JFP **21** (2): 209–213, 2011. © Cambridge University Press 2010
doi:10.1017/S0956796810000080 First published online 8 April 2010

Short note: Strict unwraps make worker/wrapper fusion totally correct

PETER GAMMIE
School of Computer Science, The Australian National University, Canberra ACT 0200
(*e-mail:* `Peter.Gammie@anu.edu.au`)

Abstract

The worker/wrapper transformation is a general way of changing the type of a recursive definition, usually applied with an eye to increasing algorithmic efficiency. This note identifies an infelicity in the program transformations presented by Gill & Hutton (The worker/wrapper transformation, *J. Funct. Program.*, vol. 19, 2009, pp. 227–251) and proposes a new totally correct worker/wrapper fusion rule.

1 Introduction

The worker/wrapper transformation has been formalised by Gill & Hutton (2009) as a technique for changing 'a computation of one type into a worker of a different type, together with a wrapper that acts as an impedance matcher between the original and new computations'. Their transformation and associated fusion rule are reproduced in Figure 1, and the reader is referred to the original paper for motivation and background.

At issue is the soundness of applying the fusion rule, which is the only essential use made by Gill and Hutton of the fold/unfold program transformation framework due to Burstall & Darlington (1977); the other transformations are directly justified by a standard fixed-point semantics. This note shows that applying the fusion rule requires extra conditions to be totally correct and proposes one such sufficient condition.

A fully formal account can be found in the Archive of Formal Proofs (Gammie 2009). This was developed in the Isabelle/HOLCF system of Müller *et al.* (1999) and more recently Huffman (2009).

2 A non-strict *unwrap* may go awry

We begin by examining how Gill and Hutton apply their worker/wrapper fusion rule in the context of the fold/unfold framework.

The key step of those left implicit in the original paper is the use of the fold rule to justify replacing the worker with the fused version. Schematically, the fold/unfold framework maintains a history of all definitions that have appeared during transformation, and the fold rule treats this as a set of rewrite rules oriented

For a recursive definition $comp = \mathit{fix}\ body$ for some $body :: A \to A$ and a pair of functions $wrap :: B \to A$ and $unwrap :: A \to B$ where $wrap \circ unwrap = id_A$, we have

$$\begin{aligned} &comp = wrap\ work \\ &\quad work :: B \qquad\qquad \text{(the worker/wrapper transformation)} \\ &\quad work = \mathit{fix}\ (unwrap \circ body \circ wrap) \end{aligned}$$

Also:

$$(unwrap \circ wrap)\ work = work \qquad \text{(worker/wrapper fusion)}$$

Fig. 1. The worker/wrapper transformation and fusion rule of Gill & Hutton (2009).

right-to-left. (The unfold rule treats the current working set of definitions as rewrite rules oriented left-to-right.) Hence as each definition $f = body$ yields a rule of the form $body \implies f$, one can always derive $f = f$. Clearly this has dire implications for the preservation of termination behaviour.

Tullsen (2002) in his §3.1.2 observes that the semantic essence of the fold rule is Park induction, viz that $f\ x = x$ implies only the partially correct $\mathit{fix}\ f \sqsubseteq x$, and not the totally correct $\mathit{fix}\ f = x$. We use this characterisation to show that if $unwrap$ is non-strict (i.e. $unwrap\ \bot \neq \bot$) then there are programs where worker/wrapper fusion as used by Gill and Hutton need only be partially correct.

Consider the scenario described in Figure 1. After applying the worker/wrapper transformation, we attempt to apply fusion by finding a residual expression $body'$ such that the body of the worker, i.e. the expression $unwrap \circ body \circ wrap$, can be rewritten as $body' \circ unwrap \circ wrap$. Intuitively this is the semantic form of workers where all self-calls are fusible. Our goal is to justify redefining $work$ to $\mathit{fix}\ body'$, i.e. to establish:

$$\mathit{fix}\ (unwrap \circ body \circ wrap) = \mathit{fix}\ body'$$

We can show partial correctness by elaborating the proof by Gill and Hutton in their §3:

$$\begin{aligned} & work \\ = & \quad \{ \text{apply } work\text{, apply computation: } \mathit{fix}\ f = f\ (\mathit{fix}\ f)\text{, unapply } work \} \\ & (unwrap \circ body \circ wrap)\ work \\ = & \quad \{ \text{apply assumption: } unwrap \circ body \circ wrap = body' \circ unwrap \circ wrap \} \\ & (body' \circ unwrap \circ wrap)\ work \\ = & \quad \{ \text{apply } work\text{, apply computation, unapply } work \} \\ & (body' \circ unwrap \circ wrap)\ ((unwrap \circ body \circ wrap)\ work) \\ = & \quad \{ \text{definition of } \circ \} \\ & (body' \circ unwrap \circ wrap \circ unwrap \circ body \circ wrap)\ work \\ = & \quad \{ \text{worker/wrapper assumption: } wrap \circ unwrap = id_A \} \\ & (body' \circ unwrap \circ body \circ wrap)\ work \\ = & \quad \{ \text{apply } \circ \text{ and } work\text{, apply computation, unapply } work \} \\ & body'\ work \end{aligned}$$

Hence $\mathit{fix}\ body' \sqsubseteq work$ by Park induction.

However it is not always the case that $work \sqsubseteq \mathit{fix}\ body'$: if *unwrap* is not strict, we can construct a $body'$ such that $\mathit{fix}\ body'$ is less defined than *work*. Consider, for example, the following two simple types:

$$\begin{array}{l} \textbf{data}\ A = A \\ \textbf{data}\ B = B\ A \end{array}$$

That is, A is a type with a single non-bottom element, and B is the non-strict lifting of A. Defining the functions *wrap* and *unwrap* for these types is straightforward:

$$\begin{array}{l} wrap\ ::\ B \to A \\ wrap\ (B\ a)\ =\ a \\ \\ unwrap\ ::\ A \to B \\ unwrap\ a\ =\ B\ a \end{array}$$

as is verifying the equation $wrap \circ unwrap = id_A$. The computation $comp = \mathit{fix}\ body$ we transform can be any where *body* uses the recursion parameter non-strictly, such as

$$\begin{array}{l} body\ ::\ A \to A \\ body\ r\ =\ A \end{array}$$

The example hinges on a definition that uses the recursion parameter strictly:

$$\begin{array}{l} body'\ ::\ B \to B \\ body'\ (B\ a)\ =\ B\ A \end{array}$$

Note that $unwrap \circ body \circ wrap = body' \circ unwrap \circ wrap$ due to the lifting in *unwrap*. However, fusing $unwrap \circ wrap$ as we did above yields:

$$\mathit{fix}\ (unwrap \circ body \circ wrap) = B\ A \not\sqsubseteq \bot = \mathit{fix}\ body'$$

This trick can be performed whenever A has at least one element and *unwrap* is not strict, which implies that we cannot expect to find an equational fusion rule without imposing extra conditions. The next section demonstrates that a strict *unwrap* is sufficient.

3 A termination-preserving fusion rule

We now show that a termination-preserving worker/wrapper fusion rule can be obtained by requiring *unwrap* to be strict. Note that *wrap* must always be strict due to the assumption that $wrap \circ unwrap = id_A$. Generalising from the starting point of the previous section, we expect that the following equation has been established:

$$unwrap \circ body \circ wrap = \lambda r.\, body'\ r\ ((unwrap \circ wrap)\ r)$$

The two parameters of $body'$ model unfusible and fusible self-calls respectively. We show

$$\mathit{fix}\ (unwrap \circ body \circ wrap) = \mathit{fix}\ (\lambda r.\, body'\ r\ r).$$

For a recursive definition $comp = body$ of type A and a pair of functions $wrap :: B \to A$ and $unwrap :: A \to B$ where $wrap \circ unwrap = id_A$ and $unwrap \perp = \perp$, define

$$\begin{aligned} comp &= wrap\ work \\ work &= unwrap\ (body[wrap\ work/comp]) \end{aligned} \qquad \text{(the worker/wrapper transformation)}$$

In the scope of *work*, the following rewrite is admissable:

$$unwrap\ (wrap\ work) \Longrightarrow work \qquad \text{(worker/wrapper fusion)}$$

Fig. 2. The syntactic worker/wrapper transformation and fusion rule.

which justifies worker/wrapper fusion in the context of the worker.

We proceed by Scott, or fixed-point, induction (see §4.2.4 of Müller *et al.* 1999): for admissible predicates P, if $P(\perp)$, and $P(x)$ implies $P(f\ x)$, then $P(\mathit{fix}\ f)$. Intuitively our P must assert that the worker lies within the part of B where $unwrap \circ wrap$ acts as the identity, which suggests this predicate:

$$P(f', g') \equiv f' = g' \wedge (unwrap \circ wrap)\, f' = f'$$

Clearly P is admissible and the assumptions about *wrap* and *unwrap* imply $P(\perp, \perp)$. The inductive case follows by standard equational reasoning.

A syntactically oriented version of this rule is shown in Figure 2; the scoping of the fusion rule ensures that correctness follows directly from the semantically oriented original.

Those familiar with the 'bananas' work of Meijer *et al.* (1991) will not be surprised that adding a strictness assumption justifies an equational fusion rule.

4 Concluding remarks

Gill and Hutton provide two examples of fusion: accumulator introduction in their §4, and the transformation in their §7 of an interpreter for a language with exceptions into one employing continuations. Both involve strict *unwrap*s and are indeed totally correct.

The example in their §5 demonstrates the unboxing of numerical computations using a different worker/wrapper rule and does not require fusion. In their §6 a non-strict *unwrap* is used to memoise functions over the natural numbers using the rule considered here. It should in fact use the same rule as the unboxing example as the scheme only correctly memoises strict functions. We can see this by considering a base case missing from their inductive proof, viz that if $f :: Nat \to a$ is not strict – in fact constant, as Nat is a flat domain – then $f \perp \neq \perp = (map\ f\ [0..])\ !!\ \perp$, where $xs\ !!\ n$ is the nth element of xs.

Acknowledgments

Much of this work was carried out while I was an Australian Youth Ambassador for Development in T. P. Hồ Chí Minh, Việt Nam, funded by the Australian Government via AusAID. I thank Kai Engelhardt, Brian Huffman, Clem Baker-Finch, Bernie

Pope, Peter Rickwood, Colin Runciman, Josef Svenningsson and the anonymous reviewers.

References

Burstall, R. M. & Darlington, J. (1977) A transformation system for developing recursive programs, *J. ACM*, 24 (1): 44–67.

Gammie, P. (2009) The worker/wrapper transformation. In *The Archive of Formal Proofs*, Klein, G., Nipkow, T. & Paulson, L. (eds). Available at: `http://afp.sf.net/entries/WorkerWrapper.shtml`. Formal proof development.

Gill, A. & Hutton, G. (2009). The worker/wrapper transformation. *J. Funct. Program.*, 19 (2): 227–251.

Huffman, B. (2009). A purely definitional universal domain. In Berghofer, S., Nipkow, T., Urban, C. & Wenzel, M. (eds), *TPHOLs*. LNCS, vol. 5674, pp. 260–275.

Meijer, E., Fokkinga, M. & Paterson, R. (1991). Functional programming with bananas, lenses, envelopes and barbed wire. In *Proceedings of the Conference on Functional Programming and Computer Architecture*. Cambridge, MA, USA, pp. 124–144.

Müller, O., Nipkow, T., von Oheimb, D. & Slotosch, O. (1999). HOLCF = HOL + LCF. *J. Funct. Program.*, **9**: 191–223.

Tullsen, M. (2002). *PATH, A Program Transformation System for Haskell*. PhD thesis, New Haven, CT: Yale University.