October 22, 2004

The Impact of *seq* on Free Theorems-Based Program Transformations

Janis Voigtländer[*]

Dresden University of Technology

`http://wwwtcs.inf.tu-dresden.de/~voigt`

(joint work with Patricia Johann)

*Supported by the "Deutsche Forschungsgemeinschaft" under grant KU 1290/2-4.

Free theorems [Wadler, 1989]: Example

$$
\begin{aligned}
&\mathit{filter} :: \forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha] \\
&\mathit{filter}\ p\ []\ = [] \\
&\mathit{filter}\ p\ (x : xs) = \text{if } p\ x \text{ then } x : \mathit{filter}\ p\ xs \\
&\qquad\qquad\qquad\qquad\quad \text{else } \mathit{filter}\ p\ xs
\end{aligned}
$$

The following law for appropriately typed p, h, and l:

$$\boxed{\mathit{filter}\ p\ (\mathit{map}\ h\ l) = \mathit{map}\ h\ (\mathit{filter}\ (p \circ h)\ l) \qquad (1)}$$

can be derived solely from the parametric polymorphic type of *filter*!

(Note: Due to the use of recursion and pattern matching, the original approach of [Wadler, 1989] would yield law (1) only for strict h, but following [Launchbury & Paterson, 1996] one can recover the unrestricted form.)

The strict evaluation primitive *seq* (in Haskell)

To avoid unneeded laziness, evaluation can be forced explicitly using the polymorphic primitive *seq*, which evaluates its first argument to weak head normal form before returning its second argument. In terms of denotational semantics:

$$\boxed{\begin{array}{l} seq :: \forall \alpha\ \beta.\ \alpha \to \beta \to \beta \\ seq\ \bot\ b = \bot \\ seq\ a\ \ b = b\ \text{, if}\ a \neq \bot \end{array}}$$

But the language definition [Peyton Jones (ed), 2003] warns:

> "However, the provision of *seq* has important semantic consequences, because it is available at every type. As a consequence, $\bot$ is not the same as $(\lambda x \to \bot)$, since *seq* can be used to distinguish them. For the same reason, the existence of *seq* weakens Haskell's parametricity properties."

Program transformation: *foldr*/*build* [Gill *et al.*, 1993]

$$\begin{aligned}
&\mathit{foldr} :: \forall \alpha\ \beta.\ (\alpha \to \beta \to \beta) \to \beta \to [\alpha] \to \beta \\
&\mathit{foldr}\ c\ n\ []\ = n \\
&\mathit{foldr}\ c\ n\ (a : as) = c\ a\ (\mathit{foldr}\ c\ n\ as) \\[1em]
&\mathit{build} :: \forall \alpha.\ (\forall \beta.\ (\alpha \to \beta \to \beta) \to \beta \to \beta) \to [\alpha] \\
&\mathit{build}\ g = g\ (:)\ []
\end{aligned}$$

Ostensibly we have for appropriately typed g, c, and n:

$$\mathit{foldr}\ c\ n\ (\mathit{build}\ g) = g\ c\ n \tag{2}$$

but each of the following instantiations makes only the right-hand side $\bot$:

$$\begin{array}{lll}
g = \mathit{seq} & c = \bot & n = 0 \\
g = (\lambda c\ n \to \mathit{seq}\ n\ (c\ \bot\ \bot)) & c = (:) & n = \bot
\end{array}$$

- Can law (2) fail also in other ways (than the rhs being less defined)?
- Under which conditions is the *foldr*/*build* transformation safe?

Key to free theorems: Relational interpretations of types

For every type τ and every relation environment η that maps all free variables of τ to relations between closed types, we define a relation $\Delta_{\tau,\eta}$ by induction on the type structure:

$$\begin{aligned}
\Delta_{\alpha,\eta} &= \eta(\alpha) \\
\Delta_{\tau\to\tau',\eta} &= \{(f,g) \mid \forall (x,y) \in \Delta_{\tau,\eta}.\ (f\ x, g\ y) \in \Delta_{\tau',\eta}\} \\
\Delta_{\forall\alpha.\,\tau,\eta} &= \{(u,v) \mid \forall \tau_1,\tau_2,\mathcal{R} \in Rel(\tau_1,\tau_2).\ (u_{\tau_1}, v_{\tau_2}) \in \Delta_{\tau,\eta[\mathcal{R}/\alpha]}\} \\
\Delta_{\mathsf{Bool},\eta} &= id_{\mathsf{Bool}} \\
\Delta_{[\tau],\eta} &= \mathit{lift}_{[]}(\Delta_{\tau,\eta})
\end{aligned}$$

The *abstraction* or *parametricity theorem* [Reynolds, 1983, Wadler, 1989] implies the following fundamental property of the relational interpretations:

if $t :: \tau$ is a closed term of closed type, then:

$$(t,t) \in \Delta_{\tau,\emptyset} \qquad (3)$$

But *seq* breaks its parametricity property!

According to the fundamental property (3), we should have:

$$
\begin{aligned}
& (seq, seq) \in \Delta_{\forall \alpha\, \beta.\, \alpha \to \beta \to \beta, \emptyset} \\
\Leftrightarrow\ & \forall \mathcal{R} \in Rel(\tau_1, \tau_2), \mathcal{S} \in Rel(\tau_1', \tau_2').\ (seq_{\tau_1\, \tau_1'}, seq_{\tau_2\, \tau_2'}) \in \Delta_{\alpha \to \beta \to \beta, [\mathcal{R}/\alpha, \mathcal{S}/\beta]} \\
\Leftrightarrow\ & \forall \mathcal{R} \in Rel(\tau_1, \tau_2), \mathcal{S} \in Rel(\tau_1', \tau_2'), (a_1, a_2) \in \Delta_{\alpha, [\mathcal{R}/\alpha, \mathcal{S}/\beta]}, (b_1, b_2) \in \Delta_{\beta, [\mathcal{R}/\alpha, \mathcal{S}/\beta]}. \\
& \quad (seq_{\tau_1\, \tau_1'}\ a_1\ b_1, seq_{\tau_2\, \tau_2'}\ a_2\ b_2) \in \Delta_{\beta, [\mathcal{R}/\alpha, \mathcal{S}/\beta]} \\
\Leftrightarrow\ & \forall \mathcal{R} \in Rel(\tau_1, \tau_2), \mathcal{S} \in Rel(\tau_1', \tau_2'), (a_1, a_2) \in \mathcal{R}, (b_1, b_2) \in \mathcal{S}. \\
& \quad (seq_{\tau_1\, \tau_1'}\ a_1\ b_1, seq_{\tau_2\, \tau_2'}\ a_2\ b_2) \in \mathcal{S}
\end{aligned}
$$

But this statement is falsified, e.g., by the following instantiation:

$$
\begin{aligned}
\mathcal{R} &= \{(a, \bot_{\mathsf{Bool}}) \mid a :: \mathsf{Bool}\} \\
\mathcal{S} &= \{(b, b) \mid b :: \mathsf{Bool}\}.
\end{aligned}
$$

Hence, also other terms that are built using *seq* might violate their parametricity properties!

Our recipe

1. Restrict quantified relations in such a way that ***seq*** does fulfill its parametricity property.
2. Adapt relational actions so that they preserve the chosen set of restrictions (but still lend themselves for the inductive proof over the structure of typing derivations).
3. Choose restrictions carefully (not too rigid) so that inequational statements come into reach when equational laws break down.

[Johann & V., 2004] Free theorems in the presence of *seq*. ***Proc. POPL'04***, SIGPLAN Notices 39(1):99–110. ACM Press.

Adapted relational interpretations of types

$$
\begin{aligned}
\Delta^{seq}_{\alpha,\eta} &= \eta(\alpha) \\
\Delta^{seq}_{\tau\to\tau',\eta} &= \{(f,g) \mid (f \neq \bot \Rightarrow g \neq \bot) \wedge \forall (x,y) \in \Delta^{seq}_{\tau,\eta}.\ (f\ x, g\ y) \in \Delta^{seq}_{\tau',\eta}\} \\
\Delta^{seq}_{\forall\alpha.\tau,\eta} &= \{(u,v) \mid \forall \tau_1,\tau_2,\mathcal{R} \in Rel^{seq}(\tau_1,\tau_2).\ (u_{\tau_1}, v_{\tau_2}) \in \Delta^{seq}_{\tau,\eta[\mathcal{R}/\alpha]}\} \\
\Delta^{seq}_{\mathsf{Bool},\eta} &= \sqsubseteq_{\mathsf{Bool}} \\
\Delta^{seq}_{[\tau],\eta} &= \sqsubseteq ; lift_{[]}(\Delta^{seq}_{\tau,\eta})
\end{aligned}
$$

where Rel^{seq} denotes the set of all relations $\mathcal{R}$ such that:

$$(\bot,\bot) \in \mathcal{R} \qquad (strictness)$$

$$(\forall i.\ (x_i,y_i) \in \mathcal{R}) \Rightarrow (\bigsqcup x_i, \bigsqcup y_i) \in \mathcal{R} \qquad (continuity)$$

$$\forall (x,y) \in \mathcal{R}.\ x \neq \bot \Rightarrow y \neq \bot \qquad (totality)$$

$$\sqsubseteq ; \mathcal{R} = \mathcal{R} \qquad (left\text{-}closedness)$$

A new parametricity theorem

In the presence of *seq* (and fixpoint recursion and algebraic datatypes) holds:

if $t :: \tau$ is a closed term of closed type, then:

$$(t, t) \in \Delta^{seq}_{\tau,\emptyset} \tag{4}$$

More generally, in analogy to Reynolds' *identity extension lemma*, we use:

if τ is a closed type, then:

$$\Delta^{seq}_{\tau,\emptyset} = \sqsubseteq_{\tau} \tag{5}$$

Restoring *foldr*/*build*

In the presence of *seq* we obtain for appropriately typed g, c, and n:

$$foldr\ c\ n\ (build\ g) \sqsupseteq g\ c\ n \tag{6}$$

if $c \perp \perp \neq \perp$ *and* $n \neq \perp$, *then:*

$$foldr\ c\ n\ (build\ g) = g\ c\ n \tag{7}$$

- *foldr*/*build* is at least partially correct in the presence of *seq*, which was not known previously
- none of the preconditions can be dropped from law (7)
- law (7) just "accidentally" coincides with what the conventional wisdom had to say about *foldr*/*build* in the presence of *seq*
- the restrictions can actually hinder fusion opportunities; consider, e.g., $concat = foldr\ (+\!\!+)\ []$

Program transformation: *destroy*/*unfoldr* [Svenningsson, 2002]

$$\begin{array}{l}
\text{data } \mathsf{Maybe}\ \alpha = \mathsf{Nothing} \mid \mathsf{Just}\ \alpha \\[1em]
\mathit{unfoldr} :: \forall\alpha\ \beta.\ (\beta \to \mathsf{Maybe}\ (\alpha,\beta)) \to \beta \to [\alpha] \\
\mathit{unfoldr}\ f\ b = \text{case } f\ b \text{ of } \mathsf{Nothing} \quad\ \ \to [] \\
\qquad\qquad\qquad\qquad\qquad\quad \mathsf{Just}\ (a, b') \to a : \mathit{unfoldr}\ f\ b' \\[1em]
\mathit{destroy} :: \forall\alpha\ \gamma.\ (\forall\beta.\ (\beta \to \mathsf{Maybe}\ (\alpha,\beta)) \to \beta \to \gamma) \to [\alpha] \to \gamma \\
\mathit{destroy}\ g = g\ \mathit{listpsi} \\
\quad \text{where } \mathit{listpsi}\ [] \qquad\quad = \mathsf{Nothing} \\
\qquad\qquad\ \ \mathit{listpsi}\ (a : as) = \mathsf{Just}\ (a, as)
\end{array}$$

We want for appropriately typed g, psi, and e:

$$\mathit{destroy}\ g\ (\mathit{unfoldr}\ psi\ e) \sqsubseteq g\ psi\ e \qquad (8)$$

But there are devious counterexamples using *seq*!

Restoring *destroy*/*unfoldr*

In the presence of *seq* we obtain for appropriately typed g, psi, and e:

if $psi \neq \bot$ *and* $psi\ \bot$ *is* $\bot$ *or* $\mathsf{Just}\ \bot$, *then:*

$$destroy\ g\ (unfoldr\ psi\ e) \sqsubseteq g\ psi\ e \tag{9}$$

if psi *is strict and total and never returns* $\mathsf{Just}\ \bot$, *then:*

$$destroy\ g\ (unfoldr\ psi\ e) \sqsupseteq g\ psi\ e \tag{10}$$

- none of the preconditions can be dropped from laws (9) and (10)
- the restrictions in law (9) are fulfilled at least for every producer of a nonempty finite list
- the restriction regarding $\mathsf{Just}\ \bot$ in law (10) is merely an artifact of using the modularly constructed type $\mathsf{Maybe}\ (\alpha, \beta)$ instead of a tailored one
- *seq* compromises the duality between *foldr*/*build* and *destroy*/*unfoldr*

Program transformation: *vanish* [V., 2002]

$$\mathit{vanish}_{++} :: \forall\alpha.\ (\forall\beta.\ \beta \to (\alpha \to \beta \to \beta) \to (\beta \to \beta \to \beta) \to \beta) \to [\alpha]$$

$$\mathit{vanish}_{++}\ g = g\ (\lambda x \to x)\ (\lambda x\ h\ ys \to x : h\ ys)\ (\circ)\ []$$

$$\mathit{vanish}_{rev} :: \forall\alpha.\ (\forall\beta.\ \beta \to (\alpha \to \beta \to \beta) \to (\beta \to \beta) \to \beta) \to [\alpha]$$

$$\begin{aligned}\mathit{vanish}_{rev}\ g = \mathit{fst}\ (g\ & (\lambda ys \to (ys, ys)) \\ & (\lambda x\ h\ ys \to (x : (\mathit{fst}\ (h\ ys)), \mathit{snd}\ (h\ (x : ys)))) \\ & (\lambda h\ ys \to \mathit{swap}\ (h\ ys)) \\ & [])\end{aligned}$$

In the absence of *seq* we have for appropriately typed gs:

$$g\ []\ (:)\ (++) = \mathit{vanish}_{++}\ g \tag{11}$$

$$g\ []\ (:)\ \mathit{reverse} \sqsubseteq \mathit{vanish}_{rev}\ g \tag{12}$$

But for (11) there are counterexamples using *seq*!

Restoring $vanish_{++}$

In the presence of *seq* we obtain for appropriately typed *g*:

$$\boxed{g\ []\ (:)\ (+\!\!+) \sqsubseteq vanish_{++}\ g \qquad (13)}$$

- **there is no way — in the presence of *seq* — to recover an equational variant of law (13) while also preserving its nature as a free theorem**
- **analogous inequational laws in the presence of *seq* hold also for the other *vanish* combinators**

Conclusions

- *seq* is really weird
- not all is lost w.r.t. program transformations
- inequational free theorems are an essential gain:
  - they appear to capture the essence of the impact of *seq* on free theorems
  - when an informed change of termination behavior is acceptable, they tend to provide useful results where an approach insisting on equational laws would have to impose much stronger restrictions or would fail completely
  - whenever an equational law exists, it seems to be available as well through combination of inequational ones

Directions for future research

- contain the weakening of free theorems due to ***seq*** using a qualified type system along the lines of [Launchbury & Paterson, 1996]
- consider "asymmetric" relational interpretations of types for purely nonstrict languages, e.g., to establish law (8) in the absence of ***seq***
- investigate free theorems for purely strict languages
- establish connections to Pitts' operational approach to parametricity [Pitts, 2000]
- study the impact of other primitives, such as the ones used to incorporate I/O and stateful references in Haskell

References

[Gill, Launchbury & Peyton Jones, 1993] A short cut to deforestation. In Proceedings of *Functional Programming Languages and Computer Architecture*. ACM Press.

[Johann & Voigtländer, 2004] Free Theorems in the Presence of *seq*. In Proceedings of *Principles of Programming Languages*. ACM Press.

[Launchbury & Paterson, 1996] Parametricity and unboxing with unpointed types. In Proceedings of *European Symposium on Programming*. LNCS, vol. 1058. Springer-Verlag.

[Peyton Jones (ed), 2003] *Haskell 98 Language and Libraries: The Revised Report*. CUP.

[Pitts, 2000] Parametric polymorphism and operational equivalence. *Math. Struct. Comput. Sci.*, **10**, 321–359.

[Reynolds, 1983] Types, abstraction and parametric polymorphism. In Proceedings of *Information Processing*. Elsevier Science Publishers B.V.

[Svenningsson, 2002] Shortcut fusion for accumulating parameters & zip-like functions. In Proceedings of *International Conference on Functional Programming*. ACM Press.

[Voigtländer, 2002] Concatenate, reverse and map vanish for free. In Proceedings of *International Conference on Functional Programming*. ACM Press.

[Wadler, 1989] Theorems for free! In Proceedings of *Functional Programming Languages and Computer Architecture*. ACM Press.