6

THEOREMS FOR FREE — BY CALCULATION

6.1 INTRODUCTION

As already stressed in previous chapters, type polymorphism remains one of the most useful and interesting ingredients of functional programming. For example, the two functions

$$
\begin{array}{l}
countBits : \mathbb{B}^* \rightarrow \mathbb{N}_0 \\
countBits\ [\,] = 0 \\
countBits\ (b : bs) = 1 + countBits\ bs
\end{array}
$$

and

$$
\begin{array}{l}
countNats : {\mathbb{N}_0}^* \rightarrow \mathbb{N}_0 \\
countNats\ [\,] = 0 \\
countNats\ (b : bs) = 1 + countNats\ bs
\end{array}
$$

are both subsumed by a single, *generic* (that is, parametric) program:

$$
\begin{array}{l}
count : (\forall\ A)\ A^* \rightarrow \mathbb{N}_0 \\
count\ [\,] = 0 \\
count\ (a : as) = 1 + count\ as
\end{array}
$$

Written as a catamorphism

$$(\!| \mathsf{in}_{\mathbb{N}_0} \cdot (id + \pi_2) |\!)$$

and thus even dispensing with a name, it becomes clear why this function is generic: nothing in

$$\mathsf{in}_{\mathbb{N}_0} \cdot (id + \pi_2)$$

is susceptible to the *type* of the elements that are being counted up!

This form of polymorphism, known as *parametric polymorphism*, is attractive because

- one writes less code (*specific* solution = *generic* solution + *customization*);
- it is intellectually rewarding, as it brings elegance and economy in programming;

- and, last but not least[1],

> "(...) *from the* type *of a polymorphic function we can derive a* theorem *that it satisfies.* (...) *How useful are the theorems so generated? Only time and experience will tell* (...)"

Recall that section 2.12 already addresses these theorems, also called *natural properties*. However, the full spread of naturality is not explored there. In particular, it does not address higher-order (exponential) types.

It turns out that the "free theorems" involving such types are easy to derive in relation algebra. The current chapter is devoted to such a generic derivation and includes a number of examples showing how vast the application of *free theorems* is.

6.2 POLYMORPHIC TYPE SIGNATURES

In any typed functional language, when declaring a polymorphic function one is bound to use the same generic format,

$$f \;:\; t$$

known as the function's *signature*: f is the name of the function and t is a functional type written according to the following "grammar" of types:

$$t \;::=\; t' \to t''$$
$$t \;::=\; \mathsf{F}(t_1, \ldots, t_n) \qquad \mathsf{F} \text{ is a } \textit{type} \text{ constructor}$$
$$t \;::=\; v \qquad \text{a type } \textit{variable}\text{, source of polymorphism.}$$

What does it mean for $f : t$ to be *parametrically* polymorphic? We shall see shortly that what :w matters in this respect is the formal structure of type t. Let

- V be the set of type variables involved in type expression t;
- $\{R_v\}_{v \in V}$ be a V-indexed family of relations (f_v in case R_v is a function);
- R_t be a relation defined inductively as follows:

$$R_{t:=v} \;=\; R_v \tag{6.1}$$
$$R_{t:=\mathsf{F}(t_1,\ldots,t_n)} \;=\; \mathsf{F}(R_{t_1}, \ldots, R_{t_n}) \tag{6.2}$$
$$R_{t:=t' \to t''} \;=\; R_{t'} \to R_{t''} \tag{6.3}$$

Two questions arise: what does F in the right handside of (6.2) mean? What kind of relation is $R_{t'} \to R_{t''}$ in (6.3)?

First of all, and to answer the first question, we need the concept of *relator*, which extends that of a *functor* (introduced in section 3.8) to relations.

1 Quoting *Theorems for free!*, by Philip Wadler [58].

6.3 RELATORS

A functor G is said to be a *relator* wherever, given a relation R from A to B, $\mathsf{G}\,R$ extends R to G-structures: it is a relation from $\mathsf{G}\,A$ to $\mathsf{G}\,B$

$$\begin{tikzcd} A \arrow[d, "R"'] \arrow[r, dotted, no head] & \mathsf{G}\,A \arrow[d, "\mathsf{G}\,R"] \\ B \arrow[r, dotted, no head] & \mathsf{G}\,B \end{tikzcd} \tag{6.4}$$

which obeys the properties of a functor,

$$\mathsf{G}\,id = id \tag{6.5}$$

$$\mathsf{G}\,(R \cdot S) = (\mathsf{G}\,R) \cdot (\mathsf{G}\,S) \tag{6.6}$$

— recall (3.55) and (3.56) — plus the properties:

$$R \subseteq S \Rightarrow \mathsf{G}\,R \subseteq \mathsf{G}\,S \tag{6.7}$$

$$\mathsf{G}\,(R^\circ) = (\mathsf{G}\,R)^\circ \tag{6.8}$$

That is, a relator is a functor that is monotonic and commutes with converse. For instance, the "Maybe" functor $\mathsf{G}\ X = 1 + X$ is an example of relator:

$$\begin{tikzcd} A \arrow[d, "R"'] \arrow[r, dotted, no head] & \mathsf{G}\,A = 1 + A \arrow[d, "\mathsf{G}\,R = id + R"] \\ B \arrow[r, dotted, no head] & \mathsf{G}\,B = 1 + B \end{tikzcd}$$

It is monotonic since $\mathsf{G}\ R = id + R$ only involves monotonic operators and commutes with converse via (5.123). Let us unfold $\mathsf{G}\,R = id + R$:

$$\begin{array}{ll} & y(id + R)x \\ \equiv & \quad \{\ \text{unfolding the sum, cf. } id + R = [i_1 \cdot id\ , i_2 \cdot R]\ (5.119)\ \} \\ & y(i_1 \cdot i_1^\circ \cup i_2 \cdot R \cdot i_2^\circ)x \\ \equiv & \quad \{\ \text{relational union (5.57); image}\ \} \\ & y(\mathsf{img}\ i_1)x \vee y(i_2 \cdot R \cdot i_2^\circ)x \\ \equiv & \quad \{\ \text{let } NIL \text{ denote } \underline{\text{the}} \text{ sole inhabitant of the singleton type}\ \} \\ & y = x = i_1 NIL \vee \langle \exists\ b, a\ :\ y = i_2\ b \wedge x = i_2\ a :\ b\ R\ a \rangle \end{array}$$

In words: two "pointer-values" x and y are $\mathsf{G}\ R$-related iff they are both null or they are both defined and hold R-related data.

Finite lists also form a relator, $\mathsf{G}\ X = X^*$. Given $B \xleftarrow{R} A$, relator $B^\star \xleftarrow{R^\star} A^\star$ is the relation

$$\begin{array}{lll} s'(R^\star)s & \Leftrightarrow & \mathsf{length}\ s' = \mathsf{length}\ s\ \wedge \\ & & \langle \forall\ i\ :\ 0 \leqslant i < \mathsf{length}\ s :\ (s'\ !!\ i)\ R\ (s\ !!\ i) \rangle \end{array} \tag{6.9}$$

***Exercise** 6.1. Check properties (6.7) and (6.8) for the list relator defined above.*

□

6.4 A RELATION ON FUNCTIONS

The next step needed to postulate free theorems requires a formal undertanding of the arrow operator written on the right handside of (6.3).

This is achieved by defining the so-called "Reynolds arrow" relational operator, which establishes a relation on two functions f and g parametric on two other arbitrary relations R and S:

$$f(R \leftarrow S)g \quad \equiv \quad f \cdot S \subseteq R \cdot g \qquad \begin{array}{ccc} A & \xleftarrow{S} & B \\ {\scriptstyle f}\downarrow & \subseteq & \downarrow{\scriptstyle g} \\ C & \xleftarrow[R]{} & D \end{array} \tag{6.10}$$

The typing rule is:

$$\frac{\begin{array}{c} A \xleftarrow{S} B \\ C \xleftarrow{R} D \end{array}}{C^A \xleftarrow{R \leftarrow S} D^B}$$

This is a powerful operator that satisfies many properties, for instance:

$$\begin{aligned}
id \leftarrow id \quad &= \quad id && (6.11)\\
(R \leftarrow S)^\circ \quad &= \quad R^\circ \leftarrow S^\circ && (6.12)\\
R \leftarrow S \ \subseteq \ V \leftarrow U \quad &\Leftarrow \quad R \subseteq V \wedge U \subseteq S && (6.13)\\
(R \leftarrow V) \cdot (S \leftarrow U) \quad &\subseteq \quad (R \cdot S) \leftarrow (V \cdot U) && (6.14)\\
(f \leftarrow g^\circ)h \quad &= \quad f \cdot h \cdot g && (6.15)\\
k(f \leftarrow g)h \quad &\equiv \quad k \cdot g = f \cdot h && (6.16)
\end{aligned}$$

From property (6.13) we learn that the combinator is monotonic on the left hand side — and thus facts

$$\begin{aligned}
S \leftarrow R \quad &\subseteq \quad (S \cup V) \leftarrow R && (6.17)\\
\top \leftarrow S \quad &= \quad \top && (6.18)
\end{aligned}$$

hold [2] — and anti-monotonic on the right hand side — and thus property

$$R \leftarrow \bot \quad = \quad \top \tag{6.19}$$

and the two distributive laws which follow:

$$\begin{aligned}
S \leftarrow (R_1 \cup R_2) \quad &= \quad (S \leftarrow R_1) \cap (S \leftarrow R_2) && (6.20)\\
(S_1 \cap S_2) \leftarrow R \quad &= \quad (S_1 \leftarrow R) \cap (S_2 \leftarrow R) && (6.21)
\end{aligned}$$

It should be stressed that (6.14) expresses *fusion* only, not *fission*.

2 Cf. $f \cdot S \cdot g^\circ \subseteq \top \Leftrightarrow$ TRUE concerning (6.18).

SUPREMA AND INFIMA Suppose relation R in (6.10) is a complete partial order $\leqslant$, that is, it has suprema and infima. What kind of relationship is established between two functions f and g such that

$$f\ ((\leqslant) \leftarrow S)\ g$$

holds? We reason:

$$\begin{array}{cl} & f\ ((\leqslant) \leftarrow S)\ g \\ \equiv & \quad \{\ (6.10)\ \} \\ & f \cdot S\ \subseteq\ (\leqslant) \cdot g \\ \equiv & \quad \{\ \text{shunting (5.46)}\ \} \\ & S\ \subseteq\ f^{\circ} \cdot (\leqslant) \cdot g \\ \equiv & \quad \{\ \text{go pointwise — (5.17), etc}\ \} \\ & \langle \forall\, a, b\ :\ a\ S\ b :\ f\ a \leqslant g\ b \rangle \\ \equiv & \quad \{\ \text{introduce supremum, for all } b\ \} \\ & g\ b = \langle \bigvee\ a\ :\ a\ S\ b :\ f\ a \rangle \end{array}$$

In summary:[3]

$$f\ ((\leqslant) \leftarrow S)\ g \quad \equiv \quad g\ b = \langle \bigvee\ a\ :\ a\ S\ b :\ f\ a \rangle \tag{6.22}$$

In words: *$g\ b$ is the largest of all $(f\ a)$ such that $a\ S\ b$ holds*.

Pattern $(\leqslant) \leftarrow \ldots$ turns up quite often in relation algebra. Consider, for instance, a Galois connection $\alpha \vdash \gamma$ (5.132), that is,

$$\begin{array}{cl} & \alpha^{\circ} \cdot (\sqsubseteq) = (\leqslant) \cdot \gamma \\ \equiv & \quad \{\ \text{ping pong}\ \} \\ & \alpha^{\circ} \cdot (\sqsubseteq)\ \subseteq\ (\leqslant) \cdot \gamma\ \wedge\ \gamma^{\circ} \cdot (\geqslant)\ \subseteq\ (\sqsupseteq) \cdot \alpha \end{array}$$

Following the same strategy as just above, we obtain pointwise definitions for the two adjoints of the connection:

$$\gamma\ x = \langle \bigvee\ y\ :\ \alpha\ y \sqsubseteq x :\ y \rangle \tag{6.23}$$

$$\alpha\ y = \langle \bigsqcap\ x\ :\ y \leqslant \gamma\ x :\ x \rangle \tag{6.24}$$

6.5 FREE THEOREM OF TYPE t

We are now ready to establish the *free theorem* (FT) of type t, which is the following remarkably simple result:[4]

---

3 Similarly, introducing infimum, for all a: $f\ a = \langle \bigwedge\ b\ :\ a\ S\ b :\ g\ b \rangle$.

4 This result is due to J. Reynolds [54], advertised by P. Wadler [58] and re-written by Backhouse [2] in the pointfree style adopted in this book.

Given any function $\theta : t$, and V as above, then

$$\theta \, R_t \, \theta$$

holds, for any relational instantiation of type variables in V.
$\square$

Note that this theorem

- is a result about t;
- holds *independently* of the actual definition of θ.

So, it holds about any polymorphic function of type t.

6.6 EXAMPLES

Let us see the simplest of all examples, where the target function is the identity:

$$\theta = id : a \leftarrow a$$

We first calculate $R_{t=a\leftarrow a}$:

$$\begin{array}{ll} & R_{a\leftarrow a} \\ \equiv & \quad \{ \text{ rule } R_{t=t'\leftarrow t''} \;=\; R_{t'} \leftarrow R_{t''} \; \} \\ & R_a \leftarrow R_a \end{array}$$

Then we derive the free theorem itself (R_a is abbreviated to R):

$$\begin{array}{ll} & id(R \leftarrow R)id \\ \equiv & \quad \{ \text{ (6.10) } \} \\ & id \cdot R \subseteq R \cdot id \end{array}$$

In case R is a function f, the FT theorem boils down to *id*'s *natural* property, $id \cdot f \;=\; f \cdot id$ — recall (2.10) — that can be read alternatively as stating that *id* is the *unit* of composition.

As a second example, consider $\theta = reverse : a^\star \leftarrow a^\star$, and first calculate $R_{t=a^\star\leftarrow a^\star}$:

$$\begin{array}{ll} & R_{a^\star\leftarrow a^\star} \\ \equiv & \quad \{ \text{ rule } R_{t=t'\leftarrow t''} \;=\; R_{t'} \leftarrow R_{t''} \; \} \\ & R_{a^\star} \leftarrow R_{a^\star} \\ \equiv & \quad \{ \text{ rule } R_{t=\mathsf{F}(t_1,\ldots,t_n)} \;=\; \mathsf{F}(R_{t_1},\ldots,R_{t_n}) \; \} \\ & {R_a}^\star \leftarrow {R_a}^\star \end{array}$$

where $s \; R^\star s'$ is given by (6.9). Next we calculate the FT itself (R_a abbreviated to R):

$$\begin{array}{ll} & reverse(R^\star \leftarrow R^\star)reverse \\ \equiv & \quad \{ \text{ definition } f(R \leftarrow S)g \;\equiv\; f \cdot S \subseteq R \cdot g \; \} \\ & reverse \cdot R^\star \subseteq R^\star \cdot reverse \end{array}$$

In case R is a function r, this FT theorem boils down to *reverse*'s *natural* property,

$$reverse \cdot r^\star \;=\; r^\star \cdot reverse$$

that is, $reverse\,[\,r\,a \mid a \leftarrow l\,] \;=\; [\,r\,b \mid b \leftarrow reverse\,l\,]$. For the general case, we obtain:

$$\begin{array}{cl} & reverse \cdot R^\star \subseteq R^\star \cdot reverse \\ \equiv & \qquad \{ \text{ shunting rule (5.46) } \} \\ & R^\star \subseteq reverse^\circ \cdot R^\star \cdot reverse \\ \equiv & \qquad \{ \text{ going pointwise (5.19, 5.17) } \} \\ & \langle \forall\, s, r \;::\; s\, R^\star r \Rightarrow (reverse\; s) R^\star (reverse\; r) \rangle \end{array}$$

An instance of this pointwise version of *reverse*-FT will state that, for example, *reverse* will respect element-wise orderings ($R := <$):[5]

$$\begin{array}{c} length\; s = length\; r \wedge \langle \forall\, i \;:\; i \in inds\; s :\; (s\, !!\, i) < (r\, !!\, i) \rangle \\ \Downarrow \\ length(reverse\; s) = length(reverse\; r) \\ \wedge \\ \langle \forall\, j \;:\; j \in inds\; s :\; (reverse\; s\, !!\, j) < (reverse\; r\, !!\, j) \rangle \end{array}$$

(Guess other instances.)

As a third example, also involving finite lists, let us calculate the FT of

$$sort : a^\star \leftarrow a^\star \leftarrow (Bool \leftarrow (a \times a))$$

where the first parameter stands for the chosen ordering relation, expressed by a binary predicate:

$$\begin{array}{cl} & sort(R_{(a^\star \leftarrow a^\star) \leftarrow (Bool \leftarrow (a \times a))})sort \\ \equiv & \qquad \{ \text{ (6.2, 6.1, 6.3); abbreviate } R_a := R \text{ } \} \\ & sort((R^\star \leftarrow R^\star) \leftarrow (R_{Bool} \leftarrow (R \times R)))sort \\ \equiv & \qquad \{ \; R_{t:=Bool} = id \text{ (constant relator) — cf. exercise 6.11 } \} \\ & sort((R^\star \leftarrow R^\star) \leftarrow (id \leftarrow (R \times R)))sort \\ \equiv & \qquad \{ \text{ (6.10) } \} \\ & sort \cdot (id \leftarrow (R \times R)) \;\;\subseteq\;\; (R^\star \leftarrow R^\star) \cdot sort \\ \equiv & \qquad \{ \text{ shunting (5.46) } \} \\ & (id \leftarrow (R \times R)) \;\;\subseteq\;\; sort^\circ \cdot (R^\star \leftarrow R^\star) \cdot sort \\ \equiv & \qquad \{ \text{ introduce variables } f \text{ and } g \text{ (5.19, 5.17) } \} \\ & f(id \leftarrow (R \times R))g \;\;\Rightarrow\;\; (sort\; f)(R^\star \leftarrow R^\star)(sort\; g) \end{array}$$

5 Let *inds s* denote the set $\{0, \ldots, \text{length } s - 1\}$.

$$\equiv \quad \{ \text{ (6.10) twice } \}$$

$$f \cdot (R \times R) \subseteq g \quad \Rightarrow \quad (sort\ f) \cdot R^\star \subseteq R^\star \cdot (sort\ g)$$

Case $R := r$:

$$f \cdot (r \times r) = g \quad \Rightarrow \quad (sort\ f) \cdot r^\star = r^\star \cdot (sort\ g)$$

$$\equiv \quad \{ \text{ introduce variables } \}$$

$$\left\langle \begin{array}{c} \forall\ a, b\ :: \\ f(r\ a, r\ b) = g(a, b) \end{array} \right\rangle \Rightarrow \left\langle \begin{array}{c} \forall\ l\ :: \\ (sort\ f)(r^\star\ l) = r^\star(sort\ g\ l) \end{array} \right\rangle$$

Denoting predicates f, g by infix orderings $\leqslant, \preceq$:

$$\left\langle \begin{array}{c} \forall\ a, b\ :: \\ r\ a \leqslant r\ b \equiv a \preceq b \end{array} \right\rangle \Rightarrow \left\langle \begin{array}{c} \forall\ l\ :: \\ sort\ (\leqslant)(r^\star\ l) = r^\star(sort\ (\preceq)\ l) \end{array} \right\rangle$$

That is, for r monotonic and injective,

$$sort\ (\leqslant)\ [\ r\ a \mid a \leftarrow l\ ]$$

is always the same list as

$$[\ r\ a \mid a \leftarrow sort\ (\preceq)\ l\ ]$$

***Exercise** 6.2. Let C be a nonempty data domain and let and $c \in C$. Let $\underline{c}$ be the* "everywhere c" *function $\underline{c} : A \to C$ (2.12). Show that the free theorem of $\underline{c}$ reduces to*

$$\langle \forall\ R\ ::\ R \subseteq \top \rangle \tag{6.25}$$

□

***Exercise** 6.3. Calculate the free theorem associated with the projections*

$$A \xleftarrow{\pi_1} A \times B \xrightarrow{\pi_2} B$$

and instantiate it to (a) functions; (b) coreflexives. Introduce variables and derive the corresponding pointwise expressions.

□

***Exercise** 6.4. As follow-up to exercise 6.2, consider higher order function $\underline{(\_)} : a \to b \to a$ such that, given any x of type a, produces the constant function $\underline{x}$. Show that the equalities*

$$\underline{f\ x} = f \cdot \underline{x} \tag{6.26}$$

$$\underline{x} \cdot f = \underline{x} \tag{6.27}$$

$$\underline{x}^\circ \cdot \underline{x} = \top \tag{6.28}$$

arise as corollaries of the free theorem *of* $\underline{(\_)}$.[6]
□

Exercise *6.5. The following is a well-known Haskell function*

$$\mathsf{filter} :: \forall\, a \cdot (a \to \mathbb{B}) \to [a] \to [a]$$

Calculate the free theorem associated with its type

$$filter : a^\star \leftarrow a^\star \leftarrow (\mathbb{B} \leftarrow a)$$

and instantiate it to the case where all relations are functions.
□

Exercise *6.6. In many sorting problems, data are sorted according to a given* ranking *function which computes each datum's numeric rank (eg. students marks, credits, etc). In this context one may parameterize sorting with an extra parameter* f *ranking data into a fixed numeric datatype, eg. the integers:* $serial : (a \to \mathbb{N}_0) \to a^\star \to a^\star$. *Calculate the FT of* $serial$.
□

Exercise *6.7. Consider the following function from Haskell's Prelude:*

$$\begin{array}{l} \mathit{findIndices} :: (a \to \mathbb{B}) \to [a] \to [\mathbb{Z}] \\ \mathit{findIndices}\ p\ xs = [i \mid (x,i) \leftarrow \mathsf{zip}\ xs\ [0\,..],\, p\ x] \end{array}$$

which yields the indices of elements in a sequence `xs` *which satisfy* `p`.

For instance, $\mathit{findIndices}\ \ (<0)\ \ [1,-2,3,0,-5] = [1,4]$. *Calculate the FT of this function.*
□

Exercise *6.8. Wherever two equally typed functions* f, g *are such that* $f\ a \leqslant g\ a$, *for all* a, *we say that* f *is* pointwise at most g *and write* $f \mathrel{\dot{\leqslant}} g$,

$$f \mathrel{\dot{\leqslant}} g \quad = \quad f \subseteq (\leqslant) \cdot g \qquad \textit{cf. diagram} \qquad \begin{tikzcd} & A \arrow[dl, "f"'] \arrow[dr, "g"] \arrow[d, phantom, "\subseteq"] & \\ B & {} & B \arrow[ll, "\leqslant"] \end{tikzcd}$$

recall (5.93). Show that implication

$$f \mathrel{\dot{\leqslant}} g \quad \Rightarrow \quad (\mathsf{map}\, f) \mathrel{\dot{\leqslant}^\star} (\mathsf{map}\, g) \tag{6.29}$$

---

6 Note that (6.27) is property (2.14) assumed in chapter 2.

follows from the FT *of the function* $\mathsf{map} : (a \to b) \to a^* \to b^*$.
□

Exercise *6.9. Infer the FT of the following function, written in Haskell syntax,*

$$\begin{array}{l} \mathbf{while} :: (a \to \mathbb{B}) \to (a \to a) \to (a \to b) \to a \to b \\ \mathbf{while}\ p\ f\ g\ x = \mathbf{if}\ \neg\ (p\ x)\ \mathbf{then}\ g\ x\ \mathbf{else}\ \mathbf{while}\ p\ f\ g\ (f\ x) \end{array}$$

which implements a generic `while`*-loop. Derive its corollary for functions.*
□

6.7 CATAMORPHISM LAWS AS FREE THEOREMS

Recall from section 3.13 the concept of a catamorphism over a parametric type $\mathsf{T}\ a$:

$$\begin{tikzcd} \mathsf{T}\ a \arrow[d, "(\!|g|\!)"'] & \mathsf{B}\,(a, \mathsf{T}\ a) \arrow[l, "in_{\mathsf{T}\ a}"'] \arrow[d, "\mathsf{B}\,(id, (\!|g|\!))"] \\ b & \mathsf{B}\,(a, b) \arrow[l, "g"] \end{tikzcd}$$

So $(\!|\_|\!)$ has generic type

$$(\!|\_|\!) : b \leftarrow \mathsf{T}\ a \leftarrow (b \leftarrow \mathsf{B}\,(a, b))$$

where $\mathsf{T}\ a \cong \mathsf{B}\,(a, \mathsf{T}\ a)$. Then the free theorem of $(\!|\_|\!)$ is

$$(\!|\_|\!) \cdot (R_b \leftarrow \mathsf{B}\,(R_a, R_b)) \quad \subseteq \quad (R_b \leftarrow \mathsf{F}\ R_a) \cdot (\!|\_|\!)$$

This unfolds into (R_a, R_b abbreviated to R, S):

$$\begin{array}{ll} & (\!|\_|\!) \cdot (S \leftarrow \mathsf{B}\,(R, S)) \quad \subseteq \quad (S \leftarrow \mathsf{T}\ R) \cdot (\!|\_|\!) \\ \equiv & \qquad \{ \text{ shunting (5.46) } \} \\ & (S \leftarrow \mathsf{B}\,(R, S)) \quad \subseteq \quad (\!|\_|\!)^\circ (S \leftarrow \mathsf{T}\ R) \cdot (\!|\_|\!) \\ \equiv & \qquad \{ \text{ introduce variables } f \text{ and } g \text{ (5.19, 5.17) } \} \\ & f(S \leftarrow \mathsf{B}\,(R, S))g \quad \Rightarrow \quad (\!|f|\!)(S \leftarrow \mathsf{T}\ R)(\!|g|\!) \\ \equiv & \qquad \{ \text{ definition } \ f(R \leftarrow S)g \quad \equiv \quad f \cdot S \subseteq R \cdot g \ \} \\ & f \cdot \mathsf{B}\,(R, S) \subseteq S \cdot g \quad \Rightarrow \quad (\!|f|\!) \cdot \mathsf{T}\ R \subseteq S \cdot (\!|g|\!) \end{array}$$

From the calculated free theorem of the catamorphism combinator,

$$f \cdot \mathsf{B}\,(R, S) \subseteq S \cdot g \quad \Rightarrow \quad (\!|f|\!) \cdot \mathsf{T}\ R \subseteq S \cdot (\!|g|\!)$$

we can infer:

- $(\!|\_|\!)$-*fusion* $(R, S := id, s)$:

$$f \cdot \mathsf{B}\,(id, s) = s \cdot g \quad \Rightarrow \quad (\!|f|\!) = s \cdot (\!|g|\!)$$

— recall (3.71), for $\mathsf{F}\, f = \mathsf{B}\ (id, f)$;

- $(\!|\_|\!)$-*absorption* $(R, S := r, id)$:

$$f \cdot \mathsf{B}\,(r, id) = g \quad \Rightarrow \quad (\!|f|\!) \cdot \mathsf{T}\ r = (\!|g|\!)$$

whereby, substituting $g := f \cdot \mathsf{B}\,(r, id)$:

$$(\!|f|\!) \cdot \mathsf{T}\ r = (\!|f \cdot \mathsf{B}\,(r, id)|\!)$$

— recall (3.77).

***Exercise** 6.10. Let*

$$iprod = (\!|[\underline{1}\,, (\times)]|\!)$$

be the function that multiplies all natural numbers in a given list, and even be the predicate which tests natural numbers for evenness. Finally, let

$$exists = (\!|[\underline{\mathrm{FALSE}}\,, (\vee)]|\!)$$

be the function that implements existential quantification over a list of Booleans.
From (6.30) infer

$$even \cdot iprod \quad = \quad exists \cdot even^{\star}$$

meaning that the product $n_1 \times n_2 \times \ldots \times n_m$ is even if and only if some n_i is so.
□

***Exercise** 6.11. Show that the* identity *relator* Id, *which is such that* $\mathsf{Id}\ R \ = \ R$ *and the* constant *relator* K *(for a given data type K) which is such that* $\mathsf{K}\ R \ = \ id_K$ *are indeed relators.*
□

***Exercise** 6.12.Show that product*

$$\begin{array}{ccc} A & C & \mathsf{G}(A, C) = A \times C \\ {\scriptstyle R}\downarrow & {\scriptstyle S}\downarrow & \downarrow{\scriptstyle \mathsf{G}(R,S) = R \times S} \\ B & D & \mathsf{G}(B, D) = B \times D \end{array}$$

is a (binary) relator.
□

6.8 BIBLIOGRAPHY NOTES

The free theorem of a polymorphic function is a result due to computer scientist John Reynolds [54]. It became popular under the "theorems for free" heading coined by Phil Wadler [58]. The original pointwise setting of this result was re-written in the pointfree style in [2] thanks to the *relation on functions* combinator (6.10) first introduced by Roland Backhouse in [3].

More recently, Janis Voigtlaender devoted a whole research project to free theorems, showing their usefulness in several areas of computer science [38]. One outcome of this project was an automatic generator of free theorems for types written in Haskell syntax. This is (was?) available from Janis Voigtlaender's home page:

`http://www-ps.iai.uni-bonn.de/ft`

The relators used in the calculational style followed in this book are implemented in this automatic generator by so-called structural functor *lifting*.