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

PETER GAMMIE
The Australian National University
(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 and Hutton (2009) and proposes a new totally correct worker/wrapper fusion rule.

1 Introduction

The worker/wrapper transformation has been formalised by Gill and 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 and 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, Nipkow, von Oheimb and Slotosch (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 right-to-left. (The unfold rule treats the current working set of definitions as rewrite rules oriented left-to-right.) Hence as each definition \( f = \text{body} \) yields a rule of the form \( \text{body} \Rightarrow f \), one can always derive \( f = f \). Clearly this has dire implications for the preservation of termination behaviour.
For a recursive definition \( \text{comp} = \text{fix \ body} \) for some \( \text{body} :: A \rightarrow A \) and a pair of functions \( \text{wrap} :: B \rightarrow A \) and \( \text{unwrap} :: A \rightarrow B \) where \( \text{wrap} \circ \text{unwrap} = \text{id}_A \), we have:

\[
\text{comp} = \text{wrap \ work}
\]

\[
\text{work} :: B \rightarrow A
\]

\[
\text{work} = \text{fix (unwrap \circ \text{body} \circ \text{wrap})}
\]

Also:

\[
(\text{unwrap} \circ \text{wrap}) \text{ work} = \text{work}
\]

(worker/wrapper fusion)

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

Tulsen (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 \( \text{fix} f \subseteq x \), and not the totally correct \( \text{fix} f = x \). We use this characterisation to show that if \( \text{unwrap} \) is non-strict (i.e. \( \text{unwrap} \perp \neq \perp \)) 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 \( \text{body}' \) such that the body of the worker, i.e. the expression \( \text{unwrap} \circ \text{body} \circ \text{wrap} \), can be rewritten as \( \text{body}' \circ \text{unwrap} \circ \text{wrap} \). Intuitively this is the semantic form of workers where all self-calls are fusible. Our goal is to justify redefining \( \text{work} \) to \( \text{fix body}' \), i.e. to establish:

\[
\text{fix (unwrap \circ body \circ wrap)} = \text{fix body}'
\]

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

\[
\text{work}
\]

\[
= \{ \text{apply work, apply computation: fix} f = f (\text{fix} f), \text{unapply work} \}
\]

\[
(\text{unwrap} \circ \text{body} \circ \text{wrap}) \text{ work}
\]

\[
= \{ \text{apply assumption: } \text{unwrap} \circ \text{body} \circ \text{wrap} = \text{body}' \circ \text{unwrap} \circ \text{wrap} \}
\]

\[
(\text{body}' \circ \text{unwrap} \circ \text{wrap}) \text{ work}
\]

\[
= \{ \text{apply work, apply computation, unapply work} \}
\]

\[
(\text{body}' \circ \text{unwrap} \circ \text{wrap}) ((\text{unwrap} \circ \text{body} \circ \text{wrap}) \text{ work})
\]

\[
= \{ \text{definition of } \circ \}
\]

\[
(\text{body}' \circ \text{unwrap} \circ \text{wrap} \circ \text{unwrap} \circ \text{body} \circ \text{wrap}) \text{ work}
\]

\[
= \{ \text{worker/wrapper assumption: } \text{wrap} \circ \text{unwrap} = \text{id}_A \}
\]

\[
(\text{body}' \circ \text{unwrap} \circ \text{body} \circ \text{wrap}) \text{ work}
\]

\[
= \{ \text{apply } \circ \text{ and work, apply computation, unapply work} \}
\]

\[
\text{body}' \text{ work}
\]

Hence \( \text{fix body}' \subseteq \text{work} \) by Park induction.

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

\[\text{data } A = A\]

\[\text{data } B = B A\]

That is, \( A \) is a type with a single non-bottom element, and \( B \) is the non-strict lifting of \( A \). Defining the functions \( \text{wrap} \) and \( \text{unwrap} \) for these types is straightforward:
Short Note: Strict unwraps make worker/wrapper fusion totally correct

\[ \text{wrap} :: B \rightarrow A \]
\[ \text{wrap} (B a) = a \]

\[ \text{unwrap} :: A \rightarrow B \]
\[ \text{unwrap} a = B a \]

as is verifying the equation \( \text{wrap} \circ \text{unwrap} = \text{id}_A \). The computation \( \text{comp} = \text{fix body} \) we transform can be any where \( \text{body} \) uses the recursion parameter non-strictly, such as:

\[ \text{body} :: A \rightarrow A \]
\[ \text{body} r = A \]

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

\[ \text{body}' :: B \rightarrow B \]
\[ \text{body}' (B a) = B A \]

Note that \( \text{unwrap} \circ \text{body} \circ \text{wrap} = \text{body}' \circ \text{unwrap} \circ \text{wrap} \) due to the lifting in \( \text{unwrap} \). However, fusing \( \text{unwrap} \circ \text{wrap} \) as we did above yields:

\[ \text{fix} (\text{unwrap} \circ \text{body} \circ \text{wrap}) = B A \not\sqsubseteq \bot \]

This trick can be performed whenever \( A \) has at least one element and \( \text{unwrap} \) 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 \( \text{unwrap} \) is sufficient.

3 A termination-preserving fusion rule

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

\[ \text{unwrap} \circ \text{body} \circ \text{wrap} = \lambda r. \text{body}' r ((\text{unwrap} \circ \text{wrap}) r) \]

The two parameters of \( \text{body}' \) model unfusible and fusible self-calls respectively. We show:

\[ \text{fix} (\text{unwrap} \circ \text{body} \circ \text{wrap}) = \text{fix} (\lambda r. \text{body}' r r) \]

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 \( P(\bot) \), and \( P(x) \) implies \( P(f x) \), then \( P(\text{fix} f) \). Intuitively our \( P \) must assert that the worker lies within the part of \( B \) where \( \text{unwrap} \circ \text{wrap} \) acts as the identity, which suggests this predicate:

\[ P(f', g') \equiv f' = g' \land (\text{unwrap} \circ \text{wrap}) f' = f' \]

Clearly \( P \) is admissible and the assumptions about \( \text{wrap} \) and \( \text{unwrap} \) imply \( P(\bot, \bot) \). 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 Fokkinga, Meijer and Paterson (1991) will not be surprised that adding a strictness assumption justifies an equational fusion rule.
For a recursive definition $comp = body$ of type $A$ and a pair of functions $wrap :: B \rightarrow A$ and $unwrap :: A \rightarrow B$ where $\text{wrap} \circ \text{unwrap} = \text{id}_A$ and $\text{unwrap} \perp = \perp$, define:

\[
\begin{align*}
\text{comp} &= \text{wrap work} \\
\text{work} &= \text{unwrap} \left( \text{body}[\text{wrap work}/\text{comp}] \right)
\end{align*}
\]

(the worker/wrapper transformation)

In the scope of $work$, the following rewrite is admissible:

\[
\text{unwrap} \left( \text{wrap work} \right) \Rightarrow \text{work}
\]

(worker/wrapper fusion)

Fig. 2. The syntactic worker/wrapper transformation and 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 $f :: \text{Nat} \rightarrow a$ is not strict – in fact constant, as $\text{Nat}$ is a flat domain – then $f \perp \neq \perp = (\text{map } f \left[ 0.. \right]) !! \perp$, where $xs !! n$ is the $n$th 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


