Corrigendum to

Bridging the gulf: a common intermediate language for ML and Haskell*

Simon Peyton Jones†
University of Glasgow
simonpj@dcs.glasgow.ac.uk

John Launchbury‡
Oregon Graduate Institute
jl@cse.ogi.edu

Mark Shields§
University of Glasgow
mbs@dcs.glasgow.ac.uk

Andrew Tolmach
Portland State University
apt@cs.pdx.edu

January, 1998

The published paper contains an error pointed out to us by Andrzej Filinski: the monad laws of Figure 3 are unsound for the denotational semantics of $\mathcal{L}_2$ given in Figure 8. As the monad laws do not hold for Haskell programs either, $\mathcal{L}_2$ could still be a suitable intermediate language for Haskell alone. Unfortunately though, this makes $\mathcal{L}_2$ unsuitable as a target for ML programs: useful transformations on ML programs are not sound on their translation into $\mathcal{L}_2$.

We first demonstrate why the laws fail. If we switch from using partial functions to total functions with a lift monad (in the categorical sense rather than the `Lift` monad of the paper), the semantics of the $\mathcal{L}_2$ constructs $\mathtt{let}_M$ and $\mathtt{ret}_M$ become

$$\begin{array}{rcl}
\mathcal{E}[\![\mathtt{let}_M\, x \leftarrow e_1 \,\mathtt{in}\, e_2]\!]\rho & = & \mathbf{let}_\bot\; e_1' \leftarrow \mathcal{E}[\![e_1]\!]\rho \;\;\mathbf{in}\,\mathbf{unit}_\bot(\mathit{bind}_M\; e_1'\; (\lambda y\,.\,\mathcal{E}[\![e_2]\!]\rho[x \mapsto y])) \\
\mathcal{E}[\![\mathtt{ret}_M\, e_2]\!]\rho & = & \mathbf{let}_\bot\; e_2' \leftarrow \mathcal{E}[\![e_2]\!]\rho \;\;\mathbf{in}\,\mathbf{unit}_\bot(\mathit{unit}_M\; e_2')
\end{array}$$

Similarly, the semantics for bind_{ST} and unit_{ST} become

$$\begin{array}{rcll}
\mathit{bind}_{ST}\; m\; k & = & \lambda s\,.\;\; \mathbf{let}_\bot (r, s') \leftarrow m\; s\,\mathbf{in} & \\
 & & \mathbf{let}_\bot\; m' \leftarrow k\; r\,\mathbf{in} & (*) \\
 & & m'\; s' & \\
\mathit{unit}_{ST}\; r & = & \lambda s\,.\,\mathbf{unit}_\bot(r, s) &
\end{array}$$

Note that the test for divergence marked $(*)$ was omitted in Figure 8, and will prove to be

*Appeared in the 25th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California, January 19–23, 1998.

†Department of Computing Science, 8–17 Lillybank Gardens, Hillhead, Glasgow, Scotland, G12 8QQ.

‡Department of Computer Science and Engineering, P.O. Box 91000, Portland, OR 97291-1000 USA.

§Research supported by ORS Award #96017029.

the source of the problem. Now consider monad beta-reduction in $\mathcal{L}_2$:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathcal{E}[\![\mathtt{let}_{ST}\ x \leftarrow \mathtt{ret}_{ST}\ e_1\ \mathtt{in}\ e_2]\!]\rho} \\
= & \mathbf{let}_\perp\ z' \leftarrow (\mathbf{let}_\perp\ z \leftarrow \mathcal{E}[\![e_1]\!]\rho\ \mathbf{in}\ \mathbf{unit}_\perp(\mathit{unit}_{ST}\ z))\ \mathbf{in} \\
 & \mathbf{unit}_\perp(\mathit{bind}_{ST}\ z'\ (\lambda y\ .\ \mathcal{E}[\![e_2]\!]\rho[x \mapsto y])) \\
= & \mathbf{let}_\perp\ z \leftarrow \mathcal{E}[\![e_1]\!]\rho\ \mathbf{in}\ \mathbf{unit}_\perp(\mathit{bind}_{ST}\ (\mathit{unit}_{ST}\ z)\ (\lambda y\ .\ \mathcal{E}[\![e_2]\!]\rho[x \mapsto y])) \\
= & \mathbf{let}_\perp\ z \leftarrow \mathcal{E}[\![e_1]\!]\rho\ \mathbf{in} \\
 & \mathbf{unit}_\perp(\lambda s\ . \\
 & \quad \mathbf{let}_\perp(r, s'') \leftarrow (\lambda s'\ .\ \mathbf{unit}_\perp(z, s'))s\ \mathbf{in} \\
 & \quad \mathbf{let}_\perp\ m' \leftarrow (\lambda y\ .\ \mathcal{E}[\![e_2]\!]\rho[x \mapsto y])r\ \mathbf{in} \\
 & \quad m'\ s'') \\
= & \mathbf{let}_\perp\ z \leftarrow \mathcal{E}[\![e_1]\!]\rho\ \mathbf{in}\ \mathbf{unit}_\perp(\lambda s\ .\ \mathbf{let}_\perp\ m' \leftarrow \mathcal{E}[\![e_2]\!]\rho[x \mapsto z])\ \mathbf{in}\ m'\ s) \\
\neq & \mathbf{let}_\perp\ z \leftarrow \mathcal{E}[\![e_1]\!]\rho\ \mathbf{in}\ \mathbf{let}_\perp\ m' \leftarrow \mathcal{E}[\![e_2]\!]\rho[x \mapsto z]\ \mathbf{in}\ \mathbf{unit}_\perp(\lambda s\ .\ m'\ s) \\
= & \mathbf{let}_\perp\ z \leftarrow \mathcal{E}[\![e_1]\!]\rho\ \mathbf{in}\ \mathcal{E}[\![e_2]\!]\rho[x \mapsto z]) \\
= & \mathcal{E}[\![\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2]\!]\rho
\end{array}
$$

The monad eta and commuting conversion rules are similarly unsound. That the omission of $(*)$ went unnoticed is a good argument against using implicit partiality.

Unfortunately this problem is intrinsic to the design of $\mathcal{L}_2$ and cannot be simply repaired.

Thankfully, all is not lost. With a little modification, $\mathcal{L}_1$ can make a very suitable intermediate language. The problems with recursion over polymorphic types discussed in Section 3.5, and the operational ambiguity of "vanilla" let discussed in Section 3.6, turn out to be mostly the fault of the Haskell translation, $\mathcal{H}$, of Figure 5 rather than intrinsic to $\mathcal{L}_1$ itself.

More information on the revised $\mathcal{L}_1$ can be found at

`http://www.dcs.gla.ac.uk/~mbs/pub`

We thank Andrzej Filinski, Ross Paterson and Phil Wadler for their helpful comments.