : (f_i b)_{i \in I} \leftarrow\!\shortmid b$$

is the respective pairing by verifying the UP of the product (Table 6). ∎

C.4.3 Definition and Lemma (coproduct). The dual notion to ‘product’ is the notion of ’coproduct’: Let $\mathcal{C}$ be a category, I a set, $(A_i)_{i \in I} \in \operatorname{Ob} \mathcal{C}^I$, and $C \in \operatorname{Ob} \mathcal{C}$. An I-family $(C \xleftarrow{\iota_i} A_i)_{i \in I}$ of $\mathcal{C}$-morphisms is called *a* **coproduct** (or **sum**) of $(A_i)_{i \in I}$ provided that for every object $B \in \operatorname{Ob} \mathcal{C}$ and every I-family $(B \xleftarrow{f_i} A_i)_{i \in I}$ of $\mathcal{C}$-morphisms there exists a unique morphism

$$[f_i]_{i \in I} : B \leftarrow C,$$

which is called **copairing** (or **case**) such that the diagram in Figure 3 commutes:
For every $i \in I$ the morphism $\iota_i : C \leftarrow A_i$ is called the **injection** of A_i into C. It is also common to say that the object C *itself* is a **coproduct** of $(A_i)_{i \in I}$ with **injections** $(C \xleftarrow{\iota_i} A_i)_{i \in I}$. If there exists a coproduct of $(A_i)_{i \in I} \in \operatorname{Ob} \mathcal{C}^I$ and the connection to the respective injections is obvious or unimportant, then we denote the coproduct-object by $\coprod_{i \in I} A_i = C$. Notice that the object $\coprod_{i \in I} A_i$ depends on the injections $(\iota_i)_{i \in I}$, which is not obvious from the notation. If I is finite, then $\coprod_{i \in I} A_i$ is called a **finite coproduct**. For finite coproducts with *e.g.* $I = \{1, \dots, n\}$, we write $A_1 + \cdots + A_n = \coprod_{i \in I} A_i$ and $[f_1, \dots, f_n] = [f_i]_{i \in I}$. Notice that an empty coproduct (*i.e.* $I = \emptyset$) is an initial object.

We say that $\mathcal{C}$ **has (finite) coproducts**, if for every (finite) set I and every $(A_i)_{i \in I} \in \operatorname{Ob} \mathcal{C}^I$ there exists a coproduct $\coprod_{i \in I} A_i$ in $\mathcal{C}$. We obtain the laws in Table 7 which are dual to the laws in Table 6.

Proof. Dually to Definition and Lemma C.4.1. ∎

C.4.4 Lemma (coproducts in $\mathcal{S}et$). The category $\mathcal{S}et$ has coproducts.

$$\forall\, j \in I::\quad \begin{tikzcd} C \arrow[d, "{[f_i]_{i\in I}}"] & A_j \arrow[l, "\iota_j"'] \arrow[dl, "f_j"] \\ B & \end{tikzcd}$$

Figure 3: Coproduct UP

Laws for coproducts	
UP	$h = [f_i]_{i\in I} \iff \forall\, i \in I::\ h \cdot \iota_i = f_i$
reflection	$[\iota_i]_{i\in I} = id_{\coprod_{i\in I} A_i}$
fusion	$h \cdot [f_i]_{i\in I} = [h \cdot f_i]_{i\in I}$
cancelation	$\forall\, j \in I::\ [f_i]_{i\in I} \cdot \iota_j = f_j$
where $h \in \operatorname{Mor} \mathcal{C}$ and $\forall\, j \in I::\ f_j : B \leftarrow A_j$	

Table 7: Laws for coproducts

Proof. Let I be a set and for every $i \in I$ let A_i be a set. We will show that

$$\coprod_{i\in I} A_i = \bigcup_{i\in I} \{(i,a) \mid a \in A_i\},$$

is a coproduct of $(A_i)_{i\in I}$ with inclusions

$$\forall\, j \in I::\ \iota_j : \coprod_{i\in I} A_i \leftarrow A_j : (j,a) \leftarrow\!\shortmid a.$$

Therefore it is sufficient to show that for every set B and every $(B \xleftarrow{f_i} A_i)_{i\in I}$

$$[f_i]_{i\in I} : B \leftarrow \coprod_{i\in I} A_i : f_j a \leftarrow\!\shortmid (j,a)$$

is the respective copairing by verifying the UP of the coproduct (Table 7). ■

C.5 The functors for (co-)products

C.5.1 Definition and Corollary (product functor). Let $\mathcal{C}$ be a category which has finite products. The **product functor** is defined by

$$\prod : \left\{ \begin{array}{ccll} \mathcal{C} & \leftarrow & \mathcal{C}^n & \\ \prod_{i=1}^n A_i & \leftarrow\!\shortmid & (A_i)_{i=1}^n & \forall\, (A_i)_{i=1}^n \in \operatorname{Ob} \mathcal{C}^n \\ \langle f_i \cdot \pi_i \rangle_{i=1}^n & \leftarrow\!\shortmid & (f_i)_{i=1}^n & \forall\, (f_i)_{i=1}^n \in \operatorname{Mor} \mathcal{C}^n, \end{array} \right.$$

We also write $f_1 \times \cdots \times f_n = \prod_{i=1}^n f_i = \prod (f_i)_{i=1}^n$. We declare that the operation symbol $\times$ binds weaker (*i.e.* has lower precedence) than the composition operator $\cdot$. For every $f \in \operatorname{Mor} \mathcal{C}$ and $n \in \mathbb{N}_0$ we define $\prod^n : \mathcal{C} \leftarrow \mathcal{C}$ by $\prod^n f = \prod_{i=1}^n f$. For every $\mathcal{C}$-object A we also write $A^n = \prod^n A$. We will never use the latter notation for morphisms. The product functor $\prod$ satisfies the laws in Table 8.

Laws for product functors	
reflection	$\prod_{i=1}^n id_{A_i} = id_{\prod_{i=1}^n A_i}$
fusion (i)	$\prod_{i=1}^n f_i \cdot \langle g_i\rangle_{i=1}^n = \langle f_i \cdot g_i\rangle_{i=1}^n$
fusion (ii)	$\prod_{i=1}^n f_i \cdot \prod_{i=1}^n h_i = \prod_{i=1}^n (f_i \cdot h_i)$
cancelation	$\forall\, j :: \ \pi_j \cdot \prod_{i\in I}^n f_i = f_j \cdot \pi_j$
where $\forall\, j :: \ f_j, g_j, h_j \in \mathrm{Mor}\,\mathcal{C}$ such that $\forall\, i, j :: \ \mathrm{dom}\, g_i = \mathrm{dom}\, g_j$	

Table 8: Laws for product functors

The projections are natural transformations:

$$\pi_j : \mathsf{P}_j \overset{\cdot}{\leftarrow} \prod .$$

Proof. The laws follow straightforward from the definition of the product functor and the laws for products from Table 6. The cancelation law is the naturalness condition for $\pi_j : \mathsf{P}_j \overset{\cdot}{\leftarrow} \prod$. ■

C.5.2 Definition and Corollary (coproduct functor). This is dual to Definition and Corollary C.5.1 so we will have nothing to prove. Let $\mathcal{C}$ be a category which has finite coproducts. The **coproduct functor** is defined by

$$\coprod : \begin{cases} \mathcal{C} & \leftarrow & \mathcal{C}^n \\ \coprod_{i=1}^n A_i & \leftarrowtail & (A_i)_{i=1}^n \quad \forall\,(A_i)_{i=1}^n \in \mathrm{Ob}\,\mathcal{C}^n \\ [\iota_i \cdot f_i]_{i=1}^n & \leftarrowtail & (f_i)_{i=1}^n \quad \forall\,(f_i)_{i=1}^n \in \mathrm{Mor}\,\mathcal{C}^n . \end{cases}$$

We also write $f_1 + \cdots + f_n = \coprod_{i=1}^n f_i = \coprod (f_i)_{i=1}^n$. We declare that the operation symbol $+$ binds weaker (*i.e.* has lower precedence) than the product functor operator $\times$. The coproduct functor $\coprod$ satisfies the laws in Table 9.

Laws for coproduct functors	
reflection	$\coprod_{i=1}^n id_{A_i} = id_{\coprod_{i=1}^n A_i}$
fusion (i)	$[f_i]_{i=1}^n \cdot \coprod_{i=1}^n g_i = [f_i \cdot g_i]_{i=1}^n$
fusion (ii)	$\coprod_{i=1}^n g_i \cdot \coprod_{i=1}^n h_i = \coprod_{i=1}^n (g_i \cdot h_i)$
cancelation	$\forall\, j :: \ \coprod_{i=1}^n g_i \cdot \iota_j = \iota_j \cdot g_j$
where $\forall\, j :: \ f_j, g_j, h_j \in \mathrm{Mor}\,\mathcal{C}$ such that $\forall\, i, j :: \ \mathrm{cod}\, f_i = \mathrm{cod}\, f_j$	

Table 9: Laws for coproduct functors

The injections are natural transformations:

$$\iota_j : \coprod \overset{\cdot}{\leftarrow} \mathsf{P}_j .$$

$*$

C.5.3 Lemma ((co-)products in the functor category). Let $\mathcal{C}$ and $\mathcal{D}$ be categories such that $\mathcal{C}$ has (finite) (co-)products. Then $\mathcal{C}^{\mathcal{D}}$ has (finite) (co-)products.

Proof. Since products and coproducts are dual to each other, it is sufficient to prove the statement for products: Let I be a set and $(\mathsf{F}_i)_{i\in I} \in \mathrm{Ob}(\mathcal{C}^{\mathcal{D}})^I$ be an I-family of $\mathcal{C}^{\mathcal{D}}$-objects. The category $\mathcal{C}$ has products $(\mathsf{F}_i D \xleftarrow{(\pi_i)_D} \prod_{j\in I}(\mathsf{F}_j D))_{i\in I}$. We claim that $(\mathsf{F}_i \xleftarrow{\pi_i} \prod_{j\in I} \mathsf{F}_j)_{i\in I}$ is a product in $\mathcal{C}^{\mathcal{D}}$ where $\pi_i = \big((pi_i)_D\big)_{D\in \mathrm{Ob}\,\mathcal{D}}$ and $\forall\, f \in \mathrm{Mor}\,\mathcal{D} :: \ (\prod_{j\in I} \mathsf{F}_j) f = \prod_{j\in I}(\mathsf{F}_j f)$. The naturalness of π_i follows from cancelation in Table 8. The UP can easily be verified for the pairing $\forall\,(\tau_i)_{i\in I} \in \mathrm{Mor}(\mathcal{C}^{\mathcal{D}})^I :: \ \forall\, D \in \mathrm{Ob}\,\mathcal{D} :: \ (\langle \tau_i\rangle_{i\in I})_D = \langle (\tau_i)_D\rangle_{i\in I}$ by pointwise calculations in $\mathcal{C}$ for every $\mathcal{D}$-object. The dual statement is true for coproducts. ■

C.5.4 Note. Let $\mathcal{C}$ be a category which has (finite) products. From Lemma C.5.3 we know that the functor category $\mathcal{C}^{\mathcal{C}}$ has (finite) products also. Let I be a (finite) set. The functors $\prod : \mathcal{C}^{\mathcal{C}} \leftarrow (\mathcal{C}^{\mathcal{C}})^I$ and $\prod : \mathcal{C} \leftarrow \mathcal{C}^I$ are related by the equation

$$\forall\, (\mathsf{F}_i)_{i\in I} \in \mathrm{Ob}(\mathcal{C}^{\mathcal{C}})^I :: \ (\prod_{i\in I} \mathsf{F}_i)f = \prod_{i\in I}(\mathsf{F}_i f).$$

Let $n \in \mathbb{N}_0$. The functors $\prod^n : \mathcal{C}^{\mathcal{C}} \leftarrow \mathcal{C}^{\mathcal{C}}$ and $\prod^n : \mathcal{C} \leftarrow \mathcal{C}$ are related by the equation $\prod^n \mathsf{Id}_{\mathcal{C}} = \prod^n$. $*$

C.5.5 Lemma. Let $\mathcal{C}$ be a category with no empty hom-classes, *i.e.* $\forall\, A, B \in \mathrm{Ob}\,\mathcal{C} ::\ \mathcal{C}(A, B) \neq \emptyset$. If $\mathcal{C}$ has (co-)products, then the respective (co-)product functors are faithful.

Proof. Let I be a set, $(A_i)_{i\in I}, (B_i)_{i\in I} \in \mathrm{Ob}\,\mathcal{C}^I$, and $(f_i)_{i\in I}, (f'_i)_{i\in I} \in \mathcal{C}^I\big((A_i)_{i\in I}, (B_i)_{i\in I}\big)$, and let $j \in I$. Since $\forall\, i \in I ::\ \mathcal{C}(B_i, B_j) \neq \emptyset$ there exist for every $i \in I$ a $\mathcal{C}$-morphism $h_i : B_i \leftarrow B_j$. We choose $h_j = id_{B_j}$ and calculate:

$$\begin{aligned}
&\prod_{i\in I} f_i = \prod_{i\in I} f'_i\\
\Longrightarrow\quad &\pi_j \cdot \prod_{i\in I} f_i \cdot \langle h_i\rangle_{i\in I} = \pi_j \cdot \prod_{i\in I} f'_i \cdot \langle h_i\rangle_{i\in I}\\
\Longrightarrow\quad &\quad \{\ \text{cancelation for product(functors) Table 8 and Table 6}\ \}\\
&f_j = f'_j.
\end{aligned}$$

This is true for all $j \in I$ and thus $(f_j)_{j\in I} = (f'_j)_{j\in I}$ and hence $\prod_{i\in I}$ is faithful. The dual proposition holds for coproducts. ∎

C.6 Initial algebras and catamorphisms

C.6.1 Definition and Lemma (initial algebra, catamorphism). Let $\mathcal{C}$ be a category and $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{C}$ an endofunctor. The category $\mathcal{A}lg_{\mathcal{C}}\mathsf{F}$, defined by:

$$\mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\mathsf{F}) = \big\{\varphi \in \mathrm{Mor}\,\mathcal{C}\ \big|\ \exists\, A \in \mathrm{Ob}\,\mathcal{C} ::\ \varphi : A \leftarrow \mathsf{F}A\big\}$$

and for every $\varphi, \varphi' \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\mathsf{F})$ where $\varphi : A \leftarrow \mathsf{F}A$ and $\varphi' : B \leftarrow \mathsf{F}B$ with $A, B \in \mathrm{Ob}\,\mathcal{C}$:

$$\mathcal{A}lg_{\mathcal{C}}\mathsf{F}\,(\varphi, \varphi') = \big\{(\varphi, f, \varphi') \in \{\varphi\} \times \mathcal{C}(A, B) \times \{\varphi'\}\ \big|\ f \cdot \varphi' = \varphi \cdot \mathsf{F}f\big\}$$

and identity and composition for every $\varphi : A \leftarrow \mathsf{F}A$, $\varphi' : A' \leftarrow \mathsf{F}A'$, $\varphi'' : A'' \leftarrow \mathsf{F}A''$, and every $f : A \leftarrow A'$ defined by

$$\begin{aligned}
id_\varphi &= (\varphi, id_A, \varphi)\\
(\varphi, f, \varphi') \cdot (\varphi', g, \varphi'') &= (\varphi, f \cdot g, \varphi'')
\end{aligned}$$

is called the category of F**-algebras** over $\mathcal{C}$. The morphisms of this category are called F**-algebra homomorphisms**. We define the functor $|\,\boldsymbol{.}\,|_{\mathsf{F}} : \mathcal{C} \leftarrow \mathcal{A}lg_{\mathcal{C}}\mathsf{F}$ by

$$\forall\, \varphi \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\mathsf{F}) ::\ |\varphi|_{\mathsf{F}} = \mathrm{cod}_{\mathcal{C}}\,\varphi$$

and

$$\forall\, (\varphi, f, \varphi') \in \mathrm{Mor}(\mathcal{A}lg_{\mathcal{C}}\mathsf{F}) ::\ |(\varphi, f, \varphi')|_{\mathsf{F}} = f.$$

For every F-algebra $\varphi : A \leftarrow \mathsf{F}A$ we call the object $A \in \mathrm{Ob}\,\mathcal{C}$ the **carrier** of φ, thus the functor $|\,\boldsymbol{.}\,|_{\mathsf{F}}$ maps F-algebras to their carriers. If $\mathcal{A}lg_{\mathcal{C}}\mathsf{F}$ has an initial object, then we denote it by in_{F} and call it the **initial algebra** (or **constructor**) of F. The carrier of the initial algebra is denoted by $\mu\mathsf{F}$ and called the **least fixed point**[5] of F. The image of the uniquely mediating morphism

$$\forall\, \varphi \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\mathsf{F}) ::\ ¡_\varphi : \varphi \xleftarrow[\mathcal{A}lg_{\mathcal{C}}\mathsf{F}]{} in_{\mathsf{F}}$$

[5]The reason is that is suffices the fixpoint equation $\mu\mathsf{F} \cong \mathsf{F}$, because initial algebras are always isomorphisms as we will see later in Lemma C.6.4. There exists an equivalent definition for the *least fixed point* of F as (carrier of) the initial object in the *category of fixpoints of* F, *i.e.* the full subcategory of $\mathcal{A}lg_{\mathcal{C}}\mathsf{F}$ where the objects (F-algebras) are $\mathcal{C}$-isomorphisms.

under the functor $|\,\boldsymbol{.}\,|_{\mathsf{F}}$, *i.e.*

$$|\mathfrak{i}_\varphi|_{\mathsf{F}} : \mathrm{cod}_{\mathcal{C}}\,\varphi \xleftarrow[\mathcal{C}]{} \mu\mathsf{F}$$

is called the **catamorphism** generated by φ (w.r.t. F) (from Greek $\kappa\alpha\tau\alpha$, downwards) and is denoted by $(\!|\varphi|\!)_{\mathsf{F}}$, *i.e.* $\mathfrak{i}_\varphi = (\varphi, (\!|\varphi|\!)_{\mathsf{F}}, in_{\mathsf{F}})$. Note that initial algebra and catamorphism are determined uniquely up to F-algebra-isomorphism only, and hence $\mu\mathsf{F}$ is determined uniquely up to $\mathcal{C}$-isomorphism only. The laws in Table 10 are a consequence of the laws in Table 4.

Laws for catamorphisms	
UP	$f = (\!|\varphi|\!)_{\mathsf{F}} \iff f \cdot in_{\mathsf{F}} = \varphi \cdot \mathsf{F}f$
reflection	$(\!|in_{\mathsf{F}}|\!)_{\mathsf{F}} = id_{\mu\mathsf{F}}$
fusion	$f \cdot \varphi' = \varphi \cdot \mathsf{F}f \implies f \cdot (\!|\varphi'|\!)_{\mathsf{F}} = (\!|\varphi|\!)_{\mathsf{F}}$
where $f \in \mathrm{Mor}\,\mathcal{C}$ and $\varphi, \varphi' \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\mathsf{F})$	

Table 10: Laws for catamorphisms

The following equivalent definition is more descriptive: the $\mathcal{A}lg_{\mathcal{C}}\mathsf{F}$-object $in_{\mathsf{F}} : \mu\mathsf{F} \leftarrow \mathsf{F}(\mu\mathsf{F})$ is an initial algebra of F, provided that for every $A \in \mathrm{Ob}\,\mathcal{C}$ and every $\varphi : A \leftarrow \mathsf{F}A$ there exists a unique $\mathcal{C}$-morphism $(\!|\varphi|\!)_{\mathsf{F}}$ such that the square in the diagram in Figure 4 commutes.

$$\begin{tikzcd}
\mathcal{A}lg_{\mathcal{C}}\mathsf{F} \arrow[rrr, dashed, "|\,\boldsymbol{.}\,|_{\mathsf{F}}"] & & & \mathcal{C} \\
in_{\mathsf{F}} \arrow[dd] & & \mu\mathsf{F} \arrow[dd, "(\!|\varphi|\!)_{\mathsf{F}}"'] & \mathsf{F}(\mu\mathsf{F}) \arrow[l, "in_{\mathsf{F}}"'] \arrow[dd, "\mathsf{F}(\!|\varphi|\!)_{\mathsf{F}}"] \\
\mathfrak{i}_\varphi \arrow[r, dashed, mapsto, "|\,\boldsymbol{.}\,|_{\mathsf{F}}"] & {} & & \\
\varphi & & A & \mathsf{F}A \arrow[l, "\varphi"]
\end{tikzcd}$$

Figure 4: Catamorphism UP

C.6.2 Example (F-algebra). For the endofunctor $\mathsf{F} = \mathsf{Id} \times \mathsf{Id}$ in the category $\mathcal{S}et$, *i.e.*

$$\mathsf{F} : \begin{cases} \mathcal{S}et & \leftarrow \mathcal{S}et \\ A \times A & \leftarrow\!\shortmid\; A \qquad \forall\, A \in \mathrm{Ob}\,\mathcal{S}et \\ f \times f & \leftarrow\!\shortmid\; f \qquad \forall\, f \in \mathrm{Mor}\,\mathcal{S}et \end{cases}$$

we consider the category $\mathcal{A}lg_{\mathcal{S}et}\mathsf{F}$. An F-Algebra $\varphi \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{S}et}\mathsf{F})$ is a function

$$\varphi : A \xleftarrow[\mathcal{S}et]{} A \times A$$

with an $A \in \mathrm{Ob}\,\mathcal{S}et$. On the other hand an F-Algebra φ can be considered as a set $A = |\varphi|_{\mathsf{F}}$ together with a binary operation $\star$, where

$$\forall\, a, b \in A :: \; a \star b = \varphi(a, b).$$

Let $\varphi, \varphi' \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{S}et}\mathsf{F})$ where $\varphi : A \leftarrow A \times A$ and $\varphi' : A' \leftarrow A' \times A'$. The underlying function $|f|_{\mathsf{F}} : A \xleftarrow[\mathcal{S}et]{} A'$ of an $\mathcal{A}lg_{\mathcal{S}et}\mathsf{F}$-morphism $f = (\varphi, |f|_{\mathsf{F}}, \varphi') : \varphi \xleftarrow[\mathcal{A}lg_{\mathcal{S}et}\mathsf{F}]{} \varphi'$ has to satisfy the equation

$$|f|_{\mathsf{F}} \cdot \varphi' = \varphi \cdot \mathsf{F}|f|_{\mathsf{F}},$$

i.e.

$$\forall\, a, b \in A :: \; |f|_{\mathsf{F}}\big(\varphi'(a,b)\big) = \varphi(|f|_{\mathsf{F}}a, |f|_{\mathsf{F}}b).$$

If we denote the functions φ and φ' by the binary operations $\star$ and $\star'$, respectively, as above, we may write this equation as follows:

$$f(a \star' b) = fa \star fb.$$

This is the homomorphism property of f for algebras (with one binary function $\star$ or $\star'$, respectively). $*$

C.6.3 Lemma (initial algebras in $\mathcal{S}et$). Let $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ be a ranked alphabet and $\mathsf{F} = \coprod_{j=1}^{m}(\prod^{\mathrm{rank}_\Sigma \sigma_j} \mathsf{Id}) : \mathcal{S}et \leftarrow \mathcal{S}et$. The category $\mathcal{A}lg_{\mathcal{S}et}\mathsf{F}$ has an initial object.

Proof. We will show that

$$in_{\mathsf{F}} = [\sigma_1, \ldots, \sigma_m] : \mu\mathsf{F} \leftarrow \mathsf{F}(\mu\mathsf{F}) \quad \text{where} \quad \mu\mathsf{F} = T_\Sigma$$

is an initial F-algebra. For every $A \in \mathrm{Ob}\,\mathcal{S}et$ and every $\varphi : A \leftarrow \mathsf{F}A$ we claim that the underlying function of the unique Σ-algebra homomorphism

$$(A; \varphi \cdot \iota_1, \ldots, \varphi \cdot \iota_m) \leftarrow (T_\Sigma; \sigma_1, \ldots, \sigma_m)$$

from the initial term algebra T_Σ (which is free over $\emptyset$) is the respective catamorphism

$$(\!|\varphi|\!)_{\mathsf{F}} : A \leftarrow \mu\mathsf{F}.$$

But this is easy to see, because the UP of the catamorphism (Table 10) is nothing else than the Σ-algebra homomorphism property. ∎

C.6.4 Lemma (Lambek's Lemma [Lam68]**).** Constructors are isomorphisms. In more detail: Let $\mathcal{C}$ be a category and $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{C}$ be an endofunctor. If the category $\mathcal{A}lg_{\mathcal{C}}\mathsf{F}$ has an initial object in_{F}, then this is an isomorphism in $\mathcal{C}$.

Proof. On the one hand

$$\begin{aligned}
& in_{\mathsf{F}} \cdot (\!|\mathsf{F}in_{\mathsf{F}}|\!)_{\mathsf{F}} \\
= \quad & \{ \text{ fusion (Table 10) } \} \\
& (\!|in_{\mathsf{F}}|\!)_{\mathsf{F}} \\
= \quad & \{ \text{ reflection (Table 10) } \} \\
& id_{\mu\mathsf{F}}
\end{aligned}$$

while on the other hand

$$\begin{aligned}
& (\!|\mathsf{F}in_{\mathsf{F}}|\!)_{\mathsf{F}} \cdot in_{\mathsf{F}} \\
= \quad & \{ \text{ UP (Table 10) } \} \\
& \mathsf{F}in_{\mathsf{F}} \cdot \mathsf{F}(\!|\mathsf{F}in_{\mathsf{F}}|\!)_{\mathsf{F}} \\
= \quad & \{ \; \mathsf{F} \text{ functor } \} \\
& \mathsf{F}\big(in_{\mathsf{F}} \cdot (\!|\mathsf{F}in_{\mathsf{F}}|\!)_{\mathsf{F}}\big) \\
= \quad & \{ \text{ see above } \} \\
& \mathsf{F}id_{\mu\mathsf{F}} \\
= \quad & \{ \; \mathsf{F} \text{ functor } \} \\
& id_{\mathsf{F}(\mu\mathsf{F})}.
\end{aligned}$$

Thus, the constructor in_{F} is a $\mathcal{C}$-isomorphism. ∎

References

[AHS90] J. Adámek, H. Herrlich, and G. E. Strecker. *Abstract and Concrete Categories*. Pure and Applied Mathematics. John Wiley & Sons, 1990.

[Bak79] B. S. Baker. Composition of top-down and bottom-up tree transductions. *Inform. and Control*, 41:186–213, 1979.

[BdM97] R. Bird and O. de Moor. *Algebra of Programming*. International Series in Computer Science. Prentice Hall, 1997.

[BH75] L. Budach and H. J. Hoehnke. *Automaten und Funktoren*, volume 35 of *Mathematische Monographien*. Akademie-Verlag, Berlin, 1975.

[Bor94] F. Borceux. *Handbook of Categorical Algebra 1*, volume 1 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Great Britain, 1994.

[CDPR97a] L. Correnson, E. Duris, D. Parigot, and G. Roussel. Attribute grammars and functional programming deforestation. In *4th International Static Analysis Symposium—Poster Session*, Paris (F), 1997.

[CDPR97b] L. Correnson, E. Duris, D. Parigot, and G. Roussel. Symbolic composition. Technical Report 3348, INRIA, January 1997.

[CF82] B. Courcelle and P. Franchi–Zannettacci. Attribute grammars and recursive program schemes. *Theoret. Comput. Sci.*, 17:163–191, 235–257, 1982.

[Dam82] W. Damm. The IO- and OI-hierarchies. *Theoret. Comput. Sci.*, 20:95–208, 1982.

[dB89] P. J. de Bruin. Naturalness of polymorphism. Technical Report CS 8916, Rijksuniversiteit Groningen, The Netherlands, 1989.

[Ehr74] H. Ehring. *Universal Theory of Automata*. Teubner Studienbücher: Informatik. Teubner, Stuttgart, Germany, 1974.

[Eng75] J. Engelfriet. Bottom-up and top-down tree transformations—a comparison. *Math. Systems Theory*, 9(3):198–231, 1975.

[Eng77] J. Engelfriet. Top-down tree transducers with regular look-ahead. *Math. Systems Theory*, 10:289–303, 1977.

[Eng80] J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R.V. Book, editor, *Formal language theory: perspectives and open problems*, pages 241–286. New York, Academic Press, 1980.

[Eng81] J. Engelfriet. Tree transducers and syntax-directed semantics. Technical Report Memorandum 363, Technische Hogeschool Twente, March 1981. also in: Proceedings of the Colloquium on Trees in Algebra and Programming (CAAP 1992), Lille, France 1992.

[Eng82] J. Engelfriet. Three Hierarchies of Transducers. *Math. Systems Theory*, 15:95–125, 1982.

[EP72] H. Ehring and M. Pfender. *Kategorien und Automaten*. de Gruyter Lehrbuch. Walter de Gruyter, 1972.

[EV85a] J. Engelfriet and H. Vogler. Characterization of high level tree transducers. In W. Brauer, editor, *Proceedings of the 12th ICALP*, volume 195 of *Lecture Notes in Computer Science*, pages 474–484, Nafplion, July 1985. Springer-Verlag Berlin.

[EV85b] J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. System Sci.*, 31:71–146, 1985.

[EV88] J. Engelfriet and H. Vogler. High level tree transducers and iterated pushdown tree transducers. *Acta Informatica*, 26:131–192, 1988.

[EV91] J. Engelfriet and H. Vogler. Modular tree transducers. *Theoret. Comput. Sci.*, 78:267–304, 1991.

[Fel78] W. Felscher. *Naive Mengen und abstrakte Zahlen*, volume I. B. I. Wissenschaftsverlag, 1978.

[FHVV93] Z. Fülöp, F. Herrmann, S. Vágvölgyi, and H. Vogler. Tree transducers with external functions. *Theoret. Comput. Sci.*, 108:185–236, 1993.

[Fok92a] M. M. Fokkinga. A gentle introduction to category theory—the calculational approach. In *Lecture Notes of the STOP 1992 Summerschool on Constructive Algorithmics*, pages 1–72 of Part 1. University of Utrecht, The Netherlands, September 1992.

[Fok92b] M. M. Fokkinga. *Law and Order in Algorithmics*. PhD thesis, University of Twente, Dept INF, Enschede, The Netherlands, 1992.

[Fok94] M. M. Fokkinga. Monadic maps and folds for arbitrary datatypes. Memoranda Informatica 94-28, University of Twente, June 1994.

[Fül81] Z. Fülöp. On attributed tree transducers. *Acta Cybernet.*, 5:261–279, 1981.

[FV98] Z. Fülöp and H. Vogler. *Syntax-directed semantics—Formal models based on tree transducers*. Monographs in Theoretical Computer Science, An EATCS Series. Springer-Verlag, 1998.

[Gie88] R. Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Inform.*, 25:355–423, 1988.

[Gil96] A. Gill. *Cheap Deforestation for Non-strict Functional Languages*. PhD thesis, Department of Computing Science, Glasgow University, January 1996.

[GLP93] A. Gill, J. Launchburry, and S. L. Peyton-Jones. A short cut to deforestation. In *Proceedings of Functional Programming Languages an Computer Architecture (FPCA '93)*, pages 223–232, Copenhagen, Denmark, June 1993. ACM Press.

[GS84] F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[GS97] F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer-Verlag, 1997.

[GTWW77] J.A. Goguen, J.W. Thatcher, E.G. Wagner, and J.B. Wright. Initial algebra semantics and continuous algebras. *J. ACM*, 24:68–95, 1977.

[HIT96] Z. Hu, H. Iwasaki, and M. Takeichi. Deriving structural hylomorphisms from recursive definitions. In *Proceedings of the 1st International Conference on Functional Programming*, pages 73–82, Philadelphia, PA, May 1996. ACM Press.

[Ihr88] Th. Ihringer. *Allgemeine Algebra*. Teubner Studienbücher: Mathematik. Teubner, Stuttgart, Germany, 1988.

[Joh01] P. Johann. Short cut fusion: Proved and improved. In W. Taha, editor, *Proceedings of the 2nd International Workshop on Semantics, Applications, and Implementation of Program Generation (SAIG 2001)*, volume 2196 of *LNCS*, pages 47–71, Florence, Italy, September 2001. Springer.

[Jür00] C. Jürgensen. A formalization of hylomorphism based deforestation with an application to an extended typed λ-calculus. Technical Report TUD-FI00-13, Technische Universität Dresden, Fakultät Informatik, D-01062 Dresden, Germany, November 2000.

[Kle65] H. Kleisli. Every standard construction is induced by a pair of adjoint functors,. In *Proc. Amer. Math. Soc.*, volume 16, pages 544–546, 1965.

[KV01] A. Kühnemann and J. Voigtländer. Tree transducer composition as deforestation method for functional programs. Technical Report TUD-FI01-07, Technische Universität Dresden, Fakultät Informatik, D-01062 Dresden, Germany, August 2001.

[Lam68] J. Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103:151–161, 1968.

[LS95] J. Launchburry and T. Sheard. Warm fusion: Deriving build-catas from recursive definitions. In *Proceedings of Functional Programming Languages an Computer Architecture (FPCA '95)*, pages 314–323, La Jolla, San Diego, CA, USA, June 1995. ACM Press.

[NV01] T. Noll and H. Vogler. The universality of higher-order attributed tree transducers. *Theory of Computing Systems*, 45–75, 2001.

[Par00] A. Pardo. Monadic corecursion – definition, fusion laws, and applications –. In B. Jacobs, L. Moss, H. Reichel, and J. Rutten, editors, *Electronic Notes in Theoretical Computer Science*, volume 11. Elsevier Science Publishers, 2000.

[PTH01] S. L. Peyton-Jones, A. Tolmach, and T. Hoare. Playing by the rules: Rewriting as a practical optimisation technique in GHC. In Ralf Hinze, editor, *Preliminary Proceedings of the 2001 ACM SIGPLAN Haskell Workshop (HW '2001)*, pages 203–233, Firenze, Italy, September 2001.

[Rou68] W. C. Rounds. *Trees, transducers and transformations*. PhD thesis, Stanford University, 1968.

[Rou70] W. C. Rounds. Mappings and grammars on trees. *Math. Systems Theory*, 4:257–287, 1970.

[Tha70] J. W. Thatcher. Generalized^2 sequential machine maps. *J. Comput. System Sci.*, 4:339–367, 1970.

[TM95] A. Takano and E. Meijer. Shortcut deforestation in calculational form. In *Proceedings of the Conference on Functional Programing Languages and Computer Architecture*, pages 306–313, La Jolla, CA, June 1995. ACM Press.

[VK01] J. Voigtländer and A. Kühnemann. Composition of functions with accumulating parameters. Technical Report TUD-FI01-08, Technische Universität Dresden, Fakultät Informatik, D-01062 Dresden, Germany, August 2001.

[Wad89] P. Wadler. Theorems for free! In *The 4th International Conference on Functional Programming Languages and Computer Architecture (FPCA '89)*, pages 347–359, London, September 1989. Imperial College, ACM Press.

[Wad90] P. Wadler. Deforestation: Transforming programs to eliminate trees. *Theoretical Computer Science*, 73(2):231–248, 1990.

[Wad92] P. Wadler. The essence of functional programming. In *Proceedings of the Symposium on Principles of Programming Languages (POPL '92), Albequerque*. ACM Press, 1992.

[Wec92] W. Wechsler. *Universal Algebra for Computer Scientists*, volume 25 of *EATCS Monograohs on Theoretical Computer Science*. Springer, 1992.