Corrections to “A Synthesis of Several Sorting Algorithms” by J. Darlington

Dean Jacobs¹ and Martin Feather²

¹ Department of Computer Science, University of Southern California, Los Angeles,
CA 90089, USA

² Information Sciences Institute, University of Southern California, 4676 Admiralty Way,
Marina del Rey, CA 90292, USA

Summary. Darlington applied program transformation to derive a number of sorting algorithms. We identify and correct two errors in his derivation of quick sort, one of which causes his resulting algorithm to be non-terminating on some inputs, the other of which could have led to an incorrect algorithm, but is fortuitously eliminated later in his derivation.

In his paper “A Synthesis of Several Sorting Algorithms” [1], John Darlington derives a Quick Sort algorithm (S1), which reads:

$$\begin{aligned}
 \text{Sort}(X) &\Leftarrow \text{Sort1}([X], X) \\
 \text{Sort1}(\langle \rangle, \emptyset) &\Leftarrow \{\emptyset\} \\
 \text{Sort1}(\langle n \rangle, \{x\}) &\Leftarrow \{\{\langle nx \rangle\}\} \\
 \text{Sort1}(N, X) &\Leftarrow \{f1 \cup f2 \mid f1 \in \text{Sort1}(N^k, Y) \text{ } f2 \in \text{Sort1}(N_n, X - Y)\} \\
 &\quad \text{where } k = \text{Card}(Y) \text{ where } Y = \text{Filter}(X) \\
 \text{Filter}(\emptyset) &\Leftarrow \emptyset \\
 \text{Filter}(x + X) &\Leftarrow \text{Filter1}(x, X) \\
 \text{Filter1}(x, \emptyset) &\Leftarrow \emptyset \\
 \text{Filter1}(x, x1 + X) &\Leftarrow x1 + \text{Filter1}(x, X) \text{ if } x1 < x \\
 &\quad \text{Filter1}(x, X) \text{ otherwise}
 \end{aligned}$$

using $[X]$, N^k , and N_k defined as follows:

$$[X] \equiv \langle 1, 2, \dots, \text{Card}(X) \rangle$$

if N is the sequence

$$\langle n_1, n_2, \dots, n_l \rangle, N^k \equiv \langle n_1, n_2, \dots, n_k \rangle$$

$$N_k \equiv \langle n_{k+1}, \dots, n_l \rangle$$

This algorithm, however, does not necessarily terminate¹. For example:

¹ Jian Xu was responsible for uncovering this

$$\begin{aligned}
\text{Sort}(\{5, 7\}) &\Leftarrow \text{Sort } 1([\{5, 7\}], \{5, 7\}) \\
&\Leftarrow \text{Sort } 1(\langle 1, 2 \rangle, \{5, 7\}) \quad \text{***** Note this call on Sort } 1. \\
&\Leftarrow \{f1 \cup f2 \mid f1 \in \text{Sort } 1(\langle 1, 2 \rangle^k, Y) \ f2 \in \text{Sort } 1(\langle 1, 2 \rangle_k, \{5, 7\} - Y)\} \\
&\quad \text{where } k = \text{Card}(Y) \\
&\quad \text{where } Y = \text{Filter}(\{5, 7\}) \\
\text{Filter}(\{5, 7\}) &\Leftarrow \text{Filter } 1(5, \{7\}) \\
&\Leftarrow \{7\} + \text{Filter } 1(5, \emptyset) \text{ if } 7 < 5 \\
&\quad \text{Filter } 1(5, \emptyset) \text{ otherwise} \\
&\Leftarrow \text{Filter } 1(5, \emptyset) \\
&\Leftarrow \emptyset
\end{aligned}$$

so, continuing evaluation of $\text{Sort}(\{5, 7\})$,

$$\begin{aligned}
\text{Sort}(\{5, 7\}) &\Leftarrow \{f1 \cup f2 \mid f1 \in \text{Sort } 1(\langle 1, 2 \rangle^k, Y) \ f2 \in \text{Sort } 1(\langle 1, 2 \rangle_k, \{5, 7\} - Y)\} \\
&\quad \text{where } k = \text{Card}(Y) \text{ where } Y = \emptyset \\
&\Leftarrow \{f1 \cup f2 \mid f1 \in \text{Sort } 1(\langle 1, 2 \rangle^k, \emptyset) \ f2 \in \text{Sort } 1(\langle 1, 2 \rangle_k, \{5, 7\} - \emptyset)\} \\
&\quad \text{where } k = 0 \\
&\Leftarrow \{f1 \cup f2 \mid f1 \in \text{Sort } 1(\langle 1, 2 \rangle^0, \emptyset) \ f2 \in \text{Sort } 1(\langle 1, 2 \rangle_0, \{5, 7\})\} \\
&\Leftarrow \{f1 \cup f2 \mid f1 \in \text{Sort } 1(\langle \rangle, \emptyset) \ f2 \in \text{Sort } 1(\langle 1, 2 \rangle, \{5, 7\})\} \\
&\Leftarrow \{f1 \cup f2 \mid f1 \in \{\emptyset\} \ f2 \in \text{Sort } 1(\langle 1, 2 \rangle, \{5, 7\})\} \\
&\Leftarrow \{\emptyset \cup f2 \mid f2 \in \text{Sort } 1(\langle 1, 2 \rangle, \{5, 7\})\} \\
&\Leftarrow \{f2 \mid f2 \in \text{Sort } 1(\langle 1, 2 \rangle, \{5, 7\})\} \\
&\Leftarrow \text{Sort } 1(\langle 1, 2 \rangle, \{5, 7\})
\end{aligned}$$

i.e., an infinite recursion on $\text{Sort } 1(\langle 1, 2 \rangle, \{5, 7\})$.

We might expect non-termination to arise from an injudicious fold step, since folding is guaranteed to preserve only partial correctness (e.g., suppose $f(x) \Leftarrow 1$, then a legal fold that leads to obvious non-termination is: $f(x) \Leftarrow f(x)$). However, investigation of Darlington's derivation reveals the non-termination stems in this case from an incorrect rewriting in his step 9, which reads:

$$\begin{aligned}
\text{Sort } 1(N, X) &\Leftarrow \{f1 \cup f2 \mid f1 \in \text{Sort } 1(N^k, Y) \ f2 \in \text{Sort } 1(N_k, X - Y)\} \\
&\quad Y \subset_k X \\
&\quad Y < X - Y \\
&\text{where } S < T \text{ means } \forall s \in S, t \in T \bullet s < t. \\
&\quad \text{The conditions on } Y, Y \subset_k X \text{ and } Y < X - Y, \\
&\quad \text{mean that for any } k \text{ there is just one such } Y. \\
&\quad \text{Thus we can rewrite the above as:} \\
\text{Sort } 1(N, X) &\Leftarrow \{f1 \cup f2 \mid f1 \in \text{Sort } 1(N^k, Y) \ f2 \in \text{Sort } 1(N_k, X - Y)\} \\
&\quad \text{where } k = \text{Card}(Y) \\
&\quad \text{for some } Y \subset X \text{ such that } Y < X - Y.
\end{aligned}$$

In step 2 of Darlington's derivation of *P1*, the set X is decomposed into subsets Y of size k using the identity

$$X = \bigcup_{Y \subset_k X} Y.$$

A particular value of k is not chosen; any value satisfying $1 \leq k < \text{Card}(X)$ may be used. The derivation proceeds to step 9 above just before a value of k is selected. However, the rewritten formula is incorrect because it permits $Y = \emptyset$ and $k = 0$. Instead, the rewritten version should read:

$$\begin{aligned} \text{Sort } 1(N, X) \Leftarrow & \{f1 \cup f2 \mid f1 \in \text{Sort } 1(N^k, Y) \text{ } f2 \in \text{Sort } 1(N_k, X - Y) \\ & \text{where } k = \text{Card}(Y) \\ & \text{for some } Y \subset X \text{ such that } Y < X - Y \text{ and } \text{Card}(Y) > 0. \end{aligned}$$

This leads to a modified definition of *Filter*:

$$\begin{aligned} \text{Filter}(X) \Leftarrow & \text{some } Y \text{ such that } Y \subset X \text{ and } Y < (X - Y) \text{ and } \text{Card}(Y) > 0 \\ & \text{where } \text{Card}(X) > 1. \end{aligned}$$

Manipulating *Filter*, we get:

$$\begin{aligned} \text{Filter}(e + X) \Leftarrow & \text{some } Y \text{ such that } Y \subset e + X \text{ and } Y < (e + X - Y) \\ & \text{and } \text{Card}(Y) > 0 \\ \Leftarrow & \text{if } \{e\} < X \\ & \text{then some } Y \text{ such that } Y \subset e + X \text{ and } Y < (e + X - Y) \\ & \quad \text{and } \text{Card}(Y) > 0 \\ & \text{else some } Y \text{ such that } Y \subset e + X \text{ and } Y < (e + X - Y) \\ & \quad \text{and } \text{Card}(Y) > 0 \\ & \quad \text{[inserting a conditional with identical branches]} \\ \Leftarrow & \text{if } \{e\} < X \\ & \text{then } \{e\} \\ & \text{else } \{d \in X \mid d < e\} \\ & \quad \text{[choosing an appropriate } Y \text{ in each branch of the conditional]} \\ \Leftarrow & \text{if } \{d \in X \mid d < e\} = \emptyset . \\ & \text{then } \{e\} \\ & \text{else } \{d \in X \mid d < e\} \\ & \quad \text{[rewriting the test of the conditional]} \\ \Leftarrow & \text{if } S = \emptyset \\ & \text{then } \{e\} \\ & \text{else } S \\ & \quad \text{where } S = \text{Filter } 1(e, X) \\ & \quad \text{[folding into a new function } \text{Filter } 1 \text{ defined as:]} \\ & \quad \text{Filter } 1(e, X) \Leftarrow \{d \in X \mid d < e\}. \end{aligned}$$

We synthesize a recursive definition for *Filter1* as follows:

$$\begin{aligned}
 \text{Filter1}(e) &\Leftarrow [\text{unfolding}] \\
 \text{Filter1}(e, a + X) &\Leftarrow \{d \in a + X \mid d < e\} \\
 &\Leftarrow \text{if } a < e \\
 &\quad \text{then } \{d \in a + X \mid d < e\} \\
 &\quad \text{else } \{d \in a + X \mid d < e\} \\
 &\Leftarrow \text{if } a < e \\
 &\quad \text{then } a + \{d \in X \mid d < e\} \\
 &\quad \text{else } \{d \in X \mid d < e\} \\
 &\Leftarrow \text{if } a < e \\
 &\quad \text{then } a + \text{Filter1}(e, X) \\
 &\quad \text{else } \text{Filter1}(e, X).
 \end{aligned}$$

Our synthesized *Filter1*(*e*, *X*) function is identical to Darlington's – it computes all the elements of *X* that are less than *e*. Our *Filter*(*X*) differs in that it takes care to always generate a *non-empty* proper subset of *X*.

Note that Darlington's *specification* of *Filter1* introduced in his step 11, namely

$$\text{Filter1}(e, X) \Leftarrow Y \text{ such that } Y \subset X \text{ and } \forall y \in Y \bullet y < e$$

need not return *all* the elements of *X* that are less than *e*. However, he synthesizes from this a version *Filter1*(*e*, *X*) that does compute the entire subset of *X* whose elements are less than *e*. This synthesis is fortuitous indeed, because the other possibilities admitted by the specification would be incorrect when used by Darlington's *Filter*, and ultimately, *Sort*, for example:

$$\begin{aligned}
 \text{Filter}(\{5, 3, 4, 6\}) &\Leftarrow \text{Filter1}(5, \{3, 4, 6\}) \\
 &\Leftarrow Y \text{ such that } Y \subset \{3, 4, 6\} \text{ and } \forall y \in Y \bullet y < 5 \\
 &\Leftarrow \{4\}.
 \end{aligned}$$

Yet *Sort* relies upon *Filter*(*X*) to return a subset of *X*, each element of which is less than the elements of the remainder of *X*, whereas clearly it is not the case that $\{4\} < \{5, 3, 6\}$.

The error arises in his derivation step 12, where he claims that

$$x \in X, Y \subset X \text{ and } \forall y \in Y \bullet y < x \text{ implies } Y < X - Y$$

This implication does not hold (e.g., $x = 5$, $X = \{5, 3, 4, 6\}$, $Y = \{4\}$), and so his *Filter* cannot be rewritten to fold with his specified *Filter1*.

Remarks

We would like to stress that Darlington's overall derivation is almost unchanged. We have merely uncovered a couple of small errors that admit to an easy correction. Thus Darlington's main purpose, namely exposing relationships

among a family tree of sorting algorithms, and demonstrating the power of program transformation techniques, remain uncompromised.

This exercise reinforces the fact that hand-conducted transformational derivations can easily contain hard-to-discern errors that might or might not compromise the derived programs. We look to machine-assisted transformation to increase our confidence in derivations. Well-constructed transformation systems reduce the likelihood of error through checking many of the steps, and clearly delineating the other steps as further lemmas upon whose correctness the entire derivation rests. (Such systems cannot, of course, be guaranteed to be correct – they too might accept incorrect derivations.) The primitive state of the art of current machine-assisted transformation systems is such that their use proves a significant impediment, rather than an aid, to conducting derivations of significant intricacy and complexity. We hope that this situation will be remedied in the near future.

References

1. Darlington, J.: A synthesis of several sorting algorithms. *Acta Inf.* **11**, 1–30 (1978)

Received April 1, 1987/April 12, 1988