Categorical semantics and composition of tree transducers

Dissertation

zur Erlangung des akademischen Grades
Doktor rerum naturalium (Dr. rer. nat.)

vorgelegt an der

Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Diplom-Mathematiker Claus Jürgensen
geboren am 05. Mai 1966 in Kappeln an der Schlei

Gutachter:

Prof. Dr.-Ing. habil. Heiko Vogler, Technische Universität Dresden
Prof. Dr. rer. nat. habil. Horst Reichel, Technische Universität Dresden
Prof. Dr. Zoltán Fülöp, Universität Szeged, Ungarn

Tag der Verteidigung: 30.01.2004

Dresden im Oktober 2003

Acknowledgments

This dissertation would not have been written without the support from many people.

I am grateful to Heiko Vogler for accepting the supervision of this thesis and for providing ideal conditions for doing research. I am obliged to Zoltán Fülöp and Horst Reichel for accepting to be referees.

Life in Dresden would not be fun without a number of people. My officemates Lutz Straßburger, Björn Borchardt and Igor Tarasyuk somehow put up with me for some years. I am particularly thankful to Janis Voigtländer, the most critical reader of my papers: His discerning remarks have always been very helpful to me. I have also been fortunate to share Dresden with Armin Kühnemann, Enrico Bormann, Alexandre Scalzitti, Andreas Maletti, and Kai Brünnler.

This thesis would not exist without the support of Gwendoline Blandin. During all the time she has been a continuous source of love and inspiration.

This PhD thesis has been written with the financial support of the postgraduate program *Specification of discrete processes and systems of processes by operational models and logics* (GRK 334) of the German Research Community (DFG).

Claus Jürgensen
October, 2003

Contents

1 Introduction

This thesis is on a program transformation of functional programs called *fusion*. Consider three types A, B, and C and two recursive programs f and g with typing[1] $C \xleftarrow{f} B \xleftarrow{g} A$. We call a program $h : C \leftarrow A$ a **fusion** of the **consumer** f and the **producer** g if two conditions are satisfied:

(i) $[\![f]\!] \cdot [\![g]\!] = [\![h]\!]$, and

(ii) the intermediate data-structure B does not occur in h.

The condition (i) is the correctness of the fusion w.r.t. the denotational semantics $[\![\,\cdot\,]\!]$. If the semantics is compositional it can be trivially satisfied setting $h = f \cdot\ g = \lambda\, x \to\ f(g\, x)$. The essential point is condition (ii): the elimination of the intermediate data-structure (which excludes the trivial solution).

An important application of fusion is the deforestation of functional programs, *i.e.* a program transformation to eliminate intermediate tree-like data structures, hoping to improve the runtime behavior of the program.

Various fusion techniques are known: *deforestation* [Wad90], *short cut fusion* [GLP93, TM95, Gil96, Joh01], or *syntactic composition of tree transducers (and attribute grammars)* [Eng75, Eng80, Fül81, CF82, EV85, Gie88, CDPR97b, CDPR97a, Küh98, FV98, VK01].

The objectives of this thesis are

(i) to model tree transducers, semantics of tree transducers, and composition of tree transducers in category theory,

(ii) to compare tree transducer composition with short cut fusion, and

(iii) to compose syntactic classes of tree transducers without doing tedious induction proofs.

We can use results from (i) for (ii), because short cut fusion can be described in category theory in form of the *acid rain theorem* [TM95]. A nice definition and proof principle used in category is the concept of *universal properties*. The latter make it possible to avoid detailed induction proofs, and thus results from (i) can also be used for (iii). However, for (i) we discovered two independent and completely different solutions: the *initial*

[1] Please forgive us for drawing all arrows from right to left. In Subsection 2.1.2 we explain why we prefer it this way. However, in program examples we follow the Haskell [PH99] syntax and use arrows from left to right.

algebra approach (Section 1.3 and Chapter 5) and the *free monad approach* (Section 1.4 and Chapter 6).

Let us start with a closer look on *short cut fusion* (Section 1.1), the *composition of tree transducers* (Section 1.2 and Chapter 3), and on the two aforementioned different approaches to model tree transducers in category theory:

1.1 Short cut fusion

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:

$$
\begin{array}{ll}
\textbf{data}\ \mathit{Tree} & = \mathit{Alpha} \mid \mathit{Sigma}(\mathit{Tree},\ \mathit{Tree}) \\
\textbf{data}\ \mathit{List} & = N \mid A\ \mathit{List} \mid B\ \mathit{List}
\end{array}
$$

$$
\begin{array}{ll}
\mathit{zig} & :: \mathit{Tree} \to \mathit{List} \\
\mathit{zag} & :: \mathit{Tree} \to \mathit{List} \\
\mathit{zig}\ \mathit{Alpha} & = N \\
\mathit{zag}\ \mathit{Alpha} & = N \\
\mathit{zig}\ (\mathit{Sigma}(x_1,\ x_2)) & = A(\mathit{zag}\ x_1) \\
\mathit{zag}\ (\mathit{Sigma}(x_1,\ x_2)) & = B(\mathit{zig}\ x_2)
\end{array}
$$

$$
\begin{array}{ll}
\mathit{bin} & :: \mathit{List} \to \mathit{Tree} \\
\mathit{bin}\ N & = \mathit{Alpha} \\
\mathit{bin}\ (A\ x) & = \mathit{Sigma}(\mathit{bin}\ x,\ \mathit{bin}\ x) \\
\mathit{bin}\ (B\ x) & = \mathit{Sigma}(\mathit{bin}\ x,\ \mathit{bin}\ x)
\end{array}
$$

$$
\begin{array}{ll}
\mathit{binzig} & :: \mathit{Tree} \to \mathit{Tree} \\
\mathit{binzig} & = \mathit{bin} \cdot \mathit{zig}
\end{array}
$$

$$
\begin{array}{ll}
\mathit{binzag} & :: \mathit{Tree} \to \mathit{Tree} \\
\mathit{binzag} & = \mathit{bin} \cdot \mathit{zag}
\end{array}
$$

We will deforest the function *binzig*: 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

$$
\begin{array}{l}
\mathit{build} \quad :: (\textbf{forall}\ c.\ c \to (c \to c) \to (c \to c) \to d \to c) \to (d \to \mathit{List}) \\
\mathit{build}\ g = g\ N\ A\ B
\end{array}
$$

where the semantics of *build* is to apply its argument to the *List*-constructors. Then we can write

$$
\begin{array}{ll}
\mathit{zig} & = \mathit{build}\ \mathit{zig}' \\
\mathit{zag} & = \mathit{build}\ \mathit{zag}'
\end{array}
$$

$$
\begin{array}{ll}
zig' & :: c \to (c \to c) \to (c \to c) \to Tree \to c \\
zag' & :: c \to (c \to c) \to (c \to c) \to Tree \to c \\
zig'\ n\ a\ b\ Alpha & = n \\
zag'\ n\ a\ b\ Alpha & = n \\
zig'\ n\ a\ b\ (Sigma(x_1,\ x_2)) & = a(zag'\ n\ a\ b\ x_1) \\
zag'\ n\ a\ b\ (Sigma(x_1,\ x_2)) & = b(zig'\ n\ a\ b\ x_2)
\end{array}
$$

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

$$
\begin{array}{ll}
cata\ n\ a\ b\ N & = n \\
cata\ n\ a\ b\ (A\ x) & = a(cata\ n\ a\ b\ x) \\
cata\ n\ a\ b\ (B\ x) & = b(cata\ n\ a\ b\ x)
\end{array}
$$

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:

$$bin\ =\ cata\ Alpha\ (\lambda x\ \to\ Sigma\ x\ x)\ (\lambda x\ \to\ Sigma\ x\ x)$$

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

$$\frac{g :: c \to (c \to c) \to (c \to c) \to d \to c}{cata\ n\ a\ b \cdot build\ g = g\ n\ a\ b}$$

which leads to

$$
\begin{array}{ll}
binzig = & bin\ \cdot\ zig \\
& = cata\ Alpha\ (\lambda x \to\ Sigma\ x\ x)\ (\lambda x \to\ Sigma\ x\ x)\ \cdot\ build\ zig' \\
& = zig'\ Alpha\ (\lambda x \to\ Sigma\ x\ x)\ (\lambda x \to\ Sigma\ x\ x)
\end{array}
$$

and similarly for *binzag*. Finally, by applying *binzig* (and *binzag*) to every possible input pattern, we obtain the program

$$
\begin{array}{ll}
binzig\ Alpha & = Alpha \\
binzag\ Alpha & = Alpha \\
binzig\ (Sigma(x_1,\ x_2)) & = Sigma((binzag\ x_1),\ (binzag\ x_1)) \\
binzag\ (Sigma(x_1,\ x_2)) & = Sigma((binzig\ x_2),\ (binzig\ x_2))
\end{array}
$$

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 type. This restricted transformation has also been implemented in the GHC (Glasgow

Haskell Compiler) [PTH01]. For arbitrary regular data types 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]. We will prove the following version:

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

where $(\!|\varphi|\!)_{\mathsf{G}}$ is the category theory notation for the catamorphism, which is the unique solution of the equation $(\!|\varphi|\!)_{\mathsf{G}} \cdot in_{\mathsf{G}} = \varphi \cdot \mathsf{G}(\!|\varphi|\!)_{\mathsf{G}}$, 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].

We also prove a generalization (Proposition 5.2.2.1):

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

where U is an arbitrary faithful endofunctor. The additional functor U makes it possible to conveniently describe mutual recursive functions.

Our above version of the *acid rain theorem* is true in any category for any endofunctors F, G, and U such that F and G have initial algebras and U is faithful. The traditional approach to prove the correctness of short cut fusion for a specific programming language uses the *free theorems* [Wad89] or relies on the existence of a parametric model for that language [Joh01]. We will use our new *acid rain theorem* to prove correctness of short cut fusion for the language of top-down tree transducers (which can be viewed as a syntactic fragment of a functional programming language) directly without *free theorems* or a parametric model. This new approach promises new possibilities for correctness proofs of short cut fusion and its generalizations for functional programming languages.

Notice that in our version of the *acid rain theorem* both functions are given as catamorphisms. In fact we can state a symmetric version (Theorem 5.2.2.2) where *fusion* turns out to be the composition in some category. Since the composition of a category is associative we can see immediately that the order of successive fusions is irrelevant, which can be of use for implementations.

1.2 Syntactic composition of tree transducers

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.*

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

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 $zig(t)$ 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 $T_\Delta\emptyset \xleftarrow{[\![T]\!]} T_\Sigma\emptyset$ where $T_\Sigma\emptyset$ and $T_\Delta\emptyset$ 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 $T_\Gamma\emptyset \xleftarrow{[\![T_2]\!]} T_\Delta\emptyset \xleftarrow{[\![T_1]\!]} T_\Sigma\emptyset$ we can construct a new top-down tree transducer $\Gamma \xleftarrow{T_2 \cdot T_1} \Sigma$ such that the following composition result holds:

$$[\![T_2]\!] \cdot [\![T_1]\!] = [\![T_2 \cdot T_1]\!] \qquad (*)$$

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:

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

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 $T_2 \cdot T_1$:

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

And this corresponds to the equations of the functional program that we have calculated for *binzig* (and *binzag*) 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, Küh98, CDPR97b, CDPR97a] for attributed tree transducers, in [Eng80, CF82, EV85] for macro tree transducers (or for primitive-recursive program schemes with parameters), and in [EV88] for high-level tree transducers (also *cf.* the survey articles and monographs [GS84, GS97, FV98]). In [Küh99, KV01] the composition of tree transducers has been compared with the deforestation method for functional programs [Wad90] in a syntactical framework.

1.3 The initial algebra approach

An obvious question is: What is the relationship between the two fusion techniques introduced in the last two sections?

In order to compare them we will introduce the notion of an *algebraic 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:

$$\begin{array}{ccc}
\boxed{\text{tree transducer}} & ? & \text{short cut fusion} \\
\text{syntactic composition} & \longleftrightarrow & \textit{cata/build-rule} \\
\textit{universal algebra} & & \textit{polymorphic } \lambda\textit{-calculi} \\
\\
\downarrow \text{ generalization} & & \downarrow \text{ generalization} \\
\\
\boxed{\text{algebraic transducer}} & & \text{functorial short cut fusion} \\
\text{composition} & \subseteq & \text{generalized } \textit{acid rain theorem} \\
\textit{category theory} & \longleftarrow & \textit{category theory}
\end{array}$$

Let us explain now a bit more the notion of an algebraic 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 the abstract level of category theory, where we have the following intuition how the ingredients of a top-down algebraic transducer are related to those of a top-down tree transducer:

top-down tree transducer			top-down algebraic transducer over $\boldsymbol{\mathcal{Set}}$	
$T = (Q, \Sigma, \Delta, q_0, R)$			$C = (\mathsf{H}, \mathsf{U}, \pi) : \mathsf{G} \leftarrow \mathsf{F}$	
finite set of states	Q	$\leftrightarrow$	$\mathsf{U} : \boldsymbol{\mathcal{Set}} \leftarrow \boldsymbol{\mathcal{Set}}$	faithful endofunctor
ranked input-alphabet	Σ	$\leftrightarrow$	$\mathsf{F} : \boldsymbol{\mathcal{Set}} \leftarrow \boldsymbol{\mathcal{Set}}$	endofunctor
ranked output-alphabet	Δ	$\leftrightarrow$	$\mathsf{G} : \boldsymbol{\mathcal{Set}} \leftarrow \boldsymbol{\mathcal{Set}}$	endofunctor
initial state	$q_0 \in Q$	$\leftrightarrow$	$\pi : \mathsf{Id} \dot{\leftarrow} \mathsf{U}$	natural transformation
finite set of rules	R	$\leftrightarrow$	$\mathsf{H} in_{\mathsf{G}}$ where H is a concrete functor $\mathsf{H} : (\boldsymbol{\mathcal{Set}}^{\mathsf{F}}, \lvert\,\boldsymbol{.}\,\rvert^{\mathsf{F}}) \leftarrow (\boldsymbol{\mathcal{Set}}^{\mathsf{G}}, \mathsf{U} \cdot \lvert\,\boldsymbol{.}\,\rvert^{\mathsf{G}})$	
set of Σ-trees	$T_\Sigma\emptyset$	$\leftrightarrow$	$\boldsymbol{\mu}\mathsf{F}$	least fixed point of F
set of Δ-trees	$T_\Delta\emptyset$	$\leftrightarrow$	$\boldsymbol{\mu}\mathsf{G}$	least fixed point of G
tree transformation $[\![T]\!] : T_\Delta\emptyset \leftarrow T_\Sigma\emptyset$		$\leftrightarrow$	$[\![C]\!] = \pi \cdot (\!|\mathsf{H}\, in_{\mathsf{G}}|\!)_{\mathsf{F}} : \boldsymbol{\mu}\mathsf{G} \leftarrow \boldsymbol{\mu}\mathsf{F}$	

We will also formalize this relationship and call the top-down tree transducer T and the top-down algebraic 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 algebraic 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 fixed point of the 'rules' by induction and the natural transformation π selects the value of the 'initial state'. We show that all algebraic transducers over some category $\boldsymbol{\mathcal{C}}$ are the morphisms of a category (denoted by $\mathbf{AT}_{\boldsymbol{\mathcal{C}}}$), 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-* $\mathbf{AT}_{\boldsymbol{\mathcal{C}}}$ of all top-down algebraic transducers is its subcategory. The crucial point is the composition $\mathsf{H}_1 \cdot \mathsf{H}_2$ where the 'rules' of the second algebraic transducer $(\mathsf{H}_2, \mathsf{U}_2, \pi_2)$ are used as 'output-symbols' for the first algebraic 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 an algebraic transducer turns out to be a generalization of short cut fusion in the following sense: With the functorial *acid rain theorem* (Theorem 5.2.2.2) we can prove that the semantics $[\![\,\boldsymbol{.}\,]\!]$ of algebraic transducers over $\boldsymbol{\mathcal{C}}$ is a functor $[\![\,\boldsymbol{.}\,]\!] : \boldsymbol{\mathcal{C}} \leftarrow \mathbf{AT}_{\boldsymbol{\mathcal{C}}}$, *i.e.*

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

On the other hand we prove that top-down tree transducers (modulo isomorphism) with syntactic composition form a category $td\text{-}\boldsymbol{\mathcal{T}t}$ and that $\mathsf{R} : td\text{-}\,\mathbf{AT}_{\boldsymbol{\mathcal{S}et}} \leftarrow td\text{-}\boldsymbol{\mathcal{T}t}$ is an embedding functor (Theorem 5.4.3.5).
This functor respects the semantics (Corollary 5.4.2.11), *i.e.*:

$$\underbrace{[\![\,\boldsymbol{\cdot}\,]\!]}_{\text{on } tdtt} \;=\; \underbrace{[\![\,\boldsymbol{\cdot}\,]\!]}_{\text{on } td\text{-}\,\mathbf{AT}} \;\cdot\; \mathsf{R}.$$

Then the syntactic composition of top-down tree transducers (*cf.* statement $(*)$: $[\![T_2]\!] \cdot [\![T_1]\!] = [\![T_2 \cdot T_1]\!]$ on page 5) is short cut fusion of the corresponding algebraic transducers (Theorem 5.4.3.5):

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

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 algebraic 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:

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

i.e.

$$[\![T_2]\!] \cdot [\![T_1]\!] = [\![(T_2 \cdot T_1]\!].$$

1.4 The free monad approach

Short cut fusion is based on the *cata/build-rule* [GLP93], *cata/augment-rule* [Gil96], or *acid rain theorem* [TM95]. Therefore it is necessary to represent the consumer as a catamorphism. A catamorphism is a generalization of the well known list-function *foldr* for arbitrary regular types. In terms of category theory a catamorphism is the unique algebra morphism from an initial algebra.

We have invented a new fusion technique using monads: instead of a catamorphism we use the unique monad morphism from a free monad.

Consider the small Haskell program:

$$\begin{array}{ll}
\textbf{data}\ \mathit{Nat} & = \mathit{Zero} \mid \mathit{Succ}\ \mathit{Nat} \\
\textbf{data}\ \mathit{Bool} & = \mathit{False} \mid \mathit{True}
\end{array}$$

$$\begin{array}{ll}
\mathit{even}\ \ \mathit{Zero} & = \mathit{True} \\
\mathit{even}\ (\mathit{Succ}\ n) & = \mathit{odd}\ \ n \\
\mathit{odd}\ \ \mathit{Zero} & = \mathit{False} \\
\mathit{odd}\ \ (\mathit{Succ}\ n) & = \mathit{even}\ n
\end{array}$$

The latter four equations define the two mutually recursive functions *even* and *odd*. We can view this system of equations as a function:

$$\begin{aligned} \varrho_X : \mathsf{T}_\Delta(\mathsf{Q}X) &\leftarrow \mathsf{Q}(\Sigma X), \\ \mathit{True} &\leftarrowtail \mathit{even}\ \mathit{Zero}, \\ \mathit{odd}\ n &\leftarrowtail \mathit{even}(\mathit{Succ}\ n), \\ \mathit{False} &\leftarrowtail \mathit{odd}\ \mathit{Zero}, \\ \mathit{even}\ n &\leftarrowtail \mathit{odd}(\mathit{Succ}\ n) \end{aligned}$$

where $X = \{n\}$ is the set of variables. The endofunctors Σ, Δ, and Q describe the application of ranked symbols[2] from $\{\mathit{Zero}^{(0)}, \mathit{Succ}^{(1)}\}$, $\{\mathit{True}^{(0)}, \mathit{False}^{(0)}\}$, and $\{\mathit{even}^{(1)}, \mathit{odd}^{(1)}\}$, respectively, to a set (*e.g.* $\Sigma X = \{\mathit{Zero}\} \cup \{\mathit{Succ}\ x | x \in X\}$). The functor T_Δ constructs all Δ-trees over a set X: $\mathsf{T}_\Delta X \cong \boldsymbol{\mu}(\Delta + \underline{X})$.

It is now possible to show (Proposition 6.1.3.1) that the function ϱ_X is *natural* in X and thus we have a natural transformation:

$$\varrho : \mathsf{T}_\Delta \cdot \mathsf{Q} \mathrel{\dot{\leftarrow}} \mathsf{Q} \cdot \Sigma$$

which we call the *rule* of the functional program. Thus we opened the door to the realm of category theory.

Using some category theory magic (like the relation of adjoint functors and monads) we can equivalently transform this rule into the form:

$$\varrho^\sharp : |\mathsf{H}\Delta^\star| \mathrel{\dot{\leftarrow}} \Sigma$$

where $\Delta^\star = (\mathsf{T}_\Delta, \eta, \mu)$ denotes the free monad over Δ, H is an endofunctor, and $|\centerdot|$ the forgetful functor mapping a monad onto its underlying endofunctor. A rule in the latter form is the main ingredient of a so called **monadic transducer** which we introduce in Definition 6.3.1.1.

Using the universal property of a free monad (Definition 4.4.3.1) we can define a denotational semantics for monadic transducers. Moreover, we have proved a new fusion theorem for monadic transducers (Theorem 6.3.2.5) w.r.t. this semantics. In particular we can prove the condition (ii), *i.e.* the elimination of the intermediate data-structure.

Our construction depends on the syntactic structure of the functional programs f and g which we want to compose. We use syntactic classes of *tree transducers* to describe sufficient syntactic forms of the functional programs. A tree transducer is a finite tree automaton with in- and output. Its integral part is a set of rules. Some classes of tree transducers can be viewed as syntactic fragments of functional programming languages. Our example Haskell program is a top-down tree transducer which has the two states *even* and *odd*.

[2]Sometimes we annotate a symbol with its rank (as a superscript in parenthesis).

The composition of top-down tree transducers is an instance of short cut fusion. But for more complicated tree transducers we have not been able to apply short cut fusion, and that is why we invented the monadic transducer.

Moreover, we are interested in the question, whether syntactic classes of tree transducers are closed under fusion. This question has been answered (positively or negatively) for many classes of tree transducers. The constructions and proofs of the classical results differ depending on the specific class of tree transducers investigated. Using our monadic transducer we can describe many kinds of tree transducers in a uniform way. Once modeled as a monadic transducer, it is easy to do a fusion and then inspect whether the result is a tree transducer of a specific class.

We will show how to compose *homomorphism top-down*, *top-down*, *simple basic macro*, *basic macro*, *macro* with our new approach. And we will outline how to extend our new approach to the fusion of *top-down tree to tree-series transducers* and *bottom-up tree transducers*. Even though we use some esoteric category theory, our results will be down-to-earth constructions which are applicable to transform real functional programs (Figure 6.3).

1.5 Structure of the thesis

Our main results are:

- A new variant of the *acid rain theorem* for mutually recursive functions where the semantics of the *build* is described by a *concrete functor* (Proposition 5.2.2.1).
- A symmetric form (*i.e.* consumer and producer have the same syntactic form) of our new *acid rain theorem* where fusion is composition in a category and thus in particular *associative* (Theorem 5.2.2.2).
- Applying *short cut fusion* (using the *acid rain theorem*) to compose top-down tree transducers yields the same result (on a syntactic level) as the classical top-down tree transducer composition (Theorem 5.4.3.2).
- A fusion theorem for monadic transducers (Theorem 6.3.2.5).
- We prove that homomorphic monadic transducers are semantically equivalent (Theorem 6.3.3.2). This makes it possible (using Corollary 6.3.3.3) to compose syntactic classes of tree transducers (or functional programs) by simply composing endofunctors
  (Subsection 6.5.2).

The structure of the thesis is as follows:

- We start with some preliminaries where we introduce symbols and notations, that we will use for universal algebra and category theory.
- In Part I we give a brief introduction to the theory of tree transducers and we collect all the notions and theorems of category theory that we will need. Most of

this can be found in standard text books. However, we give a presentation focused on our needs and use a uniform notation.

- In Part II we generalize tree transducers in category theory:
  - In Chapter 5 we start with an *initial algebra approach*: This approach follows the ideas of *short cut fusion* and the *acid rain theorem*.
  - In Chapter 6 we develop a *free monad approach*: In the same way as monoids are used in automata theory, we use monads in the theory of tree transducers.

2 Preliminaries

2.1 General notions

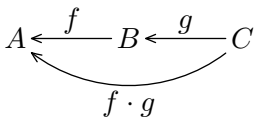
The set of natural numbers starting from *one* is denoted by $\mathbb{N}$ and the set of natural numbers starting from *zero* is denoted by $\mathbb{N}_0$.

2.1.1 Sets and classes

The *set of all sets* does not exist for set theoretical reasons. But it is possible to define the notion of *classes* and in particular the class of all sets. That is why a category is made up of a *class* of objects together with a *class* of morphisms, so it is possible to define the category $\boldsymbol{\mathcal{S}et}$ of all sets. For similar reasons the *class of all classes* does not exist. But it is possible to define the notion of *conglomerates* and in particular the conglomerate of all classes. And thus we can define the meta-category **CAT** of all categories. Notice, that the axioms for a category and for a meta-category are equal with the only exception that we use classes in the first case and conglomerates in the second. Hence all notions and theorems from category theory, as long as they do not refer to essential properties of classes, can be lifted immediately to meta-category theory.

2.1.2 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. 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[1]:



We assume that function application binds stronger than function composition and the latter binds stronger than any other binary operation. Super and subscripts have the highest precedence, *e.g.* $\Sigma^\star + \mathsf{H}\Delta^\star = (\Sigma^\star) + \big(\mathsf{H}(\Delta^\star)\big)$

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

For some functions we use an infix, outfix, or superscript notation. In order to write down such a function in a point free style, we use the symbol '$\bullet$' as a placeholder for the argument: *e.g.* $(A + \bullet) : A + B \leftarrow B$, $|\bullet| : |\mathbb{T}| \leftarrow \mathbb{T}$, or $(\bullet)^\star : \Sigma^\star \leftarrow \Sigma$. Then we can write expressions like $|\bullet| \cdot (\bullet)^\star$ for the function $|\Sigma^\star| \leftarrow \Sigma$, *i.e.* $\big(|\bullet| \cdot (\bullet)^\star\big)\Sigma = |\Sigma^\star|$.

2.2 Universal algebra

The universal algebra that we need can be found in standard textbooks like [Ihr88, Wec92].

2.2.1 Trees, terms and substitution

2.2.1.1 Definition (ranked alphabet). $\Sigma = (\Sigma, \mathrm{rank}_\Sigma)$ where Σ is a finite set and $\mathrm{rank}_\Sigma : \mathbb{N}_0 \leftarrow \Sigma$ is a function is called a **ranked alphabet**. For every $r \in \mathbb{N}_0$ we define the set $\Sigma^{(r)} = \big\{\sigma \in \Sigma \;\big|\; \mathrm{rank}_\Sigma = r\big\}$. If $\Sigma = \Sigma^{(1)}$, then Σ is called **unary**. If $\#\Sigma^{(0)} = 1$ and $\Sigma^{(k)} = \emptyset$ for every natural number $k > 2$, then Σ is called **monadic**[2].

Sometimes we write $\Sigma = \{\sigma_1^{(r_1)}, \ldots, \sigma_k^{(r_k)}\}$ to indicate that $\Sigma = (\Sigma, \mathrm{rank}_\Sigma)$ is a ranked alphabet with $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ and $\mathrm{rank}_\Sigma\, \sigma_i = r_i$. ◇

2.2.1.2 Definition (term, tree). Let Σ be a ranked alphabet and X be a set. The set $\mathsf{T}_\Sigma X$ of all X-indexed **Σ-terms** (or **Σ-trees**[3]) is the smallest set such that

$$\frac{x \in X}{\text{'}x \in \mathsf{T}_\Sigma X} \quad \text{and} \quad \frac{r \in \mathbb{N}_0 \quad \sigma \in \Sigma^{(r)} \quad t_1, \ldots, t_r \in \mathsf{T}_\Sigma X}{\sigma t_1 \cdots t_r \in \mathsf{T}_\Sigma X}.$$

The set of **variables** occurring in a term $t \in \mathsf{T}_\Sigma X$ is denoted by $var_X\, t$. The function $var_X : \boldsymbol{\mathcal{P}ot}\, X \leftarrow \mathsf{T}_\Sigma X$ is defined by

$$\frac{x \in X}{var_X(\text{'}x) = \{x\}} \quad \text{and} \quad \frac{r \in \mathbb{N}_0 \quad \sigma \in \Sigma^{(r)} \quad t_1, \ldots, t_r \in \mathsf{T}_\Sigma X}{var_X(\sigma t_1 \cdots t_r) = \bigcup_{i=1}^r (var_X\, t_i)}.$$

A Σ-term is called **monadic** if Σ is monadic. A term $t \in \mathsf{T}_\Sigma X$ is called **linear** if every variable $x \in X$ occurs at most once in t. The set of all linear terms $\mathsf{T}_\Sigma^{lin} X$ is the smallest set such that

$$\frac{x \in X}{\text{'}x \in \mathsf{T}_\Sigma^{lin} X} \quad \text{and} \quad \frac{r \in \mathbb{N}_0 \quad \sigma \in \Sigma^{(r)} \quad t_i \in \mathsf{T}_\Sigma^{lin} X_i \quad X_i \subseteq X \quad \forall\, i \neq j.\ X_i \cap X_j = \emptyset}{\sigma t_1 \cdots t_r \in \mathsf{T}_\Sigma^{lin} X}.$$

[2] This is not related to *monads* (see Note 4.4.1.2).

[3] More precisely a tree would be a labeled graph representing a term. We will not use the graph-theoretic notion of a tree. The 'X-indexed Σ-trees' are sometimes also called the 'X-incomplete Σ-branching trees'.

Moreover we define the set of all (Σ, X)**-contexts** by $\mathsf{T}^{ctx}_\Sigma X = \{t \in \mathsf{T}^{lin}_\Sigma X \mid var_X\, t = X\}$.

Obviously $\mathsf{T}^{ctx}_\Sigma X \subseteq \mathsf{T}^{lin}_\Sigma X \subseteq \mathsf{T}_\Sigma X$.

Attention: In Chapter 5 we prefer to write a term $\sigma\, t_1 \cdots t_k$ in the form $\sigma(t_1, \ldots, t_k)$, because we will identify every symbol σ of rank k with the k-ary function $\mathsf{T}_\Sigma X \leftarrow (\mathsf{T}_\Sigma X)^k : \sigma\, t_1 \cdots t_k \mapsto (t_1, \ldots, t_k)$. Moreover, we will omit the quote preceding a variable whenever it is obvious whether a symbol is intended to be a variable or a tree.

In Chapter 5 we prefer to write ranked alphabets as Σ and the set of terms as $T_\Sigma\emptyset$. In Chapter 6 we write Σ and T_Σ to emphasize that these are (or induce) functors. ◇

2.2.1.3 Definition (substitution). Let Σ be a ranked alphabet. Let $s \in \mathsf{T}_\Sigma X$ and $y \in X$.

(i) The **substitution** function $[s/y] : \mathsf{T}_\Sigma X \leftarrow \mathsf{T}_\Sigma X$ is defined by

$$[s/y](\text{'}x) = \begin{cases} s & \text{if } x = y, \\ \text{'}x & \text{otherwise,} \end{cases}$$
$$[s/y](\sigma t_1 \cdots t_r) = \sigma\big([s/x]t_1\big) \cdots \big([s/x]t_r\big).$$

The function $[s/y]$ substitutes all occurrences of the variable y by the term s. This can be generalized straightforwardly to finitely many simultaneous substitutions:

(ii) The following function $[s_1/y_1, \ldots, s_k/y_k] = [s_i/y_i]_{i=1}^k$ substitutes the variables $y_1, \ldots, y_k$ by the terms $s_1, \ldots, s_k$, respectively.

$$[s_i/y_i]_{i=1}^k(\text{'}x) = \begin{cases} s_i & \text{if } x = y_i, \\ \text{'}x & \text{otherwise} \end{cases}$$
$$[s_i/y_i]_{i=1}^k(\sigma t_1 \cdots t_r) = \sigma\big([s_i/y_i]_{i=1}^k t_1\big) \cdots \big([s_i/y_i]_{i=1}^k t_r\big).$$

(iii) Finally we define the substitution of arbitrary many variables: For every function $f : \mathsf{T}_\Sigma Y \leftarrow X$ we define $f^\dagger$ by

$$f^\dagger(\text{'}x) = fx,$$
$$f^\dagger(\sigma t_1 \cdots t_r) = \sigma(f^\dagger t_1) \cdots (f^\dagger t_r).$$

The **substitution** function $f^\dagger$ substitutes every variable from X by the terms determined by the **interpretation function** f. The function $(\bullet)^\dagger$ is called **Kleisli extension** (see Definition and Lemma 4.4.1.3).

Notice, that in the particular case $\{x_1, \ldots, x_k\} \subseteq X = Y$ where f is defined using some $t_1, \ldots, t_k \in \mathsf{T}_\Sigma X$ by $fx_j = t_j$ (and $fy = y$ otherwise) we have $f^\dagger = [t_1/x_1, \ldots, t_k/x_k]$. ◇

2.2.2 Σ-algebras

2.2.2.1 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. ◇

2.2.2.2 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 $|h| : |A| \leftarrow |B|$. ◇

2.2.2.3 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 $\forall\, x \in X.\ |f'|x = fx$. ◇

2.2.2.4 Definition (term algebra). Let $\boldsymbol{\Sigma} = (\Sigma, \mathrm{rank}_{\boldsymbol{\Sigma}})$ be a ranked alphabet. If we identify an element $\sigma \in \Sigma$ with the following function:

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

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 is denoted just by $T_\Sigma A$. Finally $T_\Sigma \emptyset$ is called 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. ◇

2.2.2.5 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. ◇

2.2.2.6 Definition (2^{nd} order 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 $\forall\, j \in \{1, \ldots, k\}.\ |g|x_j = b_j$.$\diamond$

2.3 Basic notions of category theory

In this section we give just a short overview over the category theory that we will need. A reader who is unfamiliar with category theory can find all the necessary details in Chapter 4.

We will use the following notions from category-theory: *(bi/endo)functor, natural transformation, vertical composition, initial/final object, (co)product, projection, injection, exponent, evaluation morphism, (initial)* F*-algebra, universal arrow, concrete category, construct, free object, adjunction, monad (morphism)*, and *varietor*.

Adjunctions and *monads* will only be needed in Chapter 6 and not in Chapter 5.

If possible and appropriate we will use the following fonts: $A, B, C, \ldots$ for objects; $f, g, h, \ldots$ for morphisms; $\mathsf{F}, \mathsf{G}, \mathsf{H}, \ldots$ for functors; $\sigma, \tau, \varphi, \ldots$ for natural transformations; $\boldsymbol{\mathcal{C}}, \boldsymbol{\mathcal{D}}, \boldsymbol{\mathcal{E}}, \ldots$ for categories; and $\mathbb{T}, \mathbb{T}', \ldots$ for monads.

We refer to objects and morphisms of some category $\boldsymbol{\mathcal{C}}$ as $\boldsymbol{\mathcal{C}}$-objects and $\boldsymbol{\mathcal{C}}$-morphisms and denote the classes of all objects and all morphisms of $\boldsymbol{\mathcal{C}}$ by $\mathrm{Ob}\,\boldsymbol{\mathcal{C}}$ and $\mathrm{Mor}\,\boldsymbol{\mathcal{C}}$. The subclass of all $\boldsymbol{\mathcal{C}}$-morphisms to A from B is denoted by $\boldsymbol{\mathcal{C}}(A, B)$.[4]

We denote the meta-category of all categories (with functors as morphisms) by **CAT** and the meta-category of all functors to $\boldsymbol{\mathcal{C}}$ from $\boldsymbol{\mathcal{D}}$ (with natural transformations as morphisms) by $\boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{D}}}$ (called *functor category*). We will almost always omit the word *meta* since it will make no difference for what we are doing. For the endofunctor category we use the abbreviations $\mathbf{End}\,\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{C}}}$ and $\mathbf{End^2}\,\boldsymbol{\mathcal{C}} = \mathbf{End}(\mathbf{End}\,\boldsymbol{\mathcal{C}})$.

[4] Notice, that this is more often denoted by $\boldsymbol{\mathcal{C}}(B, A)$ or $\mathrm{Hom}_{\boldsymbol{\mathcal{C}}}(B, A)$. Our notation is consistent with arrows pointing to the left which we use.

For every object A we denote the identity morphism by id_A or just id. The composition in a category is usually denoted by $f \cdot g$ or $\bigodot_i f_i$. The only exception will be the vertical composition of natural transformations denoted by $\sigma * \tau$ (Definition and Lemma 4.1.3.7) in order to distinguish it from the horizontal composition $\sigma \cdot \tau$. Moreover, we define the k^{th} power of a morphism f by $f^k = \bigodot_{i=1}^{k} f$.

We denote coproducts by $A + B$ or $\coprod_i A_i$, products by $A \times B$ or $\prod_i A_i$, and exponents by $A \Leftarrow B$ or A^B and the according evaluation morphism by $ev : A \leftarrow A^B \times B$. The symbols for (co)products and exponents will also be used for the related functors (*e.g.* $\Leftarrow : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{C}}^{\text{op}}$) where we write the bifunctors $+$, $\times$, and $\Leftarrow$ as infix binary operators. We will denote the pointwise lifting of these bifunctors to the functor category by the same symbol (*e.g.* $(\mathsf{F} + \mathsf{G})f = \mathsf{F}f + \mathsf{G}f$).

For every category $\boldsymbol{\mathcal{C}}$ we denote the identity functor by $\mathsf{Id}_{\boldsymbol{\mathcal{C}}}$ or just Id. For every object A we denote the constant functor which maps onto id_A by $\underline{A}$.

Part I

Introduction to tree transducers and category theory

3 Tree transducers

The concept of top-down tree transducers was introduced by [Rou68, Rou70] and [Tha70]. A tree transducer is a tree automaton with output. Its semantics is a function mapping terms onto terms. The syntax of a tree transducer is a system of equations (called rules) , describing how to read single symbols from the input term and how to produce a part of the output term. Moreover, a tree transducer may have *states*, *context arguments*, *etc.* A tree transducer can also be regarded as a syntacticly restricted functional program.

3.1 Syntax

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 thesis.

The syntax of a tree transducer is mainly given as a set of rules. We distinguish several classes of tree transducers:

3.1.1 Top-down tree transducers

3.1.1.1 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 *tdtt* and the subclass of all top-down tree transducers with output alphabet Δ and input alphabet Σ by $tdtt(\Delta, \Sigma)$. ◇

3.1.1.2 Example (top-down tree transducer). We define the top-down tree transducer $T_{\mathrm{zigzag}} = (Q, \Sigma, \Delta, zig, R)$ where $Q = \{zig, zag\}$; $\Sigma = \{\alpha^{(0)}, \sigma^{(2)}\}$; $\Delta =$

$\{N^{(0)}, A^{(1)}, B^{(1)}\}$ and

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

3.1.2 Macro tree transducers

We will restate the definition of top-down tree transducers (Definition 3.1.1.1) using notions of category theory in Definition 6.1.2.2. All the other syntactic classes of tree transducers (like homomorphism tree transducers, ((simple) basic) macro tree transducers, *etc.* are defined that way in Subsection 6.1.2. This makes sense, since Chapter 5 is on top-down tree transducers only.

Let us just give an example to get an idea what a macro tree transducer looks like: It is similar to a top-down tree transducer, but has got *context arguments* in addition.

3.1.2.1 Example (macro tree transducer). We use $\Delta = \{N^{(0)}, A^{(1)}, B^{(1)}\}$ again. Then

$$\begin{array}{lcl} rev\,N\,y & \to & y \\ rev\,(A\,x)\,y & \to & rev\,x\,(A\,y) \\ rev\,(B\,x)\,y & \to & rev\,x\,(B\,y) \end{array}$$

are the rules of a *macro tree transducer* with one context argument y. In particular this is a *pure basic macro tree transducer* as we will learn later in Definition 6.1.2.5.

The semantics of this tree transducer is a function which reverses a list of As and Bs. The context argument y is used to accumulate the result. ◇

3.2 Semantics

The semantics of a tree transducer with output alphabet Δ and input alphabet Σ is a function $\mathsf{T}_\Delta\emptyset \leftarrow \mathsf{T}_\Sigma\emptyset$.

3.2.1 Operational semantics

The operational semantics of a tree transducer is defined by a term reduction system: Let T be a top-down tree transducer with rules $\mapsto_T \subseteq \mathsf{T}_\Gamma X \times \mathsf{T}_\Gamma X$ where $\Gamma = \Sigma + Q + \Delta$. Let $z \notin X$. We define the **transition relation** $\Rightarrow_T \subseteq \mathsf{T}_\Gamma\emptyset \times \mathsf{T}_\Gamma\emptyset$ by

$$\begin{array}{rcl} t \Rightarrow_T t' & \iff & \exists\,\hat{t} \in \mathsf{T}_\Gamma^{ctx}\{z\}.\ \exists\,i : \mathsf{T}_\Gamma\emptyset \leftarrow X. \\ & & \quad t = \big[[ix/x]_{x\in X}\,lhs\ /\ z\big]\hat{t} \\ & & \wedge\ lhs \mapsto_T rhs \\ & & \wedge\ t' = \big[[ix/x]_{x\in X}\,rhs\ /\ z\big]\hat{t}. \end{array}$$

It is possible to prove that $\Rightarrow_T$ is confluent and terminating (see *e.g.* [FV98]). Now we define the relation $\Rightarrow_T^* \subseteq \mathsf{T}_\Gamma\emptyset \times \mathsf{T}_\Gamma\emptyset$ to be the reflexive and transitive closure of $\Rightarrow_T$. Finally we define $\Downarrow_T \subseteq \mathsf{T}_\Gamma\emptyset \times \mathsf{T}_\Gamma\emptyset$ by

$$t \Downarrow_T t' \iff t \Rightarrow_T^* t' \wedge \forall t'' \in \mathsf{T}_\Gamma\emptyset.\, t' \Rightarrow_T^* t'' \implies t' = t''.$$

Intuitively $\Rightarrow_T$ performs one computation step, $\Rightarrow_T^*$ an arbitrary (but finite) number of computation steps, and finally $\Downarrow_T$ performs as many computation steps as possible. Since $\Rightarrow_T$ is confluent and terminating, the relation $\Downarrow_T$ is in fact a function. It is easy to see, that this function maps terms from $\mathsf{Q}(\mathsf{T}_\Sigma\emptyset)$ to $\mathsf{T}_\Delta\emptyset$. We use this function to define the operational semantics $[\![\,.\,]\!]^{\mathrm{op}} : \mathsf{T}_\Delta\emptyset \leftarrow \mathsf{T}_\Sigma\emptyset$ by

$$[\![T]\!]^{\mathrm{op}} t = t' \iff q_0 t \Downarrow_T t'.$$

An important property of the operational semantics of a tree transducer is that it is *compositional*. A function f on terms is called **compositional** (or **syntax directed** [FV98]) if the value of f applied to a term t only depends on the values of f applied to subterms of t.

We will define a denotational semantics in Definition 6.3.1.2 in such a way, that it will be the unique function which satisfies certain properties. One of these properties is to be *compositional* as explained in Definition 6.3.1.2 (ii). Thus the only thing we need to know about the operational semantics in Chapter 6 is the fact that it is compositional. In Chapter 5 we will use the following denotational semantics:

3.2.2 Denotational semantics

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

$$[\![T]\!] : T_\Delta\emptyset \leftarrow T_\Sigma\emptyset$$

computed by T is defined by $[\![T]\!] = [\![T]\!]_{q_0}$ where

$$\begin{aligned} &\forall q \in Q.\, \forall k \in \mathbb{N}_0.\, \forall \sigma \in \Sigma^{(k)}.\, \forall t_1, \ldots, t_k \in T_\Sigma\emptyset. \\ &[\![T]\!]_q(\sigma(t_1, \ldots, t_k)) = [[\![T]\!]_p\, t_j / p x_j]_{\substack{p \in Q \\ x_j \in X_k}} (\mathrm{rhs}_{R,\sigma}\, q). \end{aligned}$$

Since the number of symbols in the term $\sigma(t_1, \ldots, t_k)$ is finite and for every $p \in Q$ the function $[\![T]\!]_p$ 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 $[\![T]\!]_q$ is well defined. We call the function $[\![\,.\,]\!] : (T_\Delta\emptyset)^{T_\Sigma\emptyset} \leftarrow tdtt(\Delta, \Sigma)$ the **denotational semantics** of a top-down tree transducer. $\diamond$

3.2.2.2 Theorem ([FV98] Theorem 3.25**).** Let T be a top-down tree transducer. Then $[\![T]\!] = [\![T]\!]^{\mathrm{op}}$. $\diamond$

3.2.2.3 Definition (top-down tree transformation). We denote the image class of the semantics function $[\![\,.\,]\!]$ by TOP, *i.e.* $TOP = \{[\![T]\!] \mid T \in tdtt\}$. The elements of TOP are called **top-down tree transformations**. ◇

3.2.2.4 Example (computed tree transformation). The tree transformation computed by the top-down tree transducer T_{zigzag} from Example 3.1.1.2 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 $t_1, t_2 \in T_\Sigma\emptyset$:

$$\begin{aligned}
&[\![T_{\text{zigzag}}]\!](\sigma(\sigma(t_1,\alpha),t_2))\\
&= [\![T_{\text{zigzag}}]\!]_{zig}(\sigma(\sigma(t_1,\alpha),t_2))\\
&= A([\![T_{\text{zigzag}}]\!]_{zag}(\sigma(t_1,\alpha)))\\
&= A(B([\![T_{\text{zigzag}}]\!]_{zig}\alpha))\\
&= A(BN).
\end{aligned}$$ ◇

3.2.2.5 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) $q_0 = hq_0'$ and

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

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 $T \cong T'$. ◇

3.2.2.6 Lemma (homomorphisms preserve semantics). Let $T, T' \in tdtt(\Delta, \Sigma)$ be top-down tree transducers. Then:

$$\frac{\exists\, h : T \leftarrow T'}{[\![T]\!] = [\![T']\!]}.$$

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:

$$\forall\, t \in T_\Sigma\emptyset.\ \forall\, q' \in Q'.\ [\![T]\!]_{hq'}\, t = [\![T']\!]_{q'}\, t$$

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

$$\forall\, j \in \{1, \ldots, k\}.\ \forall\, p \in Q'.\ [\![T]\!]_{hp} t_j = [\![T']\!]_p t_j.$$

Then we calculate:

$$\begin{aligned}
&[\![T]\!]_{hq'}(\sigma(t_1, \ldots, t_k)) \\
=&[[\![T]\!]_p t_j / p x_j]_{\substack{p \in Q \\ x_j \in X_k}} (\mathrm{rhs}_{R,\sigma}(hq')) \\
=&[[\![T]\!]_p t_j / p x_j]_{\substack{p \in Q \\ x_j \in X_k}} \left([hp\, x / px]_{\substack{p \in Q' \\ x \in X_k}} (\mathrm{rhs}_{R',\sigma}\, q')\right) \\
=&[[\![T]\!]_{hp} t_j / p x_j]_{\substack{p \in Q' \\ x_j \in X_k}} (\mathrm{rhs}_{R',\sigma}\, q') \\
=&[[\![T']\!]_p t_j / p x_j]_{\substack{p \in Q' \\ x_j \in X_k}} (\mathrm{rhs}_{R',\sigma}\, q') \\
=&[\![T]\!]_{q'}(\sigma(t_1, \ldots, t_k)).
\end{aligned}$$

And thus: $[\![T]\!] = [\![T]\!]_{q_0} = [\![T]\!]_{hq_0'} = [\![T']\!]_{q_0'} = [\![T']\!]$. ∎

3.3 Composition of tree transducers

3.3.1 Composition of individual tree transducers

The idea for the following construction is simple: In order to compose two top-down tree transducers, we use the right hand sides of the producer as input for the consumer. To do so, we need to extend the definition of the consumer, because it has to be able to read states of the producer as input symbols. The results of these computations are the right hand sides of a new top-down tree transducer. Then we can prove that the semantics of this new top-down tree transducer is in fact the composition of the semantics of producer and consumer.

3.3.1.1 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:

$$T_2' = \big(Q, \Delta \uplus \{(px)^{(0)}\}_{\substack{p\in P\\ x\in X_r}}, \Gamma \uplus \{((q,p)x)^{(0)}\}_{\substack{q\in Q\\ p\in P\\ x\in X_r}}, q_0, R_2'\big) \quad \text{where}$$
$$r = \max_{\sigma\in\Sigma}(\operatorname{rank}_\Sigma \sigma) \quad \text{and}$$
$$R_2' = R_2 \uplus \big\{q(px) \to (q,p)x\big\}_{\substack{q\in Q\\ p\in P\\ x\in X_r}}.$$

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

$$T_2 \cdot T_1 = \big(Q \times P, \Sigma, \Gamma, (q_0, p_0), R\big) \quad \text{where}$$
$$R = \big\{(q,p)(\sigma(x_1, \ldots, x_{\operatorname{rank}_\Sigma \sigma})) \to [\![T_2']\!]_q(\operatorname{rhs}_{R_1,\sigma}\, p)\big\}_{\substack{q\in Q\\ p\in P\\ \sigma\in\Sigma}}.$$

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. ◇

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

$$\frac{T_2 \in tdtt(\Gamma, \Delta) \qquad T_1 \in tdtt(\Delta, \Sigma)}{[\![T_2]\!] \cdot [\![T_1]\!] = [\![T_2 \cdot T_1]\!]}.$$

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

3.3.1.3 Example (syntactic composition). Consider the top-down tree transducer T_{zigzag} from Example 3.1.1.2 and the top-down tree transducer $T_{\text{bin}} = (Q', \Delta, \Sigma, bin, R')$ where $Q' = \{bin\}$, Σ and Δ are defined as in Example 3.1.1.2, and

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

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.* $[\![T_{\text{bin}}]\!]\big(A(BN)\big) = \sigma(\sigma(\alpha,\alpha),\sigma(\alpha,\alpha))$. We may construct the following syntactic composition:

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

where

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

Let $t_1, t_2 \in T_\Sigma\emptyset$ 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 3.2.2.4), thus we have:

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

In Example 3.2.2.4 we saw that $[\![T_{\text{zigzag}}]\!]t = A(B(N))$. Together with the example for T_{bin} from above we get:

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

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

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

◇

The construction from Definition 3.3.1.1 is also applicable to compose a macro tree transducer with a top-down tree transducer, where the top-down tree transducer is the producer. However, if the macro tree transducer is the producer then the construction fails: It is by no means obvious how the consuming top-down tree transducer should operate on context variable occurrences in the right hand side of the producer. See [Voi01] for a construction to compose two restricted macro tree transducers (and in particular one macro and one top-down tree transducer in either succession).

3.3.2 Composition of classes of tree transformations

For two classes of tree transformations A and B we define the composition $A \cdot B = \big\{a \cdot b \;\big|\; a \in A \;\wedge\; b \in B \;\wedge\; \operatorname{dom} a = \operatorname{cod} b\big\}$.

3.3.2.1 Lemma (identities are top-down tree transducers). For every ranked alphabet Σ:

$$id_{T_\Sigma\emptyset} \in TOP.$$

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

$$\begin{aligned} T_{id} &= (\{q^{(1)}\}, \Sigma, \Sigma, q, R) \quad \text{where} \\ R &= \{q(\sigma(x_1, \ldots, x_{\mathrm{rank}_\Sigma\,\sigma})) \to \sigma(qx_1, \ldots, qx_{\mathrm{rank}_\Sigma\,\sigma})\}_{\sigma\in\Sigma} \end{aligned}$$

is the identity function, *i.e.* $[\![T_{id}]\!] = id_{T_\Sigma\emptyset}$. ■

3.3.2.2 Corollary. From Theorem 3.3.1.2 and Lemma 3.3.2.1 we obtain that the class of top-down tree transducers is closed under composition:

$$TOP \cdot TOP = TOP.$$ ◇

In Subsection 6.5.2 we will see more compositions of different classes of tree transformations.

4 Category theory

Consider all functions to a set A from A. Then the composition of such functions is associative and there exists an identity function on A. It is easy to see, that this gives rise to a monoid $(A^A, id_A, \cdot)$. This is the archetype of all monoids, which gave the inspiration to define the notion of an abstract monoid. Many other algebraic structures (like groups, fields, vector-spaces, *etc.*) are monoids (or combinations of two monoids) satisfying some additional axioms. However, functions may not be composed in general: The functions $f : A \leftarrow B$ and $g : C \leftarrow D$ may only be composed, if $B = C$[1]. So composition is a partial binary operation, which is defined iff the codomain of the producer (*i.e.* the first or right function) and the domain of the consumer (*i.e.* the second or left function) are equal. Moreover, for every set we have an identity function. A *category* is an abstraction of this situation. In other words: A category is a generalization of a monoid, where composition is partial and we may have more than one unit[2].

4.1 Basic Definitions and Theorems

4.1.1 Categories

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

$$\forall\, A, B \in \operatorname{Ob}\mathcal{C}.\ \mathcal{C}(A,B) = \left\{ f \in \operatorname{Mor}\mathcal{C} \;\middle|\; A = \operatorname{cod} f \ \wedge\ \operatorname{dom} f = B \right\}$$

and the ternary relation $(\bullet : \bullet \leftarrow_{\mathcal{C}} \bullet) \subseteq \operatorname{Mor}\mathcal{C} \times \operatorname{Ob}\mathcal{C} \times \operatorname{Ob}\mathcal{C}$ by

$$\forall\, A, B \in \operatorname{Ob}\mathcal{C}.\ f : A \leftarrow_{\mathcal{C}} B \iff A \xleftarrow{f}_{\mathcal{C}} B \iff f \in \mathcal{C}(A,B)$$

which we will write just as

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

[1] or whenever $B \subseteq C$ which is a matter of taste and definition.

[2] Notice, that it is possible to prove that the unit of a monoid is unique. A category may have more that one unit, because the composition is partial. Since then the aforementioned proof does not work anymore.

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

$$\frac{f, g, h \in \operatorname{Mor}\mathcal{C} \qquad f \cdot g = h}{\operatorname{dom} f = \operatorname{cod} g} \tag{typing}$$

$$\frac{f : A \leftarrow B \qquad g : B \leftarrow C}{f \cdot g : A \leftarrow C} \tag{composition}$$

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

$$\frac{A \in \operatorname{Ob}\mathcal{C}}{id_A : A \leftarrow A} \qquad \frac{f : A \leftarrow B}{f \cdot id_B = f = id_A \cdot f} \tag{identity}$$

◇

4.1.1.2 Lemma (hom-classes partition morphism class). For every category $\mathcal{C}$ the class $\{\mathcal{C}(A, B) \mid A, B \in \operatorname{Ob}\mathcal{C}\}$ is a partition of $\operatorname{Mor}\mathcal{C}$, *i.e.* $\operatorname{Mor}\mathcal{C} = \biguplus_{A,B \in \operatorname{Ob}\mathcal{C}} \mathcal{C}(A, B)$. ◇

4.1.1.3 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.* $\operatorname{dom}, \operatorname{cod} : \boldsymbol{\mathcal{P}ot}(\operatorname{Ob}\mathcal{C}) \leftarrow \operatorname{Mor}\mathcal{C}$.[3] We use the same notations for pre-categories as defined for categories in Definition 4.1.1.1. For a pre-category $\mathcal{C}$ we like to mention the definition of hom-classes

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

and the typing axiom:

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

◇

4.1.1.4 Note (pre-categories give rise to categories). For every pre-category $\mathcal{C}$ we can construct a category $\mathcal{C}'$ by

$$\operatorname{Ob}\mathcal{C}' = \operatorname{Ob}\mathcal{C} \qquad \forall\, A, B \in \operatorname{Ob}\mathcal{C}.\ \mathcal{C}'(A, B) = \{(A, f, B) \mid f : A \leftarrow_{\mathcal{C}} B\}$$

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. ◇

category	objects	morphisms
$\boldsymbol{Set}$	all sets	all set functions
$\boldsymbol{Set}_{\aleph_0}$	all countable sets	all set functions between countable sets
$\boldsymbol{Set}^{\Sigma}$	all Σ-algebras	all Σ-algebra homomorphisms
$\boldsymbol{Top}$	all topological spaces	all continuous functions

Table 4.1: Some categories

4.1.1.5 Example (category). We list some categories in Table 4.1.

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

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

i.e. $\boldsymbol{\mathcal{C}}^{\mathrm{op}}$ has the same objects and morphisms as $\boldsymbol{\mathcal{C}}$, but dom and cod are exchanged and the composition is commuted. The transformation from $\boldsymbol{\mathcal{C}}$ to $\boldsymbol{\mathcal{C}}^{\mathrm{op}}$ may be viewed as the reversion of all morphism arrows. Notice, that $\boldsymbol{\mathcal{C}}^{\mathrm{op}}$ is indeed a category, because of a symmetry of the axioms of category theory (Definition 4.1.1.1). If we do this twice, then obviously we receive the original category:

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

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

$$\Big(\text{for every category } \boldsymbol{\mathcal{C}}.\ A(\boldsymbol{\mathcal{C}})\Big) \quad \Longleftrightarrow \quad \Big(\text{for every category } \boldsymbol{\mathcal{C}}.\ A^{\mathrm{op}}(\boldsymbol{\mathcal{C}})\Big)$$

[3] where $\boldsymbol{\mathcal{P}ot}$ denotes the power-class operator defined by $\boldsymbol{\mathcal{P}ot}\, C = \{P \mid P \subseteq C\}$ for every class C

Proof.

$$\begin{aligned} & \forall \boldsymbol{\mathcal{C}}.\ A(\boldsymbol{\mathcal{C}}) \\ \iff & \quad \{ \text{ since } (\boldsymbol{.})^{\mathrm{op}} \text{ is a bijection } \} \\ & \forall \boldsymbol{\mathcal{C}}.\ A(\boldsymbol{\mathcal{C}}^{\mathrm{op}}) \\ \iff & \quad \{ \text{ definition of the dual predicate } \} \\ & \forall \boldsymbol{\mathcal{C}}.\ A^{\mathrm{op}}(\boldsymbol{\mathcal{C}}) \end{aligned}$$

■

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. Sometimes notions and dual notions are named systematicly: The dual of *foo* should be called ***co****foo*. ◇

4.1.1.7 Definition (isomorphism). Let $\boldsymbol{\mathcal{C}}$ be a category and $A, B \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$. 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}. The set of all isomorphisms of $\mathrm{Mor}\,\boldsymbol{\mathcal{C}}$ is denoted by $\mathrm{Iso}\,\boldsymbol{\mathcal{C}}$. Two objects $A, B \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$ such that

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

are called **isomorphic**, and we write:

$$A \cong B.$$

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

4.1.2 Functors

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

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

which satisfy the following axioms:

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

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

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

In this case we write:

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

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 $\forall\, A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}.\ \mathsf{F}A = \mathrm{dom}(\mathsf{F}id_A)$.

For every functor $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ we define the **dual functor** $\mathsf{F}^{\mathrm{op}} : \boldsymbol{\mathcal{C}}^{\mathrm{op}} \leftarrow \boldsymbol{\mathcal{D}}^{\mathrm{op}}$, 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 $\mathsf{E} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$, which maps a category on itself, is called an **endofunctor**.

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

$$\mathsf{Id} : \left\{ \begin{array}{ccccl} \boldsymbol{\mathcal{C}} & \leftarrow & \boldsymbol{\mathcal{C}} & & \\ A & \leftarrow\!\shortmid & A & & \forall\, A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}} \\ f & \leftarrow\!\shortmid & f & & \forall\, f \in \mathrm{Mor}\,\boldsymbol{\mathcal{C}} \end{array} \right.$$

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

$$\underline{A} : \left\{ \begin{array}{ccccl} \boldsymbol{\mathcal{C}} & \leftarrow & \boldsymbol{\mathcal{D}} & & \\ A & \leftarrow\!\shortmid & B & & \forall\, B \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}} \\ id_A & \leftarrow\!\shortmid & f & & \forall\, f \in \mathrm{Mor}\,\boldsymbol{\mathcal{D}}. \end{array} \right.$$

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

4.1.2.2 Observation (constant functor fusion). The constant functors absorb other functors in compositions. More precisely: Let $\boldsymbol{\mathcal{C}}$ be a category and $A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$. For every functor F from $\boldsymbol{\mathcal{C}}$ and every functor G to $\boldsymbol{\mathcal{C}}$ holds

(i) $\mathsf{F} \cdot \underline{A} = \underline{\mathsf{F}A}$ and

(ii) $\underline{A} \cdot \mathsf{G} = \underline{A}$. ◇

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

[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.

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

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

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

holds, $\boldsymbol{\mathcal{D}}$-identities are also $\boldsymbol{\mathcal{C}}$-identities, and $\boldsymbol{\mathcal{D}}$-composition is the restriction of $\boldsymbol{\mathcal{C}}$-composition to $\operatorname{Mor}\boldsymbol{\mathcal{D}}$. If in addition

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

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

$$\mathsf{E} : \left\{ \begin{array}{ccll} \boldsymbol{\mathcal{C}} & \leftarrow & \boldsymbol{\mathcal{D}} & \\ A & \mapsto & A & \forall\, A \in \operatorname{Ob}\boldsymbol{\mathcal{D}} \\ f & \mapsto & f & \forall\, f \in \operatorname{Mor}\boldsymbol{\mathcal{D}} \end{array} \right.$$

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

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

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

and

$$\prod_{i\in I}\boldsymbol{\mathcal{C}}_i\big((A_i)_{i\in I},(B_i)_{i\in I}\big) = \Big\{(f_i)_{i\in I} \;\Big|\; \forall\, i \in I.\ f_i : A_i \leftarrow_{\boldsymbol{\mathcal{C}}_i} B_i\Big\}$$

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

$$\mathsf{P}_j : \left\{ \begin{array}{ccll} \boldsymbol{\mathcal{C}}_j & \leftarrow & \prod_{i\in I}\boldsymbol{\mathcal{C}}_i & \\ A_j & \mapsto & (A_i)_{i\in I} & \forall\,(A_i)_{i\in I} \in \operatorname{Ob}(\prod_{i\in I}\boldsymbol{\mathcal{C}}_i) \\ f_j & \mapsto & (f_i)_{i\in I} & \forall\,(f_i)_{i\in I} \in \operatorname{Mor}(\prod_{i\in I}\boldsymbol{\mathcal{C}}_i), \end{array} \right.$$

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

4.1.2.6 Definition (hom-functor). Let $\boldsymbol{\mathcal{C}}$ be a category. We introduce the **hom-functor** of $\boldsymbol{\mathcal{C}}$

$$\boldsymbol{\mathcal{C}}(\boldsymbol{.},\boldsymbol{.}) : \left\{ \begin{array}{ccll} \boldsymbol{\mathcal{S}et} & \leftarrow & \boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{C}}^{\mathrm{op}} & \\ \boldsymbol{\mathcal{C}}(A,B) & \mapsto & (A,B) & \forall\, A,B \in \operatorname{Ob}\boldsymbol{\mathcal{C}} \\ (f\cdot p\cdot g \mapsto p) & \mapsto & (f,g) & \forall\, f \in \operatorname{Mor}\boldsymbol{\mathcal{C}} \quad \forall\, g \in \operatorname{Mor}\boldsymbol{\mathcal{C}}^{\mathrm{op}}, \end{array} \right.$$

and as a special case the so called **covariant hom-functors**

$$\boldsymbol{\mathcal{C}}(\boldsymbol{.},B) : \boldsymbol{\mathcal{S}et} \leftarrow \boldsymbol{\mathcal{C}} \quad \forall\, B \in \operatorname{Ob}\boldsymbol{\mathcal{C}}.$$ ◇

4.1.3 Natural transformations

4.1.3.1 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

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

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

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

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

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

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

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

4.1.3.3 Definition and Lemma (horizontal composition). 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 4.1.3.1. ∎

This composition of natural transformations is often called **horizontal composition**. We will see *vertical composition* of natural transformations in Definition and Lemma 4.1.3.7. ◇

4.1.3.4 Definition (composition of morphisms and natural transformations). Let $\mathsf{F}, \mathsf{G} : \mathcal{C} \leftarrow \mathcal{D}$ be functors, $\tau : \mathsf{F} \dot{\leftarrow} \mathsf{G}$, and $f : \mathsf{G}A \leftarrow \mathsf{F}B$. It is common practice to drop the subscript of τ in composition expressions

(i) $\tau \cdot f = \tau_A \cdot f$,

(ii) $f \cdot \tau = f \cdot \tau_B$

since it can be derived from the type of f. ◇

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

$$\operatorname{Ob} \boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{D}}} = \left\{ \mathsf{F} \;\middle|\; \mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}} \right\}$$

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

$$\boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{D}}}(\mathsf{F}, \mathsf{G}) = \left\{ \tau \;\middle|\; \tau : \mathsf{F} \dot{\leftarrow} \mathsf{G} \right\}$$

with the composition of natural transformations and for every $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ the identity $id_{\mathsf{F}} = (id_{\mathsf{F}A})_{A \in \operatorname{Ob} \boldsymbol{\mathcal{D}}} : \mathsf{F} \dot{\leftarrow} \mathsf{F}$. Furthermore we will use the abbreviations $\mathbf{End}\, \boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{C}}}$ and $\mathbf{End}^{\mathbf{2}}\, \boldsymbol{\mathcal{C}} = \mathbf{End}(\mathbf{End}\, \boldsymbol{\mathcal{C}})$. ◇

4.1.3.6 Definition and Lemma (functors preserve naturalness). Let $\boldsymbol{\mathcal{C}}$, $\boldsymbol{\mathcal{D}}$, and $\boldsymbol{\mathcal{E}}$ be categories and $\mathsf{F}, \mathsf{G} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ 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 $\mathsf{H} : \boldsymbol{\mathcal{E}} \leftarrow \boldsymbol{\mathcal{C}}$ and τ defined by

$$\forall\, A \in \operatorname{Ob} \boldsymbol{\mathcal{D}}.\ (\mathsf{H}\tau)_A = \mathsf{H}(\tau_A)$$

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 $\mathsf{H} : \boldsymbol{\mathcal{D}} \leftarrow \boldsymbol{\mathcal{E}}$ defined by

$$\forall\, A \in \operatorname{Ob} \boldsymbol{\mathcal{E}}.\ (\tau\mathsf{H})_A = \tau_{\mathsf{H}A}$$

is a natural transformation

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

Proof. Immediately by Definition 4.1.2.1 and Definition 4.1.3.1. ∎

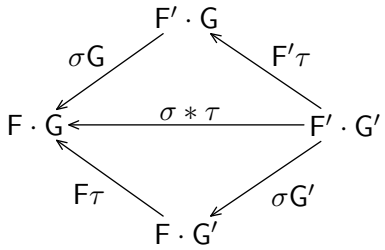
4.1.3.7 Definition and Lemma (vertical composition). Let $\boldsymbol{\mathcal{C}}$, $\boldsymbol{\mathcal{D}}$, and $\boldsymbol{\mathcal{E}}$ be categories, and

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

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:



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

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

$$
\begin{aligned}
& (\sigma\mathsf{G} \cdot \mathsf{F}'\tau)_A \\
= \quad & \{ \text{ Definition and Lemma 4.1.3.3 and Definition and Lemma 4.1.3.6 } \} \\
& \sigma_{\mathsf{G}A} \cdot \mathsf{F}'\tau_A \\
= \quad & \{ \text{ naturalness of } \sigma \text{ } \} \\
& \mathsf{F}\tau_A \cdot \sigma_{\mathsf{G}'A} \\
= \quad & \{ \text{ Definition and Lemma 4.1.3.3 and Definition and Lemma 4.1.3.6 } \} \\
& (\mathsf{F}\tau \cdot \sigma\mathsf{G}')_A
\end{aligned}
$$

■

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

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

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:

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

The latter equation is also called *middle-four exchange*[5]. Then the class of all natural transformations with horizontal and vertical composition is a *2-category*[6].

[5] A pair of categories sharing the same morphism class such that the middle-four exchange axiom holds is called a **double category**. Then the composition of one category can be extended to the object class of the other category and vice versa. This gives rise to two more categories (called the **vertical** and **horizontal edge category** of the double category) which share the same object class.

[6] A **2-category** is a double category (see preceding footnote) in which the vertical edge category is *discrete* (*i.e.* all morphisms are identities). The horizontal edge category of the 2-category of all natural transformations is **CAT**.

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

$$
\begin{aligned}
& \big((\sigma * \tau) \cdot (\sigma' * \tau')\big)_A \\
= \quad & \{ \text{ Definition and Lemma 4.1.3.7 and Definition and Lemma 4.1.3.6 } \} \\
& \sigma_{\mathsf{G}A} \cdot \mathsf{F}'\tau_A \cdot \mathsf{F}'\tau'_A \cdot \sigma'_{\mathsf{G}''A} \\
= \quad & \{ \mathsf{F}' \text{ is a functor } \} \\
& \sigma_{\mathsf{G}A} \cdot \mathsf{F}'(\tau_A \cdot \tau'_A) \cdot \sigma'_{\mathsf{G}''A} \\
= \quad & \{ \text{ naturalness of } \sigma' \ \} \\
& \sigma_{\mathsf{G}A} \cdot \sigma'_{\mathsf{G}A} \cdot \mathsf{F}''(\tau_A \cdot \tau'_A) \\
= \quad & \{ \text{ Definition and Lemma 4.1.3.7 and Definition and Lemma 4.1.3.6 } \} \\
& \big((\sigma \cdot \sigma') * (\tau \cdot \tau')\big)_A
\end{aligned}
$$

■

4.1.3.9 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 $A \in \operatorname{Ob} \boldsymbol{\mathcal{F}}$.

$$
\begin{aligned}
& \big((\sigma * \tau) * \varrho\big)_A \\
= \quad & \{ \text{ Definition and Lemma 4.1.3.7 and Definition and Lemma 4.1.3.6 } \} \\
& (\sigma * \tau)_{\mathsf{H}A} \cdot (\mathsf{F}' \cdot \mathsf{G}')\varrho_A \\
= \quad & \{ \text{ Definition and Lemma 4.1.3.7 and Definition and Lemma 4.1.3.6 } \} \\
& \sigma_{(\mathsf{G}\cdot\mathsf{H})A} \cdot \mathsf{F}'\tau_{\mathsf{H}A} \cdot (\mathsf{F}' \cdot \mathsf{G}')\varrho_A \\
= \quad & \{ \mathsf{F}' \text{ is a functor } \} \\
& \sigma_{(\mathsf{G}\cdot\mathsf{H})A} \cdot \mathsf{F}'(\tau_{\mathsf{H}A} \cdot \mathsf{G}'\varrho_A) \\
= \quad & \{ \text{ Definition and Lemma 4.1.3.7 and Definition and Lemma 4.1.3.6 } \} \\
& \sigma_{(\mathsf{G}\cdot\mathsf{H})A} \cdot \mathsf{F}'(\tau * \varrho)_A \\
= \quad & \{ \text{ Definition and Lemma 4.1.3.7 and Definition and Lemma 4.1.3.6 } \} \\
& \big(\sigma * (\tau * \varrho)\big)_A
\end{aligned}
$$

■

4.1.3.10 Definition (hom-class restricted functor). Let $\mathcal{C}$ and $\mathcal{D}$ be categories, $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{D}$ be a functor, and $A, B \in \operatorname{Ob}\mathcal{D}$. We denote the function, which we obtain from the function

$$\mathsf{F} : \operatorname{Mor}\mathcal{C} \leftarrow_{\mathbf{CLASS}} \operatorname{Mor}\mathcal{D}$$

by restricting the domain of F to the hom-class $\mathcal{D}(A, B) \subseteq \operatorname{Mor}\mathcal{D}$ and the codomain of F to the hom-class $\mathcal{C}(\mathsf{F}A, \mathsf{F}B) \subseteq \operatorname{Mor}\mathcal{D}$ by

$$\mathsf{F}_{(A,B)} : \mathcal{C}(\mathsf{F}A, \mathsf{F}B) \leftarrow_{\mathcal{S}et} \mathcal{D}(A, B).$$

$\mathsf{F}_{(A,B)}$ is called the **hom-class restriction** (or the **localization**) of F by (A, B). Using the multiplicativity of the functor F, it is easy to see that $\mathsf{F}_{(\bullet,\bullet)}$ is a natural transformation

$$\mathsf{F}_{(\bullet,\bullet)} : \mathcal{C}(\mathsf{F}\bullet, \mathsf{F}^{\mathrm{op}}\bullet) \dot{\leftarrow} \mathcal{D}(\bullet, \bullet)$$

where $\mathcal{C}(\bullet, \bullet)$ and $\mathcal{D}(\bullet, \bullet)$ are the hom-functors from Definition 4.1.2.6. ◇

4.1.4 Initial and final objects

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

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

or equivalently

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

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

$$¡_{(\bullet)\leftarrow 0} : \operatorname{Mor}\mathcal{C} \leftarrow \operatorname{Ob}\mathcal{C}$$

by

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

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

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

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

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

$$!_{(\bullet)\rightarrow 1} : \operatorname{Mor}\mathcal{C} \leftarrow \operatorname{Ob}\mathcal{C}$$

with

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

is called the **mediator** of 1. The morphism $!_{A\to 1}$ 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 $!_{A\to 1}$.

A $\mathcal{C}$-object that is initial and final as well is called a **null-object** of $\mathcal{C}$. ◇

4.1.4.2 Example (initial and final object). In the category $\mathcal{S}et$ the empty set $\emptyset$ is an initial object and every singleton set is a final object. ◇

4.1.4.3 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 $¡_{(\,.\,)\leftarrow 0}$ and $¡_{(\,.\,)\leftarrow 0'}$, respectively.

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

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

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

4.1.4.4 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 4.2 hold.

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

Table 4.2: Laws for comediators

And thus the dual laws in Table 4.3 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 4.1.4.3. The fusion law can be proven similarly: Because of $f \cdot ¡_B : A \leftarrow 0$ it is obvious from the (UP) that $f \cdot ¡_B = ¡_A$. ■

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 4.3: Laws for mediators

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

$$¡ : \mathsf{Id} \dot{\leftarrow} \underline{0}.$$

Proof. Since for every $A \in \mathrm{Ob}\,\mathcal{C}$ holds $¡_A : A \leftarrow 0$, the comediator is a transformation to Id from $\underline{0}$. It is also natural, because its naturalness condition (Definition 4.1.3.1) is equivalent to the fusion law (Table 4.2). ∎

4.1.5 (Co-)products

4.1.5.1 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 4.1 commutes.
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 4.4 in analogy to the laws in Table 4.3 from Lemma 4.1.4.4.

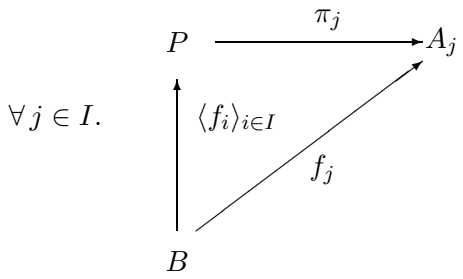


Figure 4.1: Product UP

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 4.4: Laws for products

Proof. The UP is equivalent to the definition of the product. The remaining laws follow in analogy to Lemma 4.1.4.3 and Lemma 4.1.4.4. ∎

4.1.5.2 Lemma (products in *Set*). The category ***Set*** 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:

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

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

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

is the respective pairing by verifying the UP of the product (Table 4.4). ∎

4.1.5.3 Definition and Lemma (coproduct). The dual notion to *product* is the notion of *coproduct*: Let $\mathcal{C}$ be a category, I a set, $(A_i)_{i\in I} \in \mathrm{Ob}\,\mathcal{C}^I$, and $C \in \mathrm{Ob}\,\mathcal{C}$. 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 $B \in \mathrm{Ob}\,\mathcal{C}$ 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 4.2 commutes:

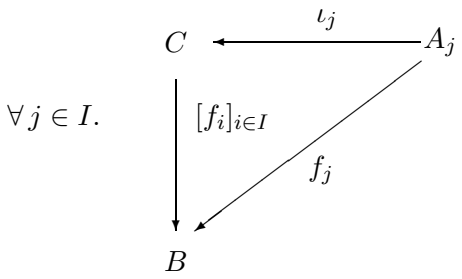


Figure 4.2: Coproduct UP

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 $(A_i)_{i\in I} \in \mathrm{Ob}\,\mathcal{C}^I$ 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.* $I = \{1, \ldots, n\}$, we write $A_1 + \cdots + A_n = \coprod_{i\in I} A_i$ and $[f_1, \ldots, f_n] = [f_i]_{i\in I}$. 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 $(A_i)_{i\in I} \in \mathrm{Ob}\,\mathcal{C}^I$ there exists a coproduct $\coprod_{i\in I} A_i$ in $\mathcal{C}$. We obtain the laws in Table 4.5 which are dual to the laws in Table 4.4.

Proof. Dually to Definition and Definition and Lemma 4.1.5.1. ∎

4.1.5.4 Lemma (coproducts in *Set*). The category ***Set*** has coproducts.

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

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

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 4.5: Laws for coproducts

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

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

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

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

is the respective copairing by verifying the UP of the coproduct (Table 4.5). ∎

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

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

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 4.6.

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 $\forall\, j.\ f_j, g_j, h_j \in \operatorname{Mor}\mathcal{C}$ such that $\forall\, i, j.\ \operatorname{dom} g_i = \operatorname{dom} g_j$	

Table 4.6: Laws for product functors

The projections are natural transformations:

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

Proof. The laws follow straightforward from the definition of the product functor and the laws for products from Table 4.4. The cancelation law is the naturalness condition for $\pi_j : \mathsf{P}_j \stackrel{\cdot}{\leftarrow} \prod$. ∎

4.1.5.6 Definition and Corollary (coproduct functor). This is dual to Definition and Definition and Corollary 4.1.5.5 so we will have nothing to prove. Let $\boldsymbol{\mathcal{C}}$ be a category which has finite coproducts. The **coproduct functor** is defined by

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

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 4.7.

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 $\forall\, j.\ f_j, g_j, h_j \in \mathrm{Mor}\, \boldsymbol{\mathcal{C}}$ such that $\forall\, i, j.\ \mathrm{cod}\, f_i = \mathrm{cod}\, f_j$	

Table 4.7: Laws for coproduct functors

The injections are natural transformations:

$$\iota_j : \coprod \stackrel{\cdot}{\leftarrow} \mathsf{P}_j. \qquad \diamond$$

4.1.5.7 Lemma ((co-)products in the functor category). Let $\boldsymbol{\mathcal{C}}$ and $\boldsymbol{\mathcal{D}}$ be categories such that $\boldsymbol{\mathcal{C}}$ has (finite) (co-)products. Then $\boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{D}}}$ 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 $(\mathsf{F}_i)_{i \in I} \in \mathrm{Ob}(\boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{D}}})^I$ be an I-family of $\boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{D}}}$-objects. The category $\boldsymbol{\mathcal{C}}$ 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 $\boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{D}}}$ where $\pi_i = \big((pi_i)_D\big)_{D \in \mathrm{Ob}\, \boldsymbol{\mathcal{D}}}$ 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 4.6. 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. ∎

4.1.5.8 Note. Let $\mathcal{C}$ be a category which has (finite) products. From Lemma 4.1.5.7 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 $\prod^n \mathsf{Id}_{\mathcal{C}} = \prod^n$. ◇

4.1.5.9 Lemma (faithful (co-)product functors). 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 (see Definition 4.2.1.1).

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:

$$\begin{aligned}
&\prod_{i\in I} f_i = \prod_{i\in I} f'_i \\
\Longrightarrow\ &\pi_j \cdot \prod_{i\in I} f_i \cdot \langle h_i\rangle_{i\in I} = \pi_j \cdot \prod_{i\in I} f'_i \cdot \langle h_i\rangle_{i\in I} \\
\Longrightarrow\ &\quad \{\ \text{cancelation for product(functors) Table 4.6 and Table 4.4}\ \} \\
&\quad f_j = f'_j.
\end{aligned}$$

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. ∎

4.1.6 Exponents

4.1.6.1 Definition and Lemma (exponent, application, abstraction, currying). Let $\mathcal{C}$ be a category with finite products. For every two objects $A, B \in \mathrm{Ob}\,\mathcal{C}$ we define the category $\mathcal{E}\mathit{xp}_{\mathcal{C}}(A,B)$ by

$$\mathrm{Ob}\big(\mathcal{E}\mathit{xp}_{\mathcal{C}}(A,B)\big) = \bigcup_{C\in\mathrm{Ob}\,\mathcal{C}} \mathcal{C}(A, C\times B)$$

and $\forall\, f, g \in \mathrm{Ob}\big(\mathcal{E}\mathit{xp}_{\mathcal{C}}(A,B)\big)$ where $f : A \leftarrow C\times B$ and $g : A \leftarrow D\times B$ with $C, D \in \mathrm{Ob}\,\mathcal{C}$:

$$\mathcal{E}\mathit{xp}_{\mathcal{C}}(A,B)(f,g) = \big\{h \in \mathcal{C}(C,D) \ \big|\ f\cdot(h\times id_B) = g\big\}.$$

If it has a final object, we denote it by

$$ev_{(A,B)} = 1_{\boldsymbol{\mathcal{E}xp}_{\mathcal{C}}(A,B)} : A \leftarrow_{\mathcal{C}} A^B \times B$$

and call the underlying $\mathcal{C}$-object A^B an **exponent** of A and B with **application** $ev_{(A,B)}$. The unique morphism

$$\operatorname{curry} f = !_f : A^B \leftarrow_{\mathcal{C}} C \quad (\text{where } C \in \operatorname{Ob}\mathcal{C} \text{ such that } f : A \leftarrow C \times B)$$

for each $f \in \operatorname{Ob}\big(\boldsymbol{\mathcal{E}xp}_{\mathcal{C}}(A,B)\big)$ is called the **abstraction** or **currying**[7]. The function

$$\operatorname{uncurry} : \operatorname{Mor}\mathcal{C} \leftarrow \operatorname{Ob}\big(\boldsymbol{\mathcal{E}xp}_{\mathcal{C}}(A,B)\big)$$

defined by

$$\operatorname{uncurry} h = ev_{(A,B)} \cdot (h \times id_B) \quad \forall\, h \in \operatorname{Ob}\big(\boldsymbol{\mathcal{E}xp}_{\mathcal{C}}(A,B)\big),$$

is the inverse of curry and we have the following laws:

Laws for currying	
UP	$h = \operatorname{curry} f \iff ev_{(A,B)} \cdot (h \times id_B) = f$
reflection	$\operatorname{curry} ev_{(A,B)} = id_{A^B}$
fusion	$\operatorname{curry} f \cdot h = \operatorname{curry}\big(f \cdot (h \times id_B)\big)$
cancelation	$ev_{(A,B)} \cdot (\operatorname{curry} f \times id_B) = f$
where $h \in \operatorname{Mor}\mathcal{C}$ and $f \in \operatorname{Ob}\big(\boldsymbol{\mathcal{E}xp}_{\mathcal{C}}(A,B)\big)$	

Proof. Immediately by Definition 4.1.4.1. ∎

4.1.7 Initial algebras and catamorphisms

4.1.7.1 Definition (F-algebra). Let $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{C}$ be an endofunctor. We define the category $\mathcal{C}^{\mathsf{F}}$ of all **F-algebras** as follows: The object class is given by

$$\operatorname{Ob}(\mathcal{C}^{\mathsf{F}}) = \big\{\varphi \in \operatorname{Mor}\mathcal{C} \;\big|\; \exists\, A \in \operatorname{Ob}\mathcal{C}.\ \varphi : A \leftarrow \mathsf{F}A\big\}$$

and for every $\varphi, \varphi' \in \operatorname{Ob}(\mathcal{C}^{\mathsf{F}})$ where $\varphi : A \leftarrow \mathsf{F}A$ and $\varphi' : B \leftarrow \mathsf{F}B$ with $A, B \in \operatorname{Ob}\mathcal{C}$ we define the hom-class by

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

The identity and composition of $\mathcal{C}^{\mathsf{F}}$ are defined for every $\varphi : A \leftarrow \mathsf{F}A$, $\varphi' : A' \leftarrow \mathsf{F}A'$, $\varphi'' : A'' \leftarrow \mathsf{F}A''$, and every $f : A \leftarrow A'$ by

$$id_\varphi = (\varphi, id_A, \varphi),$$
$$(\varphi, f, \varphi') \cdot (\varphi', g, \varphi'') = (\varphi, f \cdot g, \varphi'').$$

[7] Named after Haskell B. Curry.

The morphisms of this category are called **F-algebra homomorphisms**. We define the forgetful functor $|\,.\,|^{\mathsf{F}} : \mathcal{C} \leftarrow \mathcal{C}^{\mathsf{F}}$ by

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

and

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

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 forgetful functor $|\,.\,|^{\mathsf{F}}$ maps an F-algebras onto its carrier. If $\mathcal{C}^{\mathsf{F}}$ 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 $\boldsymbol{\mu}\mathsf{F}$ and is called the **least fixed point**[8] of F.

The image of the uniquely mediating morphism

$$\forall\, \varphi \in \mathrm{Ob}(\mathcal{C}^{\mathsf{F}}).\ \mathbf{i}_{\varphi} : \varphi \leftarrow_{\mathcal{C}^{\mathsf{F}}} in_{\mathsf{F}}$$

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

$$|\mathbf{i}_{\varphi}|^{\mathsf{F}} : \mathrm{cod}_{\mathcal{C}}\, \varphi \leftarrow_{\mathcal{C}} \boldsymbol{\mu}\mathsf{F} \qquad \diamond$$

is called the **catamorphism** generated by φ (w.r.t. F) (from Greek $\varkappa\alpha\tau\alpha$, downwards) and is denoted by $(\!|\varphi|\!)_{\mathsf{F}}$, *i.e.* $\mathbf{i}_{\varphi} = (\varphi, (\!|\varphi|\!)_{\mathsf{F}}, in_{\mathsf{F}})$. The laws in Table 4.8 are a consequence of the laws in Table 4.2.

Laws for catamorphisms	
UP	$f = (\!\|\varphi\|\!)_{\mathsf{F}} \iff f \cdot in_{\mathsf{F}} = \varphi \cdot \mathsf{F}f$
reflection	$(\!\|in_{\mathsf{F}}\|\!)_{\mathsf{F}} = id_{\boldsymbol{\mu}\mathsf{F}}$
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 $\varphi, \varphi' \in \mathrm{Ob}\,\mathcal{C}^{\mathsf{F}}$	

Table 4.8: Laws for catamorphisms

The following equivalent definition is more descriptive: the $\mathcal{C}^{\mathsf{F}}$-object $in_{\mathsf{F}} : \boldsymbol{\mu}\mathsf{F} \leftarrow \mathsf{F}(\boldsymbol{\mu}\mathsf{F})$ 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 4.3 commutes.

[8] The reason is that it satisfies the fixed-point-equation $\boldsymbol{\mu}\mathsf{F} \cong \mathsf{F}(\boldsymbol{\mu}\mathsf{F})$, because initial algebras are always isomorphisms (Lemma 4.1.7.4). There exists an equivalent definition of the *least fixed point* of F as (carrier of) the initial object in the *category of fixed points of* F, *i.e.* the full subcategory of $\mathcal{C}^{\mathsf{F}}$ where the objects are $\mathcal{C}$-isomorphisms.

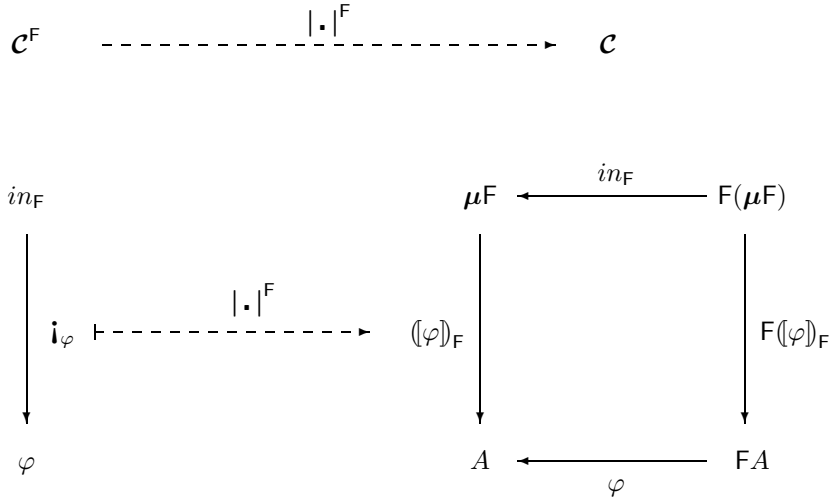


Figure 4.3: Catamorphism UP

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

$$\mathsf{F} : \left\{ \begin{array}{ccccc} \boldsymbol{\mathcal{S}et} & \leftarrow & \boldsymbol{\mathcal{S}et} & & \\ A \times A & \mapsto & A & \forall\, A \in \mathrm{Ob}\,\boldsymbol{\mathcal{S}et} \\ f \times f & \mapsto & f & \forall\, f \in \mathrm{Mor}\,\boldsymbol{\mathcal{S}et} \end{array} \right.$$

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

$$\varphi : A \leftarrow_{\boldsymbol{\mathcal{S}et}} A \times A$$

with an $A \in \mathrm{Ob}\,\boldsymbol{\mathcal{S}et}$. 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 $\varphi, \varphi' \in \mathrm{Ob}(\boldsymbol{\mathcal{S}et}^{\mathsf{F}})$ where $\varphi : A \leftarrow A \times A$ and $\varphi' : A' \leftarrow A' \times A'$. The underlying function $|f|^{\mathsf{F}} : A \leftarrow_{\boldsymbol{\mathcal{S}et}} A'$ of an $\boldsymbol{\mathcal{S}et}^{\mathsf{F}}$-morphism $f = (\varphi, |f|^{\mathsf{F}}, \varphi') : \varphi \leftarrow_{\boldsymbol{\mathcal{S}et}^{\mathsf{F}}} \varphi'$ 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). ◇

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

Proof. We will show that

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

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

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

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

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

But this is easy to see, because the UP of the catamorphism (Table 4.8) is nothing else than the Σ-algebra homomorphism property. ■

4.1.7.4 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 $\mathcal{C}^{\mathsf{F}}$ has an initial object in_{F}, then this is an isomorphism in $\mathcal{C}$.

Proof. On the one hand

$$\begin{aligned} & in_{\mathsf{F}} \cdot (\!|\mathsf{F} in_{\mathsf{F}}|\!)_{\mathsf{F}} \\ = \quad & \{ \text{ fusion (Table 4.8) } \} \\ & (\!|in_{\mathsf{F}}|\!)_{\mathsf{F}} \\ = \quad & \{ \text{ reflection (Table 4.8) } \} \\ & id_{\boldsymbol{\mu}\mathsf{F}} \end{aligned}$$

while on the other hand

$$\begin{aligned}
&(\!|\mathsf{F}in_{\mathsf{F}}|\!)_{\mathsf{F}} \cdot in_{\mathsf{F}} \\
=\ & \{\ \text{UP (Table 4.8)}\ \} \\
&\mathsf{F}in_{\mathsf{F}} \cdot \mathsf{F}(\!|\mathsf{F}in_{\mathsf{F}}|\!)_{\mathsf{F}} \\
=\ & \{\ \mathsf{F}\ \text{functor}\ \} \\
&\mathsf{F}\big(in_{\mathsf{F}} \cdot (\!|\mathsf{F}in_{\mathsf{F}}|\!)_{\mathsf{F}}\big) \\
=\ & \{\ \text{see above}\ \} \\
&\mathsf{F}id_{\boldsymbol{\mu}\mathsf{F}} \\
=\ & \{\ \mathsf{F}\ \text{functor}\ \} \\
&id_{\mathsf{F}(\boldsymbol{\mu}\mathsf{F})}.
\end{aligned}$$

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

4.1.7.5 Proposition (rolling rule [Fre90]**).** Let $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ and $\mathsf{G} : \boldsymbol{\mathcal{D}} \leftarrow \boldsymbol{\mathcal{C}}$ be functors. If $\mathsf{G} \cdot \mathsf{F}$ has an initial algebra $in_{\mathsf{G}\cdot\mathsf{F}}$, then $\mathsf{F}in_{\mathsf{G}\cdot\mathsf{F}}$ is an initial $(\mathsf{F} \cdot \mathsf{G})$-algebra. ◇

4.1.7.6 Corollary (rolling rule). Let $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ and $\mathsf{G} : \boldsymbol{\mathcal{D}} \leftarrow \boldsymbol{\mathcal{C}}$ be functors such that $\boldsymbol{\mu}(\mathsf{G} \cdot \mathsf{F})$ exists. Then:

$$\boldsymbol{\mu}(\mathsf{F} \cdot \mathsf{G}) \cong \mathsf{F}\big(\boldsymbol{\mu}(\mathsf{G} \cdot \mathsf{F})\big).$$

Proof. From Proposition 4.1.7.5 we know that $\mathsf{F}in_{\mathsf{G}\cdot\mathsf{F}}$ is an initial $(\mathsf{F} \cdot \mathsf{G})$-algebra, thus $\mathsf{F}in_{\mathsf{G}\cdot\mathsf{F}} \cong in_{\mathsf{F}\cdot\mathsf{G}}$ (isomorphic in the category $\boldsymbol{\mathcal{C}}^{\mathsf{F}\cdot\mathsf{G}}$), because initial objects are unique up to isomorphism. Applying the forgetful functor $|\,\boldsymbol{.}\,|^{\mathsf{F}\cdot\mathsf{G}}$ yields the assertion. ∎

4.1.7.7 Definition (algebraically complete [Fre90]**).** A category $\boldsymbol{\mathcal{C}}$ is called **algebraically complete** if every $\boldsymbol{\mathcal{C}}$-endofunctor has an initial algebra. ◇

4.1.7.8 Proposition ([Fre92])**.** The category $\boldsymbol{\mathcal{S}et}_{\aleph_0}$ of all countable sets is algebraically complete. ◇

4.2 Concrete categories and constructs

4.2.1 Concrete categories and concrete functors

4.2.1.1 Definition (faithful functor). Let $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ be a functor. F is called an **embedding** provided that F is injective on morphisms. F is called **faithful** provided that all its hom-class restrictions $\mathsf{F}_{(A,B)} : \boldsymbol{\mathcal{C}}(\mathsf{F}A, \mathsf{F}B) \leftarrow \boldsymbol{\mathcal{D}}(A, B)$ are injective for every $A, B \in \boldsymbol{\mathcal{D}}$. ◇

4.2.1.2 Definition ((semi)concrete category [AHS90] Def. 5.1**).** Let $\mathcal{C}$ and $\mathcal{D}$ be categories and $\mathsf{U} : \mathcal{C} \leftarrow \mathcal{D}$ be a functor. The pair $(\mathcal{D}, \mathsf{U})$ is called the **semiconcrete category** built upon $\mathcal{C}$ by U. If moreover U is faithful then $(\mathcal{D}, \mathsf{U})$ is called **concrete category** and the functor U is called the **forgetful functor** of $(\mathcal{D}, \mathsf{U})$.

As a convention we will write most of the forgetful functors just $|\mathbf{.}|$ if the connection to their concrete category is obvious. For each $\mathcal{D}$-object A we call $|A|$ the **underlying $\mathcal{C}$-object** of A and for each $\mathcal{D}$-morphism f we call $|f|$ the **underlying $\mathcal{C}$-morphism** of f. Motivated by the fact that $|\mathbf{.}|$ is faithful it is common practice to identify the morphisms f and $|f|$ and to write just f for both of them.

A concrete category built upon $\mathcal{S}et$ is called a **construct**. ◇

4.2.1.3 Example. Every mathematical structure, *i.e.* a set together with some algebraic and/or topological structure, is a *construct*. The forgetful functor maps a mathematical structure onto its carrier set, in other words it forgets the structure. The notion of a *concrete category* is a natural generalization: Consider the constructs $(\mathcal{A}b\mathcal{G}rp, |\mathbf{.}|_{\mathcal{A}b\mathcal{G}rp})$ of Abelian groups and $(\mathcal{G}rp, |\mathbf{.}|_{\mathcal{G}rp})$ of groups. Then $(\mathcal{A}b\mathcal{G}rp, \mathsf{U}_{\mathcal{A}b\mathcal{G}rp})$ is built upon $\mathcal{G}rp$ where $\mathsf{U}_{\mathcal{A}b\mathcal{G}rp}$ embeds Abelian groups into groups. Thus the forgetful functor $|\mathbf{.}|_{\mathcal{A}b\mathcal{G}rp} = |\mathbf{.}|_{\mathcal{G}rp} \cdot \mathsf{U}_{\mathcal{A}b\mathcal{G}rp}$ forgets first that the group was Abelian and secondly the entire group-structure. ◇

4.2.1.4 Definition ((semi)concrete functor, [AHS90] Def. 5.9**).** Let $\mathcal{C}$ be a category and $(\mathcal{D}, \mathsf{U})$ and $(\mathcal{D}', \mathsf{U}')$ be semiconcrete categories built upon $\mathcal{C}$. A functor $\mathsf{F} : \mathcal{D} \leftarrow \mathcal{D}'$ is called a **semiconcrete functor** provided that

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

In this case we write

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

If furthermore the codomain $(\mathcal{D}, \mathsf{U})$ is concrete then F is called a **concrete functor**. We denote the meta-category of all concrete categories built upon $\mathcal{C}$ with all concrete functors as morphisms by $\boldsymbol{c}\mathbf{CAT}\ \mathcal{C}$. ◇

4.2.1.5 Lemma ([AHS90] Prop. 5.10(2)**).** A concrete functor is completely determined by its values on objects.

Proof. Let $\mathcal{C}$ by a category, $(\mathcal{D}, \mathsf{U})$ be a concrete and $(\mathcal{D}', \mathsf{U}')$ be a semiconcrete category 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 $f : A \leftarrow_{\mathcal{D}'} B$. Then:

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

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. ∎

4.2.1.6 Definition and Lemma (concrete categories of concrete categories). Let $\boldsymbol{\mathcal{C}}$ 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 $\boldsymbol{\mathcal{C}}$ is the object class of a meta-pre-category with all concrete functors as morphisms. We denote the according meta-category (see Note 4.1.1.4) of all concrete categories built upon $\boldsymbol{\mathcal{C}}$ by $\boldsymbol{c}\mathbf{CAT}\ \boldsymbol{\mathcal{C}}$. The meta-category $\boldsymbol{c}\mathbf{CAT}\ \boldsymbol{\mathcal{C}}$ itself is built upon the meta-category **CAT** by the forgetful functor $|\,\boldsymbol{.}\,|$ which maps a $(\boldsymbol{c}\mathbf{CAT}\ \boldsymbol{\mathcal{C}})$-morphism to its underlying **CAT**-morphism, *i.e.* functor. ◇

4.2.1.7 Note. Let $\boldsymbol{\mathcal{C}}$ be a category and $(\boldsymbol{c}\mathbf{CAT}\ \ \boldsymbol{\mathcal{C}}, |\,\boldsymbol{.}\,|)$ be the concrete meta-category built upon **CAT** from Definition and Lemma 4.2.1.6. Let $(\boldsymbol{\mathcal{D}}, \mathsf{U}), (\boldsymbol{\mathcal{D}}', \mathsf{U}') \in \mathrm{Ob}(\boldsymbol{c}\mathbf{CAT}\ \boldsymbol{\mathcal{C}})$ be two concrete categories and $\mathsf{F} : (\boldsymbol{\mathcal{D}}, \mathsf{U}) \leftarrow (\boldsymbol{\mathcal{D}}', \mathsf{U}')$ be a concrete functor. We may view F out of three different perspectives:

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

(ii) as the concrete functor $\mathsf{F} : (\boldsymbol{\mathcal{D}}, \mathsf{U}) \leftarrow (\boldsymbol{\mathcal{D}}', \mathsf{U}')$, *i.e.* as a morphism in the meta-pre-category from Definition and Lemma 4.2.1.6, or

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

The connection

- from (i) to (ii) is the Definition 4.2.1.4 (and Definition and Lemma 4.2.1.6),
- from (ii) to (iii) is the construction from Note 4.1.1.4, and
- from (iii) to (i) is the forgetful functor: $\big|\big((\boldsymbol{\mathcal{D}}, \mathsf{U}), \mathsf{F}, (\boldsymbol{\mathcal{D}}', \mathsf{U}')\big)\big| = \mathsf{F}$. ◇

4.2.1.8 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 ${}_{\mathsf{U}}(\boldsymbol{.})$ 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:

$$ {}_{\mathsf{U}}(\boldsymbol{.}) : (\boldsymbol{c}\mathbf{CAT}\ \boldsymbol{\mathcal{C}}, |\,\boldsymbol{.}\,|) \leftarrow (\boldsymbol{c}\mathbf{CAT}\ \boldsymbol{\mathcal{C}}', |\,\boldsymbol{.}\,|). $$

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 $_{\mathsf{U}}(\mathbf{.})$ is also multiplicative and preserves identities, because it operates trivially on morphisms, *i.e.* it only changes domain and codomain. The concreteness property of $_{\mathsf{U}}(\mathbf{.})$ follows from $|_{\mathsf{U}}\big((\boldsymbol{\mathcal{D}}_1, \mathsf{U}_1), \mathsf{H}, (\boldsymbol{\mathcal{D}}_2, \mathsf{U}_2)\big)| = |\big((\boldsymbol{\mathcal{D}}_1, \mathsf{U} \cdot \mathsf{U}_1), \mathsf{H}, (\boldsymbol{\mathcal{D}}_2, \mathsf{U} \cdot \mathsf{U}_2)\big)| = \mathsf{H} = |\big((\boldsymbol{\mathcal{D}}_1, \mathsf{U}_1), \mathsf{H}, (\boldsymbol{\mathcal{D}}_2, \mathsf{U}_2)\big)|$. ∎

4.2.1.9 Definition (function spaces). A construct $(\boldsymbol{\mathcal{C}}, |\mathbf{.}|)$ has **function spaces** if

(i) $(\boldsymbol{\mathcal{C}}, |\mathbf{.}|)$ has finite concrete products, *i.e.* $\boldsymbol{\mathcal{C}}$ has finite products and $|\mathbf{.}|$ preserves them, and

(ii) $\boldsymbol{\mathcal{C}}$ is cartesian closed and the evaluation morphism $ev : A \leftarrow A^B \times B$ can be chosen in such a way that $|A^B| = \boldsymbol{\mathcal{C}}(A, B)$ and $|ev|(f, x) = f\,x$.

We say that a category $\boldsymbol{\mathcal{C}}$ has **function spaces** if there exists a faithful functor $|\mathbf{.}| : \boldsymbol{\mathcal{S}et} \leftarrow \boldsymbol{\mathcal{C}}$ such that the construct $(\boldsymbol{\mathcal{C}}, |\mathbf{.}|)$ has function spaces. ◇

4.3 Adjoint functors and adjunctions

This section is not necessary to understand Chapter 5. We will use adjunctions and monads in Chapter 6 only.

4.3.1 Free objects

4.3.1.1 Definition (universal arrow). Let $\mathsf{G} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ be a functor. A $\boldsymbol{\mathcal{C}}$-morphism $u_X : \mathsf{G}A \leftarrow X$ is called a **G-universal arrow** from X if for every $\varrho : \mathsf{G}B \leftarrow X$ there exists a unique $\boldsymbol{\mathcal{D}}$-morphism $f : B \leftarrow A$ such that $\mathsf{G}f \cdot u_X = \varrho$ holds. The dual notion is called a **G-co-universal arrow**. ◇

4.3.1.2 Definition (free object of a concrete category). Let $(\boldsymbol{\mathcal{D}}, \mathsf{U})$ be a concrete category built upon $\boldsymbol{\mathcal{C}}$.

(i) Let X be a $\boldsymbol{\mathcal{C}}$-object. A $\boldsymbol{\mathcal{D}}$-object A is called a **free object** over X in $(\boldsymbol{\mathcal{D}}, \mathsf{U})$ if there exists a U-universal arrow $u_X : \mathsf{U}A \leftarrow X$.

(ii) We say that $(\boldsymbol{\mathcal{D}}, \mathsf{U})$ **has free objects** if for every $\boldsymbol{\mathcal{C}}$-object X there exists a free object over X in $(\boldsymbol{\mathcal{D}}, \mathsf{U})$. ◇

4.3.1.3 Example. Let $(\boldsymbol{\mathcal{D}}, \mathsf{U})$ be a concrete category built upon $\boldsymbol{\mathcal{C}}$.

(i) If $\boldsymbol{\mathcal{C}}$ has an initial object 0, then the free objects over 0 in $(\boldsymbol{\mathcal{D}}, \mathsf{U})$ are precisely the initial objects of $\boldsymbol{\mathcal{D}}$.

(ii) If $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{S}et}$ and $(\boldsymbol{\mathcal{D}}, \mathsf{U})$ is the construct of all Σ-algebras, then the free objects over a set X are precisely the free Σ-algebras over X in the sense of universal algebra. The according universal arrows are the embeddings of X into a free algebra over X. ◇

4.3.2 Varietors

4.3.2.1 Definition (polynomial functors).

A category is called $\left\{\begin{array}{c}\textbf{(co)cartesian}\\ \textbf{bicartesian}\\ \textbf{cartesian closed}\\ \textbf{bicartesian closed}\end{array}\right\}$ if it $\begin{cases}\text{has finite (co)products,}\\ \text{is cocartesian and cartesian,}\\ \text{is cartesian and has exponents,}\\ \text{is cartesian closed and cocartesian.}\end{cases}$

The class of $\left\{\begin{array}{l}\textbf{cocartesian}\\ \textbf{bicartesian}\\ \textbf{polynomial}\end{array}\right\}$ functors to a $\left\{\begin{array}{c}\text{cocartesian}\\ \text{bicartesian}\\ \text{bicartesian closed}\end{array}\right\}$ category $\boldsymbol{\mathcal{C}}$ is the smallest class of functors to $\boldsymbol{\mathcal{C}}$ closed under $\begin{cases}\text{finite coproducts,}\\ \text{finite coproducts and products,}\\ \text{constants, finite (co)products,}\\ \qquad\qquad\text{and constant exponents.}\end{cases}$

We can describe these classes of endofunctors in a more intuitive way by the following grammars:

(i) cocartesian functors: $\mathsf{F}X ::= X \mid \mathsf{F}_1X + \mathsf{F}_2X$,

(ii) bicartesian functors: $\mathsf{F}X ::= X \mid \mathsf{F}_1X + \mathsf{F}_2X \mid \mathsf{F}_1X \times \mathsf{F}_2X$,

(iii) polynomial functors: $\mathsf{F}X ::= A \mid X \mid \mathsf{F}_1X + \mathsf{F}_2X \mid \mathsf{F}_1X \times \mathsf{F}_2X \mid \mathsf{F}_1 \Leftarrow A$. ◇

4.3.2.2 Observation. (i) Cocartesian functors to a bicartesian category are also bicartesian. Bicartesian functors to a bicartesian closed category are also polynomial.

(ii) The classes of cocartesian, bicartesian, and polynomial *endo*functors on some category are each closed under composition. ◇

4.3.2.3 Definition (varietor). An endofunctor $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$ is called a **varietor** if the concrete category of F-algebras $(\boldsymbol{\mathcal{C}}^{\mathsf{F}}, |\,.\,|^{\mathsf{F}})$ has free objects. ◇

4.3.2.4 Proposition (free algebras versus initial algebras). Let $\boldsymbol{\mathcal{C}}$ be a cocartesian category, X be a $\boldsymbol{\mathcal{C}}$-object, and $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$ be an endofunctor. The following are equivalent:

(i) There exists a free object over X in $(\boldsymbol{\mathcal{C}}^{\mathsf{F}}, |\,.\,|^{\mathsf{F}})$.

(ii) $\mathsf{F} + \underline{X}$ has an initial algebra.

Proof. The universal property of a free F-algebra over X can be translated into the universal property of an initial $(\mathsf{F} + \underline{X})$-algebra and vice versa by straight forward calculations using the following:

Let χ be free over X in $(\boldsymbol{\mathcal{C}}^{\mathsf{F}}, |\,.\,|^{\mathsf{F}})$ with universal arrow u_X. Then $[\chi, u_X]$ is initial in $\boldsymbol{\mathcal{C}}^{\mathsf{F}+\underline{X}}$. The catamorphism is given for every $(\mathsf{F} + \underline{X})$-algebra φ by $(\!|\varphi|\!)_{\mathsf{F}+\underline{X}} = |(\varphi \cdot \acute{\iota}_{(\mathsf{F}|\varphi|^{\mathsf{F}},X)})/u_X|^{\mathsf{F}+\underline{X}}$.

Let $in_{\mathsf{F}+\underline{X}}$ be initial in $\mathcal{C}^{\mathsf{F}+\underline{X}}$. Then $in_{\mathsf{F}+\underline{X}} \cdot \grave{\iota}_{(\boldsymbol{\mu}(\mathsf{F}+\underline{X}),X)}$ is free over X in $(\mathcal{C}^{\mathsf{F}}, |\,.\,|^{\mathsf{F}})$ with universal arrow $u_X = in_{\mathsf{F}+\underline{X}} \cdot \acute{\iota}_{(\boldsymbol{\mu}(\mathsf{F}+\underline{X}),X)}$. For every F-algebra ψ and every $\varrho : |\psi|^{\mathsf{F}} \leftarrow X$ the unique algebra homomorphism to ψ from the free algebra is given by $|\varrho/u_X|^{\mathsf{F}} = (\!|[\psi, \varrho]|\!)_{\mathsf{F}+\underline{X}}$. ∎

4.3.2.5 Corollary (varietors versus initial algebras). Let $\mathcal{C}$ be a cocartesian category and let $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{C}$ be an endofunctor. The following are equivalent:

(i) F is a varietor.

(ii) For every $\mathcal{C}$-object X the functor $\mathsf{F} + \underline{X}$ has an initial algebra. ◇

4.3.2.6 Corollary. Every endofunctor on an algebraically complete cocartesian category is a varietor. ◇

4.3.2.7 Theorem ([AP01]). Polynomial $\mathcal{S}et$-endofunctors are varietors. ◇

4.3.3 Adjoint functors

4.3.3.1 Definition (right (and left) adjoint functor). Let $\mathsf{G} : \mathcal{C} \leftarrow \mathcal{D}$ be a functor. Then G is called **right adjoint** (or **adjoint**) if for every $\mathcal{C}$-object X there exists a G-universal arrow from X. Dually G is called **left adjoint** (or **co-adjoint**) if for every $\mathcal{D}$-object Y there exists a G-co-universal arrow to Y. ◇

4.3.3.2 Observation. (i) Let $(\mathcal{D}, \mathsf{U})$ be a concrete category built upon $\mathcal{C}$. Then U is right adjoint iff $(\mathcal{D}, \mathsf{U})$ has free objects. We call a left adjoint of U a **free functor** of $(\mathcal{D}, \mathsf{U})$.

(ii) Let F be an endofunctor. Then the canonical forgetful functor from the category of F-algebras $|\,.\,|^{\mathsf{F}}$ is right adjoint iff F is a varietor. ◇

4.3.4 Adjunctions

4.3.4.1 Definition (adjunction). Let $\mathcal{C}$ and $\mathcal{D}$ be categories.

(i) $(\eta, \varepsilon) : \mathsf{F} \dashv \mathsf{G} : \mathcal{C} \leftarrow \mathcal{D}$ is called an **adjunction** (or **adjoint situation**) if

$$\mathsf{F} : \mathcal{C} \leftarrow \mathcal{D}, \qquad \mathsf{G} : \mathcal{D} \leftarrow \mathcal{C},$$

and

$$\eta : \mathsf{G} \cdot \mathsf{F} \dot{\leftarrow} \mathsf{Id}, \qquad \varepsilon : \mathsf{Id} \dot{\leftarrow} \mathsf{F} \cdot \mathsf{G},$$

such that

$$\varepsilon\mathsf{F} \cdot \mathsf{F}\eta = id_{\mathsf{F}} \quad \text{and} \quad \mathsf{G}\varepsilon \cdot \eta\mathsf{G} = id_{\mathsf{G}}$$

holds. The natural transformations η and ε are called the **unit** and the **co-unit** of the adjunction, respectively.

(ii) If the relation to the categories $\boldsymbol{\mathcal{C}}$ and $\boldsymbol{\mathcal{D}}$ is obvious we will only write $(\eta, \varepsilon) : \mathsf{F} \dashv \mathsf{G}$ to denote the adjunction.

(iii) If for given functors $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ and $\mathsf{G} : \boldsymbol{\mathcal{D}} \leftarrow \boldsymbol{\mathcal{C}}$ an adjunction $(\eta, \varepsilon) : \mathsf{F} \dashv \mathsf{G}$ exists, then F and G are called **adjoint** and we write $\mathsf{F} \dashv \mathsf{G}$.

(iv) If for a given functor $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ there exists a functor $\mathsf{G} : \boldsymbol{\mathcal{D}} \leftarrow \boldsymbol{\mathcal{C}}$ such that $\mathsf{F} \dashv \mathsf{G}$, then we say that F has a right adjoint.

(v) If for a given functor $\mathsf{G} : \boldsymbol{\mathcal{D}} \leftarrow \boldsymbol{\mathcal{C}}$ there exists a functor $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ such that $\mathsf{F} \dashv \mathsf{G}$, then we say that G **has a left adjoint**. ◇

4.3.4.2 Definition (adjungate). Let $(\eta, \varepsilon) : \mathsf{F} \dashv \mathsf{G} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ be an adjunction and $f : A \leftarrow_{\boldsymbol{\mathcal{C}}} \mathsf{F}B$ and $g : \mathsf{G}A \leftarrow_{\boldsymbol{\mathcal{D}}} B$.

(i) $f^\sharp = \mathsf{G}f \cdot \eta_B$ is called the **left adjungate** of f and

(ii) $g^\flat = \varepsilon_A \cdot \mathsf{F}g$ is called the **right adjungate** of g.

A left or right adjungate is also called an **adjoint transpose**. Notice that the functions $(\,\boldsymbol{.}\,)^\sharp$ and $(\,\boldsymbol{.}\,)^\flat$ are defined w.r.t. some adjunction which is invisible in the notation. Usually the according adjunction will be obvious from the context. ◇

4.3.4.3 Lemma (laws for adjunctions [Fok92a]**).** Let $(\eta, \varepsilon) : \mathsf{F} \dashv \mathsf{G} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ be an adjunction with left/right adjungates $(\,\boldsymbol{.}\,)^\sharp$ and $(\,\boldsymbol{.}\,)^\flat$, respectively. Then for all $X \xleftarrow{x}_{\boldsymbol{\mathcal{C}}} A \xleftarrow{f}_{\boldsymbol{\mathcal{C}}} \mathsf{F}B$ and $\mathsf{G}A \xleftarrow{g}_{\boldsymbol{\mathcal{D}}} B \xleftarrow{y}_{\boldsymbol{\mathcal{D}}} Y$:

(i) $\varepsilon^\sharp = id$,

(ii) $(x \cdot f \cdot \mathsf{F}y)^\sharp = \mathsf{G}x \cdot f^\sharp \cdot y$,

(iii) $\eta^\flat = id$,

(iv) $(\mathsf{G}x \cdot g \cdot y)^\flat = x \cdot g^\flat \cdot \mathsf{F}y$,

(v) $(f^\sharp)^\flat = f$ and $(g^\flat)^\sharp = g$. ◇

4.3.4.4 Proposition ([AHS90] Exer. 19A**).** Let $\boldsymbol{\mathcal{C}}$ and $\boldsymbol{\mathcal{D}}$ be categories. The functors $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ and $\mathsf{G} : \boldsymbol{\mathcal{D}} \leftarrow \boldsymbol{\mathcal{C}}$ are adjoint, *i.e.* $\mathsf{F} \dashv \mathsf{G}$ iff

$$\boldsymbol{\mathcal{C}}(A, \mathsf{F}B) \cong \boldsymbol{\mathcal{D}}(\mathsf{G}A, B) \quad \text{natural in } A \;\&\; B.$$

Proof. $\Rightarrow$ Using Lemma 4.3.4.3 (v) we show that $(\,\boldsymbol{.}\,)^\sharp : \boldsymbol{\mathcal{D}}(\mathsf{G}\boldsymbol{.}, \boldsymbol{.}) \dot{\leftarrow} \boldsymbol{\mathcal{C}}(\boldsymbol{.}, \mathsf{F}\boldsymbol{.})$ and $(\,\boldsymbol{.}\,)^\flat : \boldsymbol{\mathcal{C}}(\boldsymbol{.}, \mathsf{F}\boldsymbol{.}) \dot{\leftarrow} \boldsymbol{\mathcal{D}}(\mathsf{G}\boldsymbol{.}, \boldsymbol{.})$ are natural transformations which are each others inverses.

$\Leftarrow$ Let $(\mathbf{.})^\sharp : \boldsymbol{\mathcal{D}}(\mathsf{G}\mathbf{.}, \mathbf{.}) \dot{\leftarrow} \boldsymbol{\mathcal{C}}(\mathbf{.}, \mathsf{F}\mathbf{.})$ and $(\mathbf{.})^\flat : \boldsymbol{\mathcal{C}}(\mathbf{.}, \mathsf{F}\mathbf{.}) \dot{\leftarrow} \boldsymbol{\mathcal{D}}(\mathsf{G}\mathbf{.}, \mathbf{.})$ be mutually inverse natural isomorphisms. We show that $(id^\sharp, id^\flat) : \mathsf{F} \dashv \mathsf{G}$ is an adjunction. Moreover $(\mathbf{.})^\sharp$ and $(\mathbf{.})^\flat$ are in fact the left and right adjungates w.r.t. that adjunction. ∎

4.3.4.5 Lemma (composition of adjunctions, [AHS90] Prop. 19.13**).** Let $(\eta, \varepsilon) : \mathsf{F} \dashv \mathsf{G} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ and $(\eta', \varepsilon') : \mathsf{F}' \dashv \mathsf{G}' : \boldsymbol{\mathcal{D}} \leftarrow \boldsymbol{\mathcal{E}}$ be adjunctions, then $(\mathsf{G}'\eta\mathsf{F}' \cdot \eta', \varepsilon \cdot \mathsf{F}\varepsilon'\mathsf{G}) : \mathsf{F} \cdot \mathsf{F}' \dashv \mathsf{G}' \cdot \mathsf{G} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{E}}$ is an adjunction. It is called the **composition** of $(\eta, \varepsilon) : \mathsf{F} \dashv \mathsf{G}$ and $(\eta', \varepsilon') : \mathsf{F}' \dashv \mathsf{G}'$. ◇

4.3.4.6 Corollary. The subclass of left adjoint functors is a subcategory of **CAT**. We denote this category by **LeftAdj**. ◇

4.3.4.7 Theorem ([AHS90] Th. 19.1 & Proposition 19.7**).** Let $\boldsymbol{\mathcal{C}}$ and $\boldsymbol{\mathcal{D}}$ be categories and $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ and $\mathsf{G} : \boldsymbol{\mathcal{D}} \leftarrow \boldsymbol{\mathcal{C}}$ be functors.

(i) $\mathsf{F} \dashv \mathsf{G} \iff \mathsf{G}^{\mathrm{op}} \dashv \mathsf{F}^{\mathrm{op}}$.

(ii) G is right adjoint iff G has a left adjoint.

(iii) F is left adjoint iff F has a right adjoint. ◇

4.3.4.8 Lemma (essential uniqueness of adjoint functors, [AHS90] Prop. 19.9**).** The right adjoints of a given functor F are unique up to isomorphism. Vice versa all functors which are isomorphic to a right adjoint of F are right adjoints of F. Obviously, the dual statements hold for right adjoints. ◇

4.3.4.9 Proposition ([AHS90] Prop. 18.9**).** Right adjoint functors are continuous, *i.e.* preserve small limits. Left adjoint functors are co-continuous, *i.e.* preserve small colimits. Thus in particular: right adjoint functors preserve products and left adjoint functors preserve coproducts. ◇

4.4 Monads

This section is not necessary to understand Chapter 5. We will use adjunctions and monads in Chapter 6 only.

Monads are related to terms (or trees) like monoids are related to character strings:

- Let Σ be an alphabet and $\Sigma^* = \bigcup_{k \in \mathbb{N}_0} \Sigma^k$ be the set of all words over Σ. Then $(\Sigma^*, \varepsilon, \cdot)$ has the structure of a monoid, where $\varepsilon = () \in \Sigma^*$ denotes the empty word and $\cdot : \Sigma^* \leftarrow \Sigma^* \times \Sigma^*$ is the concatenation of words, *i.e.* $(a_1, \ldots, a_k) \cdot (a_{k+1}, \ldots, a_{k+\ell}) = (a_1, \ldots, a_{k+\ell})$. It is well known that $(\Sigma^*, \varepsilon, \cdot)$ is a free monoid over Σ and vice versa every free monoid is isomorphic to a monoid of character strings with concatenation.

- Let Σ be a functor induced by a ranked alphabet and $\mathsf{T}_\Sigma : \boldsymbol{Set} \leftarrow \boldsymbol{Set}$ be the functor defined by $\mathsf{T}_\Sigma X \cong \boldsymbol{\mu}(\Sigma + \underline{X})$. As we will see shortly: $(\mathsf{T}_\Sigma, \eta, \mu)$ has the structure of a monad, where $\eta_X : \mathsf{T}_\Sigma X \leftarrow X$ is the embedding of variables and $\mu_X : \mathsf{T}_\Sigma X \leftarrow \mathsf{T}_\Sigma(\mathsf{T}_\Sigma X)$ describes term-substitution. This monad is a free monad over Σ and vice versa every free monad (over a functor induced by a ranked alphabet) is a monad of terms with term-substitution.

It is possible to formalize the above analogy between monads and monoids: Both notions *monoid* and *monad* are instances of the more general notion *monoid over a monoidal category* [Str72, BW85].

4.4.1 Monads and Kleisli triples

4.4.1.1 Definition (monad). Let $\mathcal{C}$ be a category. $\mathbb{T} = (\mathsf{T}, \eta, \mu)$ is called a **monad**[9] on $\mathcal{C}$ if

$$\mathsf{T} : \mathcal{C} \leftarrow \mathcal{C}, \quad \eta : \mathsf{T} \dot{\leftarrow} \mathsf{Id}_{\mathcal{C}}, \quad \mu : \mathsf{T} \dot{\leftarrow} \mathsf{T}^2$$

such that

$$\mu \cdot \mathsf{T}\eta = id_{\mathsf{T}} = \mu \cdot \eta\mathsf{T}$$

and

$$\mu \cdot \mathsf{T}\mu = \mu \cdot \mu\mathsf{T}$$

holds. The natural transformations η and μ are called the **unit** and the **join** (or **multiplication**), respectively. ◇

4.4.1.2 Note (monadic). In relation with monads the adjective *monadic* is commonly used. *E.g.*. the operations 'unit' and 'join' are also called *monadic operations*. Later we will see the definitions of *monadic categories*, *monadic functors*, and *monadic transducers*. However, all this is *not* related at all to the notions *monadic ranked alphabet*, *monadic tree*, or *monadic term*. ◇

4.4.1.3 Definition and Lemma (Kleisli-triple, Haskell-Monad). There exists another equivalent description of a monad: A **Kleisli-triple** $\left(\mathsf{M}, \eta, (\,\boldsymbol{.}\,)^\dagger\right)$ on a category $\mathcal{C}$ consists of an endo*function* $\mathsf{M} : \mathrm{Ob}\,\mathcal{C} \leftarrow \mathrm{Ob}\,\mathcal{C}$, an $\mathrm{Ob}\,\mathcal{C}$-indexed family η (called **unit**) of morphisms $\eta_X : \mathsf{M}X \leftarrow X$, and an operation $(\,\boldsymbol{.}\,)^\dagger$ (called **extension operation**) taking every $\mathcal{C}$-morphism $f : \mathsf{M}X \leftarrow Y$ to a $\mathcal{C}$-morphism $f^\dagger : \mathsf{M}X \leftarrow \mathsf{M}Y$ such that

(i) $f : \mathsf{M}X \leftarrow Y \implies f^\dagger \cdot \eta_Y = f$,

(ii) $\eta_X^\dagger = id_{\mathsf{M}X}$, and

[9] The name **monad** comes from shortening and joining together two older names for the same construction: **mon***oid* and *tri***ad** [Mac71]. Other old names for a monad are *triple*, *standard construction*, or *fundamental construction*

(iii) $(f^\dagger \cdot g)^\dagger = f^\dagger \cdot g^\dagger$.

There exists a bijection between monads (T, η, μ) and Kleisli-triples $\left(\mathsf{M}, \eta, (\,\boldsymbol{.}\,)^\dagger\right)$ on $\boldsymbol{\mathcal{C}}$ given by the equations

$$\begin{array}{rcl} \mathsf{M}X & = & \mathsf{T}X, \\ f^\dagger & = & \mu \cdot \mathsf{T}f, \end{array} \qquad \text{and} \qquad \begin{array}{rcl} \mathsf{T}f & = & (\eta \cdot f)^\dagger, \\ \mu & = & id^\dagger. \end{array}$$

In Haskell a monad is defined by

```
class Monad m where
  (>>=) :: m a → (a → m b) → m b
  return :: a → m a
```

The operator $\gg\!=$ is called **bind**. The type constructor m and the functions *return* and $(=\!\ll)$, where the latter is just a variant of $(\gg\!=)$ with exchanged input arguments

```
(=<<) :: (a → m b) → m a → m b
(=<<) = flip (>>=)
```

correspond to the functions M, η, and $(\,\boldsymbol{.}\,)^\dagger$ of a Kleisli-triple, respectively. The monad laws are not part of the definition in Haskell, however it is usually assumed that they hold. ◇

4.4.1.4 Example. Let us illustrate the monadic operations of the free Δ monad $\Delta^\star = (\mathsf{T}_\Delta, \eta, \mu)$: The unit is simple:

$$X \xrightarrow{\quad \eta_X \quad} \mathsf{T}_\Delta X$$

$$x \longmapsto \textcircled{x}$$

◇

The multiplication is just term-substitution. We can express it more easily with the according Kleisli-† as shown in Figure 4.4.

4.4.2 Monad morphisms

4.4.2.1 Definition (monad morphism). Let $\mathbb{T} = (\mathsf{T}, \eta, \mu)$ and $\mathbb{T}' = (\mathsf{T}', \eta', \mu')$ be monads on $\boldsymbol{\mathcal{C}}$.

(i) A natural transformation $h : \mathsf{T} \dot{\leftarrow} \mathsf{T}'$ such that $\eta = h \cdot \eta'$ and $\mu \cdot (h * h) = h \cdot \mu'$ is called a **monad morphism** to $\mathbb{T}$ from $\mathbb{T}'$ and we write $h : \mathbb{T} \leftarrow \mathbb{T}'$.

(ii) It is easy to see that (i) gives rise to a category which we will denote by $\mathbf{Mnd}\,\boldsymbol{\mathcal{C}}$. Moreover this is a concrete category $\left(\mathbf{Mnd}\,\boldsymbol{\mathcal{C}}, |\,\boldsymbol{.}\,|\right)$ built upon $\mathbf{End}\,\boldsymbol{\mathcal{C}}$ where the forgetful functor $|\,\boldsymbol{.}\,|$ maps a monad onto its underlying endofunctor: $|(|\mathsf{T}, \eta, \mu) = \mathsf{T}$. ◇

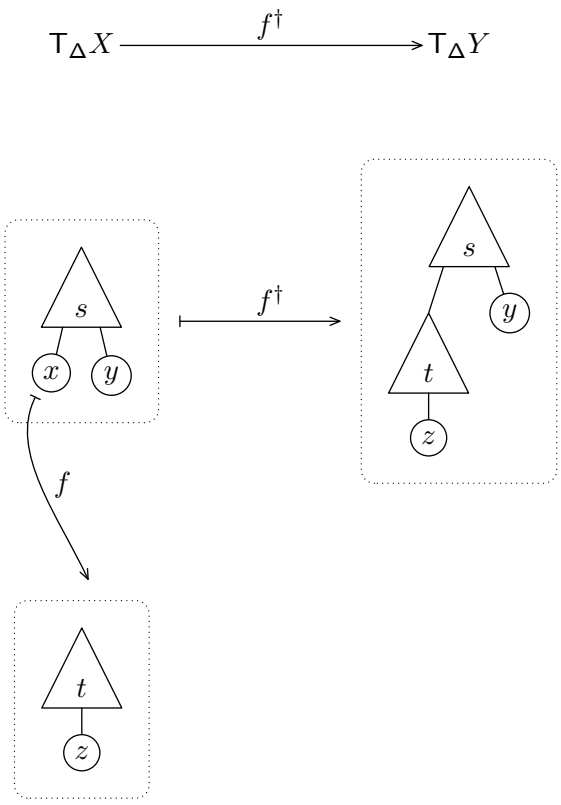


Figure 4.4: Kleisli-† of the free Δ monad

4.4.3 Free monads

4.4.3.1 Definition (free monad). Let $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$ be an endofunctor. A free object over F in $(\mathbf{Mnd}\,\boldsymbol{\mathcal{C}}, |\,\mathbf{.}\,|)$ is called a **free monad** over F. Since free monads over F are determined uniquely up to isomorphism we also talk about *the* free monad over F. We denote it by $\mathsf{F}^\star$ and its underlying endofunctor by $\mathsf{T}_\mathsf{F} = |\mathsf{F}^\star|$. ◇

4.4.4 Monads versus Adjunctions

4.4.4.1 Lemma (each adjunction gives rise to a monad, [AHS90] Prop. 20.3**).** Let $(\eta, \varepsilon) : \mathsf{F} \dashv \mathsf{G} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ be an adjunction. Then $(\mathsf{G} \cdot \mathsf{F}, \eta, \mathsf{G}\varepsilon\mathsf{F})$ is a monad on $\boldsymbol{\mathcal{D}}$. ◇

4.4.4.2 Definition (Eilenberg-Moore category). Let $\mathbb{T} = (\mathsf{T}, \eta, \mu)$ be a monad on $\boldsymbol{\mathcal{C}}$. The **Eilenberg-Moore category** $(\boldsymbol{\mathcal{C}}^\mathbb{T}, \mathsf{U}^\mathbb{T})$ of $\mathbb{T}$ is the full concrete subcategory of the concrete category $(\boldsymbol{\mathcal{C}}^\mathsf{T}, \mathsf{U})$ of all T-algebras, where the objects of $\boldsymbol{\mathcal{C}}^\mathbb{T}$ are those T-algebras $(X, \varphi) \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}^\mathsf{T}$ for which

(i) $\varphi \cdot \eta_X = id_X$ and

(ii) $\varphi \cdot \mathsf{T}\varphi = \varphi \cdot \mu_X$

holds. The objects of $\boldsymbol{\mathcal{C}}^{\mathbb{T}}$ are called $\mathbb{T}$**-algebras**. ◇

4.4.4.3 Proposition (monads induce adjunctions, [AHS90] Prop. 20.7**).** Every monad $\mathbb{T} = (\mathsf{T}, \eta, \mu)$ on $\boldsymbol{\mathcal{C}}$ gives rise to an adjunction $(\eta, \varepsilon) : \mathsf{F}^{\mathbb{T}} \dashv \mathsf{U}^{\mathbb{T}} : \boldsymbol{\mathcal{C}}^{\mathbb{T}} \leftarrow \boldsymbol{\mathcal{C}}$, where

(i) $(\boldsymbol{\mathcal{C}}^{\mathbb{T}}, \mathsf{U}^{\mathbb{T}})$ is the Eilenberg-Moore category from Definition 4.4.4.2,

(ii) $\mathsf{F}^{\mathbb{T}}(X \xleftarrow{f} Y) = (\mathsf{T}X, \mu_X) \xleftarrow{\mathsf{T}f} (\mathsf{T}Y, \mu_Y)$, and in particular $(\mathsf{T}X, \mu_X)$ is a free object over X in $(\boldsymbol{\mathcal{C}}^{\mathbb{T}}, \mathsf{U}^{\mathbb{T}})$.

(iii) $\varepsilon_{(X,\varphi)} = \varphi$.

Moreover, the monad associated with the above adjunction according to Lemma 4.4.4.1 is $\mathbb{T}$ itself. ◇

4.4.4.4 Definition (monadic category and monadic functor).

(i) A concrete category $(\boldsymbol{\mathcal{D}}, \mathsf{U})$ built upon $\boldsymbol{\mathcal{C}}$ is called **monadic** (over $\boldsymbol{\mathcal{C}}$) if there exists a monad $\mathbb{T}$ such that $(\boldsymbol{\mathcal{D}}, \mathsf{U}) \cong (\boldsymbol{\mathcal{C}}^{\mathbb{T}}, \mathsf{U}^{\mathbb{T}})$, where $(\boldsymbol{\mathcal{C}}^{\mathbb{T}}, \mathsf{U}^{\mathbb{T}})$ is the Eilenberg-Moore category of $\mathbb{T}$.

(ii) A functor $\mathsf{U} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ is called **monadic** if it is faithful and $(\boldsymbol{\mathcal{D}}, \mathsf{U})$ is monadic. ◇

4.4.4.5 Theorem ([AHS90] Th. 20.56**).** If $\Sigma : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$ is a varietor, then $(\boldsymbol{\mathcal{C}}^{\Sigma}, |\mathbf{.}|^{\Sigma})$ is monadic and the associated monad (Lemma 4.4.4.1) is a free monad over Σ. ◇

4.4.4.6 Corollary. Let $\boldsymbol{\mathcal{C}}$ be an algebraically complete cocartesian category. Then $(\mathbf{Mnd}\,\boldsymbol{\mathcal{C}}, |\mathbf{.}|)$ has free objects and thus $(\mathbf{.})^{\star} \dashv |\mathbf{.}|$ where $(\mathbf{.})^{\star}$ is the free functor (Observation 4.3.3.2 (i) & Definition 4.4.3.1 (iii)). ◇

4.4.4.7 Corollary. Let $\boldsymbol{\mathcal{C}}$ be a cocartesian category, $\Sigma : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$ be a varietor, and X a $\boldsymbol{\mathcal{C}}$-object. Then from Theorem 4.4.4.5 together with Proposition 4.3.2.4 follows: $\mathsf{T}_{\Sigma}X \cong \boldsymbol{\mu}(\Sigma + \underline{X})$. ◇

4.4.4.8 Definition (Kleisli category [Kle65]**).** The **Kleisli category** of a monad $\mathbb{T} = (\mathsf{T}, \eta, \mu)$ (or of the associated Kleisli Triple $\big(\mathsf{M}, \eta, (\mathbf{.})^{\dagger}\big)$) is the following concrete category $(\boldsymbol{\mathcal{C}}_{\mathbb{T}}, \mathsf{U}_{\mathbb{T}})$ built upon $\boldsymbol{\mathcal{C}}$ with objects $\mathrm{Ob}\,\boldsymbol{\mathcal{C}}_{\mathbb{T}} = \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$, morphisms $\boldsymbol{\mathcal{C}}_{\mathbb{T}}(X, Y) = \boldsymbol{\mathcal{C}}(\mathsf{T}X, Y)$, identities $id = \eta$, composition $f \bullet_{\mathbb{T}} g = \mu \cdot \mathsf{T}f \cdot g = f^{\dagger} \cdot g$, and forgetful functor $\mathsf{U}_{\mathbb{T}} f = \mu \cdot \mathsf{T}f = f^{\dagger}$. ◇

Part II

Tree transducer composition in category theory

5 The initial algebra approach

In this chapter we will describe the semantics and the composition of tree transducers in terms of category theory. Moreover we will describe short cut fusion in terms of category theory as well, and compare it to the composition of tree transducers. We developed this idea in [JV01], presented it in [Jür01], and published it in [JV04].

Short cut fusion is based on the *cata/build-rule* [GLP93], *cata/augment-rule* [Gil96], or –using category theory– on the *acid rain theorem* [TM95]. For the latter, it is necessary to represent the consumer as a *catamorphism*. A catamorphism is a generalization of the well known list-function *foldr* for arbitrary regular types. In terms of category theory a catamorphism is the unique algebra morphism from an *initial algebra*. To describe the polymorphic *build*-function we will use an *algebra transformer* as suggested by [Fok92b].

5.1 Algebra Transformers

A function which maps algebras onto algebras is called an *algebra transformer* in [Fok92b]. We will use algebra transformers to express functions which are parameterized by constructors: *e.g.* the function $(\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}}$ is parameterized by the constructors encoded in the initial G-algebra in_{G}. Here the function H is an algebra transformer mapping F-algebras to G-algebras.

5.1.1 Characterization of concrete algebra transformers

5.1.1.1 Definition (algebra transformer). A **((semi)concrete) algebra transformer** is a (semi(concrete)) functor between two categories of algebras over a common base category. ◇

5.1.1.2 Lemma (characterization of concrete algebra transformers). Let $\mathcal{C}$ be a category, $\mathsf{F}, \mathsf{G}, \mathsf{U} : \mathcal{C} \leftarrow \mathcal{C}$ be endofunctors where U is faithful and $\mathsf{H} : \mathrm{Ob}\,\mathcal{C}^{\mathsf{F}} \leftarrow \mathrm{Ob}\,\mathcal{C}^{\mathsf{G}}$ a function. The following two statements are equivalent:

(i) The function H can be uniquely extended on $\mathcal{C}^{\mathsf{G}}$-morphisms to a concrete algebra transformer:
$$\mathsf{H} : (\mathcal{C}^{\mathsf{F}}, |\,\boldsymbol{.}\,|^{\mathsf{F}}) \leftarrow (\mathcal{C}^{\mathsf{G}}, \mathsf{U} \cdot |\,\boldsymbol{.}\,|^{\mathsf{G}}).$$

(ii) The function H satisfies the following condition: for every $\varphi, \varphi' \in \mathrm{Ob}\,\mathcal{C}^{\mathsf{G}}$ and every

$f : |\varphi|^{\mathsf{G}} \leftarrow_{\mathcal{C}} |\varphi'|^{\mathsf{G}}$:

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

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

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

If this is a functor $\mathsf{H} : \mathcal{C}^{\mathsf{F}} \leftarrow \mathcal{C}^{\mathsf{G}}$, then it is obviously concrete, *i.e.*: $\mathsf{H} : (\mathcal{C}^{\mathsf{F}}, |\,.\,|^{\mathsf{F}}) \leftarrow (\mathcal{C}^{\mathsf{G}}, \mathsf{U} \cdot |\,.\,|^{\mathsf{G}})$ and thus uniquely determined due to Lemma 4.2.1.5. The function H satisfies the functor axioms by construction, because U is a functor. The only property that we have to verify is $\mathsf{H} : \mathrm{Mor}\, \mathcal{C}^{\mathsf{F}} \leftarrow \mathrm{Mor}\, \mathcal{C}^{\mathsf{G}}$, *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 4.1.7.1.

(i) $\Longrightarrow$ (ii): If H can be extended uniquely to a concrete algebra transformer $\mathsf{H} : (\mathcal{C}^{\mathsf{F}}, |\,.\,|^{\mathsf{F}}) \leftarrow (\mathcal{C}^{\mathsf{G}}, \mathsf{U} \cdot |\,.\,|^{\mathsf{G}})$, then in particular $\mathsf{H} : \mathrm{Mor}\, \mathcal{C}^{\mathsf{F}} \leftarrow \mathrm{Mor}\, \mathcal{C}^{\mathsf{G}}$. Let $\varphi, \varphi' \in \mathrm{Ob}\, \mathcal{C}^{\mathsf{F}}$ 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 $(*)$. ∎

5.1.2 Construction of algebra transformers

The following corollary shows how to construct concrete algebra transformers using the condition from Lemma 5.1.1.2.

5.1.2.1 Corollary (Construction of concrete algebra transformers). With the preconditions from Lemma 5.1.1.2, each of the following definitions yields an $\mathsf{H} : (\mathcal{C}^{\mathsf{G}}, |\,.\,|^{\mathsf{G}}) \leftarrow (\mathcal{C}^{\mathsf{F}}, |\,.\,|^{\mathsf{F}})$. For every $\varphi \in \mathrm{Ob}\, \mathcal{C}^{\mathsf{F}}$:

(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 $\mathsf{H}_1 : (\mathcal{C}^{\mathsf{G}_1}, |\,.\,|^{\mathsf{G}_1}) \leftarrow (\mathcal{C}^{\mathsf{F}}, |\,.\,|^{\mathsf{F}})$ and $\mathsf{H}_2 : (\mathcal{C}^{\mathsf{G}_2}, |\,.\,|^{\mathsf{G}_2}) \leftarrow (\mathcal{C}^{\mathsf{F}}, |\,.\,|^{\mathsf{F}})$ and $\mathsf{G} = \mathsf{G}_1 + \mathsf{G}_2$,

Proof. All cases are instances of Lemma 5.1.1.2 with $\mathsf{U}_{\mathsf{G}} = \mathsf{Id}$. ∎

5.2 Generalized *acid rain theorems*

5.2.1 The *acid rain theorem*

We begin with a version of the well known *acid rain theorem* [TM95]. Usually a *free theorem* [Wad89] is used to proof the *acid rain theorem*. Since we are in the setting of category theory, it will be more natural to use naturalness (Definition 4.1.3.1). However *free theorems* entail the naturalness of polymorphic functions [dB89].

5.2.1.1 Proposition (*acid rain theorem* ([TM95] Theorem 3.2)). Let $\mathcal{C}$ be a category, $B \in \mathrm{Ob}\,\mathcal{C}$, $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{C}$ have an initial algebra, and $\varphi \in \mathrm{Ob}\,\mathcal{C}^{\mathsf{F}}$. Then:

$$\frac{\tau : |\,\boldsymbol{.}\,|^{\mathsf{F}} \dot{\leftarrow} \underline{B}}{(\!|\varphi|\!)_{\mathsf{F}} \cdot \tau_{in_{\mathsf{F}}} = \tau_{\varphi}}$$

Proof.

$$\begin{aligned}
& (\!|\varphi|\!)_{\mathsf{F}} \cdot \tau_{in_{\mathsf{F}}} \\
= \quad & \{ \text{ Definition 4.1.7.1 } \} \\
& |\mathbf{i}_{\varphi}|^{\mathsf{F}} \cdot \tau_{in_{\mathsf{F}}} \\
= \quad & \{ \text{ naturalness of } \tau : |\,\boldsymbol{.}\,|^{\mathsf{F}} \dot{\leftarrow} \underline{B} \text{ (Definition 4.1.3.1) } \} \\
& \tau_{\varphi} \cdot \underline{B}(\mathbf{i}_{\varphi}) \\
= \quad & \{ \text{ Definition 4.1.2.1 } \} \\
& \tau_{\varphi}
\end{aligned}$$

■

5.2.1.2 Note (*short cut fusion*). The *acid rain theorem* is a category theory version of the *cata/build-rule* from *short cut fusion*: Let $\mathbf{\Lambda}$ be a functional language, $\mathcal{C}$ be an appropriate category, and $[\![\,\boldsymbol{.}\,]\!] : \mathcal{C} \leftarrow \mathbf{\Lambda}$ be a denotational semantics. If $[\![\,\boldsymbol{.}\,]\!]$ is sound and parametric (*i.e.* polymorphic functions are mapped onto *natural* transformations) then the *acid rain theorem* tells us that the *cata/build-rule* holds in $\mathbf{\Lambda}$ w.r.t. $[\![\,\boldsymbol{.}\,]\!]$. However, the image of $[\![\,\boldsymbol{.}\,]\!]$ may be trivial. But if moreover $[\![\,\boldsymbol{.}\,]\!]$ is correct (*i.e.* reflects equality in $\mathcal{C}$ to observational equivalence in $\mathbf{\Lambda}$) then the *cata/build-rule* holds in $\mathbf{\Lambda}$ modulo observational equivalence.

$$\frac{\tau : |\,\boldsymbol{.}\,|^{\mathsf{F}} \dot{\leftarrow} \underline{B}}{(\!|\varphi|\!)_{\mathsf{F}} \cdot \tau_{in_{\mathsf{F}}} = \tau_{\varphi}}$$

$$\frac{build :: (\forall\, \beta.(\alpha \to \beta \to \beta) \to \beta) \to [\alpha]}{foldr\, c\, n (build\, g) = g\, c\, n}$$

A semantics which maps polymorphic functions onto natural transformations is called a **parametric model**. In [Joh01] a sound and complete parametric model for the language **PolyFix** is used to prove the correctness of *short cut fusion*.

In order to establish a parametric model for a language, the **free theorems** [Wad89, dB89]] have to hold for that language. ◇

In the previous Proposition we needed a natural transformation. Here is one that we can use:

5.2.1.3 Lemma (naturalness of the catamorphism). Let $\mathcal{C}$ be a category and $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{C}$ have an initial algebra. Then:

$$(\!|\,\centerdot\,|\!)_{\mathsf{F}} = \big((\!|\varphi|\!)_{\mathsf{F}}\big)_{\varphi \in \mathrm{Ob}(\mathcal{C}^{\mathsf{F}})} : |\,\centerdot\,|^{\mathsf{F}} \dot{\leftarrow} \underline{\mu\mathsf{F}}$$

Proof. From Corollary 4.1.4.5 we know $\mathsf{i} : \mathsf{Id} \dot{\leftarrow} \underline{in_{\mathsf{F}}}$. Then $(\!|\,\centerdot\,|\!)_{\mathsf{F}} = |\mathsf{i}|^{\mathsf{F}} : |\,\centerdot\,|^{\mathsf{F}} \dot{\leftarrow} \underline{\mu\mathsf{F}}$ with Definition and Lemma 4.1.3.6 (i). ∎

Now we can state an instance of the *acid rain theorem* where we do not need *free theorems* or explicit naturalness at all:

5.2.1.4 Corollary (fusion of two catamorphisms). Let $\mathcal{C}$ be a category and $\mathsf{F}, \mathsf{G} : \mathcal{C} \leftarrow \mathcal{C}$ be functors which have initial algebras. Then:

$$\frac{\mathsf{H} : (\mathcal{C}^{\mathsf{F}}, |\,\centerdot\,|^{\mathsf{F}}) \leftarrow (\mathcal{C}^{\mathsf{G}}, |\,\centerdot\,|^{G})}{(\!|\varphi|\!)_{\mathsf{G}} \cdot (\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}} = (\!|\mathsf{H}\varphi|\!)_{\mathsf{F}}}$$

Proof. This is just an instance of Lemma 5.2.1.1 where $\tau_{\varphi} = (\!|\mathsf{H}\varphi|\!)_{\mathsf{F}}$. Where the latter is natural in φ according to Lemma 5.2.1.3 and Definition and Lemma 4.1.3.6 (ii). ∎

5.2.2 Generalized *acid rain theorems*

For our purpose we will need a more general *acid rain theorem* which we can apply conveniently to mutually recursive programs. We developed these generalizations of the *acid rain theorem* in [Jür99] and in [Jür00].

5.2.2.1 Proposition (mutual *acid rain theorem*). Let $\mathcal{C}$ be a category and $\mathsf{F}, \mathsf{U} : \mathcal{C} \leftarrow \mathcal{C}$ be endofunctors where F has an initial algebra. Let $B \in \mathrm{Ob}\,\mathcal{C}$ and $\varphi \in \mathrm{Ob}\,\mathcal{C}^{\mathsf{F}}$. Then:

$$\frac{\tau : |\,\centerdot\,|^{\mathsf{F}} \dot{\leftarrow} \underline{B} \qquad \mathsf{H} : (\mathcal{C}^{\mathsf{F}}, |\,\centerdot\,|^{\mathsf{F}}) \leftarrow (\mathcal{C}^{\mathsf{G}}, \mathsf{U} \cdot |\,\centerdot\,|^{\mathsf{G}})}{\mathsf{U}(\!|\varphi|\!)_{\mathsf{G}} \cdot \tau_{\mathsf{H}in_{\mathsf{G}}} = \tau_{\mathsf{H}\varphi}}.$$

Proof. This is a generalization of the proof of Lemma 5.2.1.1:

$$
\begin{aligned}
& \mathsf{U}(\!|\varphi|\!)_{\mathsf{G}} \cdot \tau_{\mathsf{H}in_{\mathsf{G}}} \\
= \quad & \{ \text{ Definition 4.1.7.1 \& Definition and Lemma 4.1.3.6 } \} \\
& \mathsf{U}|\mathbf{i}_{\varphi}|^{\mathsf{G}} \cdot (\tau\mathsf{H})_{in_{\mathsf{G}}} \\
= \quad & \{ \text{ naturalness of } \tau\mathsf{H} : |\,\boldsymbol{.}\,|^{\mathsf{F}} \cdot \mathsf{H} \dot{\leftarrow} \underline{B} \text{ (Definition 4.1.3.1) \& } |\,\boldsymbol{.}\,|^{\mathsf{F}} \cdot \mathsf{H} = \mathsf{U} \cdot |\,\boldsymbol{.}\,|^{\mathsf{G}} \} \\
& (\tau\mathsf{H})_{\varphi} \cdot \underline{B}(\mathbf{i}_{\varphi}) \\
= \quad & \{ \text{ Definition 4.1.2.1 \& Definition and Lemma 4.1.3.6 } \} \\
& \tau_{\mathsf{H}\varphi}
\end{aligned}
$$

■

Finally we give a symmetric version of the *acid rain theorem* (*i.e.* producer and consumer have the same structure):

5.2.2.2 Theorem (functorial *acid rain theorem*). Let $\boldsymbol{\mathcal{C}}$ be a category. Let $\boldsymbol{\mathcal{D}}$ be the full (meta-)category of the category of concrete categories with concrete functors where

$$\mathrm{Ob}(\boldsymbol{\mathcal{D}}) = \big\{(\boldsymbol{\mathcal{C}}^{\mathsf{F}}, \mathsf{U}_{\mathsf{F}} \cdot |\,\boldsymbol{.}\,|^{\mathsf{F}}) \;\big|\; \mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}},\, \boldsymbol{\mathcal{C}}^{\mathsf{F}} \text{ has an initial object, } \mathsf{U}_{\mathsf{F}} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}} \text{ faithful}\big\}.$$

We define the function M by

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

Then M is a functor

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

and thus in particular:

$$\frac{(\boldsymbol{\mathcal{C}}^{\mathsf{F}_3}, \mathsf{U}_{\mathsf{F}_3} \cdot |\,\boldsymbol{.}\,|^{\mathsf{F}_3}) \xleftarrow{\mathsf{H}_2}_{\boldsymbol{\mathcal{D}}} (\boldsymbol{\mathcal{C}}^{\mathsf{F}_2}, \mathsf{U}_{\mathsf{F}_2} \cdot |\,\boldsymbol{.}\,|^{\mathsf{F}_2}) \xleftarrow{\mathsf{H}_1}_{\boldsymbol{\mathcal{D}}} (\boldsymbol{\mathcal{C}}^{\mathsf{F}_1}, \mathsf{U}_{\mathsf{F}_1} \cdot |\,\boldsymbol{.}\,|^{\mathsf{F}_1})}{\mathsf{U}_{\mathsf{F}_2}(\!|\mathsf{H}_1 in_{\mathsf{F}_1}|\!)_{\mathsf{F}_2} \cdot \mathsf{U}_{\mathsf{F}_3}(\!|\mathsf{H}_2 in_{\mathsf{F}_2}|\!)_{\mathsf{F}_3} = \mathsf{U}_{\mathsf{F}_3}(\!|\mathsf{H}_2(\mathsf{H}_1 in_{\mathsf{F}_1})|\!)_{\mathsf{F}_3}}.$$

Proof. We have to verify the functor axioms for M:

(i) $\mathsf{M}\,\mathsf{Id} = \mathsf{U}_{\mathsf{F}}(\!|in_{\mathsf{G}}|\!)_{\mathsf{F}} = \mathsf{U}_{\mathsf{F}} id = id$ using $(\!|\,\boldsymbol{.}\,|\!)$-reflection.

(ii) $\mathsf{M}\,\mathsf{H}_2 \cdot \mathsf{M}\,\mathsf{H}_1 = \mathsf{U}_{\mathsf{F}_3}(\!|\mathsf{H}_2 in_{\mathsf{F}_2}|\!)_{\mathsf{F}_3} \cdot \mathsf{U}_{\mathsf{F}_2}(\!|\mathsf{H}_1 in_{\mathsf{F}_1}|\!)_{\mathsf{F}_2} == \mathsf{M}(\mathsf{H}_2 \cdot \mathsf{H}_1)$ using Proposition 5.2.2.1 ■

The functor M is a generalization of the *type functor* $\boldsymbol{\mu} : (\boldsymbol{\mu}\mathsf{F} \xleftarrow{(\!|in_{\mathsf{F}} \cdot \tau|\!)_{\mathsf{G}}} \boldsymbol{\mu}\mathsf{G}) \mapsto (\mathsf{F} \xleftarrow{\tau} \mathsf{G})$ (*cf.* [BdM97] Section 2.7), *i.e.* $\mathsf{M}\,\mathsf{H} = \boldsymbol{\mu}\,\tau$ where $\forall\,\varphi.\ \mathsf{H}\varphi = \varphi \cdot \tau$ (Later in Corollary 5.1.2.1 (ii) we will see that this indeed defines a concrete functor H).

5.2.2.3 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 Lemma 4.2.1.8 restricted to $\mathcal{D}$ defined in Theorem 5.2.2.2.

Proof. For every

$$(\mathcal{C}^{\mathsf{F}}, \mathsf{U}_{\mathsf{F}} \cdot |\,\centerdot\,|^{\mathsf{F}}) \xleftarrow{\mathsf{H}}_{\mathcal{D}} (\mathcal{C}^{\mathsf{G}}, \mathsf{U}_{\mathsf{G}} \cdot |\,\centerdot\,|^{\mathsf{G}})$$

we calculate:

$$\mathsf{M}({}_{\mathsf{U}}\mathsf{H}) = (\mathsf{U} \cdot \mathsf{U}_{\mathsf{F}})(\!|\mathsf{H}|in_{\mathsf{G}}|\!)_{\mathsf{F}} = \mathsf{U}\big(\mathsf{U}_{\mathsf{F}}(\!|\mathsf{H}|in_{\mathsf{G}}|\!)_{\mathsf{F}}\big) = \mathsf{U}(\mathsf{M}\,\mathsf{H}). \tag{5.2.2.1}$$

■

5.3 Algebraic transducers

5.3.1 Syntax of algebraic transducers

5.3.1.1 Definition (algebraic transducer). Let F and G be $\mathcal{C}$-endofunctors, such that $\mathcal{C}^{\mathsf{F}}$ has an initial object. A triple $(\mathsf{H}, \mathsf{U}, \pi)$ is called an **algebraic transducer**[1] (from F to G) on $\mathcal{C}$ if

(i) $\mathsf{H} : (\mathcal{C}^{\mathsf{F}}, |\,\centerdot\,|^{\mathsf{F}}) \leftarrow (\mathcal{C}^{\mathsf{G}}, \mathsf{U} \cdot |\,\centerdot\,|^{\mathsf{G}})$ is a concrete functor,

(ii) $\mathsf{U} : \mathcal{C} \leftarrow \mathcal{C}$ is a faithful endofunctor, and

(iii) $\pi : \mathsf{Id} \dot{\leftarrow} \mathsf{U}$ is a natural transformation.

We write $(\mathsf{H}, \mathsf{U}, \pi) : \mathsf{G} \leftarrow \mathsf{F}$. ◇

5.3.2 Composition of algebraic transducers

5.3.2.1 Definition (fusion of algebraic transducers). Let $\mathcal{C}$ be a category and $\mathsf{F}_1, \mathsf{F}_2, \mathsf{F}_3 : \mathcal{C} \leftarrow \mathcal{C}$. Let $(\mathsf{H}_1, \mathsf{U}_1, \pi_1) : \mathsf{F}_2 \leftarrow \mathsf{F}_1$ and $(\mathsf{H}_2, \mathsf{U}_2, \pi_2) : \mathsf{F}_3 \leftarrow \mathsf{F}_2$ be algebraic transducers. The fusion of these two is the algebraic transducer defined by

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

[1] In [JV01] we called it *categorical transducer*. The name *algebraic transducer* emphasizes that it is defined using F-*algebras*, in contrast to the *monadic transducer* (Definition 6.3.1.1) which is defined using *monads* (Definition 4.4.1.1).

Sometimes—but rarely—a *push down transducer* (*i.e.* a character string push down automaton with additional output) is also called an *algebraic transducer*. This is not related to our work.

Moreover, we define for every $\mathcal{C}$-endofunctor F the identity algebraic transducer

$$\mathbb{ID}_{\mathsf{F}} = (\mathsf{Id}_{(\mathcal{C}^{\mathsf{F}}, |\,.\,|^{\mathsf{F}})}, \mathsf{Id}_{\mathcal{C}}, id_{\mathsf{Id}_{\mathcal{C}}}).$$

Notice that the fusion is defined by component-wise (commuted) composition and the identity is defined by component-wise identities. Thus it is obvious, that this gives rise to a category $\mathbf{AT}_{\mathcal{C}}$ where the objects are all $\mathcal{C}$-endofunctors which have initial algebras, the morphisms are all algebraic transducers between such functors, and composition is fusion. ◇

5.3.3 Denotational semantics of algebraic transducers

5.3.3.1 Definition (semantics of algebraic transducers). Let $\mathcal{C}$ be a category. For every $(\mathsf{H}, \mathsf{U}, \pi) \in \mathbf{AT}_{\mathcal{C}}(\mathsf{G}, \mathsf{F})$ we define

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

where in the right hand side expression the functor U is hidden in the codomain of the concrete functor $\mathsf{H} : (\mathcal{C}^{\mathsf{F}}, |\,.\,|^{\mathsf{F}}) \leftarrow (\mathcal{C}^{\mathsf{G}}, \mathsf{U} \cdot |\,.\,|^{\mathsf{G}})$. We call $[\![\,.\,]\!] : \operatorname{Mor} \mathcal{C} \leftarrow \operatorname{Mor} \mathbf{AT}_{\mathcal{C}}$ the **semantics** of algebraic transducers. Notice that $[\![\,.\,]\!]$ depends on the choice of the initial algebras of the functors in $\operatorname{Ob} \mathbf{AT}_{\mathcal{C}}$. In Definition 5.3.1.1 we demanded only the existence of initial algebras. From Lemma 4.1.4.3 we know that initial algebras are uniquely determined up to algebra isomorphism, thus we may choose an initial algebra out of every isomorphism class. ◇

5.3.3.2 Example (algebraic transducer). Let $\mathcal{C}$ be a bicartesian category. Let $\mathsf{F} = \underline{1} + \mathsf{Id} \times \mathsf{Id} : \mathcal{C} \leftarrow \mathcal{C}$ and $\mathsf{G} = \underline{1} + \mathsf{Id} + \mathsf{Id} : \mathcal{C} \leftarrow \mathcal{C}$ be functors which have initial algebras, where $in_{\mathsf{G}} = [N, A, B] : \boldsymbol{\mu}\mathsf{G} \leftarrow \mathsf{G}(\boldsymbol{\mu}\mathsf{G})$. For every $C \in \operatorname{Ob} \mathcal{C}$ 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 5.1.1.2 to prove that H can be uniquely extended to a concrete functor:

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

More precisely we show that for every $\varphi = [\varphi_1, \varphi_2, \varphi_3], \varphi' = [\varphi'_1, \varphi'_2, \varphi'_3] \in \operatorname{Ob}(\mathcal{C}^{\mathsf{G}})$ and every $f : |\varphi|^{\mathsf{G}} \leftarrow_{\mathcal{C}} |\varphi'|^{\mathsf{G}}$ 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 4.6). Moreover we calculate:

$$\begin{aligned}
& \langle \varphi_2 \cdot \pi_2 \cdot \pi_1, \varphi_3 \cdot \pi_1 \cdot \pi_2 \rangle \cdot ((f \times f) \times (f \times f)) \\
= \quad & \{ \text{ fusion (Table 4.4) and four times cancelation (Table 4.6) } \} \\
& \langle \varphi_2 \cdot f \cdot \pi_2 \cdot \pi_1, \varphi_3 \cdot f \cdot \pi_1 \cdot \pi_2 \rangle \\
= \quad & \{ \text{ precondition of } (*) \ \} \\
& \langle f \cdot \varphi_2' \cdot \pi_2 \cdot \pi_1, f \cdot \varphi_3' \cdot \pi_1 \cdot \pi_2 \rangle \\
= \quad & \{ \text{ fusion (i) (Table 4.6) } \} \\
& (f \times f) \cdot \langle \varphi_2' \cdot \pi_2 \cdot \pi_1, \varphi_3' \cdot \pi_1 \cdot \pi_2 \rangle.
\end{aligned}$$

Using the fusion (i) law (Table 4.7) we get

$$\begin{aligned}
& [\langle \varphi_1, \varphi_1 \rangle, \langle \varphi_2 \cdot \pi_2 \cdot \pi_1, \varphi_3 \cdot \pi_1 \cdot \pi_2 \rangle] \cdot (id_1 + ((f \times f) \times (f \times f))) \\
& \qquad = (f \times f) \cdot [\langle \varphi_1', \varphi_1' \rangle, \langle \varphi_2' \cdot \pi_2 \cdot \pi_1, \varphi_3' \cdot \pi_1 \cdot \pi_2 \rangle]
\end{aligned}$$

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

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

by means of Lemma 5.1.1.2. Obviously we have a natural transformation $\pi_1 : \mathsf{Id} \dot{\leftarrow} \mathsf{Id} \times \mathsf{Id}$. Finally we obtain that

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

is an algebraic transducer over $\boldsymbol{\mathcal{C}}$. ◇

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

$$[\![\,\mathbf{.}\,]\!] : \boldsymbol{\mathcal{C}} \leftarrow \mathbf{AT}_{\boldsymbol{\mathcal{C}}}$$

and thus in particular (compare Theorem 5.2.2.2 and Theorem 3.3.1.2):

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

Proof. $[\![\,\boldsymbol{.}\,]\!]$ satisfies the typing axiom by construction where $\forall\,\mathsf{F} \in \mathrm{Ob}\,\mathbf{AT}_{\mathcal{C}}\,.\,[\![\mathsf{F}]\!] = \mu\mathsf{F}$. In order to prove the other functor axioms, we use the functor M from Theorem 5.2.2.2 to express the function $[\![\,\boldsymbol{.}\,]\!]$: For every $\mathsf{H} : (\mathcal{C}^{\mathsf{F}}, |\,\boldsymbol{.}\,|^{\mathsf{F}}) \leftarrow (\mathcal{C}^{\mathsf{G}}, \mathsf{U} \cdot |\,\boldsymbol{.}\,|^{\mathsf{G}})$, every faithful $\mathsf{U} : \mathcal{C} \leftarrow \mathcal{C}$ and every $\pi : \mathsf{Id} \dot{\leftarrow} \mathsf{U}$ we know that $(|\mathsf{H}|, \mathsf{U}, \pi) \in \mathbf{AT}_{\mathcal{C}}(\mathsf{G}, \mathsf{F})$, and we obtain:

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

Thus $[\![(\mathsf{Id}, \mathsf{Id}, id)]\!] = id \cdot \mathsf{M}\,\mathsf{Id} = id$ and hence $[\![\,\boldsymbol{.}\,]\!]$ 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 5.3.1.1 and Note 4.2.1.7)

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

and calculate

$$\begin{aligned}
&[\![(|\mathsf{H}_2|, \mathsf{U}_2, \pi_2) \cdot (|\mathsf{H}_1|, \mathsf{U}_1, \pi_1)]\!]\\
=\;& \{\text{ Definition 5.3.1.1 }\}\\
&[\![(|\mathsf{H}_1| \cdot |\mathsf{H}_2|, \mathsf{U}_1 \cdot \mathsf{U}_2, \pi_1 * \pi_2)]\!]\\
=\;& \{\text{ Definition and Lemma 4.2.1.8: } |_{\mathsf{U}_1}\mathsf{H}_2| = |\mathsf{H}_2| \text{ and } |\,\boldsymbol{.}\,| \text{ is a functor }\}\\
&[\![(|\mathsf{H}_1 \cdot_{\mathsf{U}_1} \mathsf{H}_2|, \mathsf{U}_1 \cdot \mathsf{U}_2, \pi_1 * \pi_2)]\!]\\
=\;& \{\text{ see above }\}\\
&(\pi_1 * \pi_2) \cdot \mathsf{M}(\mathsf{H}_1 \cdot_{\mathsf{U}_1} \mathsf{H}_2)\\
=\;& \{\ \mathsf{M} \text{ is a contravariant functor (short cut fusion) }\}\\
&(\pi_1 * \pi_2) \cdot \mathsf{M}(_{\mathsf{U}_1}\mathsf{H}_2) \cdot \mathsf{M}\,\mathsf{H}_1\\
=\;& \{\text{ Definition and Lemma 4.1.3.7 of } * \text{ and Lemma 5.2.2.3 }\}\\
&\pi_1 \cdot \mathsf{U}_1\pi_2 \cdot \mathsf{U}_1(\mathsf{M}\,\mathsf{H}_2) \cdot \mathsf{M}\,\mathsf{H}_1\\
=\;& \{\ \mathsf{U}_1 \text{ is a functor }\}\\
&\pi_1 \cdot \mathsf{U}_1(\pi_2 \cdot \mathsf{M}\,\mathsf{H}_2) \cdot \mathsf{M}\,\mathsf{H}_1\\
=\;& \{\ \pi_1 : \mathsf{Id} \dot{\leftarrow} \mathsf{U}_1\ \}\\
&\pi_2 \cdot \mathsf{M}\,\mathsf{H}_2 \;\cdot\; \pi_1 \cdot \mathsf{M}\,\mathsf{H}_1\\
=\;& \{\text{ see above }\}\\
&[\![(|\mathsf{H}_2|, \mathsf{U}_2, \pi_2)]\!] \;\cdot\; [\![(|\mathsf{H}_1|, \mathsf{U}_1, \pi_1)]\!].
\end{aligned}$$

■

5.3.4 Algebraic 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 algebraic transducers. Then we can prove that homomorphic algebraic transducers have equal semantics and that isomorphism is a congruence w.r.t. to fusion, *i.e.* the composition of algebraic transducers.

5.3.4.1 Definition (algebraic transducer homomorphism).
(*cf.* Definition 3.2.2.5) Let $\mathcal{C}$ be a category and $\mathbb{A} = (\mathsf{H}, \mathsf{U}, \pi) : \mathsf{G} \leftarrow \mathsf{F}$ and $\mathbb{A}' = (\mathsf{H}', \mathsf{U}', \pi') : \mathsf{G} \leftarrow \mathsf{F}$ algebraic transducers over $\mathcal{C}$. A natural transformation

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

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

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

is called a **algebraic transducer homomorphism** to $\mathbb{A}$ from $\mathbb{A}'$ and we write:

$$\eta : \mathbb{A} \leftarrow \mathbb{A}'.$$

If η is a natural isomorphism, then we call it a **algebraic transducer isomorphism**. If an algebraic transducer isomorphism to $\mathbb{A}$ from $\mathbb{A}'$ exists, then we call $\mathbb{A}$ and $\mathbb{A}'$ **isomorphic** and write $\mathbb{A} \cong \mathbb{A}'$. ◇

The condition (ii) from Definition 5.3.4.1 seems to be peculiar. The following Lemma 5.3.4.2 shows an equivalent statement which is closer to Definition 3.2.2.5 (ii):

5.3.4.2 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 $\mathsf{H}, \mathsf{H}' : (\mathcal{C}^{\mathsf{F}}, |\mathbf{.}|^{\mathsf{F}}) \leftarrow (\mathcal{C}^{\mathsf{G}}, \mathsf{U} \cdot |\mathbf{.}|^{\mathsf{G}})$ be concrete functors.

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

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

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

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

(i) From the definition of $|\mathbf{.}|^{\mathsf{F}}$ 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 $|\mathbf{.}|^{\mathsf{F}}$ is faithful, its restriction to $\mathcal{C}^{\mathsf{F}}(\mathsf{H}\varphi, \mathsf{H}'\varphi)$ is injective and we have $(\mathsf{H}\varphi, \eta_{|\varphi|_{\mathsf{G}}}, \mathsf{H}'\varphi) = \tilde{\eta}_{\varphi}$.

(ii) $\Rightarrow$:

$$\begin{aligned}
& \tilde{\eta} : \mathsf{H} \dot{\leftarrow} \mathsf{H}' \\
\Longrightarrow \quad & \{ \text{ Definition 4.1.3.1 } \} \\
& \tilde{\eta}_\varphi : \mathsf{H}\varphi \leftarrow_{\mathcal{C}^\mathsf{F}} \mathsf{H}'\varphi \\
\Longleftrightarrow \quad & \{ \text{ Definition 4.1.7.1 } \} \\
& |\tilde{\eta}_\varphi|^\mathsf{F} \cdot \mathsf{H}'\varphi = \mathsf{H}\varphi \cdot \mathsf{F}|\tilde{\eta}_\varphi|^\mathsf{F} \\
\Longleftrightarrow \quad & \{ \text{ Definition 4.1.3.4 and definition of } \tilde{\eta} \ \} \\
& \eta \cdot \mathsf{H}'\varphi = \mathsf{H}\varphi \cdot \mathsf{F}\eta.
\end{aligned}$$

$\Leftarrow$: 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:

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

■

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

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

Proof.

$$\begin{aligned}
& (\!|\mathsf{H}\varphi|\!)_\mathsf{F} \\
= \quad & \{ \text{ fusion (Table 4.8) with } |\tilde{\eta}_\varphi|^\mathsf{F} \cdot \mathsf{H}'\varphi = \mathsf{H}\varphi \cdot \mathsf{F}|\tilde{\eta}_\varphi|^\mathsf{F} \ \} \\
& |\tilde{\eta}_\varphi|^\mathsf{F} \cdot (\!|\mathsf{H}'\varphi|\!)_\mathsf{F} \\
= \quad & \{ \text{ Definition 5.3.4.1 (ii) } \} \\
& \eta_{|\varphi|_\mathsf{G}} \cdot (\!|\mathsf{H}'\varphi|\!)_\mathsf{F} \\
= \quad & \{ \text{ Definition 4.1.3.4 } \} \\
& \eta \cdot (\!|\mathsf{H}'\varphi|\!)_\mathsf{F}.
\end{aligned}$$

■

5.3.4.4 Theorem (hom. preserve semantics of algebraic transducers). (*cf.* Lemma 3.2.2.6) Let $\mathcal{C}$ be a category and $\mathbb{A}, \mathbb{A}' : \mathsf{G} \leftarrow \mathsf{F}$ algebraic transducers over $\mathcal{C}$. Then:

$$\frac{\exists\, \eta : \mathbb{A} \leftarrow \mathbb{A}'}{[\![\mathbb{A}]\!] = [\![\mathbb{A}']\!]}.$$

Proof. Let $\mathbb{A} = (\mathsf{H}, \mathsf{U}, \pi)$ and $\mathbb{A}' = (\mathsf{H}', \mathsf{U}', \pi')$ and $\eta : \mathbb{A} \leftarrow \mathbb{A}'$ be an algebraic transducer homomorphism. We calculate:

$$\begin{aligned}
& [\![\mathbb{A}]\!] \\
= \quad & \{ \text{ Definition 5.3.3.1 } \} \\
& \pi \cdot (\!|\mathsf{H} in_{\mathsf{G}}|\!)_{\mathsf{F}} \\
= \quad & \{ \text{ Lemma 5.3.4.3 } \} \\
& \pi \cdot \eta \cdot (\!|\mathsf{H}' in_{\mathsf{G}}|\!)_{\mathsf{F}} \\
= \quad & \{ \text{ Definition 5.3.4.1 } \} \\
& \pi' \cdot (\!|\mathsf{H}' in_{\mathsf{G}}|\!)_{\mathsf{F}} \\
= \quad & \{ \text{ Definition 5.3.3.1 } \} \\
& [\![\mathbb{A}']\!].
\end{aligned}$$

■

5.3.4.5 Theorem (vertical composition of algebraic transducer hom.). Let $\mathcal{C}$ be a category and

$$\mathsf{F}_3 \xleftarrow{\mathbb{A}_2 = (\mathsf{H}_2, \mathsf{U}_2, \pi_2),\ \mathbb{A}'_2 = (\mathsf{H}'_2, \mathsf{U}'_2, \pi'_2)} \mathsf{F}_2 \xleftarrow{\mathbb{A}_1 = (\mathsf{H}_1, \mathsf{U}_1, \pi_1),\ \mathbb{A}'_1 = (\mathsf{H}'_1, \mathsf{U}'_1, \pi'_1)} \mathsf{F}_1$$

be algebraic transducers over $\mathcal{C}$. Then:

$$\frac{\mathbb{A}_2 \xleftarrow{\eta_2} \mathbb{A}'_2 \qquad \mathbb{A}_1 \xleftarrow{\eta_1} \mathbb{A}'_1}{\mathbb{A}_2 \cdot \mathbb{A}_1 \xleftarrow{\eta_1 * \eta_2} \mathbb{A}'_2 \cdot \mathbb{A}'_1}.$$

Proof. We have to show that the vertical composition $\eta_1 * \eta_2$ of η_1 and η_2 is an algebraic transducer homomorphism to $\mathbb{A}_2 \cdot \mathbb{A}_1 = (\mathsf{H}_1 \cdot \mathsf{H}_2, \mathsf{U}_1 \cdot \mathsf{U}_2, \pi_1 * \pi_2)$ from $\mathbb{A}'_2 \cdot \mathbb{A}'_1 = (\mathsf{H}'_1 \cdot \mathsf{H}'_2, \mathsf{U}'_1 \cdot \mathsf{U}'_2, \pi'_1 * \pi'_2)$. according to Definition 5.3.4.1:

(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 $|\widetilde{\eta_1 * \eta_2}|^{\hat{\mathsf{F}}_1} = (\eta_1 * \eta_2)|\,.\,|^{\mathsf{F}_3}$ holds: For every $\mathcal{C}$-endofunctor F we use the abbreviation $\hat{\mathsf{F}}$ for the identical natural transformation from the forgetful functor $|\,.\,|^{\mathsf{F}}$ to

itself, *i.e.* $\hat{\mathsf{F}} = id_{|\bullet|^{\mathsf{F}}} : |\bullet|^{\mathsf{F}} \dot{\leftarrow} |\bullet|^{\mathsf{F}}$. Notice that from Definition and Lemma 4.1.3.7 for every algebraic transducer homomorphism between algebraic transducers in $\mathbf{AT}_{\mathcal{C}}(\mathsf{G},\mathsf{F})$ we have $\hat{\mathsf{F}} * \tilde{\eta} = |\tilde{\eta}|^{\mathsf{F}}$ and $\eta_{|\bullet|^{\mathsf{G}}} = \eta * \hat{\mathsf{G}}$. 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 4.1.3.9): $\hat{\mathsf{F}}_1 * \widetilde{\eta_1 * \eta_2} = \hat{\mathsf{F}}_1 * \tilde{\eta}_1 * \tilde{\eta}_2 = \eta_1 * \hat{\mathsf{F}}_2 * \tilde{\eta}_2 = \eta_1 * \eta_2 * \hat{\mathsf{F}}_3$.

(iii) With Lemma 4.1.3.8 we have $(\pi_1 * \pi_2) \cdot (\eta_1 * \eta_2) = (\pi_1 \cdot \eta_1) * (\pi_2 \cdot \eta_2) = \pi'_1 * \pi'_2$. ■

5.3.4.6 Corollary (isomorphism is a congruence w.r.t. fusion). With the preconditions from Theorem 5.3.4.5 we have

$$\frac{\mathbb{A}_2 \cong \mathbb{A}'_2 \qquad \mathbb{A}_1 \cong \mathbb{A}'_1}{\mathbb{A}_2 \cdot \mathbb{A}_1 \cong \mathbb{A}'_2 \cdot \mathbb{A}'_1}.$$

Proof. We just have to show that the vertical composition of natural transformations $*$ preserves isomorphisms, which is easy to see with Lemma 4.1.3.8 and the obvious fact that $id * id = id$. ■

5.3.5 Top-down algebraic transducers

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

5.3.5.1 Definition (top-down algebraic transducer). Let $\mathcal{C}$ be a category which has finite products and finite coproducts. An algebraic transducer $(\mathsf{H},\mathsf{U},\pi) \in \mathbf{AT}_{\mathcal{C}}(\mathsf{G},\mathsf{F})$ is called a **top-down algebraic transducer** over $\mathcal{C}$ provided that

(i) F and G are bicartesian,

(ii) U is cartesian, and

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

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

5.3.5.2 Theorem (composition of top-down algebraic transducers). Let $\mathcal{C}$ be a category which has finite products and finite coproducts. The class of all top-down algebraic transducers over $\mathcal{C}$ is a subcategory of $\mathbf{AT}_{\mathcal{C}}$ which we will denote by *td-* $\mathbf{AT}_{\mathcal{C}}$

Proof. According to Definition 4.1.2.4 it suffices to show that $td\text{-}\mathbf{AT}_{\boldsymbol{\mathcal{C}}}$ contains the identities and is closed under composition. Obviously, the identical algebraic transducer $(\mathsf{Id}, \mathsf{Id}, id)$ is a top-down algebraic 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 algebraic transducers. By Definition 5.3.5.1 there exist finite products

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

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 algebraic transducer it is sufficient to show that

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

is a finite product. We define for every $\mathsf{U}'' : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$ and every $\tau_{ij} : \mathsf{Id} \leftarrow \mathsf{U}''$ 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 4.4): Let $\sigma : \mathsf{Id} \leftarrow \mathsf{U}''$ 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 $(i,j) \in I \times J$:

$$\begin{aligned}
&\pi_j' * \pi_i \\
=\ &\{\ \text{Definition and Lemma 4.1.3.7}\ \} \\
&\pi_j' \cdot \mathsf{U}'\pi_i \\
=\ &\{\ \mathsf{U}' = \prod_{k\in J} \mathsf{Id}\ \} \\
&\pi_j' \cdot \prod_{j\in J} \pi_i \\
=\ &\{\ \text{fusion (i) for product functors (Table 4.6)}\ \} \\
&\pi_i \cdot \pi_j'.
\end{aligned}$$

With this we continue:

$$\begin{aligned}
&\forall\,(i,j) \in I \times J.\ (\pi_j' * \pi_i) \cdot \sigma = \tau_{ij} \\
\iff\ &\{\ \text{see above}\ \} \\
&\forall\,(i,j) \in I \times J.\ \pi_i \cdot \pi_j' \cdot \sigma = \tau_{ij} \\
\iff\ &\{\ \text{UP (Table 4.4) for the product } \mathsf{U}\ \} \\
&\forall\, j \in J.\ \pi_j' \cdot \sigma = \langle\tau_{ij}\rangle_{i\in I} \\
\iff\ &\{\ \text{UP (Table 4.4) for the product } \mathsf{U}'\ \} \\
&\sigma = \langle\langle\tau_{ij}\rangle_{i\in I}\rangle_{j\in J} = \langle\tau_{ij}\rangle_{(i,j)\in I\times J}.
\end{aligned}$$

■

5.3.5.3 Lemma. Let $\mathcal{C}$ be a category. The class Ob *td-* $\mathbf{AT}_{\mathcal{C}}$ is an $\cong$-block in Ob $\mathbf{AT}_{\mathcal{C}}$, *i.e.* a disjoint union of algebraic transducer isomorphism classes.

Proof. Let $\mathbb{A} = (\mathsf{H}, \mathsf{U}, \pi)$ and $\mathbb{A}' = (\mathsf{H}', \mathsf{U}', \pi')$ be algebraic transducers over $\mathcal{C}$ with $\mathbb{A} \cong \mathbb{A}'$. We have to show: if $\mathbb{A}$ is a top-down algebraic transducer then $\mathbb{A}'$ is also a top-down algebraic transducer. Using Definition 5.3.4.1 and Definition 5.3.5.1 this is obvious, because we have a natural isomorphism $\eta : \mathsf{U} \leftarrow \mathsf{U}'$ with $\pi \cdot \eta = \pi'$. ∎

5.3.5.4 Definition. We define the following class:

$$TOP_{\mathbf{AT}} = \quad \big\{ [\![\mathbb{A}]\!] \;\big|\; \mathbb{A} \in \mathrm{Mor}\; td\text{-}\; \mathbf{AT}_{\mathcal{S}et} \big\},$$

which the reader should compare with *TOP* from Definition 3.2.2.3. ◇

5.4 Relating top-down tree transducers and algebraic transducers

In this section we describe a translation (Lemma 5.4.2.9) of a top-down tree transducer into a top-down algebraic transducer.

5.4.1 Category of forests

5.4.1.1 Definition (forest). Let Σ be a ranked alphabet. A Σ-**forest** is a tuple of Σ-trees, *i.e.* an element of $(T_\Sigma \emptyset)^*$. ◇

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 (Definition 4.4.4.8).

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

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

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

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

and $\forall\, n \in \mathrm{Ob}\, \boldsymbol{\mathcal{T}}_\Sigma$.

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

Notice that $\boldsymbol{\mathcal{T}}_\Sigma$ is actually a pre-category, which we view as a category according to Note 4.1.1.4. ◇

5.4.1.3 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

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

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

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

and can easily verify the UP of the product (Table 4.4). ∎

5.4.1.4 Note. The pairing of the product in $\boldsymbol{\mathcal{T}}_\Sigma$ from Lemma 5.4.1.3 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 4.1.5.2 is not associative, because it is defined by tupling and successive tupling is not associative. However it is associative up to isomorphism. ◇

5.4.1.5 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

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

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 5.4.1.2) 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 $\forall\, n \in \operatorname{Ob} \boldsymbol{\mathcal{T}}_\Sigma = \mathbb{N}_0$. $\mathsf{E}n = (T_\Sigma \emptyset)^n$ and thus $\forall\, (m_i)_{i=1}^n \in \operatorname{Ob} \boldsymbol{\mathcal{T}}_\Sigma^n$. $\mathsf{E}(\prod_{i=1}^n m_i) = \mathsf{E}(\sum_{i=1}^n m_i) = (T_\Sigma \emptyset)^{\sum_{i=1}^n m_i} \cong \prod_{i=1}^n (T_\Sigma \emptyset)^{m_i}$. ∎

5.4.1.6 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 $(T_\Sigma \emptyset)^n$, 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 2.2.1.2). 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 $\big\langle \langle \sigma \cdot \langle \alpha \cdot \mathbf{!}, \pi_1 \rangle, \alpha \cdot \mathbf{!}, \sigma \cdot \langle \sigma, \alpha \cdot \mathbf{!} \rangle \big\rangle : (T_\Sigma \emptyset)^3 \leftarrow (T_\Sigma \emptyset)^2$◇

5.4.1.7 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. ◇

5.4.2 Relating the semantics of top-down tree transducers and top-down algebraic transducers

5.4.2.1 Example (motivating example for relation). To motivate the description of the semantics of top-down tree transducers as algebraic transducers over $\boldsymbol{\mathcal{S}et}$, consider the top-down tree transducer T_{zigzag} from Example 3.1.1.2. We view the rules of R as equations in the category $\mathsf{T}_{Q \cup \Sigma \cup \Delta}$ 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 5.4.1.6 (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 5.4.1.6 (ii):

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

We use pairing to collect all the states:

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

On the left hand side we use the fusion law for the product (Table 4.4) and on the right hand side the fusion (i) law for product functors (Table 4.7) and cancelation law for products (Table 4.4) 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:

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

To the left hand side we apply the fusion law for coproducts (Table 4.5) and to the right hand side we apply the fusion (i) law for coproduct functors (Table 4.7) 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 $\mathsf{F} = \underline{1} + \mathsf{Id} \times \mathsf{Id} : \boldsymbol{\mathcal{S}et} \leftarrow \boldsymbol{\mathcal{S}et}$ has the least fixed point $\boldsymbol{\mu}\mathsf{F} = T_\Sigma \emptyset$ where the initial F-algebra is $[\alpha, \sigma] = in_\mathsf{F} : \boldsymbol{\mu}\mathsf{F} \leftarrow \mathsf{F}(\boldsymbol{\mu}\mathsf{F})$, 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 4.8), 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 $\mathsf{G} = \underline{1} + \mathsf{Id} + \mathsf{Id}$ 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 5.3.3.2 we get that

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

is an algebraic transducer over $\boldsymbol{\mathcal{S}et}$ with

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

It is worth mentioning that the fact that H is a certain concrete functor is important in Theorem 5.3.3.3, because it is a precondition for the functorial *acid rain theorem* (Theorem 5.2.2.2). ◇

5.4.2.2 Example (top-down algebraic transducer). The algebraic transducer $\mathbb{A}_{\text{zigzag}} = (\mathsf{H}, \mathsf{Id} \times \mathsf{Id}, \pi_1)$ from the Example 5.4.2.1 is a top-down algebraic transducer over $\boldsymbol{\mathcal{S}et}$. With the notations from the Example 5.4.2.1 we can show how a top-down algebraic transducer operates: for every $f : (T_\Sigma \emptyset)^2 \leftarrow (T_\Sigma \emptyset)^0$:

$$\begin{aligned}
&[\![\mathbb{A}_{\text{zigzag}}]\!] \cdot \sigma \cdot f \\
=&\pi_1 \cdot (\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}} \cdot in_{\mathsf{F}} \cdot \iota_2 \cdot f \\
=& \quad \{ \text{ UP (Table 4.8) } \} \\
&\pi_1 \cdot \mathsf{H}in_{\mathsf{G}} \cdot \mathsf{F}(\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}} \cdot \iota_2 \cdot f \\
=& \quad \{ \text{ definition of } \mathsf{H} \text{ and } \mathsf{F} \text{ and cancelation for coproduct functors (Table 4.7) } \} \\
&\pi_1 \cdot [\langle N, N \rangle, \langle A \cdot \pi_2 \cdot \pi_1, B \cdot \pi_1 \cdot \pi_2 \rangle] \cdot \iota_2 \cdot ((\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}} \times (\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}}) \cdot f \\
=& \quad \{ \text{ cancelation (Table 4.5 and Table 4.4) } \} \\
&A \cdot \pi_2 \cdot \pi_1 \cdot ((\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}} \times (\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}}) \cdot f \\
=& \quad \{ \text{ cancelation (Table 4.6) } \} \\
&A \cdot \pi_2 \cdot (\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}} \cdot \pi_1 \cdot f \\
=&A \cdot [\![(\mathsf{H}, \mathsf{Id} \times \mathsf{Id}, \pi_2)]\!] \cdot \pi_1 \cdot f.
\end{aligned}$$ ◇

5.4.2.3 Note (motivation for relation). We want to generalize the construction from Example 5.4.2.1. Let $T \in tdtt(\Delta, \Sigma)$ be a top-down tree transducer and $\mathbb{A} = (\mathsf{H}, \mathsf{U}, \pi) : \mathsf{G} \leftarrow \mathsf{F}$ be a top-down algebraic transducer over $\boldsymbol{\mathcal{S}et}$. In Table 5.1 we list the parts of T and $\mathbb{A}$ which correspond to each other. The following Definition 5.4.2.5 will define this correspondence formally.

5.4.2.4 Lemma (right hand side form). 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

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

top-down tree transducer	$\longleftrightarrow$	top-down algebraic transducer
$T = (Q, \Sigma, \Delta, q_0, R)$	$\longleftrightarrow$	$\mathbb{A} = (\mathsf{H}, \mathsf{U}, \pi) : \mathsf{G} \leftarrow \mathsf{F}$
Q	$\longleftrightarrow$	U
Σ	$\longleftrightarrow$	F
Δ	$\longleftrightarrow$	G
q_0	$\longleftrightarrow$	π
R	$\longleftrightarrow$	$\mathsf{H}in_{\mathsf{G}}$
$T_\Sigma\emptyset$	$\longleftrightarrow$	$\boldsymbol{\mu}\mathsf{F}$
$T_\Delta\emptyset$	$\longleftrightarrow$	$\boldsymbol{\mu}\mathsf{G}$
$[\![T]\!]$	$\longleftrightarrow$	$[\![\mathbb{A}]\!]$

Table 5.1: Relation between the components of top-down tree and top-down algebraic transducers

of R in the form (see Definition and Corollary 4.1.5.5 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

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

Proof.

$$\begin{aligned}
&\mathrm{rhs}'_{R,\sigma}\, q \cdot \prod^r \big(\langle q_i \rangle_{i=1}^l\big) \\
=\ & \{ \text{ Lemma 5.4.1.3 } \} \\
&\mathrm{rhs}'_{R,\sigma}\, q \cdot \prod_{t=1}^r \big(q_1 x_t, \ldots, q_l x_t\big) \\
=\ & \{ \text{ Lemma 5.4.1.3 and Definition and Corollary 4.1.5.5 } \} \\
&\mathrm{rhs}'_{R,\sigma}\, q \cdot (q_1 x_1, \ldots, q_l x_1,\ q_1 x_2, \ldots, q_l x_2,\ \ldots\ q_1 x_r, \ldots, q_l x_r) \\
=\ & \{ \text{ Definition 5.4.1.2 } \} \\
&[q_s x_t / x_{(t-1)\cdot l+s}]_{s=1\,t=1}^{l\quad r}(\mathrm{rhs}'_{R,\sigma}\, q) \\
=\ & \{ \text{ definition of rhs}' \text{ } \} \\
&\mathrm{rhs}_{R,\sigma}\, q.
\end{aligned}$$

■

5.4.2.5 Definition (relation). Let $T = (Q, \Sigma, \Delta, q_0, R)$ be a top-down tree transducer and $\mathbb{A} = (\mathsf{H}, \mathsf{U}, \pi) \in \mathbf{AT}_{\boldsymbol{\mathcal{S}et}}(\mathsf{G}, \mathsf{F})$ be an algebraic transducer. We call T and $\mathbb{A}$ **related** and write

$$T \approx \mathbb{A}$$

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

5.4.2.6 Example (relation). The top-down tree transducer T_{zigzag} from Example 3.1.1.2 and the algebraic transducer $\mathbb{A}_{\mathrm{zigzag}}$ from Example 5.4.2.1 are related $T_{\mathrm{zigzag}} \approx \mathbb{A}_{\mathrm{zigzag}}$. It is obvious by construction that (i) and (ii) from Definition 5.4.2.5 are satisfied. Let us have a look at (iii): From the definition of T_{zigzag} in Example 3.1.1.2 and with Lemma 5.4.2.4 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 5.4.1.2 and Lemma 5.4.1.3 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: $\pi_1, \pi_2 : T_\Delta\emptyset \leftarrow (T_\Delta\emptyset)^2$ and $\pi_1, \pi_2 : (T_\Delta\emptyset)^2 \leftarrow (T_\Delta\emptyset)^4$. We do this according to Definition 4.1.3.4, because projections are natural transformations due to Definition and Definition and Corollary 4.1.5.5. ◇

5.4.2.7 Theorem (relation implies semantic equivalence). Let $\mathbb{A}$ be a top-down algebraic transducer over $\boldsymbol{\mathcal{S}et}$ and T be a top-down tree transducer. If T and $\mathbb{A}$ are related, then they have equal semantics:

$$T \approx \mathbb{A} \implies [\![T]\!] = [\![\mathbb{A}]\!].$$

Proof. We use the notations from Definition 5.4.2.5 and let $Q = \{q_1, \ldots, q_l\}$ with $l \in \mathbb{N}$.

$$
\begin{aligned}
& (\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}} = \langle [\![T]\!]_q \rangle_{q \in Q} \\
\iff & \quad \{ \text{ UP (Table 4.8) } \} \\
& \langle [\![T]\!]_q \rangle_{q \in Q} \cdot [\sigma]_{\sigma \in \Sigma} = \mathsf{H}in_{\mathsf{G}} \cdot \mathsf{F} \langle [\![T]\!]_q \rangle_{q \in Q} \\
\iff & \quad \{ \text{ UP (Table 4.4 and Table 4.5) } \} \\
& \forall\, \sigma \in \Sigma.\, \forall\, q \in Q.\, [\![T]\!]_q \cdot \sigma = \pi_q \cdot \mathsf{H}in_{\mathsf{G}} \cdot \mathsf{F} \langle [\![T]\!]_q \rangle_{q \in Q} \cdot \iota_\sigma \\
\iff & \quad \{ \text{ definition of } \mathsf{F} \text{ in Definition 5.4.2.5; cancelation (Table 4.7) } \} \\
& \forall\, \sigma \in \Sigma.\, \forall\, q \in Q.\, [\![T]\!]_q \cdot \sigma = \pi_q \cdot \mathsf{H}in_{\mathsf{G}} \cdot \iota_\sigma \cdot \prod^{\mathrm{rank}_\Sigma\, \sigma} \big(\langle [\![T]\!]_q \rangle_{q \in Q} \big)
\end{aligned}
$$

$$
\begin{aligned}
\iff & \quad \{ \text{ Definition 5.4.2.5 (iii) } \} \\
& \forall\, \sigma \in \Sigma.\, \forall\, q \in Q.\, [\![T]\!]_q \cdot \sigma = \mathrm{rhs}'_{R,\sigma}\, q \cdot \prod^{\mathrm{rank}_\Sigma\, \sigma} \big(\langle [\![T]\!]_q \rangle_{q \in Q} \big) \\
\iff & \quad \{ \text{ pointwise on terms with Lemma 5.4.2.4 and Definition 5.4.1.2 } \} \\
& \forall\, \sigma \in \Sigma.\, \forall\, q \in Q.\, \forall\, (t_r)_{r=1}^{\mathrm{rank}_\Sigma\, \sigma} \in (T_\Sigma \emptyset)^{\mathrm{rank}_\Sigma\, \sigma}. \\
& \quad [\![T]\!]_q \big(\sigma(t_1, \ldots, t_{\mathrm{rank}_\Sigma\, \sigma}) \big) \\
& \qquad = \big[[\![T]\!]_{q_1} t_1/x_1, \ldots, [\![T]\!]_{q_l} t_1/x_l, [\![T]\!]_{q_1} t_2/x_{l+1}, \ldots, [\![T]\!]_{q_l} t_2/x_{2 \cdot l}, \ldots \big] (\mathrm{rhs}'_{R,\sigma}\, q) \\
& \qquad = \big[[\![T]\!]_p t_j/p x_j \big]_{\substack{p \in Q \\ x_j \in X_{\mathrm{rank}_\Sigma\, \sigma}}} (\mathrm{rhs}_{R,\sigma}\, q).
\end{aligned}
$$

The latter is the definition of $[\![\,.\,]\!]$ in Definition 3.2.2.1 and thus with Definition 5.4.2.5 (ii):

$$[\![\mathbb{A}]\!] = \pi \cdot (\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}} = [\![T]\!]_{q_0} = [\![T]\!].$$ ∎

5.4.2.8 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 2^{nd}-order substitution operators from Definition 2.2.2.6 (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): $g = f \cdot h$. ∎

5.4.2.9 Lemma. For every top-down tree transducer $T \in tdtt$ there exists a related top-down algebraic transducer $\mathbb{A} \in \mathrm{Mor}\, td\text{-}\mathbf{AT}_{\boldsymbol{\mathcal{S}et}}$, *i.e.* $T \approx \mathbb{A}$.

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 5.4.1.3 and Lemma 4.1.5.4 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 4.1.7.3 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

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

Lemma 4.1.5.9 ensures that U is faithful. We claim that the function $\mathsf{H} : \mathrm{Ob}(\boldsymbol{\mathcal{S}et}^\mathsf{F}) \leftarrow \mathrm{Ob}(\boldsymbol{\mathcal{S}et}^\mathsf{G})$ defined by

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

can be uniquely extended to a concrete functor:

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

To prove this we use Lemma 5.1.1.2, thus we have to verify that for every $\varphi, \varphi' \in \mathrm{Ob}(\boldsymbol{\mathcal{S}et}^\mathsf{G})$ 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 $(*)$:

$$\begin{aligned}
&\varphi \cdot \mathsf{G}f = f \cdot \varphi' \\
\iff &\quad \{ \text{ fusion and reflection in Table 4.5 and definition of } \mathsf{G} \ \} \\
&[\varphi \cdot \iota_\delta]_{\delta\in\Delta} \cdot \coprod_{\delta\in\Delta} (\prod^{\mathrm{rank}_\Delta\, \delta} f) = f \cdot [\varphi' \cdot \iota_\delta]_{\delta\in\Delta} \\
\iff &\quad \{ \text{ fusion (i) in Table 4.7 } \} \\
&[\varphi \cdot \iota_\delta \cdot \prod^{\mathrm{rank}_\Delta\, \delta} f]_{\delta\in\Delta} = f \cdot [\varphi' \cdot \iota_\delta]_{\delta\in\Delta} \\
\iff &\quad \{ \text{ UP and cancelation in Table 4.5 } \} \\
&\forall\, \delta \in \Delta.\ \varphi \cdot \iota_\delta \cdot \prod^{\mathrm{rank}_\Delta\, \delta} f = f \cdot \varphi' \cdot \iota_\delta \\
\iff &\quad \{ \text{ Definition 2.2.2.2 } \} \\
&f : (|\varphi|_{\mathsf{G}}; (\varphi \cdot \iota_\delta)_{\delta\in\Delta}) \leftarrow (|\varphi'|_{\mathsf{G}}; (\varphi' \cdot \iota_\delta)_{\delta\in\Delta}) \text{ is a } \Delta\text{-algebra homomorphism.}
\end{aligned}$$

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

$$\begin{aligned}
&\mathsf{H}\varphi \cdot \mathsf{F}(\mathsf{U}f) \\
=&\left[\left\langle [\varphi \cdot \iota_\delta/\delta]_{\delta\in\Delta}(\mathrm{rhs}'_{R,\sigma}\, q)\right\rangle_{q\in Q}\right]_{\sigma\in\Sigma} \cdot \coprod_{\sigma\in\Sigma} (\prod^{\mathrm{rank}_\Sigma\, \sigma} (\mathsf{U}f)) \\
=&\left[\left\langle [\varphi \cdot \iota_\delta/\delta]_{\delta\in\Delta}(\mathrm{rhs}'_{R,\sigma}\, q) \cdot \prod^{\mathrm{rank}_\Sigma\, \sigma} (\mathsf{U}f)\right\rangle_{q\in Q}\right]_{\sigma\in\Sigma} \\
=&\quad \{ \text{ Lemma 5.4.2.8 with } \prod^{\mathrm{rank}_\Sigma\, \sigma\cdot \#Q} = \prod^{\mathrm{rank}_\Sigma\, \sigma} \cdot \mathsf{U} \text{ and precondition of } (*) \ \} \\
&\left[\left\langle f \cdot [\varphi' \cdot \iota_\delta/\delta]_{\delta\in\Delta}(\mathrm{rhs}'_{R,\sigma}\, q)\right\rangle_{q\in Q}\right]_{\sigma\in\Sigma} \\
=&\quad \left\{ \begin{array}{l} \text{fusion (i) for product functors (Table 4.6), definition of } \mathsf{U}, \\ \text{and fusion for coproducts (Table 4.5)} \end{array} \right\} \\
&\mathsf{U}f \cdot \left[\left\langle [\varphi' \cdot \iota_\delta/\delta]_{\delta\in\Delta}(\mathrm{rhs}'_{R,\sigma}\, q)\right\rangle_{q\in Q}\right]_{\sigma\in\Sigma} \\
=&\mathsf{U}f \cdot \mathsf{H}\varphi'.
\end{aligned}$$

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 $\mathbb{A} = (\mathsf{H}, \mathsf{U}, \pi_{q_0})$ is an algebraic transducer over $\boldsymbol{\mathcal{S}et}$ with $T \approx \mathbb{A}$. ∎

5.4.2.10 Definition. We use the construction from the preceding Lemma 5.4.2.9 to define a function:

$$\mathsf{R} : \mathrm{Mor}\ td\text{-}\,\mathbf{AT}_{\boldsymbol{\mathcal{S}et}} \leftarrow tdtt.$$

Notice that the functors F, G, and U from Lemma 5.4.2.9 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- $\mathbf{AT}_{\mathcal{S}et}$ modulo algebraic transducer isomorphisms, because of Corollary 5.3.4.6, Lemma 5.3.5.3, and Theorem 5.3.4.4.◇

5.4.2.11 Corollary. From Lemma 5.4.2.9 and Theorem 5.4.2.7 follows:

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

(ii) $\underbrace{[\![\,.\,]\!]}_{\text{on } tdtt} = \underbrace{[\![\,.\,]\!]}_{\text{on } td\text{-}\,\mathbf{AT}} \cdot \mathsf{R}.$ ◇

5.4.2.12 Corollary. With Lemma 5.4.2.9 and Theorem 5.4.2.7 we get:

$$TOP \subseteq TOP_{\mathbf{AT}}.$$ ◇

5.4.2.13 Note. We have not yet proven the other direction, and thus equality, in Corollary 5.4.2.12, because there may be top-down algebraic transducers, which are not in the image of R. Perhaps we would need additional preconditions on the concrete functors of the algebraic transducers, to force R to be surjective. ◇

5.4.3 Relating syntactic composition and fusion

5.4.3.1 Example (composition of top-down algebraic transducers). Consider the two top-down tree transducers T_{zigzag} and T_{bin} from the Example 3.1.1.2 and Example 3.3.1.3. For both we may construct a related algebraic transducer over $\mathcal{S}et$ according to Lemma 5.4.2.9, *i.e.* $T_{\text{zigzag}} \approx \mathbb{A}_{\text{zigzag}}$ and $T_{\text{bin}} \approx \mathbb{A}_{\text{bin}}$. In Example 5.4.2.1 we have already seen $\mathbb{A}_{\text{zigzag}} = (\mathsf{H}, \mathsf{Id} \times \mathsf{Id}, \pi_1)$ 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 $\mathbb{A}_{\text{bin}} = (\mathsf{H}', \mathsf{Id}, id)$ 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 3.3.1.3):

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

In the same way we could construct the composition:

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

The following theorem gives an answers a question which motivated this thesis: What is the relation between short-cut fusion and composition of tree transducers? For top-down tree transducers both fusion techniques yield the same result (modulo relation):

5.4.3.2 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

$$R = \big\{(q,p)(\sigma(x_1, \ldots)) \to [\![T_2']\!]_q(\mathrm{rhs}_{R_1,\sigma}\, p) \;\big|\; q \in Q \;\wedge\; p \in P \;\wedge\; \sigma \in \Sigma\big\}$$

where T_2' is constructed from T_2 as in the proof of Theorem 3.3.1.2. Let $r = \max(\mathrm{rank}_\Sigma\, \sigma)$. Without loss of generality we can assume that Δ and Γ are disjoint. We define the top-down algebraic tree transducers

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

by $\mathbb{A}_1 = \mathsf{R}\,T_1$, $\mathbb{A}_2 = \mathsf{R}\,T_2$, and $\mathbb{A} = \mathsf{R}(T_2 \cdot T_1)$ where we use the construction of Lemma 5.4.2.9 and thus in particular:

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

We have to show that $\mathbb{A}_2 \cdot \mathbb{A}_1 = \mathbb{A}$ holds. From Definition 5.3.1.1 we obtain $\mathbb{A}_2 \cdot \mathbb{A}_1 = (\mathsf{H}_1 \cdot \mathsf{H}_2, \mathsf{U}_1 \cdot \mathsf{U}_2, \pi_1 * \pi_2)$. 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 5.3.5.2).

The essential statement we have to show is $\mathsf{H}_1 \cdot \mathsf{H}_2 = \mathsf{H}$: From the definition of $[\![\,\boldsymbol{.}\,]\!]$ in Definition 3.2.2.1 and with Lemma 5.4.2.4 we obtain:

$$\forall\, \delta \in \Delta.\ \langle [\![T_2']\!]_q\rangle_{q\in Q} \cdot \delta = \langle \mathrm{rhs}'_{R_2,\sigma}\, q\rangle_{q\in Q} \cdot \prod^{\mathrm{rank}_\Sigma\,\sigma} \langle [\![T_2']\!]_q\rangle_{q\in Q}$$

and $\forall\, px \in PX_r.\ \langle [\![T_2']\!]_q\rangle_{q\in Q}(px) = \big((q,p)x\big)_{q\in Q}$, *i.e.* $\langle [\![T_2']\!]_q\rangle_{q\in Q}$ is a unique Δ-algebra homomorphism on the Δ-algebra $T_\Delta(PX_r)$ which is free on PX_r. With

Definition 2.2.2.6 (ii) we get

$$
\begin{array}{ll}
& \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 [\![T'_2]\!]_q\rangle(\mathrm{rhs}_{R_1,\sigma}\, p) \\
\Longrightarrow & \quad \{ \text{ definition of } R \text{ } \} \\
& \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 5.4.2.4 } \} \\
& \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{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 \text{ } \} \\
& \forall\, \varphi \in \mathrm{Ob}(\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 4.5) } \} \\
& \forall\, \varphi \in \mathrm{Ob}(\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 4.5) } \} \\
& \forall\, \varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{S}et}^{\mathsf{F}_3}).\, \mathsf{H}_1(\mathsf{H}_2\varphi) = \mathsf{H}\varphi \\
\Longrightarrow & \quad \{ \text{ Lemma 4.2.1.5 } \} \\
& \mathsf{H}_1\cdot\mathsf{H}_2 = \mathsf{H}.
\end{array}
$$

■

5.4.3.3 Lemma. Let T_{id} be the top-down tree transducer from Lemma 3.3.2.1. 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 5.4.2.9 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 4.2.1.5 that $\mathsf{H} = \mathsf{Id}$. ■

5.4.3.4 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 3.2.2.5.

Proof. The direction $\Leftarrow$ is obvious, because the construction from Lemma 5.4.2.9 which is the definition of R (Definition 5.4.2.10) depends only on the number of states rather

than on the set of states. Let $\mathbb{A}_1 = (\mathsf{H}_1, \mathsf{U}_1, \pi_{p_0}) = \mathsf{R}\,T_1$ and $\mathbb{A}_2 = (\mathsf{H}_2, \mathsf{U}_2, \pi_{q_0}) = \mathsf{R}\,T_2$ where $\mathbb{A}_1, \mathbb{A}_2 : \mathsf{G} \leftarrow \mathsf{F}$. 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 5.4.2.9 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

$$
\begin{array}{ll}
& \mathsf{R}\,T_1 = \mathsf{R}\,T_2 \\
\Longrightarrow & \quad \{ \text{ see above } \} \\
& \mathsf{H}_1 = \mathsf{H}_2 \\
\Longrightarrow & \mathsf{H}_1\, in_{\mathsf{G}} = \mathsf{H}_2\, in_{\mathsf{G}} \\
\Longrightarrow & \quad \{ \text{ Corollary 5.4.2.11: } T_1 \approx \mathbb{A}_1 \text{ and } T_2 \approx \mathbb{A}_2 \text{ and Definition 5.4.2.5 } \} \\
& \forall\, \sigma \in \Sigma.\ \langle \mathrm{rhs}'_{R_1,\sigma}\, p \rangle_{p \in P} = \langle \mathrm{rhs}'_{R_2,\sigma}\, q \rangle_{q \in Q} \\
\Longrightarrow & \quad \{ \text{ choose the appropriate bijection } h : P \leftarrow Q \ \} \\
& \forall\, q \in Q.\ \forall\, \sigma \in \Sigma.\ \mathrm{rhs}'_{R_1,\sigma}(hq) = \mathrm{rhs}'_{R_2,\sigma}\, q \\
\Longrightarrow & \forall\, q \in Q.\ \forall\, \sigma \in \Sigma.\ \mathrm{rhs}'_{R_1,\sigma}(hq) \cdot \prod^{\mathrm{rank}_\Sigma\, \sigma} (\langle px \rangle_{p \in P}) = \mathrm{rhs}'_{R_2,\sigma}\, q \cdot \prod^{\mathrm{rank}_\Sigma\, \sigma} (\langle px \rangle_{p \in P}) \\
\Longrightarrow & \forall\, q \in Q.\ \forall\, \sigma \in \Sigma.\ \mathrm{rhs}'_{R_1,\sigma}(hq) \cdot \prod^{\mathrm{rank}_\Sigma\, \sigma} (\langle px \rangle_{p \in P}) = \mathrm{rhs}'_{R_2,\sigma}\, q \cdot \prod^{\mathrm{rank}_\Sigma\, \sigma} (\langle hq\, x \rangle_{q \in Q}) \\
\Longrightarrow & \quad \{ \text{ Lemma 5.4.2.4 } \} \\
& \forall\, q \in Q.\ \forall\, \sigma \in \Sigma.\ \mathrm{rhs}_{R_1,\sigma}(hq) = [hq\, x / qx]^{q \in Q}_{x \in X}(\mathrm{rhs}_{R_2,\sigma}\, q)
\end{array}
$$

The latter is the property (iii) from Definition 3.2.2.5. ∎

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

(ii) The function R is an embedding functor $\mathsf{R} : td\text{-}\mathbf{AT}_{\boldsymbol{\mathcal{S}et}} \leftarrow td\text{-}\boldsymbol{\mathcal{T}t}$.

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 5.4.3.2) 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 5.4.3.4 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 5.4.3.4 we get that $\mathsf{R}\,T_1 = \mathsf{R}\,T'_1$ and $\mathsf{R}\,T_2 = \mathsf{R}\,T'_2$ and thus with Theorem 5.4.3.2: $\mathsf{R}(T_2 \cdot T_1) = \mathsf{R}\,T_2 \cdot \mathsf{R}\,T_1 = \mathsf{R}\,T'_2 \cdot \mathsf{R}\,T'_1 = \mathsf{R}(T'_2 \cdot T'_1)$. We

use Lemma 5.4.3.4 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 $td\text{-}\boldsymbol{\mathcal{T}t}$ where Mor $td\text{-}\boldsymbol{\mathcal{T}t} = tdtt/\cong$ and Ob $td\text{-}\boldsymbol{\mathcal{T}t}$ is the class of all finite ranked alphabets.

(ii) We already know from Theorem 5.4.3.2 that R is multiplicative, from Lemma 5.4.3.3 that R preserves identities, and from Lemma 5.4.3.4 that R does not depend on the representative of the isomorphism class. Thus together with (i), we obtain that R is a functor $\mathsf{R} : td\text{-}\mathbf{AT}_{\boldsymbol{\mathcal{S}et}} \leftarrow td\text{-}\boldsymbol{\mathcal{T}t}$. It is obvious from Lemma 5.4.3.4 that R is also an embedding. ∎

5.4.3.6 Note. We try to visualize the relation between top-down tree transducers and top-down algebraic transducers over $\boldsymbol{\mathcal{S}et}$ in a diagram in Figure 5.1.

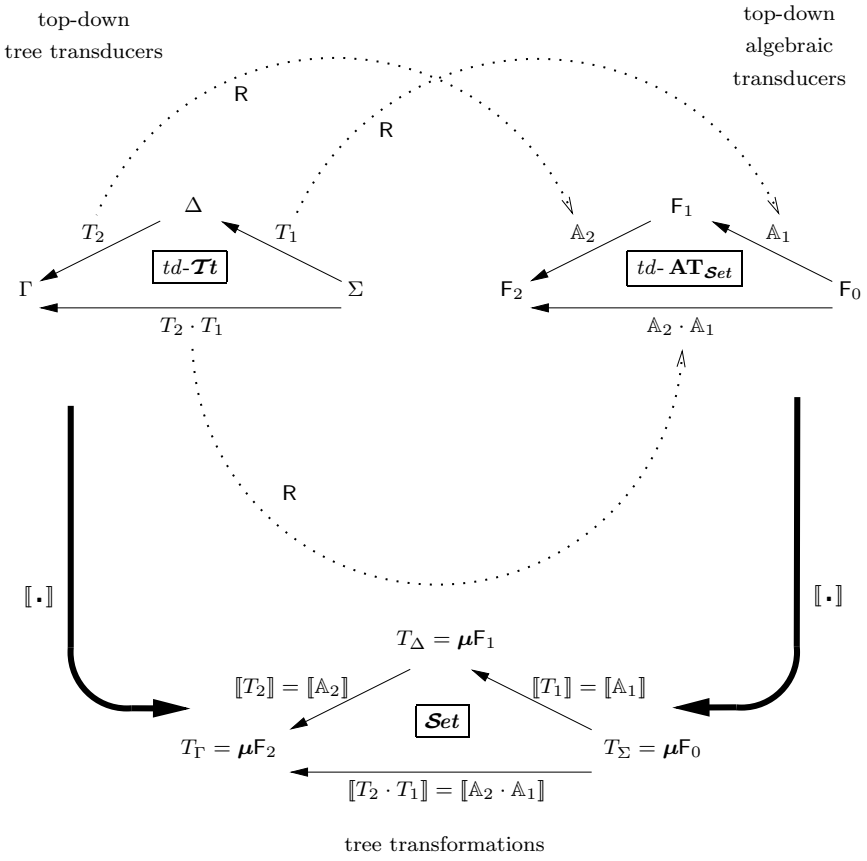


Figure 5.1: Relation between $td\text{-}\boldsymbol{\mathcal{T}t}$ and $td\text{-}\mathbf{AT}_{\boldsymbol{\mathcal{S}et}}$

6 The free monad approach

The generalization from monoids to monads corresponds to the generalization from character strings to trees on an abstract level.

As the theories of character string automata and generalized sequential machines use monoids, it seems thus to be natural to use monads in the theory of tree transducers. This idea has been developed in [Jür02] and has been presented in [Jür03].

Let us motivate the denotational semantics of tree transducers, that we will define using a free monad in Subsection 6.3.1. We define a denotational semantics for deterministic finite state automata (FSA) using a free monoid in three steps:

Consider an FSA $M = (Q, \Sigma, r, q_0, F)$ where Q is the set of states, Σ is the input alphabet, $r : Q \leftarrow Q \times \Sigma$ is the transition function[1], $q_0 \in Q$ is the initial state, and $F \subseteq Q$ the set of final states.

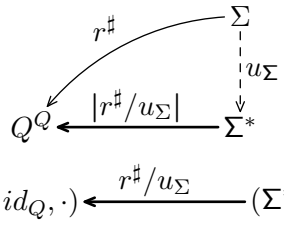
Step 1: The transition function

$$r : Q \leftarrow Q \times \Sigma$$

can equivalently be described by

$$r^\sharp : Q^Q \leftarrow \Sigma.$$

Step 2: The set Q^Q is the carrier of the monoid $(Q^Q, id_Q, \cdot)$. The character string monoid $(\Sigma^*, \varepsilon, \cdot)$ is a *free* monoid over Σ with universal arrow $u_\Sigma : \Sigma^* \leftarrow \Sigma : a \leftarrow\!\shortmid a$. Then there exists a unique monoid morphism $r^\sharp/u_\Sigma : (Q^Q, id_Q, \cdot) \leftarrow (\Sigma^*, \varepsilon, \cdot)$ such that the following diagram commutes:



This *unique* monoid morphism can be used as a denotational semantics $(\!|M|\!) = r^\sharp/u_\Sigma$ of the FSA M. We call $(\!|M|\!)$ the *generalized semantics*, because it is independent from the initial state q_0. Notice, that a compositional function is a monoid

[1] Without loss of generality we may assume that r is a function, rather than just a partial function. Otherwise we could introduce one additional state $Q' = Q \uplus \{\bot\}$, and define $r' : Q' \leftarrow Q' \times \Sigma$ for every $q \in Q'$ and $a \in \Sigma$ by $r'(q, a) = r(q, a)$ if $q \neq \bot$ and $r(q, a)$ is defined, and $r'(q, a) = \bot$ otherwise.

homomorphism from a free monoid. Since the above generalized semantics is defined to be the *unique* monoid homomorphism extending $r^\sharp$, it is obvious, that any compositional semantics of the automaton M will be equal to $\langle\!|M|\!\rangle$.

Step 3: We define the *semantics* $[\![M]\!] : Q \leftarrow \Sigma^*$ of M by $[\![M]\!]w = |\langle\!|M|\!\rangle|\, w\, q_0$, *i.e.*

$$[\![M]\!] = \omega \cdot |\langle\!|M|\!\rangle|$$

where $\omega : Q \leftarrow Q^Q : f\, q_0 \mapsfrom f$ (called *observation* function) and $|\mathbf{.}| : \boldsymbol{Set} \leftarrow \mathbf{Mon} : A \mapsfrom (A, 1, \cdot)$ is the canonical *forgetful functor* on the category of monoids **Mon**.

The language recognized by M can now be described as the F-pre-image of this semantics: $\mathcal{L}M = [\![M]\!]^{-1}[F] = \{w \in \Sigma^* \mid [\![M]\!]w \in F\}$.

Before we continue let us summarize the three steps from above: **Step 1:** We used the *adjunction* $(Q \times \mathbf{.}) \dashv (\mathbf{.})^Q : \boldsymbol{Set} \leftarrow \boldsymbol{Set}$ to transform the transition function r into $r^\sharp$. **Step 2:** We used the *free* character string monoid over Σ to iterate the transformed transition $r^\sharp$ deriving $\langle\!|M|\!\rangle$. **Step 3:** We used the *observation* function ω (encoding the initial state q_0) and the *forgetful functor* $|\mathbf{.}|$ to get the *semantics* $[\![M]\!]$.

We will see these three steps again in Subsection 6.3.1 on the level of terms and monads.

For the following we will need a more abstract view on tree transducers and/or functional programs:

6.1 Tree transducers as functional programs

6.1.1 Terms, types, and functors

In the following we want to use the language of category theory to describe terms and types. We have already seen how to relate a ranked alphabet to a functor in Note 5.4.2.3. In the last chapter this relation was rather ad hoc. In this chapter we will use a formal definition:

6.1.1.1 Definition (ranked alphabets induce endofunctors on $\boldsymbol{Set}$). (i) Let $\mathbf{\Sigma} = (\Sigma, \mathrm{rank}_{\mathbf{\Sigma}})$ be a ranked alphabet. The function $\mathbf{\Sigma}'$ defined by

$$\forall\, f \in \boldsymbol{Set}(A, B).\ \forall\, r \in \mathbb{N}_0.\ \forall\, \sigma \in \Sigma^{(r)}.\ \forall\, b_1, \ldots, b_r.\ \mathbf{\Sigma}' f(\sigma b_1 \cdots b_r) = \sigma(f b_1) \cdots (f b_r)$$

is an endofunctor $\mathbf{\Sigma}' : \boldsymbol{Set} \leftarrow \boldsymbol{Set}$ which is defined on objects by $\forall A \in \mathrm{Ob}\,\boldsymbol{Set}.\ \mathbf{\Sigma}' A = \{\sigma a_1 \cdots a_{\mathrm{rank}_{\mathbf{\Sigma}}} \mid \sigma \in \Sigma \wedge a_i \in A\}$.

(ii) Obviously $\mathbf{\Sigma}'\emptyset = \Sigma^{(0)}$ and $\mathbf{\Sigma}'\{\emptyset\} \cong \Sigma$ and we will assume, without loss of generality, that $\mathbf{\Sigma}'\{\emptyset\} = \Sigma$.

(iii) To simplify our notation we will use the same symbol $\mathbf{\Sigma}$ to denote a ranked alphabet $\mathbf{\Sigma} = (\Sigma, \mathrm{rank}_{\mathbf{\Sigma}})$, its underlying set of symbols Σ, and the functor $\mathbf{\Sigma}'$ induced by $\mathbf{\Sigma}$. ◇

6.1.1.2 Definition (algebraic data type, regular type). An algebraic data type in Haskell [PH99] is declared by the expression

$$\textbf{data}\, T\, \alpha_1 \cdots \alpha_n \;=\; C_1\, \vec{\tau}_1 \mid \cdots \mid C_m\, \vec{\tau}_m$$

where $n \in \mathbb{N}_0$, $\alpha_1, \ldots, \alpha_n$ are type variables, $m \in \mathbb{N}$, and $\vec{\tau}_1, \ldots, \vec{\tau}_m$ are types with a syntax given by the following grammar

$\vec{\tau}$	::=	$\{\tau\}$		
τ	::=	α		type variable
	$\mid$	$\tau \to \tau$		function type
	$\mid$	$T\{\tau\}$		recursive call
	$\mid$	$T'\{\tau\}$	(where $T' \neq T$)	algebraic data type

where $\{\,\centerdot\,\}$ denotes 'zero or more' occurrences.[2] The symbol T is called a **type constructor** and $C_1, \ldots, C_m$ are called **data constructors**. For any types $\beta_1, \ldots, \beta_n$ the above defines a type $T\, \beta_1 \cdots \beta_n$.[3]

An algebraic data type is called **regular type** (or **uniform type**) if the recursive call occurs only *positively*. This can be described by the following extension to the above grammar, where we introduce the new nonterminals τ^+ and τ^- for *positive* and *negative* type expressions, respectively:

$\vec{\tau}$	::=	$\{\tau^+\}$		
τ^+	::=	α		type variable
	$\mid$	$\tau^- \to \tau^+$		function type
	$\mid$	$T\, \alpha_1 \ldots \alpha_n$		recursive call
	$\mid$	$T'\{\tau^+\}$	(where $T' \neq T$)	regular type
τ^-	::=	α		type variable
	$\mid$	$\tau^+ \to \tau^-$		function type
	$\mid$	$T'\{\tau^-\}$	(where $T' \neq T$)	regular type

If the recursive call never occurs in the domain of a function type (*i.e.* left of $\to$), then the regular type is called a **polynomial** type. ◇

6.1.1.3 Definition (regular types as endofunctors). Consider a regular type

$$\textbf{data}\, T\, \alpha_1 \cdots \alpha_n \;=\; C_1\, \vec{\tau}_1 \mid \cdots \mid C_m\, \vec{\tau}_m$$

as defined in Definition 6.1.1.2.

Notice, that in the special case that all $\vec{\tau}_i$ have the form $\vec{\tau} ::= \{T\alpha_1 \cdots \alpha_n\}$, the values of the regular type are just Σ-trees for $\Sigma = \{C_1, \ldots, C_m\}$ where the ranks are equal to the arities. Then we have the induced $\boldsymbol{\mathcal{S}et}$-endofunctor Σ from Definition 6.1.1.1.

[2] Please distinguish the Haskell | from the | used in grammars!

[3] More precisely T is a type of kind $\underbrace{* \to \cdots \to *}_{n \text{ times}}$, where a **kind** is a 'meta type' of types. Like a value may be of some type, a type my be of some kind. A kind is either $*$ (for base types) or $\varkappa_1 \to \varkappa_2$ for some kinds $\varkappa_1$ and $\varkappa_2$. If α has kind $\varkappa_1 \to \varkappa_2$ and β has kind $\varkappa_1$ then $\alpha\beta$ is a type of kind $\varkappa_2$.

Let us extend the definition of the endofunctor Σ to regular types: The restriction to positive recursion calls is essential to guaranty that the algebraic data type can be described by a (covariant) endofunctor. Otherwise the functor may become contravariant (and thus cannot be an endofunctor) or even have some 'mixed variance'. Let $\mathcal{C}$ be a cartesian closed category that we use as semantic domain. For every free type variable α let $[\![\alpha]\!] \in \mathrm{Ob}\,\mathcal{C}$ be the semantics of α. We define the functor $\Sigma : \mathcal{C} \leftarrow \mathcal{C}$ by

$$\begin{aligned}
\Sigma = \Sigma_{C_1\vec{\tau}_1|\cdots|C_m\vec{\tau}_m} &= \Sigma_{C_1\vec{\tau}_1} + \cdots + \Sigma_{C_m\vec{\tau}_m} \\
\Sigma_{C_i\tau_{i,1}\cdots\tau_{i,k_i}} &= \Sigma_{\tau_{i,1}} \times \cdots \times \Sigma_{\tau_{i,m}} \\
\Sigma_{\alpha} &= \underline{[\![\alpha]\!]} \\
\Sigma_{T\,\alpha_1\cdots\alpha_n} &= \mathsf{Id} \\
\Sigma_{\tau\to\tau'} &= \Sigma_{\tau'} \Leftarrow \Sigma_{\tau}
\end{aligned}$$

Finally, we define the denotational semantics of $T\,\alpha_1\cdots\alpha_n$ by $[\![T\,\alpha_1\cdots\alpha_n]\!] = [\![\Sigma]\!] = \boldsymbol{\mu}\Sigma$ where we assume that Σ has an initial algebra $in_\Sigma : \boldsymbol{\mu}\Sigma \leftarrow \Sigma(\boldsymbol{\mu}\Sigma)$.

Notice, that for a ranked alphabet Σ (which we identify with an endofunctor Σ according to Definition 6.1.1.1 (iii)) the semantics $[\![\,.\,]\!]$ given by Definition 5.3.3.1 is a functor (according to Theorem 5.3.3.3) which is defined on objects by $[\![\Sigma]\!] = \boldsymbol{\mu}\Sigma$. Later we will see a semantics $[\![\,.\,]\!]$ (Definition 6.3.1.3) which will have the same property for every varietor Σ. ◇

6.1.1.4 Observation.

The endofunctors induced by polynomial types, ranked alphabets, and unary ranked alphabets are polynomial, bicartesian, and cocartesian $\boldsymbol{\mathcal{S}et}$-endofunctors, respectively. *E.g.* the functor for the list data type $\mathbf{data}\,[\alpha] = [] \mid \alpha : [\alpha]$ is the polynomial functor $\underline{1} + \underline{[\![\alpha]\!]} \times \mathsf{Id}$.

The endofunctors induced by ranked alphabets or polynomial functors with only finite exponents are $\boldsymbol{\mathcal{S}et}_{\aleph_0}$-endofunctors. ◇

6.1.2 Syntax and semantics of tree transducers

Throughout this subsection the symbols Q, Σ and Δ usually denote ranked alphabets where Q is unary, *i.e.* Σ and Δ are bicartesian $\boldsymbol{\mathcal{S}et}$-endofunctors and Q is a cocartesian $\boldsymbol{\mathcal{S}et}$-endofunctor. It is possible to generalize all definitions in this subsection, by allowing Σ and Δ to be *Seta*-endofunctors or polynomial $\boldsymbol{\mathcal{S}et}$-endofunctors. Of course this generalization enlarges the classes of functions computable by a tree transducer. However, it has no effect on the fusion results in Section 6.5.

As pointed out in Subsection 3.2.1, an important property of the operational semantics of a tree transducer is that it is *compositional*. A function f on terms is called **compositional** (or **syntax directed** [FV98]) if the value of f applied to a term t only depends on the values of f applied to subterms of t.

We will define a denotational semantics in Definition 6.3.1.2 in such a way, that it will be the unique function which satisfies certain properties. One of these properties is to be *compositional* as explained in Definition 6.3.1.2 (ii). Thus the only thing we need to know about the operational semantics in this chapter is the fact that it is compositional. And thus we will not need further details about the operational semantics of tree transducers here.

6.1.2.1 Definition ('left hand side' term). Let Σ be a ranked alphabet and $X = \{x_1, x_2, x_3, \ldots\}$ be a countably infinite set of variables. The set $\mathsf{T}_\Sigma^{lhs} X$ of all **'left hand side' terms** is the smallest subset of $\mathsf{T}_\Sigma X$ such that $'x_1 \in \mathsf{T}_\Sigma^{lhs} X$ and

$$\frac{\begin{array}{ccccc} r \in \mathbb{N}_0 & \sigma \in \Sigma^{(r)} & t_1, \ldots, t_r \in \mathsf{T}_\Sigma^{lhs} X & k_1, \ldots, k_r \in \mathbb{N} & k_1 = 1 \\ \forall\, i < j.\ k_i < k_j & \forall\, i < r.\ var_X\, t_i = \{x_{k_i}, \ldots, x_{k_{i+1}-1}\} & & var_X\, t_r = \{x_{k_r}, \ldots\} & \end{array}}{\sigma t_1 \cdots t_r \in \mathsf{T}_\Sigma^{lhs} X}.$$

Obviously $\mathsf{T}_\Sigma^{lhs} X \subseteq \mathsf{T}_\Sigma^{lin} X$. Notice that $\mathsf{T}_\Sigma^{lhs} X$ depends on the order in which we enumerated the elements of X. The essential property of $\mathsf{T}_\Sigma^{lhs} X$ is, that for *every term* $t \in \mathsf{T}_\Sigma X$ there is *precisely one* term $t' \in \mathsf{T}_\Sigma^{lhs} X$ such that t' is *linear* and t' is equal to t up to renaming of variables. ◇

The following is essentialy equivalent to Definition 3.1.1.1 but now we use some functors to describe the syntactic restrictions of the left and right hand side terms. This will make it easier to handle the tree transducers in our category theroy framework.

6.1.2.2 Definition ((pure) top-down tree transducer [Rou68, Rou70, Tha70]**).** A functional program (*e.g.* a Haskell program [PH99]) of the form

$$\begin{array}{l} \lambda\, x \to \mathbf{let} \\ \qquad\quad lhs_1 = rhs_1 \\ \qquad\qquad \vdots \\ \qquad\quad lhs_m = rhs_m \\ \qquad \mathbf{in}\, f\ x \end{array}$$

is called a **top-down tree transducer** with **initial state** f if there exists a set of variables X and ranked alphabets $\mathsf{Q}, \Sigma, \Delta$ where Q is unary and finite ($\ell = \#\mathsf{Q} \in \mathbb{N}$), such that for all $i \in \{1, \ldots, m\}$:

(i) $lhs_i \in \mathsf{Q}(\Sigma\, X) \cap \mathsf{T}_{\mathsf{Q}+\Sigma}^{lhs} X$,

(ii) $rhs_i \in \mathsf{T}_\Delta(\mathsf{Q}X)$,

(iii) $var_X\, rhs_i \subseteq var_X\, lhs_i$, and

(iv) $f\ x \in \mathsf{Q}\{x\}$.

The **let** ... **in** construction is only used to make the initial state f explicit. The λ-abstraction $\lambda\, x \to$ is only used to make explicit that f is a unary function (and may be dropped using η-conversion). In fact the body of the program, *i.e.* the equation system

$$\begin{aligned} lhs_1 &= rhs_1 \\ &\vdots \\ lhs_m &= rhs_m \end{aligned}$$

can be viewed as the relation of a top-down tree transducer in the sense of Definition 3.1.1.1 where we have writen $\rightarrow$ instead of $=$.

Notice, that it is part of the definition of a functional program, that every left hand side of the above equation system has a unique right hand side.

Notice, that in the case of a top-down tree transducer the operational semantics of the according functional program, and the operational semantics as described in Subsection 3.2.1, as well as the denotational semantics given in Subsection 3.2.2 coincide.

We denote the class of all functions which may be computed by an ℓ-state top-down tree transducer by TOP_ℓ and according to Definition 3.2.2.3 we have $TOP = \bigcup_{\ell \in \mathbb{N}} TOP_\ell$.

A top-down tree transducer is called **pure** if $\ell = 1$, *i.e.* the top-down tree transducer has only one state. A pure top-down tree transducer is also called a **homomorphism tree transducer**. The class of all functions computable by a homomorphism tree transducer is also denoted by $HOM = TOP_1$. Obviously $HOM \subseteq TOP$. ◇

6.1.2.3 Example (top-down tree transducer). Consider the following two regular types:

$$\begin{aligned} &\textbf{data}\ Nat\ = Zero \mid Succ\ Nat \\ &\textbf{data}\ Bool = False \mid True \end{aligned}$$

The program

$$\begin{aligned} &\textbf{let} \\ &\quad even\ Zero\ \ \ \ = True \\ &\quad even(Succ\ n) = odd\ \ n \\ &\quad odd\ \ Zero\ \ \ \ = False \\ &\quad odd\ \ (Succ\ n) = even\ n \\ &\textbf{in}\ even \end{aligned}$$

is a top-down tree transducer with two states and initial state *even*. Moreover the program

$$\begin{aligned} even\ Zero\ \ \ \ &= True \\ even(Succ\ n) &= odd\ \ n \\ odd\ \ Zero\ \ \ \ &= False \\ odd\ \ (Succ\ n) &= even\ n \end{aligned}$$

can be viewed as a top-down tree transducer with initial state *even* or a top-down tree transducer with initial state *odd*. ◇

Before we can define macro tree transducers we need one more definition in order to express *applicative terms* [Dam82], *i.e.* terms, where some subterms are treated as functions which can be applied to other terms.

6.1.2.4 Definition (application functor). Let $\mathcal{C}$ be a cartesian closed category and $I \in \operatorname{Ob}\mathcal{C}$. We define the functor $\mathsf{A}_I : \mathbf{End}\,\mathcal{C} \leftarrow \mathcal{C}$ by

$$\forall\, f, g \in \operatorname{Mor}\mathcal{C}.\ \mathsf{A}_I f\, g = (g \Leftarrow id_I) \times f.$$

Obviously A_I is polynomial.

Notice, that since $\Leftarrow$ and $\times$ are only defined up to isomorphism, so is A_I. For every pair of objects $X, Y \in \operatorname{Ob}\mathcal{C}$ we have $\mathsf{A}_I X\, Y = Y^I \times X$.

For finite sets $I = \{1, \ldots, k\}$ we can easily define A_I on cartesian categories (*e.g.* on $\boldsymbol{\mathcal{S}et}_{\aleph_0}$). For $\mathcal{C} = \boldsymbol{\mathcal{S}et}$ or $\mathcal{C} = \boldsymbol{\mathcal{S}et}_{\aleph_0}$ and $k \in \mathbb{N}_0$ we use $\mathsf{A}_k X\, Y = \left\{ x\, y_1 \cdots y_k \;\middle|\; x \in X \ \wedge\ y_i \in Y \right\} \cong Y^k \times X \cong \mathsf{A}_{\{1,\ldots,k\}} X\, Y$. ◇

6.1.2.5 Definition ((pure) ((simple) basic) macro tree transducer). A functional program of the form

$$\begin{array}{l} \lambda\, x \rightarrow \mathbf{let} \\ \qquad\quad lhs_1 \;=\; rhs_1 \\ \qquad\qquad \vdots \\ \qquad\quad lhs_m \;=\; rhs_m \\ \qquad \mathbf{in}\, f\ x\ e_1\ \cdots\ e_k \end{array}$$

is called a **macro tree transducer** with **initial state** f and **environment** $(e_1, \ldots,\ e_k)$ if there exist sets X, Y and ranked alphabets $\mathsf{Q}, \Sigma, \Delta$ where Q is unary and finite ($\ell = \#\mathsf{Q} \in \mathbb{N}$), and a finite index set $I = \{1, \ldots, k\}$, such that for all $i \in \{1, \ldots, m\}$:

(i) $lhs_i \in \mathsf{A}_I\big(\mathsf{Q}(\Sigma X)\big)\, Y \ \cap\ \mathsf{T}^{lhs}_{\mathsf{F}+\Sigma}(X + Y)$, where $\mathsf{F}X = \mathsf{A}_I(\mathsf{Q}X)X$

(ii) $rhs_i \in \mathsf{T}_{\Delta+\mathsf{A}_I(\mathsf{Q}X)} Y$,

(iii) $var_{X \uplus Y}\, rhs_i \subseteq var_{X \uplus Y}\, lhs_i$, and

(iv) $f\ x\ e_1 \cdots e_k \in \mathsf{A}_I(\mathsf{Q}\{x\})\,\{e_1, \ldots, e_k\}$.

The **let** ... **in** construction is only used to make explicit the initial state f and the environment $(e_1, \ldots,\ e_k)$.

We denote the class of all functions which may be computed by an ℓ-state macro tree transducer by MAC_ℓ and set $MAC = \bigcup_{\ell \in \mathbb{N}} MAC_\ell$. A macro tree transducer is called **pure** if $\ell = 1$, *i.e.* the macro tree transducer has only one state. If $\{rhs_1, \ldots, rhs_m\} \subseteq \mathsf{T}_\Delta\big(Y + \mathsf{A}_I(\mathsf{Q}X)(\mathsf{T}_\Delta Y)\big)$ it is called **basic** and if $\{rhs_1, \ldots, rhs_m\} \subseteq \mathsf{T}_\Delta\big(Y + \mathsf{A}_I(\mathsf{Q}X)Y\big)$ it is called **simple basic**. In other words basic macro tree transducers have no nested states in any right hand side, and simple basic macro tree transducers are basic macro tree transducers where constructors may not occur inside a context parameter in any right hand side.

The classes of all functions computable by (simple) basic macro tree transducers are denoted by *b-MAC* and *sb-MAC*, respectively. The classical definition of a macro tree transducer allows different states to have different numbers of context parameters, whereas we use a fixed number for all the states. This can be done without loss of

generality because we need not to use all the context parameters in the right hand sides. It is easy to see that $TOP \subseteq sb\text{-}MAC \subseteq b\text{-}MAC \subseteq MAC$. ◇

6.1.2.6 Example (macro tree transducer). Consider the list data type:

$$\textbf{data}\,[\alpha] \;=\; [] \mid \alpha : [\alpha]$$

The program

$$\begin{aligned} \mathit{reverse}\; x \;=\; & \textbf{let} \\ & \quad \mathit{rev}\; [] \qquad\; \mathit{ys} = \mathit{ys} \\ & \quad \mathit{rev}\; (x : \mathit{xs})\mathit{ys} = \mathit{rev}\; \mathit{xs}\; (x : \mathit{ys}) \\ & \textbf{in}\; \mathit{rev}\; x\; [] \end{aligned}$$

is a pure basic macro tree transducer with one context parameter ys. It is not simple.

Notice that this is not a tree transducer in the classical sense, since the functor $\Sigma = \Delta$ for the input and output data type (see Observation 6.1.1.4) is polynomial rather than just bicartesian (compare Example 3.1.2.1). As we will see later, the monadic transducer (Definition 6.3.1.1) makes it possible to describe this kind of polymorphic tree transducers without any additional effort. ◇

6.1.3 The rule of a tree transducer

The body of a tree transducer is a system of defining equations

$$\begin{aligned} \mathit{lhs}_1 \;&=\; \mathit{rhs}_1 \\ &\;\;\vdots \\ \mathit{lhs}_m \;&=\; \mathit{rhs}_m \end{aligned}$$

Since every left hand side has a unique right hand side, we can describe these equations by the function $\varrho_X : \mathit{lhs}_i \mapsto \mathit{rhs}_i$ where X is the set of variables occurring in the equations. We call this function *the* **rule** of the tree transducer. The type of this function depends on the syntactic class of the tree transducer. We give an overview over the syntactic classes of tree transducers that we have defined (and some more) and the types of their rules in Table 6.1 where Σ, Δ are polynomial $\boldsymbol{\mathcal{S}et}$-endofunctors; Q is a cocartesian $\boldsymbol{\mathcal{S}et}$-endofunctor; X, Y are sets; $k \in \mathbb{N}$; A is a complete semiring; and $\mathbb{B}$ is the boolean semiring. The symbol $\langle\!\langle \,\boldsymbol{.}\, \rangle\!\rangle$ will be explained later in Definition 7.1.3.3.

Moreover, every tree transducer rule ϱ_X is natural in X:

6.1.3.1 Proposition (tree transducer rules are natural transformations). Every tree transducer rule can be uniquely extended to a natural transformation, and vice versa every natural transformation of the appropriate type can be restricted to a tree transducer rule.

syntactic class	class of tree transformations	type of rules
homomorphism	HOM	$\mathsf{T}_\Delta X \leftarrow X \leftarrow \Sigma X$
top-down	TOP	$\mathsf{T}_\Delta(\mathsf{Q}X) \leftarrow \mathsf{Q}(\Sigma X)$
simple basic macro	$sb\text{-}MAC$	$\mathsf{T}_\Delta\big(Y + \mathsf{A}_k(\mathsf{Q}X)\ Y\big) \leftarrow \mathsf{A}_k\big(\mathsf{Q}(\Sigma X)\big)Y$
basic macro	$b\text{-}MAC$	$\mathsf{T}_\Delta\big(Y + \mathsf{A}_k(\mathsf{Q}X)(\mathsf{T}_\Delta Y)\big) \leftarrow \mathsf{A}_k\big(\mathsf{Q}(\Sigma X)\big)Y$
macro	MAC	$\mathsf{T}_{\Delta+\mathsf{A}_k(\mathsf{Q}X)}Y \leftarrow \mathsf{A}_k\big(\mathsf{Q}(\Sigma X)\big)Y$
top-down tree-series	TOP_A	$A\langle\!\langle \mathsf{T}_\Delta(\mathsf{Q}X)\rangle\!\rangle \leftarrow \mathsf{Q}(\Sigma X)$
nondeterm. top-down	$TOP_{\mathbb{B}}$	$\mathbb{B}\langle\!\langle \mathsf{T}_\Delta(\mathsf{Q}X)\rangle\!\rangle \leftarrow \mathsf{Q}(\Sigma X)$
bottom-up	BOT	$\mathsf{Q}(\mathsf{T}_\Delta X) \leftarrow \Sigma(\mathsf{Q}X)$

Table 6.1: Some classes of tree transducers

Proof. We begin with a precise formalization of the statement:

Let $\Sigma = (\Sigma, \mathrm{rank}_\Sigma)$ and Δ be ranked alphabets and $X = \{x_1, x_2, x_3, \ldots\}$ a countably infinite set. As described in Definition 6.1.1.1 we can view these as $\boldsymbol{\mathcal{S}et}$-endofunctors.

We formalize and prove the statement for homomorphism tree transducers only. It can be generalized mechanically to other subsets of $\mathsf{T}_\Sigma^{lhs} X$.

We define the sets $L = \big\{\sigma\text{'}x_1 \cdots\text{'}x_{\mathrm{rank}_\Sigma \sigma} \mid \sigma \in \Sigma\big\} = \Sigma X \cap \mathsf{T}_\Sigma^{lhs} X$ of left hand sides and $R = \big\{r : \mathsf{T}_\Delta X \leftarrow L \mid \forall \ell \in L.\ var_X(r\ell) \subseteq var_X \ell\big\}$ of tree transducer rules. Finally, we define the function Ψ by

$$\begin{array}{rccc} \Psi: & R & \leftarrow & \mathbf{End}\,\boldsymbol{\mathcal{S}et}(\mathsf{T}_\Delta, \Sigma) \\ & \varrho_X|_L & \mapsto & \varrho \end{array}$$

where $\mathbf{End}\,\boldsymbol{\mathcal{S}et}(\mathsf{T}_\Delta, \Sigma)$ denotes the set of all natural transformations to T_Δ from Σ and $\varrho_X|_L$ denotes the restriction of the function ϱ_X on the set L.

The statement may now be given as follows: The function Ψ is a bijection.

Now we are going to prove that statement: We will show that the inverse of Ψ is the function $\hat{\cdot}$ given for every $r \in R$, for every set Y, every $k \in \mathbb{N}_0$, $\sigma \in \Sigma^{(k)}$, and $y_1, \ldots, y_k \in Y$ by

$$(\hat{r})_Y(\sigma\text{'}y_1 \cdots\text{'}y_k) = \mathsf{T}_\Delta f\big(r(\sigma\text{'}x_1 \cdots\text{'}x_k)\big) \quad \text{where} \quad f : Y \leftarrow \{x_1, \ldots, x_k\} : y_i \mapsto x_i.$$

Notice, that the above definition of f is possible because the variables $\{x_1, \ldots, x_k\}$ are pairwise distinct. We have to prove the following four statements:

(i) The values of Ψ are indeed elements of R:
Let $\varrho : \mathsf{T}_\Delta \leftarrow \Sigma$ and $\ell \in L$. We assume that there exists an $x \in X$ such that $x \in var_X(\Psi\varrho\,\ell)$ but $x \notin var_X \ell$. Let $x' \in X \setminus var_X(\Psi\varrho\ell)$. We define $h : X \leftarrow X$

by $hx = x'$ and $h|_{X\setminus\{x,x'\}} = id$.

$$
\begin{aligned}
& \varrho_X \ell \\
= \quad & \{\ h|_{var_X\,\ell} = id\ \} \\
& \varrho_X(\Sigma h \ell) \\
= \quad & \{\ \text{naturalness of } \varrho, \textit{ i.e. } \varrho_X \cdot \Sigma h = \mathsf{T}_\Delta h \cdot \varrho_X\ \} \\
& \mathsf{T}_\Delta h(\varrho_X \ell) \\
\neq \quad & \{\ x' \in var_X\big(\mathsf{T}_\Delta h(\varrho_X \ell)\big) \setminus var_X(\varrho_X \ell)\ \} \\
& \varrho_X \ell.
\end{aligned}
$$

And thus by contraposition $var_X(\Psi\varrho\,\ell) \subseteq var_X\,\ell$.

(ii) The values of $\hat{\ }$ are indeed *natural* transformations:
Let $\varrho = \hat{r}$. Let Y and Z be sets, $h : Y \leftarrow Z$, $\{y_1, \ldots, y_k\} \subseteq Y$, $\{z_1, \ldots, z_k\} \subseteq Z$, $fx_i = hz_i$, and $gx_i = z_i$.

$$
\begin{aligned}
& \varrho_Y\big(\Sigma h(\sigma\,{}'z_1 \cdots {}'z_k)\big) \\
= \quad & \{\ \text{definition of } \Sigma \text{ in Definition 6.1.1.1}\ \} \\
& \varrho_Y\big(\sigma(hz_1) \cdots (hz_k)\big) \\
= \quad & \{\ \text{definition of } \hat{\ }\ \} \\
& \mathsf{T}_\Delta f\big(r(\sigma\,{}'x_1 \cdots {}'x_k)\big) \\
= \quad & \{\ f = h \cdot g\ \} \\
& \mathsf{T}_\Delta(h \cdot g)\big(r(\sigma\,{}'x_1 \cdots {}'x_k)\big) \\
= \quad & \{\ \mathsf{T}_\Delta \text{ is a functor}\ \} \\
& \mathsf{T}_\Delta h\Big(\mathsf{T}_\Delta g\big(r(\sigma\,{}'x_1 \cdots {}'x_k)\big)\Big) \\
= \quad & \{\ \text{definition of } \hat{\ }\ \} \\
& \mathsf{T}_\Delta h\big(\varrho_Z(\sigma\,{}'z_1 \cdots {}'z_k)\big)
\end{aligned}
$$

And thus $\varrho_Y \cdot \Sigma h = \mathsf{T}_\Delta h \cdot \varrho_Z$.

(iii) Obviously $\Psi \cdot \hat{\ } = id$ because $\hat{\ }$ is an extension and Ψ the according restriction.

(iv) $\hat{\ } \cdot \Psi = id$:

Let Y be a set, $\varrho : \mathsf{T}_\Delta \dot{\leftarrow} \Sigma$, and $\sigma\,'y_1 \cdots\, 'y_k \in \Sigma Y$.

$$
\begin{aligned}
&\widehat{\Psi\varrho_Y}(\sigma\,'y_1 \cdots\, 'y_k)\\
=\;& \{ \text{ definition of } \Psi \ \}\\
&\widehat{\varrho_X|_{LY}}(\sigma\,'y_1 \cdots\, 'y_k)\\
=\;& \{ \text{ definition of } \hat{\ } \text{ with } fx_i = y_i \ \}\\
&\mathsf{T}_\Delta f\big(\varrho_X(\sigma\,'x_1 \ldots\, 'x_k)\big)\\
=\;& \{ \text{ naturalness of } \varrho, \textit{ i.e. } \mathsf{T}_\Delta f \cdot \varrho_X = \varrho_Y \cdot \Sigma f \ \}\\
&\varrho_Y\big(\Sigma f(\sigma\,'x_1 \cdots\, 'x_k)\big)\\
=\;& \{ \text{ definition of } \Sigma \text{ in Definition 6.1.1.1 } \}\\
&\varrho_Y(\sigma\,'y_1 \cdots\, 'y_k)
\end{aligned}
$$

■

6.2 Monads and Monad transformers

A monad $\mathbb{T} = (\mathsf{T}, \eta, \mu)$ over a category $\boldsymbol{\mathcal{C}}$ is a triple consisting of a $\boldsymbol{\mathcal{C}}$-endofunctor T and two natural transformations $\eta : \mathsf{T} \dot{\leftarrow} \mathsf{Id}$ and $\mu : \mathsf{T} \dot{\leftarrow} \mathsf{T}^2$ which have to satisfy some axioms (see Definition 4.4.1.1). The category of all monads together with all monad morphisms (*i.e.* natural transformations commuting with the monadic operations, see Definition 4.4.3.1) is denoted by $\mathbf{Mnd}\,\boldsymbol{\mathcal{C}}$.

The intuition for a monad, that we will need is, that it can be viewed as a description of a recursive data structure together with a notion of substitution.

The easiest example for a monad is the **trivial monad** $\mathbb{O}_{\boldsymbol{\mathcal{C}}} = (\mathsf{Id}_{\boldsymbol{\mathcal{C}}}, id_{\mathsf{Id}_{\boldsymbol{\mathcal{C}}}}, id_{\mathsf{Id}_{\boldsymbol{\mathcal{C}}}})$ on a category $\boldsymbol{\mathcal{C}}$. It is easy to see, that $\mathbb{O}_{\boldsymbol{\mathcal{C}}}$ is an initial object in $\mathbf{Mnd}\,\boldsymbol{\mathcal{C}}$ (with unique mediating arrow $|\mathbf{i}_{(T,\eta,\mu)}| = \eta$). Moreover, if 0 is an initial object of $\boldsymbol{\mathcal{C}}$ then $\mathbb{O}_{\boldsymbol{\mathcal{C}}}$ is free over $\underline{0}$ (Example 4.3.1.3 (i)).

An important example are the **tree monads**:

6.2.1 Tree monads and free monads

Consider a $\boldsymbol{\mathcal{S}et}$-endofunctor Σ induced by a ranked alphabet. Then $\mathsf{T}_\Sigma X \cong \boldsymbol{\mu}(\Sigma + \underline{X})$ denotes the set of all Σ-terms over X. It is easy to see that $(\mathsf{T}_\Sigma, '(\centerdot), (\centerdot)^\dagger)$ is a Kleisli triple, *i.e.* the embedding of variables into trees $'(\centerdot)$ and the substitution $(\centerdot)^\dagger$ satisfy the axioms (i)–(iii) given in Definition and Lemma 4.4.1.3. Moreover the according monad $(\mathsf{T}_\Sigma, \eta, \mu)$ is a free monad (Definition 4.4.3.1) over Σ with universal arrow u_Σ given by $(u_\Sigma)_X = id_{\mathsf{T}_\Sigma}|_{\Sigma X}$ (see Theorem 4.3.2.7, Theorem 4.4.4.5, and Corollary 4.4.4.7).

Thus a free monad over a $\boldsymbol{\mathcal{S}et}$-endofunctor Σ induced by a ranked alphabet describes the free term-algebra together with the common term-substitution.

We will use monads (which are not necessarily free) to express the calculations of the right hand sides of tree transducer rules during the computation. And we will use *free*

monads to describe the input (and output) terms of tree transducers. Moreover we will use the (UP) of free monads to define a denotational semantics for tree transducers.

We know that varietors have free monads (Theorem 4.4.4.5). We denote the free monad over a varietor Σ by $\Sigma^\star$ and its underlying endofunctor by $\mathsf{T}_\Sigma = |\Sigma^\star|$ (Definition 4.4.3.1).

The theory of tree transducers can be modeled in the category $\boldsymbol{\mathcal{S}et}$ or even in the full subcategory $\boldsymbol{\mathcal{S}et}_{\aleph_0}$, because sets of terms over finite ranked alphabets are countable. Thus we may choose one of the following to get the existence of the free monads we need:

(i) Every polynomial $\boldsymbol{\mathcal{S}et}$-endofunctor is a varietor (Theorem 4.3.2.7).

(ii) Every $\boldsymbol{\mathcal{S}et}_{\aleph_0}$-endofunctor is a varietor (Proposition 4.1.7.8 & Corollary 4.3.2.6).

In case (i) we have to verify that the functors we deal with are indeed polynomial. In case (ii) we have to be careful, since $\boldsymbol{\mathcal{S}et}_{\aleph_0}$ lacks many of the convenient properties of $\boldsymbol{\mathcal{S}et}$, *e.g.* $\boldsymbol{\mathcal{S}et}_{\aleph_0}$ is not cartesian closed, since function spaces may be uncountable.

However, for us the choice (ii) is more convenient, since $(\mathbf{Mnd}\,\boldsymbol{\mathcal{S}et}_{\aleph_0}, |\mathbf{.}|)$ has free objects and thus $(\mathbf{.})^\star \dashv |\mathbf{.}|$ (Proposition 4.1.7.8 & Corollary 4.4.4.6).

6.2.2 Monad transformers

One part of the monadic transducer, which we define later in Definition 6.3.1.1, is an endofunctor on the category of all monads (on some category). Such a functor is sometimes called a *monad transformer* [Mog90].

Many different definitions of monad transformers exist in the literature. In [Hin00] a monad transformer $(\mathsf{H}, \pi, \omega)$ is an endofunctor H mapping monads onto monads together with two natural transformations $\pi : \mathsf{H} \leftarrow \mathsf{Id}$ (called **promote** or **lift**) and $\omega : \mathsf{Id} \leftarrow \mathsf{H}$ (called **observe**). We will need a natural transformation $\omega : |\mathbf{.}|_0 \cdot |\mathbf{.}|^{\underline{\Delta^\star}} \leftarrow |\mathbf{.}|_0 \cdot \mathsf{H} \cdot |\mathbf{.}|^{\underline{\Delta^\star}}$ to observe the final result of the monadic computation. However, this function ω will not be part of our definition of a monad transformer.

6.2.2.1 Definition (monad transformer). A **pointed functor** (F, π) on a category $\boldsymbol{\mathcal{C}}$ is a pair consisting of a $\boldsymbol{\mathcal{C}}$-endofunctor F and a natural transformation $\pi : \mathsf{F} \leftarrow \mathsf{Id}_{\boldsymbol{\mathcal{C}}}$. A **monad transformer** on $\boldsymbol{\mathcal{C}}$ is a pointed functor on the category of monads $\mathbf{Mnd}\,\boldsymbol{\mathcal{C}}$. $\diamond$

In the following two subsections we will see how to construct monad transformers from adjunctions or coproducts of monads:

6.2.3 Monad transformers from adjunctions

6.2.3.1 Lemma (composition of an adjunction and a monad). Let $(\eta, \varepsilon) : \mathsf{Q} \dashv \mathsf{U} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ be an adjunction and $\mathbb{T} = (\mathsf{T}, \tilde{\eta}, \mu)$ be a monad on $\boldsymbol{\mathcal{D}}$. Then

$$\mathbb{S} = \big(\mathsf{U} \cdot \mathsf{T} \cdot \mathsf{Q}, \mathsf{U}\tilde{\eta}\mathsf{Q} \cdot \eta, \mathsf{U}(\mu \cdot \mathsf{T}\varepsilon\mathsf{T})\mathsf{Q}\big) \tag{6.2.3.1}$$

is a monad on $\boldsymbol{\mathcal{C}}$.

Proof. According to Proposition 4.4.4.3 we have an adjunction $(\tilde{\eta}, \tilde{\varepsilon}) : \mathsf{F}^{\mathbb{T}} \dashv \mathsf{U}^{\mathbb{T}}$ with

$$(\mathsf{U}^{\mathbb{T}} \cdot \mathsf{F}^{\mathbb{T}}, \tilde{\eta}, \mathsf{U}^{\mathbb{T}} \tilde{\varepsilon} \mathsf{F}^{\mathbb{T}}) = \mathbb{T}. \qquad (*)$$

We compose this adjunction with the adjunction $(\eta, \varepsilon) : \mathsf{Q} \dashv \mathsf{U}$ as described in Lemma 4.3.4.5 and get the adjunction

$$(\mathsf{U}\tilde{\eta}\mathsf{Q} \cdot \eta, \tilde{\varepsilon} \cdot \mathsf{F}^{\mathbb{T}} \varepsilon \mathsf{U}^{\mathbb{T}}) : \mathsf{F}^{\mathbb{T}} \cdot \mathsf{Q} \dashv \mathsf{U} \cdot \mathsf{U}^{\mathbb{T}}$$

which itself gives rise to a monad (see Lemma 4.4.4.1):

$$\big(\mathsf{U} \cdot \mathsf{U}^{\mathbb{T}} \cdot \mathsf{F}^{\mathbb{T}} \cdot \mathsf{Q}, \mathsf{U}\tilde{\eta}\mathsf{Q} \cdot \eta, (\mathsf{U} \cdot \mathsf{U}^{\mathbb{T}})(\tilde{\varepsilon} \cdot \mathsf{F}^{\mathbb{T}} \varepsilon \mathsf{U}^{\mathbb{T}})(\mathsf{F}^{\mathbb{T}} \cdot \mathsf{Q})\big).$$

We claim that the latter is equal to $\mathbb{S}$ which would finish our proof. The first components can seen to be equal using the fact $\mathsf{U}^{\mathbb{T}} \cdot \mathsf{F}^{\mathbb{T}} = \mathsf{T}$ from $(*)$. The second components are equal. To show that the third components are equal as well we use $(*)$ and calculate:

$$(\mathsf{U} \cdot \mathsf{U}^{\mathbb{T}})(\tilde{\varepsilon} \cdot \mathsf{F}^{\mathbb{T}} \varepsilon \mathsf{U}^{\mathbb{T}})(\mathsf{F}^{\mathbb{T}} \cdot \mathsf{Q}) = \mathsf{U}\big(\underbrace{\mathsf{U}^{\mathbb{T}} \tilde{\varepsilon} \mathsf{F}^{\mathbb{T}}}_{=\mu} \cdot (\underbrace{\mathsf{U}^{\mathbb{T}} \cdot \mathsf{F}^{\mathbb{T}}}_{=\mathsf{T}}) \varepsilon (\underbrace{\mathsf{U}^{\mathbb{T}} \cdot \mathsf{F}^{\mathbb{T}}}_{=\mathsf{T}})\big)\mathsf{Q}. \qquad \blacksquare$$

We have just seen the function $\mathsf{U} \cdot \mathsf{T} \cdot \mathsf{Q} \leftarrow\!\shortmid \mathsf{T}$. For the following it will be useful to give it a name:

6.2.3.2 Definition. Let $\boldsymbol{\mathcal{C}}$, $\boldsymbol{\mathcal{D}}$, $\boldsymbol{\mathcal{E}}$, and $\boldsymbol{\mathcal{F}}$ be categories. We define the binary operator $\leftarrow\!\circ$ for every $\alpha \in \operatorname{Mor} \boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{D}}}$ and $\beta \in \operatorname{Mor} \boldsymbol{\mathcal{E}}^{\boldsymbol{\mathcal{F}}}$ by

$$\forall\, \mathsf{H} \in \operatorname{Ob} \boldsymbol{\mathcal{D}}^{\boldsymbol{\mathcal{E}}}.\ (\alpha \leftarrow\!\circ\, \beta)_{\mathsf{H}} = \alpha * id_{\mathsf{H}} * \beta.$$

Using Definition and Lemma 4.1.3.7, Lemma 4.1.3.8, and Definition and Lemma 4.1.3.6 it is easy to see that $\leftarrow\!\circ$ is a bifunctor

$$\boldsymbol{.} \leftarrow\!\circ\, \boldsymbol{.} : (\boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{F}}})^{\boldsymbol{\mathcal{D}}^{\boldsymbol{\mathcal{E}}}} \leftarrow \boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{D}}} \times \boldsymbol{\mathcal{E}}^{\boldsymbol{\mathcal{F}}}$$

where the value of $\leftarrow\!\circ$ applied to a pair of objects $\mathsf{F} \in \operatorname{Ob} \boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{D}}}$ and $\mathsf{G} \in \operatorname{Ob} \boldsymbol{\mathcal{E}}^{\boldsymbol{\mathcal{F}}}$ is given by

$$\forall\, \varphi \in \operatorname{Mor} \boldsymbol{\mathcal{D}}^{\boldsymbol{\mathcal{E}}}.\ (\mathsf{F} \leftarrow\!\circ\, \mathsf{G})\varphi = \mathsf{F}\varphi\mathsf{G}$$

where $\mathsf{F} \leftarrow\!\circ\, \mathsf{G}$ is a functor

$$\mathsf{F} \leftarrow\!\circ\, \mathsf{G} : \boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{F}}} \leftarrow \boldsymbol{\mathcal{D}}^{\boldsymbol{\mathcal{E}}}$$

given on objects $\mathsf{H} \in \operatorname{Ob} \boldsymbol{\mathcal{D}}^{\boldsymbol{\mathcal{E}}}$ by $(\mathsf{F} \leftarrow\!\circ\, \mathsf{G})\mathsf{H} = \mathsf{F} \cdot \mathsf{H} \cdot \mathsf{G}$. Notice, that the latter makes $\leftarrow\!\circ$ to a bifunctor

$$\boldsymbol{.} \leftarrow\!\circ\, \boldsymbol{.} : \mathbf{CAT} \leftarrow \mathbf{CAT} \times \mathbf{CAT}^{\mathrm{op}}. \qquad \diamond$$

6.2.3.3 Lemma. The binary operator $\leftarrow\!\!\circ$ is a functor

$$\mathbf{.} \leftarrow\!\!\circ \mathbf{.} : \mathbb{CAT} \leftarrow \mathbb{CAT} \times \mathbb{CAT}^{\mathrm{op}}$$

which is given on objects by

$$\forall \boldsymbol{\mathcal{C}}, \boldsymbol{\mathcal{D}} \in \mathrm{Ob}\,\mathbb{CAT}.\ \boldsymbol{\mathcal{C}} \leftarrow\!\!\circ \boldsymbol{\mathcal{D}} = \boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{D}}}.$$

Proof. Let $\boldsymbol{\mathcal{C}}$, $\boldsymbol{\mathcal{D}}$, $\boldsymbol{\mathcal{E}}$, and $\boldsymbol{\mathcal{F}}$ be categories. Let $\mathsf{F}, \mathsf{F}', \mathsf{G}, \mathsf{G}' : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ and $\mathsf{H}, \mathsf{H}', \mathsf{I}, \mathsf{I}' : \boldsymbol{\mathcal{E}} \leftarrow \boldsymbol{\mathcal{F}}$. Let $\alpha : \mathsf{F} \leftarrow \mathsf{F}'$, $\beta : \mathsf{G} \leftarrow \mathsf{G}'$, $\gamma : \mathsf{H} \leftarrow \mathsf{H}'$ and $\delta : \mathsf{I} \leftarrow \mathsf{I}'$. Then

$$\begin{aligned}
&(\alpha * \beta) \leftarrow\!\!\circ (\gamma * \delta)\\
=\ &\{\ \text{Definition and Lemma 4.1.3.7}\ \}\\
&(\alpha\mathsf{G} \cdot \mathsf{F}'\beta) \leftarrow\!\!\circ (\mathsf{H}\delta \cdot \gamma\mathsf{I}')\\
=\ &\{\ \text{Definition 6.2.3.2: } \mathbf{.} \leftarrow\!\!\circ \mathbf{.} : (\boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{F}}})^{\boldsymbol{\mathcal{D}}^{\boldsymbol{\mathcal{E}}}} \leftarrow \boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{D}}} \times \boldsymbol{\mathcal{E}}^{\boldsymbol{\mathcal{F}}}\ \}\\
&(\alpha\mathsf{G} \leftarrow\!\!\circ \mathsf{H}\delta) \cdot (\mathsf{F}'\beta \leftarrow\!\!\circ \gamma\mathsf{I}')\\
=\ &\{\ \text{Definition 6.2.3.2: } \mathbf{.} \leftarrow\!\!\circ \mathbf{.} : \mathbf{CAT} \leftarrow \mathbf{CAT} \times \mathbf{CAT}^{\mathrm{op}}\ \}\\
&(\alpha \leftarrow\!\!\circ \delta)(\mathsf{G} \leftarrow\!\!\circ \mathsf{H}) \cdot (\mathsf{F}' \leftarrow\!\!\circ \mathsf{I}')(\beta \leftarrow\!\!\circ \gamma)\\
=\ &\{\ \text{Definition and Lemma 4.1.3.7}\ \}\\
&(\alpha \leftarrow\!\!\circ \delta) * (\beta \leftarrow\!\!\circ \gamma)
\end{aligned}$$

■

6.2.3.4 Definition. Let $\mathsf{Q} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ be a left adjoint functor. We use the construction from Lemma 6.2.3.1 to define a functor $\overline{\mathsf{Q}} : \mathbf{Mnd}\,\boldsymbol{\mathcal{D}} \leftarrow \mathbf{Mnd}\,\boldsymbol{\mathcal{C}}$ by

$$\forall\,(\mathsf{T}, \tilde{\eta}, \mu) \in \mathrm{Ob}\big(\mathbf{Mnd}\,\boldsymbol{\mathcal{C}}\big).\ \overline{\mathsf{Q}}(\mathsf{T}, \tilde{\eta}, \mu) = (\mathsf{U} \cdot \mathsf{T} \cdot \mathsf{Q}, \mathsf{U}\tilde{\eta}\mathsf{Q} \cdot \eta, \mathsf{U}(\mu \cdot \mathsf{T}\varepsilon\mathsf{T})\mathsf{Q}),$$

$$\forall\, h \in \mathrm{Mor}\big(\mathbf{Mnd}\,\boldsymbol{\mathcal{C}}\big).\ \overline{\mathsf{Q}}h = \mathsf{U}h\mathsf{Q}$$

where $(\eta, \varepsilon) : \mathsf{Q} \dashv \mathsf{U}$ is an adjunction. That $\overline{\mathsf{Q}}$ is indeed a functor follows from Definition and Lemma 4.1.3.6 and the fact that U is a functor. Moreover $\overline{\mathsf{Q}}$ is a concrete functor $\overline{\mathsf{Q}} : \big(\mathbf{Mnd}\,\boldsymbol{\mathcal{D}}, |\mathbf{.}|\big) \leftarrow \big(\mathbf{Mnd}\,\boldsymbol{\mathcal{C}}, (\mathsf{U} \leftarrow\!\!\circ \mathsf{Q}) \cdot |\mathbf{.}|\big)$. Notice that $\overline{\mathsf{Q}}$ depends on the choice of the right adjoint U. However, according to Lemma 4.3.4.8 Q determines U uniquely up to isomorphism. ◇

6.2.3.5 Lemma. The function $\overline{\mathbf{.}}$ from Definition 6.2.3.4 is a functor

$$\overline{\mathbf{.}} : \mathbf{CAT} \leftarrow \mathbf{LeftAdj}^{\mathrm{op}}$$

where **LeftAdj** denotes the subcategory of **CAT** where the morphisms are all left adjoint functors (see Corollary 4.3.4.6).

Proof. Obviously $\overline{\mathsf{Id}} = \mathsf{Id}$. Let $\mathsf{Q}, \mathsf{Q}' : \mathcal{C} \leftarrow \mathcal{D}$ be left adjoint endofunctors with adjunctions $\mathsf{Q} \dashv \mathsf{U}$ and $\mathsf{Q}' \dashv \mathsf{U}'$, respectively. Then $\mathsf{Q}' \cdot \mathsf{Q} \dashv \mathsf{U} \cdot \mathsf{U}'$ is an adjunction according to Lemma 4.3.4.5. We show that $\overline{\mathsf{Q}} \cdot \overline{\mathsf{Q}'} = \overline{\mathsf{Q}' \cdot \mathsf{Q}}$ holds by pointwise calculation for all $h \in \mathrm{Mor}\big(\mathbf{Mnd}\,\mathcal{D}\big)$ as follows: $(\overline{\mathsf{Q}} \cdot \overline{\mathsf{Q}'})h = \overline{\mathsf{Q}}(\overline{\mathsf{Q}'}h) = \mathsf{U}(\mathsf{U}'h\mathsf{Q}')\mathsf{Q} = (\mathsf{U} \cdot \mathsf{U}')h(\mathsf{Q}' \cdot \mathsf{Q}) = \overline{\mathsf{Q}' \cdot \mathsf{Q}}$. ■

6.2.3.6 Lemma. The bifunctor $\leftarrow\!\circ$ preserves adjunctions: Let Q, U, Q', and U' be functors. Then

$$\mathsf{Q} \dashv \mathsf{U} \;\wedge\; \mathsf{Q}' \dashv \mathsf{U}' \implies (\mathsf{Q} \leftarrow\!\circ\, \mathsf{U}') \dashv (\mathsf{U} \leftarrow\!\circ\, \mathsf{Q}').$$

Proof. We claim that

$$(\eta, \varepsilon) : \mathsf{Q} \dashv \mathsf{U} \;\wedge\; (\eta', \varepsilon') : \mathsf{Q}' \dashv \mathsf{U}' \implies (\eta \leftarrow\!\circ\, \eta', \varepsilon \leftarrow\!\circ\, \varepsilon') : (\mathsf{Q} \leftarrow\!\circ\, \mathsf{U}') \dashv (\mathsf{U} \leftarrow\!\circ\, \mathsf{Q}')$$

holds. We have to show that $\eta \leftarrow\!\circ\, \eta'$ and $\varepsilon \leftarrow\!\circ\, \varepsilon'$ satisfy the axioms of an adjunction given in Definition 4.3.4.1:

$$\begin{aligned}
&(\varepsilon \leftarrow\!\circ\, \varepsilon')(\mathsf{Q} \leftarrow\!\circ\, \mathsf{U}') \;\cdot\; (\mathsf{Q} \leftarrow\!\circ\, \mathsf{U}')(\eta \leftarrow\!\circ\, \eta') \\
=\;& \quad \{\ \text{Definition 6.2.3.2}\ \} \\
&(\varepsilon\mathsf{Q} \leftarrow\!\circ\, \mathsf{U}'\varepsilon') \;\cdot\; (\mathsf{Q}\eta \leftarrow\!\circ\, \eta'\mathsf{U}') \\
=\;& \quad \{\ \text{Definition 6.2.3.2}\ \} \\
&(\varepsilon\mathsf{Q} \cdot \mathsf{Q}\eta) \leftarrow\!\circ\, (\mathsf{U}'\varepsilon' \cdot \eta'\mathsf{U}') \\
=\;& \quad \{\ (\eta, \varepsilon) : \mathsf{Q} \dashv \mathsf{U},\ (\eta', \varepsilon') : \mathsf{Q}' \dashv \mathsf{U}' \ \&\ \text{Definition 4.3.4.1}\ \} \\
&id_{\mathsf{Q}} \leftarrow\!\circ\, id_{\mathsf{U}'} \\
=\;& \quad \{\ \text{Definition 6.2.3.2}\ \} \\
&id_{\mathsf{Q} \leftarrow\!\circ\, \mathsf{U}'}.
\end{aligned}$$

And similar $(\mathsf{U} \leftarrow\!\circ\, \mathsf{Q}')(\varepsilon \leftarrow\!\circ\, \varepsilon') \;\cdot\; (\eta \leftarrow\!\circ\, \eta')(\mathsf{U} \leftarrow\!\circ\, \mathsf{Q}') = id_{\mathsf{U} \leftarrow\!\circ\, \mathsf{Q}'}$. ■

6.2.3.7 Proposition. Let $\mathcal{C}$ be a category. The bifunctor $\leftarrow\!\circ\, : \mathbf{End}^{\mathbf{2}}\,\mathcal{C} \leftarrow \mathbf{End}\,\mathcal{C} \times \mathbf{End}\,\mathcal{C}$ can be extended to a concrete bifunctor:

$$\mathbf{.} \leftarrow\!\circ\, \mathbf{.} : (\mathbf{Mnd}(\mathbf{End}\,\mathcal{C}), |\mathbf{.}|) \leftarrow (\mathbf{Mon}\,\mathcal{C}, |\mathbf{.}|) \times (\mathbf{Mnd}\,\mathcal{C}, |\mathbf{.}|)$$

Proof. We define $\leftarrow\!\circ$ on monads (T, η, μ) and $(\tilde{\mathsf{T}}, \tilde{\eta}, \tilde{\mu})$ on $\mathcal{C}$ by

$$(\mathsf{T}, \eta, \mu) \leftarrow\!\circ\, (\tilde{\mathsf{T}}, \tilde{\eta}, \tilde{\mu}) = (\mathsf{T} \leftarrow\!\circ\, \tilde{\mathsf{T}}, \eta \leftarrow\!\circ\, \tilde{\eta}, \mu \leftarrow\!\circ\, \tilde{\mu}). \qquad (*)$$

First we have to show that the right hand side of $(*)$ is indeed a monad: We could just verify the monad axioms. But we will do it a little more complicated: By means of

Proposition 4.4.4.3 we have adjunctions $(\eta, \varepsilon) : \mathsf{F}^{\mathbb{T}} \dashv \mathsf{U}^{\mathbb{T}}$ and $(\tilde{\eta}, \tilde{\varepsilon}) : \tilde{\mathsf{F}}^{\tilde{\mathbb{T}}} \dashv \tilde{\mathsf{U}}^{\tilde{\mathbb{T}}}$ such that $(\mathsf{U}^{\mathbb{T}} \cdot \mathsf{F}^{\mathbb{T}}, \eta, \mathsf{U}^{\mathbb{T}} \varepsilon \mathsf{F}^{\mathbb{T}}) = \mathbb{T}$ and $\big(\tilde{\mathsf{U}}^{\tilde{\mathbb{T}}} \cdot \tilde{\mathsf{F}}^{\tilde{\mathbb{T}}}, \tilde{\eta}, \tilde{\mathsf{U}}^{\tilde{\mathbb{T}}} \tilde{\varepsilon} \tilde{\mathsf{F}}^{\tilde{\mathbb{T}}}\big) = \tilde{\mathbb{T}}$. Then $(\eta \leftarrow\!\circ\, \tilde{\eta}, \varepsilon \leftarrow\!\circ\, \tilde{\varepsilon}) : (\mathsf{F}^{\mathbb{T}} \leftarrow\!\circ\, \tilde{\mathsf{U}}^{\tilde{\mathbb{T}}}) \dashv (\mathsf{U}^{\mathbb{T}} \leftarrow\!\circ\, \tilde{\mathsf{F}}^{\tilde{\mathbb{T}}})$ is an adjunction according to Lemma 6.2.3.6. Using Lemma 4.4.4.1 we can make this into a monad $\big((\mathsf{U}^{\mathbb{T}} \leftarrow\!\circ\, \tilde{\mathsf{F}}^{\tilde{\mathbb{T}}}) \cdot (\mathsf{F}^{\mathbb{T}} \leftarrow\!\circ\, \tilde{\mathsf{U}}^{\tilde{\mathbb{T}}}), \eta \leftarrow\!\circ\, \tilde{\eta}, (\mathsf{U}^{\mathbb{T}} \leftarrow\!\circ\, \tilde{\mathsf{F}}^{\tilde{\mathbb{T}}})(\varepsilon \leftarrow\!\circ\, \tilde{\varepsilon})(\mathsf{F}^{\mathbb{T}} \leftarrow\!\circ\, \tilde{\mathsf{U}}^{\tilde{\mathbb{T}}})\big)$. The latter is equal to the right hand side of $(*)$ which can easily be seen using Definition 6.2.3.2. Notice, that we in fact derived the right hand side of $(*)$ rather that just verifying that it is a monad.

It remains to show that $\leftarrow\!\circ$ maps monad morphisms onto monad morphisms: Let $h : \mathbb{T} \leftarrow \mathbb{T}'$ and $\tilde{h} : \tilde{\mathbb{T}} \leftarrow \tilde{\mathbb{T}}'$ be monad morphisms. We have to show $h \leftarrow\!\circ\, \tilde{h} : (\mathbb{T} \leftarrow\!\circ\, \tilde{\mathbb{T}}) \leftarrow (\mathbb{T}' \leftarrow\!\circ\, \tilde{\mathbb{T}}')$ *i.e.* we have to verify the axioms of a monad morphism given in Definition 4.4.3.1 (i). We will only show that $h \leftarrow\!\circ\, \tilde{h}$ respects joins:

$$
\begin{aligned}
& (\mu \leftarrow\!\circ\, \tilde{\mu}) \cdot \big((h \leftarrow\!\circ\, \tilde{h}) * (h \leftarrow\!\circ\, \tilde{h})\big) \\
= \quad & \{ \text{ Lemma 6.2.3.3 } \} \\
& (\mu \leftarrow\!\circ\, \tilde{\mu}) \cdot \big((h * h) \leftarrow\!\circ\, (\tilde{h} * \tilde{h})\big) \\
= \quad & \{ \text{ Definition 6.2.3.2 } \} \\
& \big(\mu \cdot (h * h)\big) \leftarrow\!\circ\, \big(\tilde{\mu} \cdot (\tilde{h} * \tilde{h})\big) \\
= \quad & \{ \text{ precondition } \} \\
& (h \cdot \mu') \leftarrow\!\circ\, (\tilde{h} \cdot \tilde{\mu}') \\
= \quad & \{ \text{ Definition 6.2.3.2 } \} \\
& (h \leftarrow\!\circ\, \tilde{h}) \cdot (\mu' \leftarrow\!\circ\, \tilde{\mu}')
\end{aligned}
$$

Similarity $h \leftarrow\!\circ\, \tilde{h}$ respects units also. ∎

6.2.3.8 Definition. We define the functor $\text{Ł} : \mathbf{LeftAdj} \leftarrow \mathbf{LeftAdj}$ for every left adjoint functor Q by

$$\text{Ł}\mathsf{Q} = \mathsf{Q} \leftarrow\!\circ\, \mathsf{U}$$

where U is a right adjoint of Q. Notice that Ł is determined uniquely up to isomorphism with Lemma 4.3.4.8. That Ł is indeed a functor follows from Lemma 4.3.4.5 and Definition 6.2.3.2. Finally Ł preserves left adjoint functors due to Lemma 6.2.3.6. ◇

6.2.3.9 Lemma. Let $\mathsf{Q} \dashv \mathsf{U} : \mathcal{C} \leftarrow \mathcal{D}$ be an adjunction and $\mathbb{T}$ and $\mathbb{T}'$ be monads on $\mathcal{C}$. Then:

$$\overline{\mathsf{Q} \leftarrow\!\circ\, \mathsf{U}}(\mathbb{T} \leftarrow\!\circ\, \mathbb{T}') \cong \overline{\mathsf{Q}}\mathbb{T} \leftarrow\!\circ\, \overline{\mathsf{Q}}\mathbb{T}'$$

Proof. Let $\mathbb{T} = (\mathsf{T}, \eta, \mu)$, $\mathbb{T}' = (\mathsf{T}', \eta', \mu')$, and $(\tilde{\eta}, \varepsilon) : \mathsf{Q} \dashv \mathsf{U}$. With Lemma 6.2.3.6 we get:

$$(\tilde{\eta} \leftarrow\!\circ\, \tilde{\eta}, \varepsilon \leftarrow\!\circ\, \varepsilon) : (\mathsf{Q} \leftarrow\!\circ\, \mathsf{U}) \dashv (\mathsf{U} \leftarrow\!\circ\, \mathsf{Q}). \qquad (*)$$

We calculate:

$$
\begin{aligned}
&\overline{\mathsf{Q} \longleftarrow\circ\, \mathsf{U}}(\mathbb{T} \longleftarrow\circ\, \mathbb{T}') \\
=\;& \{\ \text{Proposition 6.2.3.7}\ \} \\
&\overline{\mathsf{Q} \longleftarrow\circ\, \mathsf{U}}(\mathsf{T} \longleftarrow\circ\, \mathsf{T}', \eta \longleftarrow\circ\, \eta', \mu \longleftarrow\circ\, \mu') \\
=\;& \{\ (*)\ \&\ \text{Definition 6.2.3.4}\ \} \\
&\Big((\mathsf{U} \longleftarrow\circ\, \mathsf{Q})(\mathsf{T} \longleftarrow\circ\, \mathsf{T}')(\mathsf{Q} \longleftarrow\circ\, \mathsf{U}),\ (\mathsf{U} \longleftarrow\circ\, \mathsf{Q})(\eta \longleftarrow\circ\, \eta')(\mathsf{Q} \longleftarrow\circ\, \mathsf{U}) \cdot (\tilde{\eta} \longleftarrow\circ\, \tilde{\eta}), \\
&\qquad\qquad (\mathsf{U} \longleftarrow\circ\, \mathsf{Q})\big((\mu \longleftarrow\circ\, \mu') \cdot (\mathsf{T} \longleftarrow\circ\, \mathsf{T}')(\varepsilon \longleftarrow\circ\, \varepsilon)(\mathsf{T} \longleftarrow\circ\, \mathsf{T}')\big)(\mathsf{Q} \longleftarrow\circ\, \mathsf{U})\Big) \\
=\;& \{\ \text{Definition 6.2.3.2}\ \} \\
&\big(\mathsf{U} \cdot \mathsf{T} \cdot \mathsf{Q} \longleftarrow\circ\, \mathsf{U} \cdot \mathsf{T}' \cdot \mathsf{Q},\ \mathsf{U}\eta\mathsf{Q} \cdot \tilde{\eta} \longleftarrow\circ\, \mathsf{U}\eta'\mathsf{Q} \cdot \tilde{\eta},\ \mathsf{U}(\mu \cdot \mathsf{T}\varepsilon\mathsf{T})\mathsf{Q} \longleftarrow\circ\, \mathsf{U}(\mu' \cdot \mathsf{T}'\varepsilon\mathsf{T}')\mathsf{Q}\big) \\
=\;& \{\ \text{Proposition 6.2.3.7}\ \} \\
&\big(\mathsf{U} \cdot \mathsf{T} \cdot \mathsf{Q}, \mathsf{U}\eta\mathsf{Q} \cdot \tilde{\eta}, \mathsf{U}(\mu \cdot \mathsf{T}\varepsilon\mathsf{T})\mathsf{Q}\big)\ \cdot\ \big(\mathsf{U} \cdot \mathsf{T}' \cdot \mathsf{Q}, \mathsf{U}\eta'\mathsf{Q} \cdot \tilde{\eta}, \mathsf{U}(\mu' \cdot \mathsf{T}'\varepsilon\mathsf{T}')\mathsf{Q}\big) \\
=\;& \{\ \text{Definition 6.2.3.4}\ \} \\
&\overline{\mathsf{Q}}\mathbb{T} \longleftarrow\circ\, \overline{\mathsf{Q}}\mathbb{T}'
\end{aligned}
$$

■

6.2.3.10 Lemma. Let $(\eta, \varepsilon) : \mathsf{Q} \dashv \mathsf{U} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$ be an adjunction where Q is a cocartesian endofunctor. Then $\overline{\mathsf{Q}}$ is a monad transformer.

Proof. Since Q is cocartesian we have a product $(\mathsf{Q} \xleftarrow{\iota_q} \mathsf{Id})_{q \in Q}$ where Q is a finite set. We claim that $(\overline{\mathsf{Q}}, \pi)$ is a pointed functor where for every monad $\mathbb{T} = (\mathsf{T}, \tilde{\eta}, \mu)$:

$$\pi_{\mathbb{T}} = \mathsf{U}[\mathsf{T}\iota_q]_{q \in Q} \cdot \eta\mathsf{T}$$

where $m = [\mathsf{T}\iota_q]_{q \in Q}$ denotes the unique mediating morphism satisfying $\forall\, q \in Q.\ m \cdot \iota_q = \mathsf{T}\iota_q$. That π is indeed natural in $\mathbb{T}$ and that π is a monad morphism (*i.e.* it respects units and joins according to Definition 4.4.3.1) can be demonstrated by straightforward calculations. ■

6.2.4 Monad transformers from coproducts of monads

The coproduct of monads on a category $\boldsymbol{\mathcal{C}}$ is just the usual coproduct in the category $\mathbf{Mnd}\,\boldsymbol{\mathcal{C}}$.

Colimits of monads have been studied in [Kel80]. Coproducts of monads have been used in [LG02a, LG02b] to construct monad transformers.

6.2.4.1 Lemma (coproduct of free monads). Let $\boldsymbol{\mathcal{C}}$ be a cocartesian category and Σ and Δ be $\boldsymbol{\mathcal{C}}$-varietors. Then:

$$\Sigma^\star + \Delta^\star \cong (\Sigma + \Delta)^\star$$

Proof. The free functor $(\mathbf{.})^\star$ mapping a varietor onto its free monad is left adjoint and thus preserves coproducts (Proposition 4.3.4.9). ∎

6.2.4.2 Definition and Lemma. Let $\mathcal{C}$ be a cocartesian category with initial object 0 and A a $\mathcal{C}$-object. Let us denote the left and right injections of binary coproducts by $\grave{\iota}$ and $\acute{\iota}$, respectively.

(i) $A^+ = \big((A + \mathbf{.}), \acute{\iota}, [\grave{\iota}, id]\big)$ is a monad on $\mathcal{C}$.

(ii) The monad A^+ is free over $\underline{A}$.

(iii) The function $(\mathbf{.})^+$ is a functor

$$(\mathbf{.})^+ : \mathbf{Mnd}\,\mathcal{C} \leftarrow \mathcal{C}$$

defined on $\mathcal{C}$-morphisms f by $f^+ = \underline{f} + id$.

Proof. (i) Elementary.

(ii) We claim that $(u_{\underline{A}})_X = \grave{\iota}_{(A,X)}$ is a universal arrow, *i.e.* for every monad $\mathbb{T} = (\mathsf{T}, \eta, \mu)$ and every $\varrho : \mathsf{T} \leftarrow \underline{A}$ there exists a unique $\varrho/u_{\underline{A}} : \mathbb{T} \leftarrow A^+$ such that $|\varrho/u_{\underline{A}}| \cdot u_{\underline{A}} = \varrho$. This can easily be verified for $|\varrho/u_{\underline{A}}| = [\varrho, \eta]$ using the UP of the coproduct.

(iii) Since $\underline{\mathbf{.}}$ and $+$ are functors $(\mathbf{.})^+$ is also a functor. That $(\mathbf{.})^+$ maps onto monad morphisms can easily be checked. ∎

6.2.4.3 Definition (abstraction functor). Let $\mathcal{C}$ be a category and $I \in \mathrm{Ob}\,\mathcal{C}$.

(i) We define the functor $\Lambda_I : \mathcal{C} \leftarrow \mathbf{End}\,\mathcal{C}$ by

$$\forall\, \mathsf{T} \in \mathrm{Ob}(\mathbf{End}\,\mathcal{C}).\ \Lambda_I \mathsf{T} = \mathsf{T}I \quad \text{and}$$
$$\forall\, h \in \mathrm{Mor}(\mathbf{End}\,\mathcal{C}).\ \Lambda_I h = h_I.$$

That this is indeed a *functor* is easy to see, using the definition of the horizontal composition of natural transformations.

(ii) We define the functor $|\mathbf{.}|_I : \mathcal{C} \leftarrow \mathbf{Mnd}\,\mathcal{C}$ by $\Lambda_I \cdot |\mathbf{.}|$ where $\big(\mathbf{Mnd}\,\mathcal{C}, |\mathbf{.}|\big)$ is the concrete category of monads on $\mathcal{C}$ and $|\mathbf{.}| : \mathbf{End}\,\mathcal{C} \leftarrow \mathbf{Mnd}\,\mathcal{C}$ the default forgetful functor mapping a monad onto its underlying endofunctor. Thus in particular $|(\mathsf{T}, \eta, \mu)|_0 = \mathsf{T}0$. ◇

6.2.4.4 Corollary. Let $\mathcal{C}$ be a category such that $\mathbf{Mnd}\,\mathcal{C}$ is cocartesian. Then:

(i)

$$(\mathbf{.})^+ : \big(\mathbf{Mnd}(\mathbf{Mnd}\,\mathcal{C}), |\mathbf{.}| \cdot |\mathbf{.}|_{\mathbb{O}_\mathcal{C}}\big) \leftarrow \big(\mathbf{Mnd}\,\mathcal{C}, |\mathbf{.}|\big)$$

is a semi-concrete functor.

(ii) For every monad $\mathbb{T}$ on $\boldsymbol{\mathcal{C}}$ the functor $(\mathbb{T} + \boldsymbol{.}) = |\mathbb{T}^+|$ is a monad transformer: $\big((\mathbb{T} + \boldsymbol{.}), \acute{\imath}\big)$ where $\acute{\imath}$ denotes a right injection into the coproduct of two monads. $\diamond$

We have already seen how to construct a coproduct of two free monads. The following theorem gives us a coproduct of a free monad and an *arbitrary* monad:

6.2.4.5 Theorem. Let $\mathsf{Q} \dashv \mathsf{U} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ and $\Delta : \boldsymbol{\mathcal{D}} \leftarrow \boldsymbol{\mathcal{D}}$ such that Δ and $\mathsf{Q}\Delta \cdot \mathsf{U}$ have free monads. Then:

$$\overline{\mathsf{Q}}\mathbb{O}_{\boldsymbol{\mathcal{C}}} + \Delta^\star \cong \overline{\mathsf{Q}}(\mathsf{Q} \cdot \Delta \cdot \mathsf{U})^\star \quad \text{natural in } \Delta.$$

Proof. Using Corollary 4.4.4.7 and the rolling rule (Corollary 4.1.7.6) the right hand side can be shown to be isomorphic to the *resumptions monad* from [CM93]. Then we can apply Proposition 5.3 of [SP00] as demonstrated in Theorem 4 and Corollary 2 of [HPP02]. ∎

6.2.4.6 Corollary. Let Q be left adjoint. Then:

$$\overline{(\boldsymbol{.})^\star} \cdot (\boldsymbol{.})^+ \cdot \overline{\mathsf{Q}} \cong \overline{\text{Ł}\mathsf{Q}} \cdot \overline{(\boldsymbol{.})^\star} \cdot (\boldsymbol{.})^+$$

provided the coproducts exist.

Proof. Let $\mathsf{Q} \dashv \mathsf{U}$. We can generalize Theorem 6.2.4.5 to

$$\overline{\mathsf{Q}}\mathbb{T} + \Delta^\star \cong \overline{\mathsf{Q}}\big(\mathbb{T} + (\mathsf{Q} \cdot \Delta \cdot \mathsf{U})^\star\big) \quad \text{natural in } \Delta.$$

Then the assertion is just a point free version of the latter equation. ∎

6.2.4.7 Corollary. For every $\boldsymbol{\mathcal{C}}$-object A we have a concrete functor

$$(A^+ + \boldsymbol{.}) : (\mathbf{Mnd}\,\boldsymbol{\mathcal{C}}, |\boldsymbol{.}|) \leftarrow \Big(\mathbf{Mnd}\,\boldsymbol{\mathcal{C}}, \big(\mathsf{Id} \leftarrow\!\circ\, (A + \boldsymbol{.})\big) \cdot |\boldsymbol{.}|\Big). \qquad \diamond$$

6.3 Monadic transducers

A monadic transducer is a generalization of a tree transducer described in terms of category theory. The advantage of monadic transducers is a higher level of abstraction which leads to much more elegant proofs and enables us to treat different kinds of tree transducers (homomorphism, top-down, tree-series, ...) in a unified framework. Monadic transducers can be used to give denotational semantics to fragments of functional programs. We will use this denotational semantics to prove the correctness of our monadic fusion.

6.3.1 Syntax and semantics of monadic transducers

6.3.1.1 Definition (monadic transducer). Let $\mathcal{C}$ be a category which has an initial object 0 and let $\Sigma, \Delta : \mathcal{C} \leftarrow \mathcal{C}$ be varietors. A triple $\mathbb{M} = (\mathsf{H}, \varrho, \omega)$ is called a **monadic transducer** (to Δ from Σ) on $\mathcal{C}$ if

(i) $\mathsf{H} : \mathbf{Mnd}\,\mathcal{C} \leftarrow \mathbf{Mnd}\,\mathcal{C}$ (called **pattern**) is an endofunctor,

(ii) $\varrho : |\mathsf{H}\Delta^\star| \dot{\leftarrow} \Sigma$ (called **rule**), and

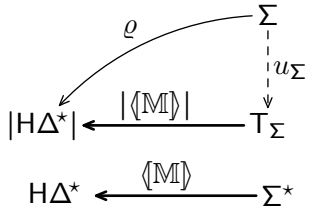
(iii) $\omega : |\mathbf{.}|_0 \cdot |\mathbf{.}|^{\underline{\Delta^\star}} \dot{\leftarrow} |\mathbf{.}|_0 \cdot \mathsf{H} \cdot |\mathbf{.}|^{\underline{\Delta^\star}}$ (called **observe**) are natural transformations.

We denote this by $\mathbb{M} = (\mathsf{H}, \varrho, \omega) : \Delta \leftarrow \Sigma$.

In the above definition we used $|\mathbf{.}|_0 : \mathcal{C} \leftarrow \mathbf{Mnd}\,\mathcal{C}$ from Definition 6.2.4.3 (ii) and $|\mathbf{.}|^{\underline{\Delta^\star}} : \mathbf{Mnd}\,\mathcal{C} \leftarrow (\mathbf{Mnd}\,\mathcal{C})^{\underline{\Delta^\star}}$ from Definition 4.1.7.1, with the convention $\underline{\Delta^\star} = \underline{(\Delta^\star)}$. ◇

We define the semantics of a monadic transducer in two phases (corresponding to **Step 2** and **Step 3** from the motivation in the beginning of Chapter 6): We wait with **Step 1** until Section 6.4 where we are going to describe tree transducers of different syntactic classes as monadic transducers. The first phase corresponds to **Step 2:** from the motivation in the beginning of Chapter 6:

6.3.1.2 Definition (generalized semantics of a monadic transducer). Let $\mathbb{M} = (\mathsf{H}, \varrho, \omega) : \Delta \leftarrow \Sigma$ be a monadic transducer on $\mathcal{C}$. Since $\Sigma^\star$ is free over Σ, there exists a universal arrow $u_\Sigma : \mathsf{T}_\Sigma \dot{\leftarrow} \Sigma$ (see Theorem 4.4.4.5 and Definition 4.3.1.2). Then there exists a unique monad morphism $(\!|\mathbb{M}|\!) : \mathsf{H}\Delta^\star \leftarrow \Sigma^\star$ such that $|(\!|\mathbb{M}|\!)| \cdot u_\Sigma = \varrho$ holds:



The underlying natural transformation $(\!|\mathbb{M}|\!) : |\mathsf{H}\Delta^\star| \dot{\leftarrow} \mathsf{T}_\Sigma$ is called the **generalized semantics** of $\mathbb{M}$. The generalized semantics is independent from ω. However, it depends on the choice of the universal arrow u_Σ. To make things simpler, we choose for every varietor Σ a universal arrow u_Σ from Σ (to the free monad over Σ) and use this choice implicitly for the generalized semantics of every monadic transducer. To simplify our notation we will sometimes omit the forgetful functor $|\mathbf{.}|$ on morphisms.

It is worth mentioning that 'being a monad morphism' (Definition 4.4.3.1) is a natural property for the generalized semantics $(\!|\mathbb{M}|\!)$:

(i) $\eta = (\!|\mathbb{M}|\!) \cdot \eta'$ means that variables will be throughput and

(ii) $\mu \cdot ((\!|\mathbb{M}|\!) * (\!|\mathbb{M}|\!)) = (\!|\mathbb{M}|\!) \cdot \mu'$ states that $(\!|\mathbb{M}|\!)$ is compositional (or syntax directed).

Thus, in other words, the generalized semantics is the *unique* compositional (and variable through passing) extension of the rule. ◇

The second phase corresponds to **Step 3:** from the motivation in the beginning of Chapter 6:

6.3.1.3 Definition (semantics of a monadic transducer). Let $\mathbb{M} = (\mathsf{H}, \varrho, \omega) : \Delta \leftarrow \Sigma$ be a monadic transducer on $\boldsymbol{\mathcal{C}}$ which has an initial object 0. The **semantics** $[\![\mathbb{M}]\!] : \mathsf{T}_\Delta 0 \leftarrow \mathsf{T}_\Sigma 0$ of $\mathbb{M}$ is defined by

$$[\![\mathbb{M}]\!] = \omega_{id_{\Delta^\star}} \cdot |\langle\!\langle \mathbb{M} \rangle\!\rangle|_0. \qquad \diamond$$

6.3.2 Fusion of monadic transducers

6.3.2.1 Definition. Let $\boldsymbol{\mathcal{C}}$ be a category. We define the functor $(\,\centerdot\,)^\bullet : \mathbf{CAT} \leftarrow \boldsymbol{\mathcal{C}}^{\mathrm{op}}$ on objects by $\forall\, A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}.\ A^\bullet = \boldsymbol{\mathcal{C}}^{\underline{A}}$. For every $f : A \leftarrow B$ we define the functor $f^\bullet : \boldsymbol{\mathcal{C}}^{\underline{B}} \leftarrow \boldsymbol{\mathcal{C}}^{\underline{A}}$ on objects by

$$\forall\, \varphi \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}^{\underline{A}}.\ f^\bullet \varphi = \varphi \cdot f.$$

It is easy to see that the latter induces a unique concrete functor $f^\bullet : (\boldsymbol{\mathcal{C}}^{\underline{B}}, |\,\centerdot\,|^{\underline{B}}) \leftarrow (\boldsymbol{\mathcal{C}}^{\underline{A}}, |\,\centerdot\,|^{\underline{A}})$. $\diamond$

6.3.2.2 Definition. Let $\boldsymbol{\mathcal{C}}$ and $\boldsymbol{\mathcal{D}}$ be categories and $A \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}}$. Every functor $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ induces a functor $\mathsf{F}^{(A)} : \boldsymbol{\mathcal{C}}^{\underline{\mathsf{F}A}} \leftarrow \boldsymbol{\mathcal{D}}^{\underline{A}}$ where F operates on $\underline{A}$-algebras and on $\underline{A}$-algebra morphisms just in the same way as on $\boldsymbol{\mathcal{D}}$-morphism. Obviously, F maps every $\underline{A}$-algebra $B \xleftarrow{\varphi} \underline{A}B = A$ onto an $\underline{\mathsf{F}A}$-algebra $\mathsf{F}B \xleftarrow{\mathsf{F}\varphi} \mathsf{F}A = \underline{\mathsf{F}A}(\mathsf{F}B)$. Since every functor maps commuting diagrams onto commuting diagrams, the functor F will map $\underline{A}$-algebra morphisms onto $\underline{\mathsf{F}A}$-algebra morphisms. Moreover, $\mathsf{F}^{(A)} : (\boldsymbol{\mathcal{C}}^{\underline{\mathsf{F}A}}, |\,\centerdot\,|^{\underline{\mathsf{F}A}}) \leftarrow (\boldsymbol{\mathcal{D}}^{\underline{A}}, \mathsf{F} \cdot |\,\centerdot\,|^{\underline{A}})$ is concrete. Notice, that this is not possible for F-algebras in general, since it depends on properties of constant functors. $\diamond$

6.3.2.3 Definition (fusion of monadic transducers). Let $\mathbb{M} = (\mathsf{H}, \varrho, \omega) : \Gamma \leftarrow \Delta$ and $\mathbb{M} = (\mathsf{H}', \varrho', \omega') : \Delta \leftarrow \Sigma$ be monadic transducers on $\boldsymbol{\mathcal{C}}$. The **fusion** $\mathbb{M} \cdot \mathbb{M}' : \Gamma \leftarrow \Sigma$ of $\mathbb{M}$ and $\mathbb{M}'$ is the monadic transducer on $\boldsymbol{\mathcal{C}}$ defined by

$$\mathbb{M} \cdot \mathbb{M}' = \big(\mathsf{H}' \cdot \mathsf{H}, \mathsf{H}'\langle\!\langle \mathbb{M} \rangle\!\rangle \cdot \varrho', \omega \ \cdot\ \omega'(\langle\!\langle \mathbb{M} \rangle\!\rangle^\bullet \cdot \mathsf{H}^{(\Gamma^\star)})\big).$$

Moreover we define for every $\boldsymbol{\mathcal{C}}$-varietor Σ the **identity monadic transducer** by

$$\mathbb{ID}_\Sigma = (\mathsf{Id}, u_\Sigma, id_{id_{\Sigma^\star}}) : \Sigma \leftarrow \Sigma. \qquad \diamond$$

For the next theorem we will have to calculate with $(\,\centerdot\,)^\bullet$:

6.3.2.4 Lemma. Let $\boldsymbol{\mathcal{C}} \xleftarrow{\mathsf{F}} \boldsymbol{\mathcal{D}} \xleftarrow{\mathsf{G}} \boldsymbol{\mathcal{E}}$, $\mathsf{F}B \xleftarrow{f}_{\boldsymbol{\mathcal{C}}} A$, $\mathsf{G}C \xleftarrow{g}_{\boldsymbol{\mathcal{D}}} B$, and $X \xleftarrow{h}_{\boldsymbol{\mathcal{D}}} Y$. Then:

(i) $(\mathsf{F}h)^\bullet \cdot \mathsf{F}^{(X)} = \mathsf{F}^{(Y)} \cdot h^\bullet$,

(ii) $\mathsf{F}^{(\mathsf{G}C)} \cdot \mathsf{G}^{(C)} = (\mathsf{F} \cdot \mathsf{G})^{(C)}$,

(iii) $(\mathsf{F}g \cdot f)^\bullet \cdot (\mathsf{F} \cdot \mathsf{G})^{(C)} = f^\bullet \cdot \mathsf{F}^{(B)} \cdot g^\bullet \cdot \mathsf{G}^{(C)}$,

(iv) $(f^\bullet \cdot \mathsf{F}^{(B)}) id_B = f \quad$ where $\quad id_B \in \mathrm{Ob}\, \boldsymbol{\mathcal{D}}^{\underline{B}}$.

Proof. We use Definition 6.3.2.1 and Definition 6.3.2.2:

(i) The left and the right hand side of (i) are concrete functors, thus (with Lemma 4.2.1.5) it suffices to show that (i) holds for every object $\varphi \in \mathrm{Ob}\, \boldsymbol{\mathcal{D}}^{\underline{A}}$:

$$\big((\mathsf{F}h)^\bullet \cdot \mathsf{F}^{(X)}\big)\varphi = \mathsf{F}\varphi \cdot \mathsf{F}h = \mathsf{F}(\varphi \cdot h) = (\mathsf{F}^{(Y)} \cdot h^\bullet)\varphi.$$

(ii) Immediately from the definition.

(iii) With (i) and (ii): $(\mathsf{F}g \cdot f)^\bullet \cdot (\mathsf{F} \cdot \mathsf{G})^{(C)} = f^\bullet \cdot (\mathsf{F}g)^\bullet \cdot \mathsf{F}^{(\mathsf{G}C)} \cdot \mathsf{G}^{(C)} = f^\bullet \cdot \mathsf{F}^{(B)} \cdot g^\bullet \cdot \mathsf{G}^{(C)}$.

(iv) $(f^\bullet \cdot \mathsf{F}^{(B)}) id_B = f^\bullet(\mathsf{F}id_B) = f^\bullet id_{\mathsf{F}B} = id_{\mathsf{F}B} \cdot f = f \quad$ where id_B is considered an *object*. ∎

The following is the main theorem in Chapter 5:

6.3.2.5 Theorem (monadic fusion). Let $\mathbb{M} = (\mathsf{H}, \varrho, \omega) : \Gamma \leftarrow \Delta$ and $\mathbb{M}' = (\mathsf{H}', \varrho', \omega') : \Delta \leftarrow \Sigma$ be monadic transducers on $\boldsymbol{\mathcal{C}}$. Then the following holds:

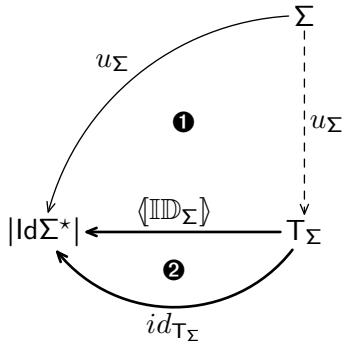
(i) $\langle\!|\mathbb{ID}|\!\rangle = id$, and

(ii) $\langle\!|\mathbb{M} \cdot \mathbb{M}'|\!\rangle = \mathsf{H}'\langle\!|\mathbb{M}|\!\rangle \cdot \langle\!|\mathbb{M}'|\!\rangle$.

(iii) The monadic transducers on $\boldsymbol{\mathcal{C}}$ are the morphisms of a category $\mathbf{MT}\,\boldsymbol{\mathcal{C}}$ where composition is fusion and the objects are all $\boldsymbol{\mathcal{C}}$-varietors.

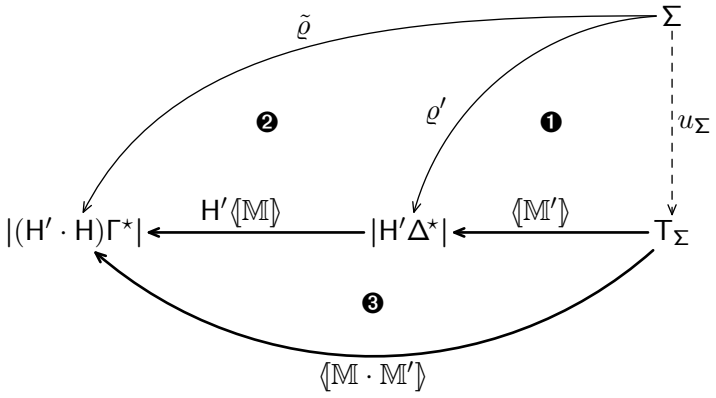
(iv) The semantics $[\![\,\mathbf{.}\,]\!]$ is a functor: $[\![\,\mathbf{.}\,]\!] : \boldsymbol{\mathcal{C}} \leftarrow \mathbf{MT}\,\boldsymbol{\mathcal{C}}$.

Proof. (i) Consider the following diagram:



The outside triangle around ❶❷ commutes trivially and ❶ commutes by definition of $\langle\!|\,\mathbf{.}\,|\!\rangle$ (Definition 6.3.1.2). Thus ❷ also commutes, because u_Σ is universal.

(ii) Let $\tilde{\varrho} = \mathsf{H}'\langle\!|\mathbb{M}|\!\rangle \cdot \varrho'$.



The outside triangle around ❶❷❸ and the triangle ❶ commute by definition (Definition 6.3.1.2). Obviously ❷ commutes by definition of $\tilde{\varrho}$. Thus ❸ also commutes, because u_Σ is universal.

(iii) We define the category $\mathbf{MT}\,\mathcal{C}$ by

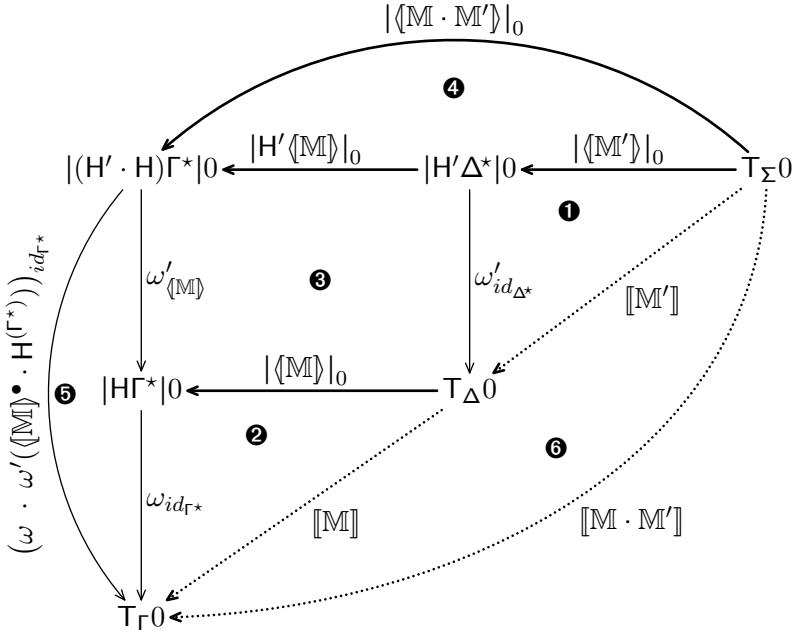
$$\begin{aligned}\mathrm{Ob}(\mathbf{MT}\,\mathcal{C}) &= \big\{\Sigma \;\big|\; \Sigma : \mathcal{C} \leftarrow \mathcal{C} \text{ varietor}\big\}\\ \mathbf{MT}\,\mathcal{C}(\Delta,\Sigma) &= \big\{\mathbb{M} \;\big|\; \mathbb{M} : \Delta \leftarrow \Sigma \text{ monadic transducer on } \mathcal{C}\big\}\end{aligned}$$

where the identity for every $\Sigma \in \mathrm{Ob}(\mathbf{MT}\,\mathcal{C})$ is the monadic transducer $\mathbb{ID}_\Sigma$ and composition is fusion. That the identities are neutral elements w.r.t. fusion is obvious by definition. It remains to show that fusion is associative: Let $\mathbb{M}'' = (\mathsf{H}'', \varrho'', \omega'') : \Sigma \leftarrow \Theta$ be a monadic transducer on $\mathcal{C}$. Then:

$$\begin{aligned}&(\mathbb{M} \cdot \mathbb{M}') \cdot \mathbb{M}''\\ =\;& \{ \text{ two times Definition 6.3.2.3 } \}\\ &\Big(\mathsf{H}'' \cdot \mathsf{H}' \cdot \mathsf{H},\quad \mathsf{H}''\langle\!|\mathbb{M} \cdot \mathbb{M}'|\!\rangle \cdot \varrho'',\\ &\qquad \omega \cdot \omega'(\langle\!|\mathbb{M}|\!\rangle^\bullet \cdot \mathsf{H}^{(\Gamma^\star)}) \cdot \omega''\big(\langle\!|\mathbb{M} \cdot \mathbb{M}'|\!\rangle^\bullet \cdot (\mathsf{H}' \cdot \mathsf{H})^{(\Gamma^\star)}\big)\Big)\\ =\;& \{ \text{ (ii) } \}\\ &\Big(\mathsf{H}'' \cdot \mathsf{H}' \cdot \mathsf{H},\quad \mathsf{H}''(\mathsf{H}'\langle\!|\mathbb{M}|\!\rangle \cdot \langle\!|\mathbb{M}'|\!\rangle) \cdot \varrho'',\\ &\qquad \omega \cdot \omega'(\langle\!|\mathbb{M}|\!\rangle^\bullet \cdot \mathsf{H}^{(\Gamma^\star)}) \cdot \omega''\big((\mathsf{H}'\langle\!|\mathbb{M}|\!\rangle \cdot \langle\!|\mathbb{M}'|\!\rangle)^\bullet \cdot (\mathsf{H}' \cdot \mathsf{H})^{(\Gamma^\star)}\big)\Big)\\ =\;& \{ \text{ Lemma 6.3.2.4 (iii) } \}\\ &\big(\mathsf{H}'' \cdot \mathsf{H}' \cdot \mathsf{H},\quad \mathsf{H}''(\mathsf{H}'\langle\!|\mathbb{M}|\!\rangle \cdot \langle\!|\mathbb{M}'|\!\rangle) \cdot \varrho'',\\ &\qquad \omega \cdot \omega'(\langle\!|\mathbb{M}|\!\rangle^\bullet \cdot \mathsf{H}^{(\Gamma^\star)}) \cdot \omega''(\langle\!|\mathbb{M}'|\!\rangle^\bullet \cdot \mathsf{H}'^{(\Delta^\star)} \cdot \langle\!|\mathbb{M}|\!\rangle^\bullet \cdot \mathsf{H}^{(\Gamma^\star)})\big)\end{aligned}$$

$$
\begin{aligned}
&= \quad \{ \text{ functor } \} \\
&\quad \Big(\mathsf{H}'' \cdot \mathsf{H}' \cdot \mathsf{H}, \quad (\mathsf{H}'' \cdot \mathsf{H}')\langle\!\langle \mathbb{M} \rangle\!\rangle \cdot \mathsf{H}''\langle\!\langle \mathbb{M}' \rangle\!\rangle \cdot \varrho'', \\
&\qquad\qquad \omega \cdot \big(\omega' \cdot \omega''(\langle\!\langle \mathbb{M}' \rangle\!\rangle^{\bullet} \cdot \mathsf{H}'^{(\Delta^\star)}) \big)(\langle\!\langle \mathbb{M} \rangle\!\rangle^{\bullet} \cdot \mathsf{H}^{(\Gamma^\star)}) \Big) \\
&= \quad \{ \text{ Definition 6.3.2.3 } \} \\
&\quad \mathbb{M} \cdot \big(\mathsf{H}'' \cdot \mathsf{H}', \ \mathsf{H}''\langle\!\langle \mathbb{M}' \rangle\!\rangle \cdot \varrho'', \ \omega' \cdot \omega''(\langle\!\langle \mathbb{M}' \rangle\!\rangle^{\bullet} \cdot \mathsf{H}'^{(\Delta^\star)}) \big) \\
&= \quad \{ \text{ Definition 6.3.2.3 } \} \\
&\quad \mathbb{M} \cdot (\mathbb{M}' \cdot \mathbb{M}'')
\end{aligned}
$$

(iv) With Definition 6.3.1.3 and (i) we calculate $[\![\mathbb{ID}_{\mathbf{\Sigma}}]\!] = id_{id_{\mathbf{\Sigma}^\star}} \cdot |\langle\!\langle \mathbb{ID} \rangle\!\rangle|_0 = id_{\mathsf{T}_{\mathbf{\Sigma}}0}$. Consider the diagram:



The triangles ❶ and ❷ commute by definition and the square ❸ because $\omega' : |\centerdot|_0 \cdot |\centerdot|^{\underline{\Delta^\star}} \dot{\leftarrow} |\centerdot|_0 \cdot \mathsf{H}' \cdot |\centerdot|^{\underline{\Delta^\star}}$ is natural. The triangle ❹ is just an instance of (ii). The triangle ❺ commutes due to Lemma 6.3.2.4 (iv) and Definition and Lemma 4.1.3.6 (ii). The outside triangle around ❶-❻ commutes by definition. Thus ❻ also commutes. Altogether we have that $[\![\centerdot]\!] : \mathcal{C} \leftarrow \mathbf{MT}\,\mathcal{C}$ is a functor. ∎

6.3.3 Monadic transducer homomorphisms

In order to class the results of fusions in Subsection 6.5.2 we will have to compare monadic transducers. We have two obvious notions of equivalence: Monadic transducers

$\mathbb{M}$ and $\mathbb{M}'$ may be syntacticly equivalent ($\mathbb{M} = \mathbb{M}'$) or they may be semanticly equivalent ($[\![\mathbb{M}]\!] = [\![\mathbb{M}']\!]$). Artlessly, the former implies the latter.

It will become obvious (in Theorem 6.5.2.11) that a more subtle relation between monadic transducers is of use:

6.3.3.1 Definition (monadic transducer homomorphism). Let $\mathbb{M} = (\mathsf{H}, \varrho, \omega)$, $\mathbb{M}' = (\mathsf{H}', \varrho', \omega') : \Delta \leftarrow \Sigma$ be monadic transducers on $\boldsymbol{\mathcal{C}}$. A natural transformation $\tau : \mathsf{H} \dot{\leftarrow} \mathsf{H}'$ such that

$$\omega \cdot |\,\centerdot\,|_0 \tau |\,\centerdot\,|^{\underline{\Delta^\star}} = \omega' \quad \text{and} \quad \varrho = \tau_{\Delta^\star} \cdot \varrho'$$

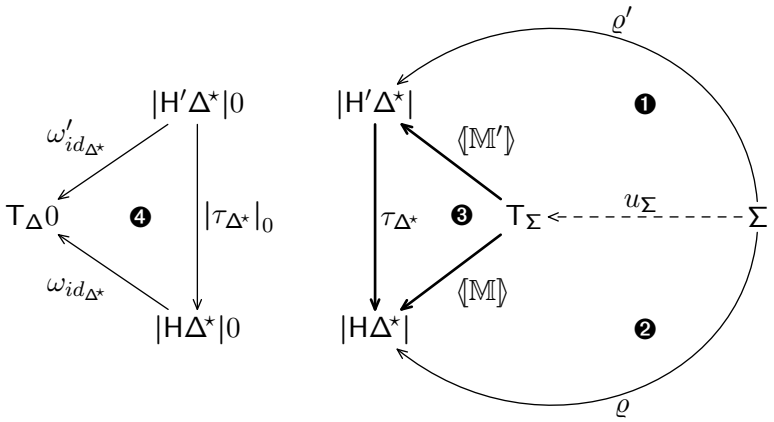
holds is called a **monadic transducer homomorphism** to $\mathbb{M}$ from $\mathbb{M}'$ and we write it

$$\tau : \mathbb{M} \leftarrow \mathbb{M}'. \qquad \diamond$$

Obviously, 'being homomorphic' is a preorder on the class of monadic transducers, moreover it implies semantic equivalence as demonstrated in the following:

6.3.3.2 Theorem (monadic transducer homomorphisms preserve semantics). Let $\mathbb{M} = (\mathsf{H}, \varrho, \omega)$, $\mathbb{M}' = (\mathsf{H}', \varrho', \omega') : \Delta \leftarrow \Sigma$ be monadic transducers on $\boldsymbol{\mathcal{C}}$. If there exists a monadic transducer homomorphism $\tau : \mathbb{M} \leftarrow \mathbb{M}'$ then $[\![\mathbb{M}]\!] = [\![\mathbb{M}']\!]$.

Proof. Consider the following diagrams:



The triangles ❶ and ❷ commute by definition and the triangle around ❶❷❸ commutes according to the precondition. Thus the square around ❷❸ also commutes and since u_Σ is universal ❸ commutes. The triangle ❹ commutes according to the precondition and finally we calculate: $[\![\mathbb{M}]\!] = \omega_{id_{\Delta^\star}} \cdot |\langle\!|\mathbb{M}|\!\rangle|_0 = \omega_{id_{\Delta^\star}} \cdot |\tau_{\Delta^\star} \cdot \langle\!|\mathbb{M}'|\!\rangle|_0 = \omega_{id_{\Delta^\star}} \cdot |\tau_{\Delta^\star}|_0 \cdot |\langle\!|\mathbb{M}'|\!\rangle|_0 = \omega'_{id_{\Delta^\star}} \cdot |\langle\!|\mathbb{M}'|\!\rangle|_0 = [\![\mathbb{M}']\!]$. ∎

In the next section we will characterize classes of tree transducers by patterns of monadic transducers. The following corollary to Theorem 6.3.3.2 allows us to do calculations with these patterns up to isomorphism.

6.3.3.3 Corollary (isomorphic patterns imply semantic equivalence). Let $\mathbb{M} = (\mathsf{H}, \varrho, \omega) : \Delta \leftarrow \Sigma$ be a monadic transducer on $\boldsymbol{\mathcal{C}}$ and $\mathsf{H}' \cong \mathsf{H}$. Then there exists a monadic transducer $\mathbb{M}' : \Delta \leftarrow \Sigma$ on $\boldsymbol{\mathcal{C}}$ such that H' is the pattern of $\mathbb{M}'$ and $[\![\mathbb{M}]\!] = [\![\mathbb{M}']\!]$.

Proof. There exists a natural isomorphism $\alpha : \mathsf{H} \dot{\leftarrow} \mathsf{H}'$. We define

$$\varrho' = \alpha^{-1}_{\Delta^\star} \cdot \varrho \quad \text{and} \quad \omega' = \omega \cdot |\centerdot|_0 \alpha |\centerdot|^{\underline{\Delta^\star}}.$$

Then $\mathbb{M}' = (\mathsf{H}', \varrho', \omega') : \Delta \leftarrow \Sigma$ is a monadic transducer on $\boldsymbol{\mathcal{C}}$ such that $\alpha : \mathbb{M} \leftarrow \mathbb{M}'$ (Definition 6.3.3.1) and with Theorem 6.3.3.2 follows $[\![\mathbb{M}]\!] = [\![\mathbb{M}']\!]$. ∎

6.3.4 Algebraic transducers versus Monadic transducers

Obviously, there are some similarities between algebraic and monadic transducers: In both cases we use the language of category theory to define them and in both cases we use a universal property to define the denotational semantics and to prove the respective fusion theorem. However, the aim of both approaches is completely different: The algebraic transducer is used to bridge the gap between syntactic composition of top-down tree transducers and short cut fusion. The monadic transducer is used to directly fuse classes of tree transducers. A main difference is the way in which the tree transducer is being transformed into an algebraic or monadic transducer, respectively. To derive an algebraic transducer from a top-down tree transducer, a transformation of the equations of the top-down tree transducer is necessary using pairings and copairings (Lemma 5.4.2.4). The result is a concrete functor which encodes the equations (Lemma 5.4.2.9). On the other hand side, the transformation of a tree transducer into a monadic transducer is (in principle) quite simple: The rule of the tree transducer can be transformed into the rule of a monadic transducer via the two simple bijections $\hat{\centerdot}$ (Proposition 6.1.3.1) and $(\centerdot)^\sharp$ (Chapter 6 **Step 1** and Section 6.4). The result is a natural transformation which encodes the equations. The difficult part is to identify a monad such that its carrier is the codomain of that rule (Section 6.4). Another important difference is that in one case the generalized semantics is an algebra morphism (Subsection 5.3.3) and in the other case it is a monad morphism (Definition 6.3.1.2). It is easy to see that a monad morphism from a free monad is a *compositional* (or *syntax directed*) function. Since the generalized semantics of a monadic transducer is the *unique* monad morphism from a free monad, and since any reasonable semantics of a tree transducer is compositional, it is obvious that the two coincide. Thus no more effort is needed to verify that a tree transducer and the according monadic transducer are semanticly equivalent. The latter is by no means obvious in the case of algebraic transducers, where we had to define the *relation* (Definition 5.4.2.5) to prove semantic equivalence (Theorem 5.4.2.7). Finally, the theory of monadic transducers is more elegant and on a higher level of abstraction then the theory of algebraic transducers. We believe that every kind of tree transducer can be described as a monadic transducer, whereas we only managed to describe *top-down* tree transducers as algebraic transducers.

6.4 Tree transducers as monadic transducers

In this section we will model tree transducers by monadic transducers. Moreover we will see that syntactic classes of tree transducers can be characterized by the patterns of monadic transducers.

We have already seen, that the *rule* of a tree transducer can be written as a natural transformation (Proposition 6.1.3.1). Now we will see how to transform this natural transformation into the form used in a monadic transducer (Definition 6.3.1.1).

This corresponds to **Step 1** from the motivation in the beginning of Chapter 6.

Notice, that we will get semantic equivalence for free in the following subsections, because the generalized semantics of a monadic transducer is defined to be the *unique compositional* extension of its rule (Definition 6.3.1.2). As we will see in detail shortly, the rule of a tree transducer (Proposition 6.1.3.1) can be transformed into the rule of a monadic transducer just by applying the bijection $(\,\cdot\,)^\sharp$ which is the adjungate of some adjunction. We will investigate in the following, how to find this adjunction.

6.4.1 Homomorphism tree transducers as monadic transducers

We start with the easiest case: the homomorphism tree transducer (Definition 6.1.2.2). The rule of a homomorphism tree transducer has the form

$$\mathsf{T}_\Delta X \xleftarrow{r} \Sigma X \cap \mathsf{T}_\Sigma^{lhs} X.$$

According to Proposition 6.1.3.1 this can be equivalently described by a natural transformation

$$|\mathsf{Id}\,\Delta^\star| = \mathsf{T}_\Delta \xleftarrow{\varrho = \hat{r}} \Sigma.$$

This is already the desired rule of a monadic transducer on $\boldsymbol{Set}_{\aleph_0}$ where the *pattern* is the identity functor Id. Since $\hat{\cdot}$ is a bijection, we can reverse the above transformation. Thus we have the following:

6.4.1.1 Proposition. The homomorphism tree transducers are equivalent[4] to the monadic transducers

$$\mathbb{M} = (\mathsf{Id}, \varrho, id) : \Delta \leftarrow \Sigma$$

on $\boldsymbol{Set}_{\aleph_0}$ where Σ and Δ are bicartesian. ◇

6.4.2 Top-down tree transducers as monadic transducers

The rule of a top-down tree transducer (Definition 6.1.2.2) has the form

$$\mathsf{T}_\Delta(\mathsf{Q}X) \xleftarrow{r} \mathsf{Q}(\Sigma\, X) \cap \mathsf{T}^{lhs}_{\mathsf{Q}+\Sigma} X$$

[4] *I.e.* syntacticly and semanticly equivalent via the bijections described in Proposition 6.1.3.1 and Proposition 4.3.4.4).

where Q is cocartesian. According to Proposition 6.1.3.1 this can be equivalently described by a natural transformation

$$\mathsf{T}_\Delta \cdot \mathsf{Q} \xleftarrow{\hat{r}} \mathsf{Q} \cdot \Sigma.$$

This rule can be understood as a definition for a couple of functions (*e.g. even* and *odd* in Example 6.1.2.3). Alternatively we could define just *one* function mapping onto tuples (*e.g.* $f\ x\ =\ (even\ x,\ odd\ x)$). Let us describe the tupling by U (*i.e.* $\mathsf{U}A$ is the set of all Q-tuples with elements in A). Then the new rule would have the form $\mathsf{U} \cdot \mathsf{T}_\Delta \cdot \mathsf{Q} \leftarrow \Sigma$. The latter transformation corresponds to **Step 1** from the motivation in the beginning of Chapter 6. Before we continue, let us repeat the last step in a more formal way:

6.4.2.1 Lemma. Let $\mathcal{C}$ be a bicartesian category and $\mathsf{Q} : \mathcal{C} \leftarrow \mathcal{C}$ a cocartesian functor. Then Q is left adjoint.

Proof. Since Q is cocartesian, there exists an $\ell \in \mathbb{N}_0$ such that $\mathsf{Q} \cong \coprod^\ell$. We set $\mathsf{U} = \prod_\ell$ and denote the projections and injections by $\pi_i : \mathsf{Id} \leftarrow \mathsf{U}$ and $\iota_i : \mathsf{Q} \leftarrow \mathsf{Id}$, respectively. We claim that

$$(\eta, \varepsilon) : \mathsf{Q} \dashv \mathsf{U} : \mathcal{C} \leftarrow \mathcal{C}$$

where $\eta = \langle \iota_i \rangle_{i=1}^\ell$ and $\varepsilon = [\pi_i]_{i=1}^\ell$ are the unique mediating morphisms with $\pi_i \cdot \eta = \iota_i$ and $\varepsilon \cdot \iota_i = \pi_i$, respectively, according to the universal property of the (co)product. The axioms of the adjunction (Definition 4.3.4.1) can now be verified by straight forward calculations using the universal property of the (co)product. ∎

We continue with the rule of a top-down tree transducer in the form

$$\mathsf{T}_\Delta \cdot \mathsf{Q} \xleftarrow{\hat{r}} \mathsf{Q} \cdot \Sigma.$$

Since Q is cocartesian, Lemma 6.4.2.1 tells us that Q has a right adjoint U. Using Lemma 6.2.3.6 and Definition 6.2.3.4 we can write the rule equivalently as

$$|\overline{\mathsf{Q}}\Delta^\star| = \mathsf{U} \cdot \mathsf{T}_\Delta \cdot \mathsf{Q} \xleftarrow{\varrho = \hat{r}^\sharp} \Sigma$$

where $(\,\centerdot\,)^\sharp$ refers to $(\mathsf{Q} \longleftarrow\!\circ\, \mathsf{Id}) \dashv (\mathsf{U} \longleftarrow\!\circ\, \mathsf{Id})$. This is the desired rule of a monadic transducer with *pattern* $\overline{\mathsf{Q}}$.

The inclusion $\iota : \mathsf{Q} \leftarrow \mathsf{Id}$ of the initial state can be equivalently described by the projection $\pi = \iota^\flat : \mathsf{Id} \leftarrow \mathsf{U}$ where $(\,\centerdot\,)^\flat$ refers to $(\mathsf{Id} \longleftarrow\!\circ\, \mathsf{U}) \dashv (\mathsf{Id} \longleftarrow\!\circ\, \mathsf{Q})$. Using $\mathsf{Q}\emptyset = \emptyset$ we can define the *observation function* of the monadic transducer by

$$\forall\, h : (\mathsf{T}, \eta, \mu) \leftarrow \Delta^\star. \quad \omega_h = \pi_{\mathsf{T}\emptyset}$$

i.e. $\omega = \pi(|\,\centerdot\,|_\emptyset \cdot |\,\centerdot\,|^{\overline{\Delta^\star}})$.

Since $\hat{\centerdot}$, $(\,\centerdot\,)^\sharp$, and $(\,\centerdot\,)^\flat$ are bijections, we can reverse the above transformations. Thus we have the following:

6.4.2.2 Proposition. The top-down tree transducers are equivalent[5] to the monadic transducers

$$\mathbb{M} = (\overline{\mathsf{Q}}, \varrho, \omega) : \Delta \leftarrow \Sigma$$

on $\boldsymbol{Set}_{\aleph_0}$ where Q is cocartesian and Σ and Δ are bicartesian. ◇

6.4.2.3 Example. Let us illustrate the monadic operations of the monad $\overline{\mathsf{Q}}\Delta^\star$ which models the computation of a top-down tree transducer according to Proposition 6.4.2.2 (see also Definition 6.3.1.1 and Definition 6.3.1.2). It is helpful to have a look at Example 4.4.1.4 before. In diagrams we draw triangles for arbitrary terms and for every variable x we draw $\textcircled{x}$ for the term $'x$. The unit is simple:

$$X \xrightarrow{\quad\eta_X\quad} \mathsf{U}(\mathsf{T}_\Delta(\mathsf{Q}Y))$$

$$x \xmapsto{\quad\eta_X\quad} \Big(\begin{array}{c} q \\ | \\ \textcircled{x} \end{array} \Big)_{q\in Q}$$

◇

The multiplication is like term-substitution but tags are used to project from tuples. We show the according Kleisli-† in Figure 6.1.

6.4.3 Simple basic macro tree transducers as monadic transducers

Writing a macro tree transducer as a monadic transducer is a little more involved: We start with a simple case: the simple basic macro tree transducer.

Using Proposition 6.1.3.1 we can write the rule of a simple basic macro tree transducer (Definition 6.1.2.5) in the form

$$\mathsf{T}_\Delta\big(Y + \mathsf{A}_I(\mathsf{Q}X)\,Y\big) \xleftarrow{\big((\hat{r})_X\big)_Y} \mathsf{A}_I\big(\mathsf{Q}(\Sigma X)\big)\,Y$$

where $\big((\hat{r})_X\big)_Y$ is natural in X and Y, Q is cocartesian, and I is finite. Abstraction from Y yields

$$\mathsf{T}_\Delta \cdot \big(\mathsf{Id} + \mathsf{A}_I(\mathsf{Q}X)\big) \xleftarrow{(\hat{r})_X} \mathsf{A}_I\big(\mathsf{Q}(\Sigma X)\big)$$

and then abstracting from X gives us

$$(\mathsf{T}_\Delta \leftarrow\!\circ\, \mathsf{Id}) \cdot (\mathsf{Id} + \,\centerdot\,) \cdot \mathsf{A}_I \cdot \mathsf{Q} \xleftarrow{\hat{r}} \mathsf{A}_I \cdot \mathsf{Q} \cdot \Sigma$$

[5] *I.e.* syntacticly and semanticly equivalent via the bijections described in Proposition 6.1.3.1 and Proposition 4.3.4.4).

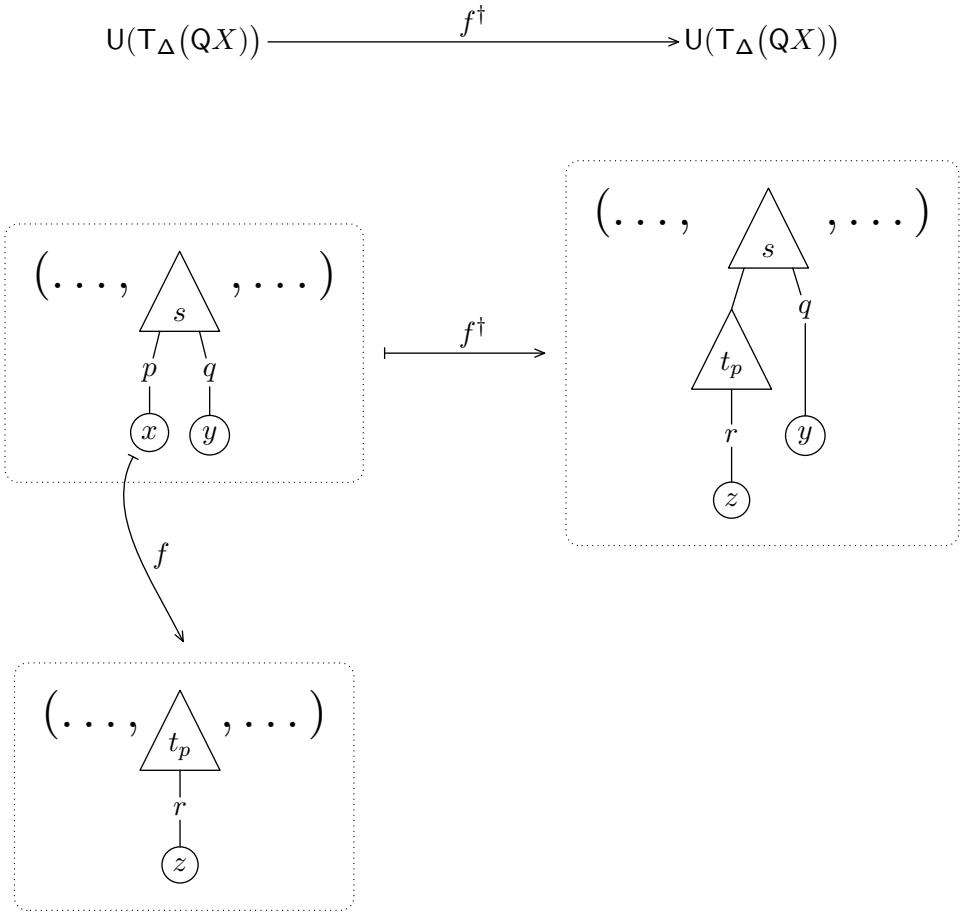


Figure 6.1: Kleisli-† of the monad $\overline{\mathsf{Q}}\Delta^\star$

where we used the bifunctor $\leftarrow\!\circ$ from Definition 6.2.3.2. Now we need an adjunction to continue in a similar way as in Subsection 6.4.2:

6.4.3.1 Lemma. Let $\boldsymbol{\mathcal{C}}$ be a category which has function spaces (Definition 4.2.1.9). Then

$$\mathsf{A}_I \dashv \Lambda_I : \mathbf{End}\,\boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$$

where $\Lambda_I : \boldsymbol{\mathcal{C}} \leftarrow \mathbf{End}\,\boldsymbol{\mathcal{C}}$ is the functor from Definition 6.2.4.3.

Proof. Let $|\mathbf{.}| : \boldsymbol{\mathcal{S}et} \leftarrow \boldsymbol{\mathcal{C}}$ be the faithful functor according to Definition 4.2.1.9. We claim that $(\eta, \varepsilon) : \mathsf{A}_I \dashv \Lambda_I$ where $\eta_X : I^I \times X \leftarrow X$ and $(\varepsilon_\mathsf{T})_Y : \mathsf{T}Y \leftarrow Y^I \times \mathsf{T}I$ are defined by $|\eta_X|x = (id_I, x)$ and $|(\varepsilon_\mathsf{T})_Y|(f,t) = |\mathsf{T}f|t$, respectively. Since $|\mathbf{.}|$ is faithful, η and ε are well defined. That η and ε are indeed natural transformation can be verified by straight forward calculations. It remains to show the axioms of an

adjunction (Definition 4.3.4.1):

$$\begin{aligned}
&|(\Lambda_I\varepsilon \cdot \eta\Lambda_I)_{\mathsf{T}}|t \\
=\ & \{ \text{ Definition and Lemma 4.1.3.6 and Definition 6.2.4.3 } \} \\
&|(\varepsilon_{\mathsf{T}})_I|\big(|\eta_{\mathsf{T}I}|t\big) \\
=\ & \{ \text{ definition of } \eta \ \} \\
&|(\varepsilon_{\mathsf{T}})_I|(id_I, t) \\
=\ & \{ \text{ definition of } \varepsilon \text{ and Definition 6.2.4.3 } \} \\
&|\mathsf{T}id_I|t = |id_{\mathsf{T}I}|t = |(id_{\Lambda_I})_{\mathsf{T}}|t.
\end{aligned}$$

Since $|\centerdot|$ is faithful and the above is true for arbitrary $\mathsf{T} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$ and $t \in |\mathsf{T}I|$ we get $\Lambda_I\varepsilon \cdot \eta\Lambda_I = id_{\Lambda_I}$. Analogously we can show that $\varepsilon\mathsf{A}_I \cdot \mathsf{A}_I\eta = id_{\mathsf{A}_I}$ holds. ∎

6.4.3.2 Note. Let us illustrate the above theorem in the category $\boldsymbol{\mathcal{S}et}$ for $I = \{1, \ldots, k\}$. Notice that in this case where I is finite the proof of Lemma 6.4.3.1 also works in $\boldsymbol{\mathcal{S}et}_{\aleph_0}$.

We use $\mathsf{A}_I X\, Y = \big\{x\, y_1 \cdots y_k \ \big|\ x \in X \ \wedge\ y_i \in Y\big\} \cong Y^I \times X$ and $\Lambda_I\mathsf{T} = \big\{\lambda 1 \cdots k.\, t \ \big|\ t \in \mathsf{T}I\big\} \cong \mathsf{T}I$. Then we have an adjunction:

$$(\eta, \varepsilon) : \mathsf{A}_I \dashv \Lambda_I : \mathbf{End}\,\boldsymbol{\mathcal{S}et} \leftarrow \boldsymbol{\mathcal{S}et}$$

where $\eta_X x = (id_I, x)$ and $(\varepsilon_{\mathsf{T}})_Y(f, t) = \mathsf{T}\, f\, t$. Moreover, the function η and ε describe η-conversion and β-reduction, respectively:

$$\Lambda_I(\mathsf{A}_I X) = \big\{\lambda 1 \cdots k.\, t \ \big|\ t \in \mathsf{A}_I X I\big\}$$

$$\begin{array}{rccll}
\eta : & \Lambda_I \cdot \mathsf{A}_I & \leftarrow & \mathsf{Id} & \\
\eta_X : \lambda 1 \cdots k.\, x\, 1 \cdots k & \mapsto & x & \eta\text{-conversion}
\end{array}$$

and

$$\mathsf{A}_I(\Lambda_I\mathsf{T})\, Y = \big\{(\lambda 1 \cdots k.\, t)\, y_1 \cdots y_k \ \big|\ t \in \mathsf{T}I \ \wedge\ y_i \in Y\big\}$$

$$\begin{array}{rccll}
\varepsilon : & \mathsf{Id} & \leftarrow & \Lambda_I \cdot \mathsf{A}_I & \\
(\varepsilon_{\mathsf{T}})_Y : [y_i/i]_{i=1}^k t & \mapsto & (\lambda 1 \cdots k.\, t)\, y_1 \cdots y_k & \beta\text{-reduction.}
\end{array}$$

◇

Let us continue with the rule of a simple basic macro tree transducer in the form

$$(\mathsf{T}_\Delta \multimap \mathsf{Id}) \cdot (\mathsf{Id} + \centerdot) \cdot \mathsf{A}_I \cdot \mathsf{Q} \xleftarrow{\hat{r}} \mathsf{A}_I \cdot \mathsf{Q} \cdot \Sigma.$$

With Lemma 6.4.2.1 and Lemma 6.4.3.1 we get the adjunctions $\mathsf{Q} \dashv \mathsf{U}$ and $\mathsf{A}_I \dashv \Lambda_I$, respectively. Now we can use $(\centerdot)^\sharp$ w.r.t. $\mathsf{A}_I \cdot \mathsf{Q} \dashv \mathsf{U} \cdot \Lambda_I$, Definition 6.2.3.4, and Corollary 6.2.4.7 to write the rule as

$$|\overline{\mathsf{A}_I \cdot \mathsf{Q}} \cdot (\mathsf{Id}^+ + \centerdot) \cdot (\centerdot \multimap \mathbb{O})| = \mathsf{U} \cdot \Lambda_I \cdot (\mathsf{T}_\Delta \multimap \mathsf{Id}) \cdot (\mathsf{Id} + \centerdot) \cdot \mathsf{A}_I \cdot \mathsf{Q} \xleftarrow{\hat{r}^\sharp} \Sigma.$$

The inclusion $\iota : \mathsf{Q} \dot{\leftarrow} \mathsf{Id}$ of the initial state can be equivalently described by the projection $\pi = \iota^\flat : \mathsf{Id} \leftarrow \mathsf{U}$ where $(\,\centerdot\,)^\flat$ refers to $(\mathsf{Id} \leftarrow\!\!\circ\, \mathsf{U}) \dashv (\mathsf{Id} \leftarrow\!\!\circ\, \mathsf{Q})$. Using $\mathsf{A}_I(\mathsf{Q}\emptyset) = \underline{\emptyset}$ we can define the *observation function* of the monadic transducer by

$$\forall\, h : (\mathsf{T}, \eta, \mu) \leftarrow \Delta^\star. \quad \omega_h = \pi_{\mathsf{T}\emptyset} \cdot \mathsf{U}\big(\mu_\emptyset \cdot \mathsf{T}(h_\emptyset \cdot e)\big).$$

Thus altogether we have got:

6.4.3.3 Proposition. The simple basic macro tree transducers are equivalent[6] to the monadic transducers

$$\mathbb{M} = \big(\overline{\mathsf{A}_I \cdot \mathsf{Q}} \cdot (\mathsf{Id}^+ + \centerdot) \cdot (\centerdot \leftarrow\!\!\circ\, \mathbb{O}), \varrho, \omega\big) : \Delta \leftarrow \Sigma$$

on $\boldsymbol{\mathcal{S}et}_{\aleph_0}$ where I is a finite set, Q is a cocartesian and Σ and Δ are bicartesian. ◇

6.4.4 Basic macro tree transducers as monadic transducers

The basic macro tree transducer case is just a little more complicated than the simple basic macro tree transducer: The only difference to Proposition 6.4.3.3 is that we have to replace the functor $(\centerdot \leftarrow\!\!\circ\, \mathbb{O})$ by the functor $\langle \leftarrow\!\!\circ \rangle$ defined by $\forall\, h.\ \langle \leftarrow\!\!\circ \rangle h = h \leftarrow\!\!\circ\, h$.

The rule of a basic macro tree transducer can be written in the form

$$\mathsf{T}_\Delta\big(\mathsf{T}_\Delta Y + \mathsf{A}_I(\mathsf{Q}X)(\mathsf{T}_\Delta Y)\big) \xleftarrow{\big((\hat{r})_X\big)_Y} \mathsf{A}_I\big(\mathsf{Q}(\Sigma X)\big)\, Y$$

where we have introduced (without loss of generality) an additional T_Δ. Similar to Subsection 6.4.3 we can abstract from Y and then from X:

$$(\mathsf{T}_\Delta \leftarrow\!\!\circ\, \mathsf{T}_\Delta) \cdot (\mathsf{Id} + \centerdot) \cdot \mathsf{A}_I \cdot \mathsf{Q} \xleftarrow{\hat{r}} \mathsf{A}_I \cdot \mathsf{Q} \cdot \Sigma.$$

The only difference to Subsection 6.4.3 is that we now have an additional T_Δ.

Similar to Subsection 6.4.3 we get:

6.4.4.1 Proposition. The basic macro tree transducers are equivalent[6] to the monadic transducers

$$\mathbb{M} = (\overline{\mathsf{A}_I \cdot \mathsf{Q}} \cdot (\mathsf{Id}^+ + \centerdot) \cdot \langle \leftarrow\!\!\circ \rangle, \varrho, \omega) : \Delta \leftarrow \Sigma$$

on $\boldsymbol{\mathcal{S}et}_{\aleph_0}$ where I is a finite set, Q is a cocartesian and Σ and Δ are bicartesian. ◇

6.4.5 Macro tree transducers as monadic transducers

The rule of a macro tree transducer (Definition 6.1.2.5) can be written in the form

$$\mathsf{T}_{\Delta + \mathsf{A}_I(\mathsf{Q}X)} Y \xleftarrow{\big((\hat{r})_X\big)_Y} \mathsf{A}_I\big(\mathsf{Q}(\Sigma X)\big)\, Y.$$

[6] *I.e.* syntacticly and semanticly equivalent via the bijections described in Proposition 6.1.3.1 and Proposition 4.3.4.4).

Abstraction from Y yields

$$\mathsf{T}_{\Delta+\mathsf{A}_I(\mathsf{Q}X)} \xleftarrow{(\hat{r})_X} \mathsf{A}_I\big(\mathsf{Q}(\Sigma X)\big).$$

Now we need the coproduct of monads: With $\mathsf{T}_\Delta = |\Delta^\star|$ and Lemma 6.2.4.1 we can write the rule as

$$|\Delta^\star + \big(\mathsf{A}_I(\mathsf{Q}X)\big)^\star| \xleftarrow{(\hat{r})_X} \mathsf{A}_I\big(\mathsf{Q}(\Sigma X)\big).$$

Abstracting from X gives us

$$|\,\centerdot\,| \cdot (\Delta^\star + \centerdot) \cdot (\centerdot)^\star \cdot \mathsf{A}_I \cdot \mathsf{Q} \xleftarrow{\hat{r}} \mathsf{A}_I \cdot \mathsf{Q} \cdot \Sigma.$$

We have the adjunctions $\mathsf{Q} \dashv \mathsf{U}$ and $\mathsf{A}_I \dashv \Lambda_I$ as in Subsection 6.4.3 and $(\centerdot)^\star \dashv |\centerdot|$ from Corollary 4.4.4.6. Now we use $(\centerdot)^\sharp$ w.r.t. $(\centerdot)^\star \cdot \mathsf{A}_I \cdot \mathsf{Q} \dashv \mathsf{U} \cdot \Lambda_I \cdot |\centerdot|$, Definition and Lemma 6.2.4.2 (i), and Definition 6.2.3.4 to write the rule as

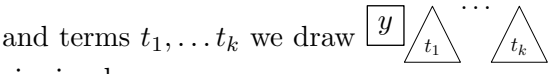
$$\overline{(\centerdot)^\star \cdot \mathsf{A}_I \cdot \mathsf{Q}}\,(\Delta^\star)^+ \xleftarrow{\hat{r}^\sharp} \Sigma.$$

The latter is the *rule* of a monadic transducer with *pattern* $\overline{(\centerdot)^\star \cdot \mathsf{A}_I \cdot \mathsf{Q}} \cdot (\centerdot)^+$. The *observation function* is defined just as in Subsection 6.4.3. Altogether we have:

6.4.5.1 Proposition. The macro tree transducers are equivalent[7] to the monadic transducers

$$\mathbb{M} = \big(\overline{(\centerdot)^\star \cdot \mathsf{A}_I \cdot \mathsf{Q}} \cdot (\centerdot)^+, \varrho, \omega\big) : \Delta \leftarrow \Sigma$$

on $\boldsymbol{Set}_{\aleph_0}$ where I is a finite set, Q is a cocartesian and Σ and Δ are bicartesian. ◇

6.4.5.2 Example. Let us now illustrate the monadic operations of the monad $\overline{\mathsf{A}_I}(\overline{(\centerdot)^\star}(\Delta^\star)^+)$. It is helpful to have a look at Example 4.4.1.4 and Example 6.4.2.3 before. For every context variable y and terms $t_1, \dots t_k$ we draw  for the applicative term $\text{'}y\,t_1 \cdots t_k$. The unit is simple:

$$X \xrightarrow{\eta_X} \Lambda_I \mathsf{T}_{\Delta+\mathsf{A}_I X}$$

$$x \longmapsto \lambda\, 1 \cdots k.\ \boxed{x}\,\textcircled{1}\cdots\textcircled{k}$$

◇

The multiplication is like term-substitution but with an extra β-reduction. We show the according Kleisli-† in Figure 6.2.

[7] *I.e.* syntacticly and semanticly equivalent via the bijections described in Proposition 6.1.3.1 and Proposition 4.3.4.4).

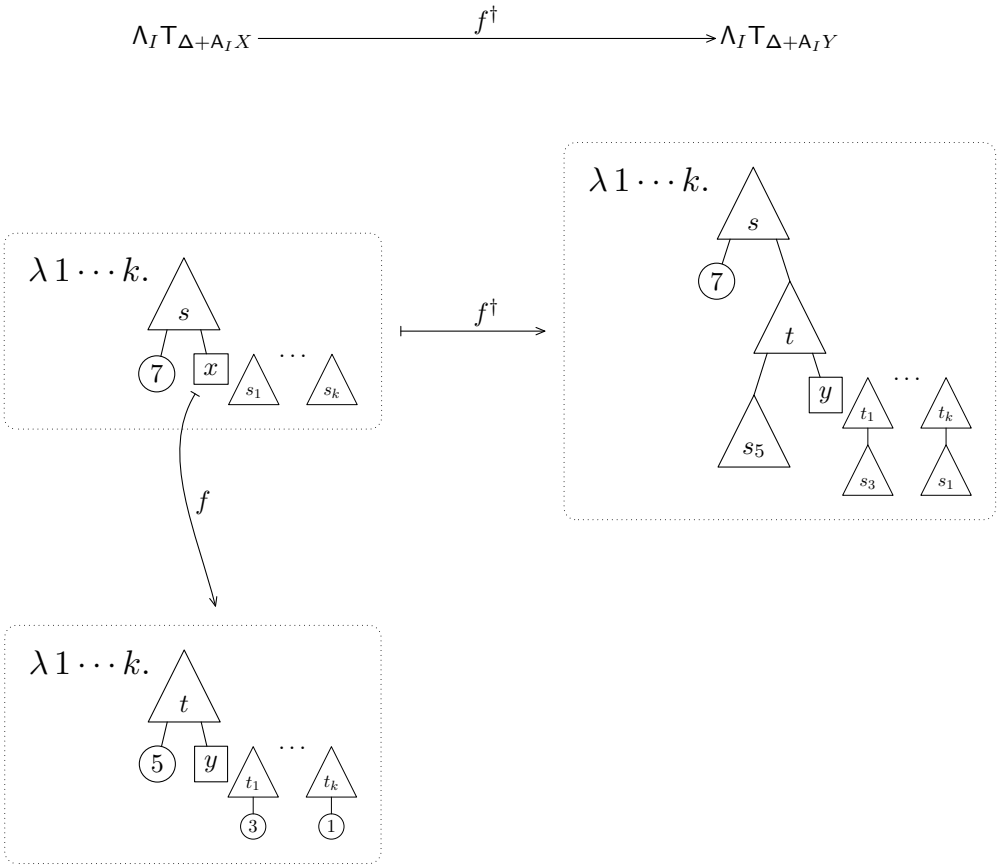


Figure 6.2: Kleisli-† of the monad $\overline{\mathsf{A}_I}(\overline{(\blacksquare^\star)}(\Delta^\star)^+)$

6.5 Fusion of tree transducers

6.5.1 Fusion of particular functional programs

In Section 6.4 we have seen how particular functional programs (*i.e.*some syntactic classes of tree transducers) can be equivalently transformed into monadic transducers. Now we are ready to apply the monadic fusion Theorem 6.3.2.5 to functional programs: Given regular types A, B, and C and functional programs $C \xleftarrow{g} B \xleftarrow{f} A$ we construct a new program $C \xleftarrow{h} A$ such that

$$\mathsf{T}_\Gamma\emptyset \xleftarrow{[\![g]\!]} \mathsf{T}_\Delta\emptyset \xleftarrow{[\![f]\!]} \mathsf{T}_\Sigma\emptyset \qquad [\![h]\!]$$

where the initial term-algebras T_Σ, T_Δ, andT_Γ are supposed to be the semantics of the regular types A, B, and C, respectively. The construction of h is shown in Figure 6.3.

The rule of h is constructed according to Definition 6.3.2.3. The correctness of the fusion transformation w.r.t. the denotational semantics $[\![\,\cdot\,]\!]$ follows from Theorem 6.3.2.5. Notice, that there occurs no Δ in the rule of h (as shown in Figure 6.3), *i.e.* the intermediate data structure B is indeed eliminated.

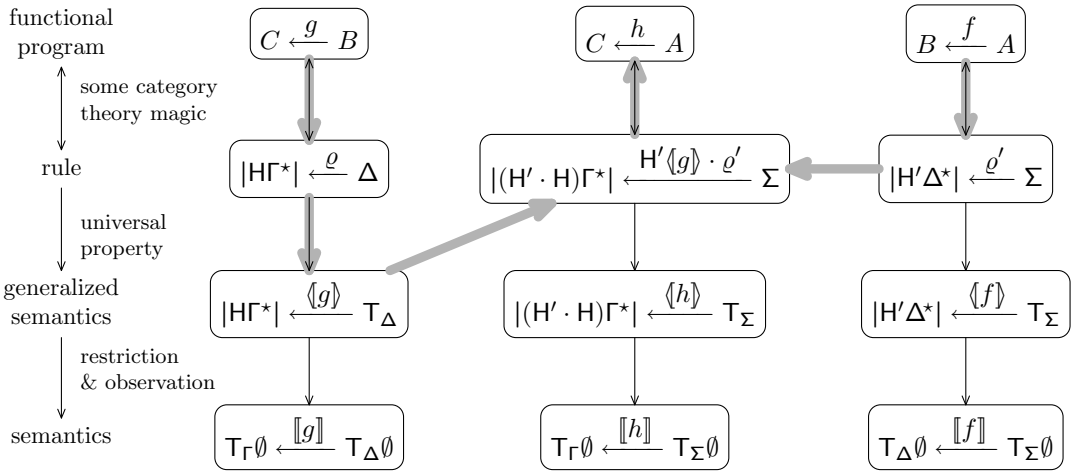


Figure 6.3: Monadic fusion algorithm

6.5.2 Fusion of classes of tree transformations

Now we are ready to use our machinery to harvest tree transducer fusion results: In the last subsection we have characterized some classes of tree transducers by patterns of monadic transducers. We derive fusion results as follows: First we compose the patterns of the consumer and producer to get the pattern of the fusion (Definition 6.3.2.3 and Theorem 6.3.2.5). Then we do calculations (using *e.g.* Lemma 6.2.3.9 or Corollary 6.2.4.6) to transform isomorphicly the pattern such that we are able to recognize the associated class of tree transducers. Here it is worth mentioning that it suffices to calculate the pattern up to isomorphism to derive the fusion up to semantic equivalence (with Corollary 6.3.3.3).

We have seen patterns for the following classes of tree transformations: $HOM \subseteq TOP \subseteq sb\text{-}MAC \subseteq b\text{-}MAC \subseteq MAC$.

For two classes of tree transformations A and B we define the composition $A \cdot B = \{a \cdot b \mid a \in A \wedge b \in B \wedge \operatorname{dom} a = \operatorname{cod} b\}$. Moreover we define the class ID to be the class of all identity tree transformations.

6.5.2.1 Definition (characterization by patterns). Let A be a class of tree transformations and P be a class of (**Mnd** $\mathcal{S}et$)-endofunctors.

(i) We say that A is **characterized** by P if

$$A = \{[\![\mathbb{M}]\!] \mid \mathbb{M} \text{ is a monadic transducer with a pattern from } P\}$$

i.e. A is the class of all functions computable by monadic transducers where the pattern is an endofunctor from the class P.

(ii) We say that A **can be characterized by patterns** if there exists a class P such that A is characterized by P. ◇

6.5.2.2 Lemma (fusion with homomorphism tree transducers). Let A be characterized by patterns. Then $A \cdot HOM \subseteq A$ and $HOM \cdot A \subseteq A$.

Proof. The class HOM is characterized by the single pattern Id (Proposition 6.4.1.1). Obviously, any class of patterns is closed under composition with the identity functor Id. ■

The other inclusion direction also holds:

6.5.2.3 Lemma. Let A be characterized by patterns. Then $A \subseteq A \cdot HOM$ and $A \subseteq HOM \cdot A$.

Proof. Every identity monadic transducer $\mathbb{ID}_\Sigma$ (Definition 6.3.2.3) is a homomorphism tree transducer (Proposition 6.4.1.1). Thus $ID \subseteq HOM$ and then $A \subseteq A \cdot ID \subseteq A \cdot HOM$. ■

The following corollary can be understood in a negative way: It states that a class of tree transducers can only be characterized by patterns if it is closed under composition with homomorphisms tree transducers:

6.5.2.4 Corollary. Let A be characterized by patterns. Then $A \cdot HOM = A = HOM \cdot A$.
Or a little more general: Let $HOM \subseteq B$. Then

(i) $A \cdot B \subseteq A \implies A \cdot B = A$ and

(ii) $B \cdot A \subseteq A \implies B \cdot A = A$.

Proof. Immediately with Lemma 6.5.2.2 and Lemma 6.5.2.3. ■

6.5.2.5 Theorem (fusion of top-down tree transducers [Eng75]**).**

$$TOP_\ell \cdot TOP_k = TOP_{\ell \cdot k}.$$

Proof. According to Proposition 6.4.2.2 the consumer and producer are characterized by the patterns $\overline{\coprod^\ell}$ and $\overline{\coprod^k}$, respectively. With Definition 6.3.2.3 we get the pattern of

the fusion

$$\begin{aligned}
&\overline{\coprod}^{k} \cdot \overline{\coprod}^{\ell} \\
=\ & \{ \text{ Lemma 6.2.3.5 } \} \\
&\overline{\coprod}^{\ell} \cdot \coprod^{k} \\
\cong\ & \{ \text{ coproducts are associative up to isomorphism } \} \\
&\overline{\coprod}^{\ell \cdot k}
\end{aligned}$$

The latter is the pattern of a top-down tree transducers (with $\ell \cdot k$ states) which is semanticly equivalent to the fusion according to Theorem 6.3.2.5 and Corollary 6.3.3.3. Vice versa the pattern of top-down tree transducer with $\ell \cdot k$ states can be factorized into the patterns of two top-down tree transducers with ℓ and k states, respectively. ∎

Thus (with a little additional help from Corollary 6.5.2.4) we reproduced the result from Corollary 3.3.2.2:

6.5.2.6 Corollary. The class of all top-down tree transducers is closed under fusion: $TOP \cdot TOP = TOP$. ◇

6.5.2.7 Theorem (fusion of a macro and a top-down tree transd. [Eng81]**).**

$$\begin{aligned}
\textit{sb-MAC}_\ell \cdot \textit{TOP}_k &= \textit{sb-MAC}_{\ell \cdot k} \\
\textit{b-MAC}_\ell \cdot \textit{TOP}_k &= \textit{b-MAC}_{\ell \cdot k} \\
\textit{MAC}_\ell \cdot \textit{TOP}_k &= \textit{MAC}_{\ell \cdot k}.
\end{aligned}$$

Proof. We can almost copy the proof of Theorem 6.5.2.5: The classes $\textit{b-MAC}_\ell$, $\textit{sb-MAC}_\ell$, and $\textit{MAC}_\ell$ are all characterized by patterns of the form $(\cdots) \cdot \overline{\coprod}^{\ell}$. The class $\textit{TOP}_k$ is characterized by the pattern $\overline{\coprod}^{k}$. Thus the pattern of the fusion is $(\cdots) \cdot \overline{\coprod}^{\ell \cdot k}$, where the $(\cdots)$ part is left untouched. ∎

For a commuted version of Theorem 6.5.2.7 we need to calculate with patterns:

6.5.2.8 Lemma. Let $\mathcal{C}$ be a category which has function spaces (Definition 4.2.1.9), I be a $\mathcal{C}$ object, and $\mathsf{Q} \dashv \mathsf{U} : \mathcal{C} \leftarrow \mathcal{C}$. Then the following holds:

$$\mathsf{A}_{\mathsf{Q}I} \cong (\mathsf{Id} \longleftarrow\!\circ\, \mathsf{U}) \cdot \mathsf{A}_I \quad \text{natural in } I.$$

Proof.

$$
\begin{aligned}
& \mathsf{Q} \dashv \mathsf{U} \\
\Longrightarrow \quad & \{ \text{ Proposition 4.3.4.4 } \} \\
& \boldsymbol{\mathcal{C}}(Y, \mathsf{Q}I) \cong \boldsymbol{\mathcal{C}}(\mathsf{U}Y, I) \quad \text{natural in } I \ \& \ Y \\
\Longrightarrow \quad & \{ \ \boldsymbol{\mathcal{C}} \text{ has function spaces } \} \\
& Y^{\mathsf{Q}I} \cong (\mathsf{U}Y)^I \quad \text{natural in } I \ \& \ Y \\
\Longrightarrow \quad & Y^{\mathsf{Q}I} \times X \cong (\mathsf{U}Y)^I \times X \quad \text{natural in } I, X, \ \& \ Y \\
\Longrightarrow \quad & \{ \text{ Definition 6.1.2.4 } \} \\
& \mathsf{A}_{\mathsf{Q}I} X\, Y \cong \mathsf{A}_I X\, (\mathsf{U}Y) \quad \text{natural in } I, X, \ \& \ Y \\
\Longrightarrow \quad & \mathsf{A}_{\mathsf{Q}I} X \cong (\mathsf{A}_I X) \cdot \mathsf{U} \quad \text{natural in } I \ \& \ X \\
\Longrightarrow \quad & \{ \text{ Definition 6.2.3.2 } \} \\
& \mathsf{A}_{\mathsf{Q}I} \cong (\mathsf{Id} \longleftarrow\!\circ\, \mathsf{U}) \cdot \mathsf{A}_I \quad \text{natural in } I.
\end{aligned}
$$

■

6.5.2.9 Lemma. Let $\boldsymbol{\mathcal{C}}$ be a bicartesian closed category, I be a $\boldsymbol{\mathcal{C}}$-object, and Q a cocartesian $\boldsymbol{\mathcal{C}}$-endofunctor. Then the following holds:

$$\mathsf{A}_I \cdot \mathsf{Q} \cong (\mathsf{Q} \longleftarrow\!\circ\, \mathsf{Id}) \cdot \mathsf{A}_I \quad \text{natural in } I.$$

Proof. Let X and Y be $\boldsymbol{\mathcal{C}}$ objects. The $\boldsymbol{\mathcal{C}}$-endofunctor $(Y^I \times \boldsymbol{.})$ preserves coproducts since it is left adjoint. The functor Q is cocartesian and thus $\mathsf{Q} \cong \coprod^{\ell}$ for some $\ell \in \mathbb{N}_0$. Thus the functors $(Y^I \times \boldsymbol{.})$ and Q commute:

$$
\begin{aligned}
\Longrightarrow \quad & Y^I \times \mathsf{Q}X \cong \mathsf{Q}(Y^I \times X) \quad \text{natural in } I, X \ \& \ Y \\
\Longrightarrow \quad & \{ \text{ Definition 6.1.2.4 } \} \\
& \mathsf{A}_I(\mathsf{Q}X)\, Y \cong \mathsf{Q}(\mathsf{A}_I X\, Y) \quad \text{natural in } I, X, \ \& \ Y \\
\Longrightarrow \quad & \{ \text{ Definition 6.2.3.2 } \} \\
& \mathsf{A}_I(\mathsf{Q}X) \cong (\mathsf{Q} \longleftarrow\!\circ\, \mathsf{Id})(\mathsf{A}_I X) \quad \text{natural in } I \ \& \ X \\
\Longrightarrow \quad & \mathsf{A}_I \cdot \mathsf{Q} \cong (\mathsf{Q} \longleftarrow\!\circ\, \mathsf{Id}) \cdot \mathsf{A}_I \quad \text{natural in } I.
\end{aligned}
$$

■

6.5.2.10 Corollary. Let $\boldsymbol{\mathcal{C}}$ be a bicartesian closed category which has function spaces (Definition 4.2.1.9), I be a $\boldsymbol{\mathcal{C}}$ object, and Q a cocartesian $\boldsymbol{\mathcal{C}}$-endofunctor.Putting together Lemma 6.5.2.8 and Lemma 6.5.2.9 and with a little help from Definition 6.2.3.2, Lemma 6.2.3.6 and Lemma 6.4.3.1 we get:

$$\mathsf{A}_{\mathsf{Q}I} \cdot \mathsf{Q} \cong \text{Ł}\mathsf{Q} \cdot \mathsf{A}_I \quad \text{natural in } I.$$

◇

6.5.2.11 Theorem (fusion of a top-down and a macro tree transd. [EV85]**).**

$$TOP_\ell \cdot MAC_k = MAC_{\ell \cdot k}.$$

Proof. The class TOP_ℓ is characterized by the pattern $\overline{\mathsf{Q}}$ where $\mathsf{Q} = \coprod^\ell$. The class MAC_k is characterized by patterns of the form $\overline{(\centerdot)^\star \cdot \mathsf{A}_I \cdot \mathsf{Q}'} \cdot (\centerdot)^+$ where I is finite and $\mathsf{Q}' = \coprod^k$. Then the pattern of the fusion is

$$\begin{aligned}
&\overline{(\centerdot)^\star \cdot \mathsf{A}_I \cdot \mathsf{Q}'} \cdot (\centerdot)^+ \cdot \overline{\mathsf{Q}} \\
\cong\ &\{ \text{ Lemma 6.2.3.5 } \} \\
&\overline{\mathsf{A}_I \cdot \mathsf{Q}'} \cdot \overline{(\centerdot)^\star} \cdot (\centerdot)^+ \cdot \overline{\mathsf{Q}} \\
\cong\ &\{ \text{ Corollary 6.2.4.6 } \} \\
&\overline{\mathsf{A}_I \cdot \mathsf{Q}'} \cdot \overline{\mathsf{Ł}\mathsf{Q}} \cdot \overline{(\centerdot)^\star} \cdot (\centerdot)^+ \\
\cong\ &\{ \text{ Lemma 6.2.3.5 } \} \\
&\overline{(\centerdot)^\star \cdot \mathsf{Ł}\mathsf{Q} \cdot \mathsf{A}_I \cdot \mathsf{Q}'} \cdot (\centerdot)^+ \\
\cong\ &\{ \text{ Corollary 6.5.2.10 } \} \\
&\overline{(\centerdot)^\star \cdot \mathsf{A}_{\mathsf{Q}I} \cdot \mathsf{Q} \cdot \mathsf{Q}'} \cdot (\centerdot)^+.
\end{aligned}$$

The latter is a pattern of $MAC_{\ell \cdot k}$ since $\mathsf{Q} \cdot \mathsf{Q}' \cong \coprod^{\ell \cdot k}$ (see Theorem 6.5.2.5).

Since $\mathsf{Q}I$ is a coproduct, we have an injection to $\mathsf{Q}I$ from I. Then every macro tree transducer with pattern $\overline{(\centerdot)^\star \cdot \mathsf{A}_I \cdot \mathsf{Q}'} \cdot (\centerdot)^+$ can be transformed into a semanticly equivalent macro tree transducer with pattern $\overline{(\centerdot)^\star \cdot \mathsf{A}_{\mathsf{Q}I} \cdot \mathsf{Q} \cdot \mathsf{Q}'} \cdot (\centerdot)^+$ just by adding superfluous context parameters and states. The resulting pattern can then be factorized using the above equations in reverse order. ∎

6.5.2.12 Corollary. $MAC_1 \cdot TOP = MAC = TOP \cdot MAC_1$ and $MAC \cdot TOP = MAC = TOP \cdot MAC$. ◇

The following is a variant of $MAC = YIELD \cdot TOP$ in [Eng80] and [FV98] where $YIELD \subseteq MAC_1$ is a class of tree transformations expressing β-reductions.

6.5.2.13 Corollary.

$$MAC^n = MAC_1^n \cdot TOP.$$

Proof.

$$
\begin{aligned}
& MAC^n \\
= \quad & \{ \text{ Corollary 6.5.2.12 } \} \\
& (MAC_1 \cdot TOP)^n \\
= \quad & \{ \text{ by induction using Corollary 6.5.2.12 to swap } MAC_1 \text{ with } TOP \text{ } \} \\
& MAC_1^n \cdot TOP^n \\
= \quad & \{ \text{ by induction using Theorem 6.5.2.5 } \} \\
& MAC_1^n \cdot TOP.
\end{aligned}
$$

■

7 Open problems and future work

7.1 Generalized monadic transducers

7.1.1 Generalized monadic transducers

Consider the rule $\varrho : |\mathsf{H}\Delta^\star| \leftarrow \Sigma$ of a monadic transducers according to Definition 6.3.1.1. It is easy to see that (i) and (ii) of Theorem 6.3.2.5 make no use of the fact that $\Delta^\star$ is a *free* monad over Δ. Hence we can define the **generalized monadic transducer** where the rule has the form $\varrho : |\mathsf{H}\mathbb{T}| \leftarrow \Sigma$ where $\mathbb{T}$ is an *arbitrary* monad. Then we can still apply our fusion theorem (Theorem 6.3.2.5 (ii)) even if the consuming monadic transducer is generalized

7.1.2 Internal functions

An application of the generalization described in Subsection 7.1.1 is the following: Instead of the free monad $\Delta^\star$ we consider an arbitrary Δ-algebra A. The construct of all super Δ-algebras of A has free objects, namely the Δ-algebras of A-polynomials. Let $u_X : A[X] \leftarrow X$ be the universal arrow to the set of all A-polynomials over the set X. Then $A[\,\boldsymbol{.}\,] : \boldsymbol{\mathcal{S}et} \leftarrow \boldsymbol{\mathcal{S}et}$ is an endofunctor and we can define a monad $\mathbb{A} = (A[\,\boldsymbol{.}\,], \eta, \mu)$ where $\eta_X = u_X$ and $\mu_X : A\big[A[X]\big] \leftarrow A[X]$ is the unique Δ-algebra-homomorphism such that $\mu_X \cdot u_{A[X]} = id_{A[X]}$ holds. This monad $\mathbb{A}$ describes the substitution of variables in A-polynomials and in the special case that A is the initial Δ-term-algebra we have $\mathbb{A} \cong \Delta^\star$. As outlined in Subsection 7.1.1 we can apply our fusion theorem (Theorem 6.3.2.5 (ii)) even if the consuming (generalized) monadic transducer uses arbitrary functions (operations of a Δ-algebra A) to construct its output instead of using constructors (operations of $\mathsf{T}_\Delta\emptyset$).

7.1.3 Tree to tree-series transducers

It is possible to generalize tree transducers to so called *tree to tree-series transducers*, by allowing formal tree-series as output rather than just trees. The composition of tree to tree-series transducers is investigated in [EFV02]. In the following we will give a brief outline, how tree to tree-series transducers should be described as monadic transducers.

Let us first introduce the basic notions regarding tree-series:

7.1.3.1 Definition (semiring). $A = (|A|, +, \cdot, 0, 1)$ is called a **semiring** if $(|A|, +, 0)$ is a commutative monoid and $(|A|, \cdot, 1)$ is a monoid, such that for all $a, b, c \in |A|$ holds

(i) $a \cdot (b + c) = a \cdot b + a \cdot c$,

(ii) $(a+b)\cdot c = a\cdot c + b\cdot c$, and

(iii) $0\cdot a = 0 = a\cdot 0$. ◇

7.1.3.2 Definition (complete semiring). A semiring $A = (|A|, +, \cdot, 0, 1)$ is called **complete** if sums are defined over arbitrary index sets X, *i.e.* we have a function $\sum_X : |A| \leftarrow |A|^X$ which is natural in X, such that for every finite set $X = \{x_1, \ldots, x_n\}$ and every $p : |A| \leftarrow X$ holds $\sum_X p = \sum_{x\in X} p_x = p_{x_1} + \cdots + p_{x_n}$. ◇

7.1.3.3 Definition (formal series). Let $A = (|A|, +, \cdot, 0, 1)$ be a semiring and X be a set. The set of all **formal series** over X is defined by $A\langle\langle X\rangle\rangle = |A|^X$. It is common to denote elements $p = (p_x)_{x\in X} \in A\langle\langle X\rangle\rangle$ as formal series: $p = \sum_{x\in X} p_x \cdot x$. We define the **support** of a formal series $p \in A\langle\langle X\rangle\rangle$ by $\operatorname{supp} p = \{x \in X \mid f \neq 0\}$. If the support of a formal series p is finite, say $\operatorname{supp} p = \{x_1, \ldots, x_n\}$ we also write: $p = p_{x_1}\cdot x_1 + \cdots + p_{x_n}\cdot x_n$. ◇

7.1.3.4 Lemma (formal series functor). Let A be a complete semiring. For every function $f : X \leftarrow Y$ we define

$$A\langle\langle f\rangle\rangle\big(\sum_{y\in Y} p_y \cdot y\big) = \sum_{y\in Y} p_y \cdot fy = \sum_{x\in X}\big(\sum_{\substack{y\in Y\\ fy=x}} p_y\big)\cdot x.$$

Then $A\langle\langle \mathbf{.} \rangle\rangle : \boldsymbol{\mathcal{S}et} \leftarrow \boldsymbol{\mathcal{S}et}$ is a functor. ◇

7.1.3.5 Definition (formal series monad). Let A be a complete semiring. For every set X we define the functions $\eta_X : A\langle\langle X\rangle\rangle \leftarrow X$ and $\mu_X : A\langle\langle X\rangle\rangle \leftarrow A\langle\langle A\langle\langle X\rangle\rangle\rangle\rangle$ by

$$\eta_X x = 1\cdot x$$
$$\mu_X\big(\sum_{p\in A\langle\langle X\rangle\rangle} f_p \cdot p\big) = \sum_{x\in X}\big(\sum_{p\in A\langle\langle X\rangle\rangle} f_p \cdot p_x\big)\cdot x.$$

Then $\mathbb{A} = (A\langle\langle \mathbf{.} \rangle\rangle, \eta, \mu)$ is a monad. ◇

7.1.3.6 Definition (distributive law [BW85] 9.2**).** Let $\mathbb{T} = (\mathsf{T}, \eta, \mu)$ and $\mathbb{T}' = (\mathsf{T}', \eta', \mu')$ be monads on $\boldsymbol{\mathcal{C}}$. A natural transformation $\lambda : \mathsf{T}\cdot\mathsf{T}' \leftarrow \mathsf{T}'\cdot\mathsf{T}$ is called a **distributive law** of $\mathbb{T}'$ over $\mathbb{T}$ if the following axioms hold:

(i) $\lambda\cdot\mathsf{T}'\eta = \eta\mathsf{T}'$,

(ii) $\lambda\cdot\eta'\mathsf{T} = \mathsf{T}\eta'$,

(iii) $\lambda\cdot\mathsf{T}'\mu = \mu\mathsf{T}'\cdot\mathsf{T}\lambda\cdot\lambda\mathsf{T}$, and

(iv) $\lambda\cdot\mu'\mathsf{T} = \mathsf{T}\mu'\cdot\lambda\mathsf{T}'\cdot\mathsf{T}'\lambda$. ◇

7.1.3.7 Proposition (composition of monads [BW85] 9.2 and [JD93] Section 3.4**).** Let $\mathbb{T} = (\mathsf{T}, \eta, \mu)$ and $\mathbb{T}' = (\mathsf{T}', \eta', \mu')$ be monads on $\mathcal{C}$ and let $\lambda : \mathsf{T} \cdot \mathsf{T}' \leftarrow \mathsf{T}' \cdot \mathsf{T}$ be a distributive law of $\mathbb{T}'$ over $\mathbb{T}$. Then $\mathbb{T}\lambda\mathbb{T}' = \big(\mathsf{T} \cdot \mathsf{T}', \eta * \eta', (\mu * \mu') \cdot \mathsf{T}\lambda\mathsf{T}'\big)$ is a monad on $\mathcal{C}$ called the **compatible monad** with $\mathbb{T}$ and $\mathbb{T}'$ w.r.t. λ. ◇

7.1.3.8 Definition (tree-series monad). Let A be a complete semiring and Σ a ranked alphabet. Let $\mathbb{A}$ be the formal series monad w.r.t. the complete semiring A and $\Sigma^\star$ the free monad over Σ. For every set X an element of $A\langle\langle \mathsf{T}_\Sigma X \rangle\rangle$ is called a *tree-series* over A and X. We will use Proposition 7.1.3.7 to impose monadic structure on tree-series, *i.e.* on the functor $A\langle\langle \centerdot \rangle\rangle \cdot \mathsf{T}_\Sigma$: Let $\mathbb{A}\lambda\Sigma^\star$ be the compatible monad with $\mathbb{A}$ and $\Sigma^\star$ w.r.t. the distributive law λ. Then $\mathbb{A}\lambda\Sigma^\star$ is called the **tree-series monad** over A and Σ w.r.t. the distributive law λ. Notice that $|\mathbb{A}\lambda\Sigma^\star|X = A\langle\langle \mathsf{T}_\Sigma X \rangle\rangle$. ◇

Now we are ready to describe tree to tree-series transducers as generalized monadic transducers: Let A be a semiring and Σ and Δ be a ranked alphabet. The *rule* (Subsection 6.1.3) of a tree to tree-series transducer has the form $\varrho_X : A\langle\langle \mathsf{T}_\Delta X \rangle\rangle \leftarrow \mathsf{T}_\Sigma X$ or (abstracted from X) the form $\varrho : A\langle\langle \centerdot \rangle\rangle \cdot \mathsf{T}_\Delta \leftarrow \mathsf{T}_\Sigma$. We have to choose a distributive law λ satisfying the conditions from Proposition 7.1.3.7 then $\varrho : |\mathbb{A}\lambda\Delta^\star| \leftarrow \mathsf{T}_\Sigma$ is the rule of a generalized monadic transducer. Notice, that we did not touch the natural transformation ϱ at all, we have only rewritten its type. Now we can fuse arbitrary monadic transducers (as producer) with this tree series transducer (as consumer). However, some more work is needed to investigate the pattern of the fusion.

7.2 High-level tree transducers as monadic transducers

A generalization of a macro tree transducer where the context arguments may also be functions is called a *high-level tree transducer* [EV88]. In [EV88] it is demonstrated, that the composition of finitely many macro tree transducers is semanticly equivalent to a high-level tree transducer. Thus the pattern of MAC^n characterizes a high-level tree transducer (of level n). However, we have not yet been able to establish a satisfying relation between the syntax of a high-level tree transducer and such a pattern.

7.3 Bottom-up tree transducers as comonadic transducers

This work is only on tree transducers which read input trees starting from the root towards the leaves (*top-down*). However, there also exists the notion of *bottom-up* tree transducers which read input trees in the opposite direction from the leaves towards the root. We can define *co*monadic transducers as the dual notion of monadic transducers using *co*varietors (which have *co*free *co*monads) and *co*monad transformers. Then it turns out that bottom-up tree transducers are comonadic transducers. We have not yet formalized this precisely.

7.4 Efficiency improvement

In [Voi02] conditions for efficiency improvement by tree transducer composition are given. It is worth mentioning that the composition of tree transducers may also deteriorate the efficiency of a functional program. We would like to have this kind of conditions on the level of monadic transducers.

7.5 Implementation

It is possible to implement the monadic fusion algorithm Figure 6.3 using the rewrite rules of the GHC [PTH01]. However, the respective recursive functions would have to be given in the form of monadic transducers. In the particular case that the recursive programs are tree transducers—which can be checked syntacticly—it should be possible to use the results from Section 6.4 to derive the according monadic transducers automaticly.

Bibliography

[AHS90] J. Adámek, H. Herrlich, and G. E. Strecker. *Abstract and Concrete Categories*. Pure and Applied Mathematics. John Wiley & Sons, 1990. 4.2.1.2, 4.2.1.4, 4.2.1.5, 4.3.4.4, 4.3.4.5, 4.3.4.7, 4.3.4.8, 4.3.4.9, 4.4.4.1, 4.4.4.3, 4.4.4.5

[AP01] J. Adámek and H.-E. Porst. From varieties of algebras to covarieties of coalgebras. In *Proceedings of the 4th Workshop on Coalgebraic Methods in Computer Science (CMCS '01)*, pages 27–46, Genova, Italy, April 2001. Elsevier Science Publishers. 4.3.2.7

[Bak79] B. S. Baker. Composition of top-down and bottom-up tree transductions. *Inform. and Control*, 41:186–213, 1979. 1.2, 3.3.1.1

[BdM97] R. Bird and O. de Moor. *Algebra of Programming*. International Series in Computer Science. Prentice Hall, 1997. 5.2.2

[BW85] M. Barr and C. Wells. *Toposes, triples and theories*, volume 278 of *Grundlehren der mathematischen Wissenschaften*. Springer, New York; Heidelberg, 1985. 4.4, 7.1.3.6, 7.1.3.7

[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. 1, 1.2

[CDPR97b] L. Correnson, E. Duris, D. Parigot, and G. Roussel. Symbolic composition. Technical Report 3348, INRIA, January 1997. 1, 1.2

[CF82] B. Courcelle and P. Franchi–Zannettacci. Attribute grammars and recursive program schemes. *Theoret. Comput. Sci.*, 17:163–191, 235–257, 1982. 1, 1.2

[CM93] P. Cenciarelli and E. Moggi. A syntactic approach to modularity in denotational semantics. In *Proceedings of the 5th Biennial Meeting on Category Theory and Computer Science*, 1993. 65

[Dam82] W. Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20:95–206, 1982. 6.1.2

[dB89] P. J. de Bruin. Naturalness of polymorphism. Technical Report CS 8916, Rijksuniversiteit Groningen, The Netherlands, 1989. 5.2.1, 5.2.1.2

[EFV02] J. Engelfriet, Z. Fülöp, and H. Vogler. Bottom-up and top-down tree series transformations. *J. Automata, Languages and Combinatorics*, 2002. accepted. 7.1.3

[Eng75] J. Engelfriet. Bottom-up and top-down tree transformations—a comparison. *Math. Systems Theory*, 9(3):198–231, 1975. 1, 1.2, 6.5.2.5

[Eng77] J. Engelfriet. Top-down tree transducers with regular look-ahead. *Math. Systems Theory*, 10:289–303, 1977. 1.2

[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. 1, 1.2, 6.5.2

[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 '92), Lille, France 1992. 6.5.2.7

[Eng82] J. Engelfriet. Three Hierarchies of Transducers. *Math. Systems Theory*, 15:95–125, 1982. 1.2

[EV85] J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. System Sci.*, 31:71–146, 1985. 1, 1.2, 6.5.2.11

[EV88] J. Engelfriet and H. Vogler. High level tree transducers and iterated pushdown tree transducers. *Acta Informatica*, 26:131–192, 1988. 1.2, 7.2

[EV91] J. Engelfriet and H. Vogler. Modular tree transducers. *Theoret. Comput. Sci.*, 78:267–304, 1991. 1.2

[FHVV93] Z. Fülöp, F. Herrmann, S. Vágvölgyi, and H. Vogler. Tree transducers with external functions. *Theoret. Comput. Sci.*, 108:185–236, 1993. 1.2

[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. 4.3.4.3

[Fok92b] M. M. Fokkinga. *Law and Order in Algorithmics*. PhD thesis, University of Twente, Dept INF, Enschede, The Netherlands, 1992. 5, 5.1

[Fre90] P. J. Freyd. Agebaically complete categories. In G. Rosolini, A. Carboni, and M. C. Pedicchio, editors, *Category Theory*, volume 1488 of *Lecture Notes in Mathematics*, pages 95–104, Como, Italy, 1990. Springer Verlag. 4.1.7.5, 4.1.7.7

[Fre92] P. J. Freyd. Remarks on algebraically compact categories. In *Applications of Categories in Computer Science*, volume 77 of *London Math. Society Lecture Notes Series*. Cambridge University Press, 1992. 4.1.7.8

[Fül81] Z. Fülöp. On attributed tree transducers. *Acta Cybernet.*, 5:261–279, 1981. 1, 1.2

[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. 1, 1.2, 3.2.1, 3.2.2.2, 2, 6.1.2, 6.5.2

[Gie88] R. Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Inform.*, 25:355–423, 1988. 1, 1.2

[Gil96] A. Gill. *Cheap Deforestation for Non-strict Functional Languages*. PhD thesis, Department of Computing Science, Glasgow University, January 1996. 1, 1.1, 1.4, 5

[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. 1, 1.1, 1.4, 5

[GS84] F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984. 1.2, 5.3.4

[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. 1.2

[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. 5.4.1

[Hin00] R. Hinze. Deriving backtracking monad transformers. In P. Wadler, editor, *Proceedings of the 2000 International Conference on Functional Programming (ICFP '03)*, Montreal, Canada, September 2000. 6.2.2

[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. 1.1

[HPP02] M. Hyland, G. Plotkin, and J. Power. Combining computational effects: Commutativity and sum. In *2nd IFIP International Conference on Computer Science (TCS 2002)*, Montreal, 2002. 65

[Ihr88] Th. Ihringer. *Allgemeine Algebra*. Teubner Studienbücher: Mathematik. Teubner, Stuttgart, Germany, 1988. 2.2

[JD93] M. Jones and L. Duponcheel. Composing monads. Yale technical report YALEU/DCS/RR-1004, Yale University, December 1993. 7.1.3.7

[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 '01)*, volume 2196 of *LNCS*, pages 47–71, Florence, Italy, September 2001. Springer. 1, 1.1, 5.2.1.2

[Jür99] C. Jürgensen. Kategorientheoretisch fundierte Programmtransformationen für Haskell. Diplomarbeit in Mathematik, Lehrstuhl für Informatik II, RWTH-Aachen, Germany, October 1999. 5.2.2

[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. 5.2.2

[Jür01] C. Jürgensen. Composition of tree transducers versus categorical deforestation. In J. Dassow and B. Reichel, editors, *Proceedings of the Workshop on Coding Theory and Formal Languages, 11. Theorietag der GI-Fachgruppe 0.1.5. 'Automaten und Formale Sprachen', Wendgräben*, pages 73–77, Magdeburg, Germany, October 2001. Otto-von-Guericke-Universität. 5

[Jür02] C. Jürgensen. Monadic fusion of functional programs. Technical Report TUD-FI02-12, Technische Universität Dresden, Fakultät Informatik, D-01062 Dresden, Germany, December 2002. *Submitted*. 6

[Jür03] C. Jürgensen. Monadic fusion of functional programs. In Z. Ésik and I. Walukiewicz, editors, *Preliminary Proceedings of the 2003 International Workshop on Fixed Points in Computer Science (FICS '03), Satellite Workshop at the ETAPS '03*, pages 44–63, Warsaw, Poland, April 2003. 6

[JV01] C. Jürgensen and H. Vogler. Syntactic composition of top-down tree transducers is short cut fusion. Technical Report TUD-FI01-10, Technische Universität Dresden, Fakultät Informatik, D-01062 Dresden, Germany, November 2001. 5, 1

[JV04] C. Jürgensen and H. Vogler. Syntactic composition of top-down tree transducers is short cut fusion. *Math. Structures in Comput. Sci.*, 14(2):215–282, 2004. 5

[Kel80] G. M. Kelly. A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves and so on. *Bulletins of the Australian Mathematical Society*, 22:1–83, 1980. 6.2.4

[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. 4.4.4.8

[Küh98] A. Kühnemann. Benefits of tree transducers for optimizing functional programs. In V. Arvind and R. Ramanujam, editors, *Proceedings of the 18th INternational Conference on Foundations of Software Technology & Theoretical Computer Science (FST&TCS '98)*, volume 1530 of *LNCS*, pages 146–157, Chennai, India, December 1998. Springer-Verlag. 1, 1.2

[Küh99] A. Kühnemann. Comparison of deforestation techniques for functional programs and for tree transducers. In A. Middeldorp and T. Sato, editors, *Proceedings of the 4th Fuji International Symposium on Functional and Logic Programming (FLOPS '99)*, volume 1722 of *LNCS*, pages 114–130, Tsukuba, Japan, November 1999. Springer-Verlag. 1.2

[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. 1.2

[Lam68] J. Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103:151–161, 1968. 4.1.7.4

[LG02a] Ch. Lüth and N. Ghani. Composing monads using coproducts. In *International Conference on Functional Programming (ICFP '02)*, pages 133– 144. ACM Press, September 2002. 6.2.4

[LG02b] Ch. Lüth and N. Ghani. Monads and modularity. In Alessandro Armando, editor, *Frontiers of Combining Systems (FroCos '02), 4th International Workshop*, number 2309 in Lecture Notes in Artificial Intelligence, pages 18–32. Springer Verlag, 2002. 6.2.4

[LS95] J. Launchbury and T. Sheard. Warm fusion: Deriving build-catas from recursive definitions. In *Proceedings of Functional Programming Languages and Computer Architecture (FPCA '95)*, pages 314–323, La Jolla, San Diego, CA, USA, June 1995. ACM Press. 1.1

[Mac71] S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, 1971. 9

[Mog90] E. Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, LFCS, 1990. 6.2.2

[NV01] T. Noll and H. Vogler. The universality of higher-order attributed tree transducers. *Theory of Computing Systems*, pages 45–75, 2001. 1.2

[PH99] S. L. Peyton-Jones and J. Hughes, editors. *Haskell 98: A Non-strict, Purely Functional Language*. `http://www.haskell.org/onlinereport/`, 1999. 1, 6.1.1.2, 6.1.2.2

[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 '01)*, pages 203–233, Firenze, Italy, September 2001. 1.1, 7.5

[Rou68] W. C. Rounds. *Trees, transducers and transformations*. PhD thesis, Stanford University, 1968. 1.2, 3, 6.1.2.2

[Rou70] W. C. Rounds. Mappings and grammars on trees. *Math. Systems Theory*, 4:257–287, 1970. 1.2, 3, 3.3.1.1, 2, 6.1.2.2

[SP00] A. Simpson and G. D. Plotkin. Complete axioms for categorical fixed-point operators. In *Proceedings of the 15th Symposium on Logic in Computer Science*, pages 30–41. IEEE Computer Society Press, 2000. 65

[Str72] R. Street. The formal theory of monads. *Journal of Pure and Applied Algebra*, 2:149–168, 1972. 4.4

[Tha70] J. W. Thatcher. Generalized^2 sequential machine maps. *J. Comput. System Sci.*, 4:339–367, 1970. 1.2, 3, 6.1.2.2

[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. 1, 1, 1.1, 1.4, 5, 5.2.1, 5.2.1.1

[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. *To appear in Journal of Functional Programming.* 1

[Voi01] J. Voigtländer. Composition of restricted macro tree transducers. Diplomarbeit in Informatik, Lehrstuhl Grundlagen der Programmierung, Institut für Theoretische Informatik, TU-Dresden, Germany, March 2001. 3.3.1

[Voi02] Janis Voigtländer. Conditions for efficiency improvement by tree transducer composition. In Sophie Tison, editor, *13th International Conference on Rewriting Techniques and Applications, Copenhagen, Denmark, Proceedings*, volume 2378 of *LNCS*, pages 222–236. Springer-Verlag, July 2002. 7.4

[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. 1.1, 5.2.1, 5.2.1.2

[Wad90] P. Wadler. Deforestation: Transforming programs to eliminate trees. *Theoretical Computer Science*, 73(2):231–248, 1990. 1, 1.2

[Wec92] W. Wechsler. *Universal Algebra for Computer Scientists*, volume 25 of *EATCS Monograohs on Theoretical Computer Science*. Springer, 1992. 2.2

Index

Symbols

A

U

V