Embarrassingly Simple Generation of Free Theorems

Stefan Mehner and Janis Voigtländer

March 26th, 2014

Free Theorems

Statements about polymorphic functions based solely on their types, obtained from relational parametricity [Rey83, Wad89].

For example,

- for every $f :: [\alpha] \to [\alpha]$ and every g and x,

$$map\ g\ (f\ x) = f\ (map\ g\ x)$$

Free Theorems

Statements about polymorphic functions based solely on their types, obtained from relational parametricity [Rey83, Wad89].

For example,

- for every $f :: [\alpha] \to [\alpha]$ and every g and x,

$$map\ g\ (f\ x) = f\ (map\ g\ x)$$

- for every $f :: (\alpha \to \text{Bool}) \to [\alpha] \to \text{Maybe}\ \alpha$ and every g, h, x,

$$fmap\ g\ (f\ (h \circ g)\ x) = f\ h\ (map\ g\ x)$$

Free Theorems

Statements about polymorphic functions based solely on their types, obtained from relational parametricity [Rey83, Wad89].

For example,

- for every $f :: [\alpha] \to [\alpha]$ and every g and x,

$$map\ g\ (f\ x) = f\ (map\ g\ x)$$

- for every $f :: (\alpha \to \mathrm{Bool}) \to [\alpha] \to \mathrm{Maybe}\ \alpha$ and every g, h, x,

$$fmap\ g\ (f\ (h \circ g)\ x) = f\ h\ (map\ g\ x)$$

- for every $f :: (\alpha \to \mathrm{Bool}) \to \alpha \to \mathrm{Int}$ and every g, h and x,

$$f\ (h \circ g)\ x = f\ h\ (g\ x)$$

Free Theorems

Statements about polymorphic functions based solely on their types, obtained from relational parametricity [Rey83, Wad89].

For example,

- for every $f :: [\alpha] \to [\alpha]$ and every g and x,
$$map\ g\ (f\ x) = f\ (map\ g\ x)$$
- for every $f :: (\alpha \to \mathrm{Bool}) \to [\alpha] \to \mathrm{Maybe}\ \alpha$ and every g, h, x,
$$fmap\ g\ (f\ (h \circ g)\ x) = f\ h\ (map\ g\ x)$$
- for every $f :: (\alpha \to \mathrm{Bool}) \to \alpha \to \mathrm{Int}$ and every g, h and x,
$$f\ (h \circ g)\ x = f\ h\ (g\ x)$$
- for every $f :: (([\alpha] \to \mathrm{Int}) \to \alpha) \to \alpha$ and every g and h,

Free Theorems

Statements about polymorphic functions based solely on their types, obtained from relational parametricity [Rey83, Wad89].

For example,

- for every $f :: [\alpha] \to [\alpha]$ and every g and x,
$$map\ g\ (f\ x) = f\ (map\ g\ x)$$
- for every $f :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to \mathsf{Maybe}\ \alpha$ and every g, h, x,
$$fmap\ g\ (f\ (h \circ g)\ x) = f\ h\ (map\ g\ x)$$
- for every $f :: (\alpha \to \mathsf{Bool}) \to \alpha \to \mathsf{Int}$ and every g, h and x,
$$f\ (h \circ g)\ x = f\ h\ (g\ x)$$
- for every $f :: (([\alpha] \to \mathsf{Int}) \to \alpha) \to \alpha$ and every g and h,
$$g\ (f\ h) = f\ (\lambda k \to g\ (h\ (k \circ map\ g)))$$

Free Theorems – How they are usually derived

Take polymorphic type, say $f :: (\alpha \rightarrow \mathsf{Bool}) \rightarrow ([\alpha] \rightarrow \mathsf{Maybe}\ \alpha)$,

Free Theorems – How they are usually derived

Take polymorphic type, say $f :: (\alpha \to \mathsf{Bool}) \to ([\alpha] \to \mathsf{Maybe}\ \alpha)$, replace type variables by relation variables, for the example yielding $(\mathcal{R} \to \mathsf{Bool}) \to ([\mathcal{R}] \to \mathsf{Maybe}\ \mathcal{R})$,

Free Theorems – How they are usually derived

Take polymorphic type, say $f :: (\alpha \to \text{Bool}) \to ([\alpha] \to \text{Maybe}\ \alpha)$, replace type variables by relation variables, for the example yielding $(\mathcal{R} \to \text{Bool}) \to ([\mathcal{R}] \to \text{Maybe}\ \mathcal{R})$, invoke a parametricity theorem stating $(f, f) \in \ldots$,

Free Theorems – How they are usually derived

Take polymorphic type, say $f :: (\alpha \to \mathsf{Bool}) \to ([\alpha] \to \mathsf{Maybe}\ \alpha)$, replace type variables by relation variables, for the example yielding $(\mathcal{R} \to \mathsf{Bool}) \to ([\mathcal{R}] \to \mathsf{Maybe}\ \mathcal{R})$, invoke a parametricity theorem stating $(f, f) \in \ldots$, unfold a given set of definitions, such as:

- base types like Bool and Int are read as identity relations,
- $\mathcal{R}_1 \to \mathcal{R}_2 = \{\,(f, g) \mid \forall (a, b) \in \mathcal{R}_1.\ (f\ a, g\ b) \in \mathcal{R}_2\,\}$
- $\mathsf{Maybe}\ \mathcal{R} = \{\,(\mathsf{N}, \mathsf{N})\,\} \cup \{\,(\mathsf{J}\ a, \mathsf{J}\ b) \mid (a, b) \in \mathcal{R}\,\}$

Free Theorems – How they are usually derived

Take polymorphic type, say $f :: (\alpha \to \mathsf{Bool}) \to ([\alpha] \to \mathsf{Maybe}\ \alpha)$, replace type variables by relation variables, for the example yielding $(\mathcal{R} \to \mathsf{Bool}) \to ([\mathcal{R}] \to \mathsf{Maybe}\ \mathcal{R})$, invoke a parametricity theorem stating $(f, f) \in \ldots$, unfold a given set of definitions, such as:

- base types like Bool and Int are read as identity relations,
- $\mathcal{R}_1 \to \mathcal{R}_2 = \{(f, g) \mid \forall (a, b) \in \mathcal{R}_1.\ (f\ a, g\ b) \in \mathcal{R}_2\}$
- $\mathsf{Maybe}\ \mathcal{R} = \{(\mathsf{N}, \mathsf{N})\} \cup \{(\mathsf{J}\ a, \mathsf{J}\ b) \mid (a, b) \in \mathcal{R}\}$

... and then try to massage and simplify the resulting statement.

Free Theorems – How they are usually derived

Take polymorphic type, say $f :: (\alpha \to \text{Bool}) \to ([\alpha] \to \text{Maybe}\ \alpha)$, replace type variables by relation variables, for the example yielding $(\mathcal{R} \to \text{Bool}) \to ([\mathcal{R}] \to \text{Maybe}\ \mathcal{R})$, invoke a parametricity theorem stating $(f, f) \in \ldots$, unfold a given set of definitions, such as:

- base types like Bool and Int are read as identity relations,
- $\mathcal{R}_1 \to \mathcal{R}_2 = \{(f, g) \mid \forall (a, b) \in \mathcal{R}_1.\ (f\ a, g\ b) \in \mathcal{R}_2\}$
- $\text{Maybe}\ \mathcal{R} = \{(\text{N}, \text{N})\} \cup \{(\text{J}\ a, \text{J}\ b) \mid (a, b) \in \mathcal{R}\}$

... and then try to massage and simplify the resulting statement.

For example:

$$
\begin{array}{ll}
& (f, f) \in (\mathcal{R} \to id) \to ([\mathcal{R}] \to \text{Maybe}\ \mathcal{R}) \\
\Leftrightarrow & \llbracket \text{ definition of } \mathcal{R}_1 \to \mathcal{R}_2 \ \rrbracket \\
& \forall (a, b) \in \mathcal{R} \to id.\ (f\ a, f\ b) \in [\mathcal{R}] \to \text{Maybe}\ \mathcal{R} \\
\Leftrightarrow & \llbracket \text{ again } \rrbracket \\
& \forall (a, b) \in \mathcal{R} \to id, (c, d) \in [\mathcal{R}].(f\ a\ c, f\ b\ d) \in \text{Maybe}\ \mathcal{R} \\
\Leftrightarrow & \ldots
\end{array}
$$

Free Theorems – How they are usually derived

For example:

$$
\begin{array}{ll}
& (f, f) \in (\mathcal{R} \to id) \to ([\mathcal{R}] \to \text{Maybe}\ \mathcal{R}) \\
\Leftrightarrow & [\![\ \text{definition of } \mathcal{R}_1 \to \mathcal{R}_2\ ]\!] \\
& \forall (a, b) \in \mathcal{R} \to id.\ (f\ a, f\ b) \in [\mathcal{R}] \to \text{Maybe}\ \mathcal{R} \\
\Leftrightarrow & [\![\ \text{again}\ ]\!] \\
& \forall (a, b) \in \mathcal{R} \to id, (c, d) \in [\mathcal{R}].(f\ a\ c, f\ b\ d) \in \text{Maybe}\ \mathcal{R} \\
\Leftrightarrow & \ldots
\end{array}
$$

Observations:

- Even when we in principle “know” what the free theorem is, we have to go through these steps.

Free Theorems – How they are usually derived

For example:

$$
\begin{array}{ll}
 & (f, f) \in (\mathcal{R} \to id) \to ([\mathcal{R}] \to \text{Maybe}\ \mathcal{R}) \\
\Leftrightarrow & [\![\text{ definition of } \mathcal{R}_1 \to \mathcal{R}_2 \ ]\!] \\
 & \forall (a, b) \in \mathcal{R} \to id.\ (f\ a, f\ b) \in [\mathcal{R}] \to \text{Maybe}\ \mathcal{R} \\
\Leftrightarrow & [\![\text{ again }]\!] \\
 & \forall (a, b) \in \mathcal{R} \to id, (c, d) \in [\mathcal{R}].(f\ a\ c, f\ b\ d) \in \text{Maybe}\ \mathcal{R} \\
\Leftrightarrow & \ldots
\end{array}
$$

Observations:

- Even when we in principle “know” what the free theorem is, we have to go through these steps.
- We have no guarantee that we will end up with a nice enough statement (depends on the massage/simplification heuristics).

Free Theorems – How they are usually derived

For example:

$$
\begin{array}{ll}
 & (f, f) \in (\mathcal{R} \to id) \to ([\mathcal{R}] \to \text{Maybe}\ \mathcal{R}) \\
\Leftrightarrow & [\![\text{ definition of } \mathcal{R}_1 \to \mathcal{R}_2 \ ]\!] \\
 & \forall (a, b) \in \mathcal{R} \to id.\ (f\ a, f\ b) \in [\mathcal{R}] \to \text{Maybe}\ \mathcal{R} \\
\Leftrightarrow & [\![\text{ again }]\!] \\
 & \forall (a, b) \in \mathcal{R} \to id, (c, d) \in [\mathcal{R}].(f\ a\ c, f\ b\ d) \in \text{Maybe}\ \mathcal{R} \\
\Leftrightarrow & \ldots
\end{array}
$$

Observations:

- Even when we in principle “know” what the free theorem is, we have to go through these steps.
- We have no guarantee that we will end up with a nice enough statement (depends on the massage/simplification heuristics).
- Depending on what language we are actually interested in, there will be deviations in the relation unfolding definitions, hence also in the derivations.

Relational Parametricity

Usually,

- definition of a family of relations $\Delta_{\rho,\tau}$ capturing the interpretation of types by relations, such that, e.g., $\Delta_{[\alpha\mapsto\mathcal{R}],(\alpha\to\mathsf{Bool})\to([\alpha]\to\alpha)} = (\mathcal{R} \to \mathit{id}) \to ([\mathcal{R}] \to \mathcal{R})$

Relational Parametricity

Usually,

- definition of a family of relations $\Delta_{\rho,\tau}$ capturing the interpretation of types by relations, such that, e.g., $\Delta_{[\alpha\mapsto\mathcal{R}],(\alpha\to\mathsf{Bool})\to([\alpha]\to\alpha)} = (\mathcal{R} \to \mathit{id}) \to ([\mathcal{R}] \to \mathcal{R})$
- proof that for closed type τ, $\Delta_{\emptyset,\tau}$ is the identity relation

Relational Parametricity

Usually,

- definition of a family of relations $\Delta_{\rho,\tau}$ capturing the interpretation of types by relations, such that, e.g., $\Delta_{[\alpha\mapsto\mathcal{R}],(\alpha\to\mathsf{Bool})\to([\alpha]\to\alpha)} = (\mathcal{R} \to \mathit{id}) \to ([\mathcal{R}] \to \mathcal{R})$
- proof that for closed type τ, $\Delta_{\emptyset,\tau}$ is the identity relation
- proof that for each valid typing judgement $\Gamma \vdash e :: \tau$, if for each $x :: \tau'$ in Γ we choose e_1^x and e_2^x with $(e_1^x, e_2^x) \in \Delta_{\rho,\tau'}$, then $(e[\overline{e_1^x}/\overline{x}], e[\overline{e_2^x}/\overline{x}]) \in \Delta_{\rho,\tau}$

Relational Parametricity

Usually,

- definition of a family of relations $\Delta_{\rho,\tau}$ capturing the interpretation of types by relations, such that, e.g., $\Delta_{[\alpha\mapsto\mathcal{R}],(\alpha\to\mathsf{Bool})\to([\alpha]\to\alpha)} = (\mathcal{R} \to \mathit{id}) \to ([\mathcal{R}] \to \mathcal{R})$
- proof that for closed type τ, $\Delta_{\emptyset,\tau}$ is the identity relation
- proof that for each valid typing judgement $\Gamma \vdash e :: \tau$, if for each $x :: \tau'$ in Γ we choose e_1^x and e_2^x with $(e_1^x, e_2^x) \in \Delta_{\rho,\tau'}$, then $(e[\overline{e_1^x}/\overline{x}], e[\overline{e_2^x}/\overline{x}]) \in \Delta_{\rho,\tau}$

From the above, we prove the "conjuring lemma of parametricity". Crucially, it does not even mention Δ.

The Conjuring Lemma

Let τ, τ_1 and τ_2 be closed types. Let $e :: \tau$ be a term possibly involving α (but not in its own overall type, which is closed by assumption) and term variables $pre :: \tau_1 \to \alpha$ and $post :: \alpha \to \tau_2$, but no other free variables. Then for every $g :: \tau_1 \to \tau_2$,

$$e[\tau_1/\alpha, id_{\tau_1}/pre, g/post] = e[\tau_2/\alpha, g/pre, id_{\tau_2}/post]$$

The Conjuring Lemma

Let τ, τ_1 and τ_2 be closed types. Let $e :: \tau$ be a term possibly involving α (but not in its own overall type, which is closed by assumption) and term variables $pre :: \tau_1 \to \alpha$ and $post :: \alpha \to \tau_2$, but no other free variables. Then for every $g :: \tau_1 \to \tau_2$,

$$e[\tau_1/\alpha, id_{\tau_1}/pre, g/post] = e[\tau_2/\alpha, g/pre, id_{\tau_2}/post]$$

- How could such an e look like?

The Conjuring Lemma

Let τ, τ_1 and τ_2 be closed types. Let $e :: \tau$ be a term possibly involving α (but not in its own overall type, which is closed by assumption) and term variables $pre :: \tau_1 \to \alpha$ and $post :: \alpha \to \tau_2$, but no other free variables. Then for every $g :: \tau_1 \to \tau_2$,

$$e[\tau_1/\alpha, id_{\tau_1}/pre, g/post] = e[\tau_2/\alpha, g/pre, id_{\tau_2}/post]$$

- How could such an e look like?

  For example $e = \lambda xs \to map\ post\ (f\ (map\ pre\ xs))$ with $f :: [\alpha] \to [\alpha]$.

The Conjuring Lemma

Let τ, τ_1 and τ_2 be closed types. Let $e :: \tau$ be a term possibly involving α (but not in its own overall type, which is closed by assumption) and term variables $pre :: \tau_1 \to \alpha$ and $post :: \alpha \to \tau_2$, but no other free variables. Then for every $g :: \tau_1 \to \tau_2$,

$$e[\tau_1/\alpha, id_{\tau_1}/pre, g/post] = e[\tau_2/\alpha, g/pre, id_{\tau_2}/post]$$

- How could such an e look like?

  For example $e = \lambda xs \to map\ post\ (f\ (map\ pre\ xs))$ with $f :: [\alpha] \to [\alpha]$.

- Why is this interesting?

The Conjuring Lemma

Let τ, τ_1 and τ_2 be closed types. Let $e :: \tau$ be a term possibly involving α (but not in its own overall type, which is closed by assumption) and term variables $pre :: \tau_1 \to \alpha$ and $post :: \alpha \to \tau_2$, but no other free variables. Then for every $g :: \tau_1 \to \tau_2$,

$$e[\tau_1/\alpha, id_{\tau_1}/pre, g/post] = e[\tau_2/\alpha, g/pre, id_{\tau_2}/post] \quad (*)$$

- How could such an e look like?

  For example $e = \lambda xs \to map\ post\ (f\ (map\ pre\ xs))$ with $f :: [\alpha] \to [\alpha]$.

- Why is this interesting?

  Because in this case, $(*)$ specializes to

  $\lambda xs \to map\ g\ (f\ (map\ id\ xs)) = \lambda xs \to map\ id\ (f\ (map\ g\ xs))$

Turning this into a Generator

Given some f of polymorphic type, can we come up with some term e of closed type and only $\mathit{pre} :: \tau_1 \rightarrow \alpha$ and $\mathit{post} :: \alpha \rightarrow \tau_2$ (for any closed types τ_1 and τ_2) as free term variables?

Turning this into a Generator

Given some f of polymorphic type, can we come up with some term e of closed type and only $\mathit{pre} :: \tau_1 \to \alpha$ and $\mathit{post} :: \alpha \to \tau_2$ (for any closed types τ_1 and τ_2) as free term variables?

Well, e should of course use f in some interesting way.

Turning this into a Generator

Given some f of polymorphic type, can we come up with some term e of closed type and only *pre* $:: \tau_1 \rightarrow \alpha$ and *post* $:: \alpha \rightarrow \tau_2$ (for any closed types τ_1 and τ_2) as free term variables?

Well, e should of course use f in some interesting way.
In essence, we want to build e around f, using *pre* and *post* to do away with the polymorphism of f.

Turning this into a Generator

Given some f of polymorphic type, can we come up with some term e of closed type and only $pre :: \tau_1 \to \alpha$ and $post :: \alpha \to \tau_2$ (for any closed types τ_1 and τ_2) as free term variables?

Well, e should of course use f in some interesting way.
In essence, we want to build e around f, using *pre* and *post* to do away with the polymorphism of f.

Let's see on a few examples:

- $f :: [\alpha] \to [\alpha] \quad \rightsquigarrow \quad e = map\ post \circ f \circ map\ pre \ :: [\tau_1] \to [\tau_2]$

Turning this into a Generator

Given some f of polymorphic type, can we come up with some term e of closed type and only $pre :: \tau_1 \to \alpha$ and $post :: \alpha \to \tau_2$ (for any closed types τ_1 and τ_2) as free term variables?

Well, e should of course use f in some interesting way.
In essence, we want to build e around f, using *pre* and *post* to do away with the polymorphism of f.

Let's see on a few examples:

- $f :: [\alpha] \to [\alpha] \quad \rightsquigarrow \quad e = map\ post \circ f \circ map\ pre\ :: [\tau_1] \to [\tau_2]$
- $f :: (\alpha \to \text{Bool}) \to [\alpha] \to \text{Maybe}\ \alpha$
  $\rightsquigarrow \quad e = \lambda h \to fmap\ post \circ f\ (h \circ post) \circ map\ pre$
  $\quad :: (\tau_2 \to \text{Bool}) \to [\tau_1] \to \text{Maybe}\ \tau_2$

Turning this into a Generator

Given some f of polymorphic type, can we come up with some term e of closed type and only $\mathit{pre} :: \tau_1 \to \alpha$ and $\mathit{post} :: \alpha \to \tau_2$ (for any closed types τ_1 and τ_2) as free term variables?

Well, e should of course use f in some interesting way.
In essence, we want to build e around f, using *pre* and *post* to do away with the polymorphism of f.

Let's see on a few examples:

- $f :: [\alpha] \to [\alpha] \quad \leadsto \quad e = \mathit{map}\ \mathit{post} \circ f \circ \mathit{map}\ \mathit{pre}\ :: [\tau_1] \to [\tau_2]$
- $f :: (\alpha \to \mathrm{Bool}) \to [\alpha] \to \mathrm{Maybe}\ \alpha$
  $\leadsto \quad e = \lambda h \to \mathit{fmap}\ \mathit{post} \circ f\ (h \circ \mathit{post}) \circ \mathit{map}\ \mathit{pre}$
  $:: (\tau_2 \to \mathrm{Bool}) \to [\tau_1] \to \mathrm{Maybe}\ \tau_2$
- $f :: (\alpha \to \mathrm{Bool}) \to \alpha \to \mathrm{Int}$
  $\leadsto \quad e = \lambda h \to f\ (h \circ \mathit{post}) \circ \mathit{pre}$
  $:: (\tau_2 \to \mathrm{Bool}) \to \tau_1 \to \mathrm{Int}$

Turning this into a Generator

The following does the trick:

$$\begin{aligned}
\mathit{mono}_{\mathit{pre},\mathit{post}}(\alpha) &= \mathit{post} \\
\mathit{mono}_{\mathit{pre},\mathit{post}}(\mathsf{Bool}) &= \mathit{id} \\
\mathit{mono}_{\mathit{pre},\mathit{post}}(\mathsf{Int}) &= \mathit{id} \\
\mathit{mono}_{\mathit{pre},\mathit{post}}([\sigma]) &= \mathit{map}\ \mathit{mono}_{\mathit{pre},\mathit{post}}(\sigma) \\
\mathit{mono}_{\mathit{pre},\mathit{post}}(\mathsf{Maybe}\ \sigma) &= \mathit{fmap}\ \mathit{mono}_{\mathit{pre},\mathit{post}}(\sigma) \\
\mathit{mono}_{\mathit{pre},\mathit{post}}(\sigma_1 \to \sigma_2) &= \lambda h \to \mathit{mono}_{\mathit{pre},\mathit{post}}(\sigma_2) \\
&\qquad\quad \circ\ h \circ \\
&\qquad\quad \mathit{mono}_{\mathit{post},\mathit{pre}}(\sigma_1)
\end{aligned}$$

Turning this into a Generator

The following does the trick:

$$
\begin{aligned}
\mathit{mono}_{\mathit{pre},\mathit{post}}(\alpha) &= \mathit{post}\\
\mathit{mono}_{\mathit{pre},\mathit{post}}(\mathsf{Bool}) &= \mathit{id}\\
\mathit{mono}_{\mathit{pre},\mathit{post}}(\mathsf{Int}) &= \mathit{id}\\
\mathit{mono}_{\mathit{pre},\mathit{post}}([\sigma]) &= \mathit{map}\ \mathit{mono}_{\mathit{pre},\mathit{post}}(\sigma)\\
\mathit{mono}_{\mathit{pre},\mathit{post}}(\mathsf{Maybe}\ \sigma) &= \mathit{fmap}\ \mathit{mono}_{\mathit{pre},\mathit{post}}(\sigma)\\
\mathit{mono}_{\mathit{pre},\mathit{post}}(\sigma_1 \to \sigma_2) &= \lambda h \to \mathit{mono}_{\mathit{pre},\mathit{post}}(\sigma_2)\\
&\qquad\quad \circ\, h \,\circ\\
&\qquad\quad \mathit{mono}_{\mathit{post},\mathit{pre}}(\sigma_1)
\end{aligned}
$$

... in the sense that $e = \mathit{mono}_{\mathit{pre},\mathit{post}}(\sigma)\ f$ is the term we seek if f has polymorphic type σ.

Turning this into a Generator

The following does the trick:

$$\begin{aligned}
mono_{pre,post}(\alpha) &= post \\
mono_{pre,post}(\mathrm{Bool}) &= id \\
mono_{pre,post}(\mathrm{Int}) &= id \\
mono_{pre,post}([\sigma]) &= map\ mono_{pre,post}(\sigma) \\
mono_{pre,post}(\mathrm{Maybe}\ \sigma) &= fmap\ mono_{pre,post}(\sigma) \\
mono_{pre,post}(\sigma_1 \to \sigma_2) &= \lambda h \to mono_{pre,post}(\sigma_2) \\
&\qquad\quad \circ\, h \circ \\
&\qquad\quad mono_{post,pre}(\sigma_1)
\end{aligned}$$

... in the sense that $e = mono_{pre,post}(\sigma)\ f$ is the term we seek if f has polymorphic type σ.

In other words, given $f :: \sigma$, we now generate the free theorem

$$mono_{id,g}(\sigma)\ f = mono_{g,id}(\sigma)\ f$$

. . . and doing deterministic Simplifications

Well, actually, we generate

$$\lfloor mono_{id,g}(\sigma)\ f \rfloor = \lfloor mono_{g,id}(\sigma)\ f \rfloor$$

where:

$$
\begin{aligned}
\lfloor id\ \ t \rfloor &= t \\
\lfloor map\ f\ \ t \rfloor &= map\ (\lambda v \to \lfloor f\ \ v \rfloor)\ t \\
\lfloor fmap\ f\ \ t \rfloor &= fmap\ (\lambda v \to \lfloor f\ \ v \rfloor)\ t \\
\lfloor (\lambda h \to body)\ t \rfloor &= \lambda v \to \lfloor body[t/h]\ v \rfloor \\
\lfloor (f \circ g)\ t \rfloor &= \lfloor f\ \lfloor g\ \ t \rfloor \rfloor \\
\lfloor f\ \ t \rfloor &= f\ \ t
\end{aligned}
$$

... and doing deterministic Simplifications

Well, actually, we generate

$$\lfloor mono_{id,g}(\sigma)\ f \rfloor = \lfloor mono_{g,id}(\sigma)\ f \rfloor$$

where:

$$\begin{aligned}
\lfloor id\ \ t \rfloor &= t \\
\lfloor map\ f\ \ t \rfloor &= map\ (\lambda v \rightarrow \lfloor f\ \ v \rfloor)\ t \\
\lfloor fmap\ f\ \ t \rfloor &= fmap\ (\lambda v \rightarrow \lfloor f\ \ v \rfloor)\ t \\
\lfloor (\lambda h \rightarrow body)\ \ t \rfloor &= \lambda v \rightarrow \lfloor body[t/h]\ \ v \rfloor \\
\lfloor (f \circ g)\ \ t \rfloor &= \lfloor f\ \ \lfloor g\ \ t \rfloor \rfloor \\
\lfloor f\ \ t \rfloor &= f\ \ t
\end{aligned}$$

Thanks to the types used for syntax in the implementation, and GHC's exhaustiveness checker, we know that this simple recursive definition cannot accidentally skip any simplification opportunities.

When it "doesn't work"

For types like $f :: (\alpha \to \alpha) \to (\alpha \to \alpha)$ we lose some generality.

The general free theorem would be:

$$(g \circ h = k \circ g) \Rightarrow (g \circ f\ h = f\ k \circ g)$$

When it “doesn’t work”

For types like $f :: (\alpha \to \alpha) \to (\alpha \to \alpha)$ we lose some generality.

The general free theorem would be:

$$(g \circ h = k \circ g) \Rightarrow (g \circ f\ h = f\ k \circ g)$$

We instead generate:

$$g \circ f\ (p \circ g) = f\ (g \circ p) \circ g$$

When it “doesn’t work”

For types like $f :: (\alpha \to \alpha) \to (\alpha \to \alpha)$ we lose some generality.

The general free theorem would be:

$$(g \circ h = k \circ g) \Rightarrow (g \circ f\ h = f\ k \circ g)$$

We instead generate:

$$g \circ f\ (p \circ g) = f\ (g \circ p) \circ g$$

Why? And what does “like” mean above?

When it "doesn't work"

For types like $f :: (\alpha \to \alpha) \to (\alpha \to \alpha)$ we lose some generality.

The general free theorem would be:

$$(g \circ h = k \circ g) \Rightarrow (g \circ f\ h = f\ k \circ g)$$

We instead generate:

$$g \circ f\ (p \circ g) = f\ (g \circ p) \circ g$$

Why? And what does "like" mean above?

In a nutshell, "because" of: $(\alpha^+ \to \alpha^-)^- \to (\alpha^- \to \alpha^+)^+$

References

- J.C. Reynolds.
  Types, abstraction and parametric polymorphism.
  In *Information Processing, Proceedings*, pages 513–523. Elsevier, 1983.
- P. Wadler.
  Theorems for free!
  In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 347–359. ACM Press, 1989.