List reversal: back into the frying pan

Richard Bornat

March 20, 2006

Abstract

More than thirty years ago Rod Burstall showed how to do a proof of a neat little program, shown in a modern notation in figure 1(a) [4]. The program, which was well understood by programmers even before Burstall's efforts, reverses the order of a straight-line linked list of cells in the heap using only three variables and without moving or copying anything in the heap. An algorithm which does copy cells in order to produce a separate reversed list, is shown in figure 1(b). A copying algorithm which doesn't use assignment is shown in figure 1(c).

Each of the algorithms has its advantages and disadvantages: reverse by copying means that you don't disturb the original list and you have a separate copy to play with, but if you don't need the original again it's wasteful of time and space; copying without assignment is even more wasteful because it needs a stack of procedure executions, unless you have a compiler clever enough to recognise that it's equivalent to the with-assignment version. Copying without assignment was historically the first to have a specification and a proof, then copying with assignment and finally in-place list reversal. The in-place reversal problem influenced early developments in separation logic, by showing us so clearly that separation was important [3]. A proof in separation logic is shown in figure 3, using the heap predicate listp defined in figure 2. Note that the predicate pins down the precise location in memory of the cells of the list as well as the values of the sequence that they represent, so that the proof can show that the reversed list uses exactly the cells of the original list with exactly their original value content.

I concentrate on in-place reversal because it remains the most economical and the most 'dangerous' of the three algorithms, and because it has a useful peculiarity: it terminates even if the input list is circular, whereas the other two terminate only on finite straight-line lists. More peculiarly still, it terminates even if the input list has a handle and then a circle, a 'frying pan' list as in figure 4, which reads 1, 2, 3, 4, 5, 6, 7, 8, 4, 5, 6, ... It reverses the handle, then goes round the pan reversing it, then back up the handle de-reversing it, finishing up just where it started, so figure 4 'reversed' reads 1, 2, 3, 8, 7, 6, 5, 4, 8, 7, 6, ... It works because the nil which it puts in y is inserted into the heap by $[z + 1] := y$, and so acts a termination marker when it's seen again. The copying algorithms don't change the original list, so they go round and round the pan for ever looking for a nil that isn't there.

1 Proof of frying-pan 'reversal'

The copying algorithms don't work on the structure in figure 4 because it isn't a list. Lists represent sequences, and sequences don't have loops. To specify what the in-place list reversal algorithm does

```
y := nil;
while x ≠ nil do
   z := x; x := [x + 1]; [z + 1] := y; y := z
od
```

(a) 'in-place'

```
y := nil;
while x ≠ nil do
   z := [x]; x := [x + 1]; y := cons(w, y)
od
```

(b) copying

```
let f z =
   if z = nil then nil else cons([z], f [z + 1]) fi
in f z ni
```

(c) copying without assignment

Figure 1: reversing a list

$$\begin{array}{rcl} \text{listp}\, x\, \langle\,\rangle & \hat{=} & x = \text{nil} \wedge \mathbf{emp} \\ \text{listp}\, x\, (\langle (n, c)\rangle +\!\!+ xs) & \hat{=} & x = c \wedge \exists x' \cdot (x \mapsto n, x' \star \text{listp}\, x'\, xs) \end{array}$$

Figure 2: a heap predicate for straight-line lists

1. $\{\text{listp}\, x\, L\}$
2. $\quad y := \text{nil};$
3. $\{\text{listp}\, x\, L \wedge y = \text{nil}\}\ \therefore$
4. $\{\text{listp}\, x\, L \star \text{listp}\, y\, \langle\,\rangle\}\ \therefore$
5. $\{\exists xs, ys \cdot (\ (\text{listp}\, x\, xs \star \text{listp}\, y\, ys) \wedge \text{rev}\, xs +\!\!+ ys = \text{rev}\, L\ )\}$
6. $\quad \text{while } x \neq \text{nil do}$
7. $\qquad \{\exists xs, ys \cdot (\ (\text{listp}\, x\, xs \star \text{listp}\, y\, ys) \wedge \text{rev}\, xs +\!\!+ ys = \text{rev}\, L\ ) \wedge x \neq \text{nil}\}\ \therefore$
8. $\qquad \{\exists n, xs', ys \cdot (\ (\text{listp}\, x\, (\langle (n, x)\rangle +\!\!+ xs') \star \text{listp}\, y\, ys) \wedge \text{rev}(\langle n\rangle +\!\!+ xs') +\!\!+ ys = \text{rev}\, L\ )\}\ \therefore$
9. $\qquad \{\exists n, x', xs', ys \cdot (\ (x \mapsto n, x' \star \text{listp}\, x'\, xs' \star \text{listp}\, y\, ys) \wedge \text{rev}\, xs' +\!\!+ \langle (n, x)\rangle +\!\!+ ys = \text{rev}\, L\ )\}$
10. $\qquad\quad z := x$
11. $\qquad \{\exists x', xs', ys \cdot (\ (x \mapsto n, x' \star \text{listp}\, x'\, xs' \star \text{listp}\, y\, ys) \wedge \text{rev}\, xs' +\!\!+ \langle (n, x)\rangle +\!\!+ ys = \text{rev}\, L\ ) \wedge z = x\}$
12. $\qquad \{\exists x', xs', ys \cdot (\ (z \mapsto n, x' \star \text{listp}\, x'\, xs' \star \text{listp}\, y\, ys) \wedge \text{rev}\, xs' +\!\!+ \langle (n, z)\rangle +\!\!+ ys = \text{rev}\, L\ ) \wedge x = z\}$
13. $\qquad\quad x := [x + 1]$
14. $\qquad \{\exists n, xs', ys \cdot (\ (z \mapsto n, x \star \text{listp}\, x\, xs' \star \text{listp}\, y\, ys) \wedge \text{rev}\, xs' +\!\!+ \langle (n, z)\rangle +\!\!+ ys = \text{rev}\, L\ )\}$
15. $\qquad\quad [z + 1] := y$
16. $\qquad \{\exists n, xs', ys \cdot (\ (z \mapsto n, y \star \text{listp}\, x\, xs' \star \text{listp}\, y\, ys) \wedge \text{rev}\, xs' +\!\!+ \langle (n, z)\rangle +\!\!+ ys = \text{rev}\, L\ )\}\ \therefore$
17. $\qquad \{\exists n, xs', ys \cdot (\ (\text{listp}\, x\, xs' \star \text{listp}\, z\, (\langle (n, z)\rangle + ys)) \wedge \text{rev}\, xs' +\!\!+ \langle (n, z)\rangle +\!\!+ ys = \text{rev}\, L\ )\}\ \therefore$
18. $\qquad \{\exists xs', ys' \cdot (\ (\text{listp}\, x\, xs' \star \text{listp}\, z\, ys') \wedge \text{rev}\, xs' +\!\!+ ys' = \text{rev}\, L\ )\}$
19. $\qquad\quad y := z$
20. $\qquad \{\exists xs', ys' \cdot (\ (\text{listp}\, x\, xs' \star \text{listp}\, y\, ys') \wedge \text{rev}\, xs' +\!\!+ ys' = \text{rev}\, L\ )\}$
21. $\quad \text{od}$
22. $\{\exists xs, ys \cdot (\ (\text{listp}\, x\, xs \star \text{listp}\, y\, ys) \wedge \text{rev}\, xs +\!\!+ ys = \text{rev}\, L\ ) \wedge xs = \text{nil}\}\ \therefore$
23. $\{\text{listp}\, y\, (\text{rev}\, L)\}$

Figure 3: a separation-logic proof of in-place list reversal

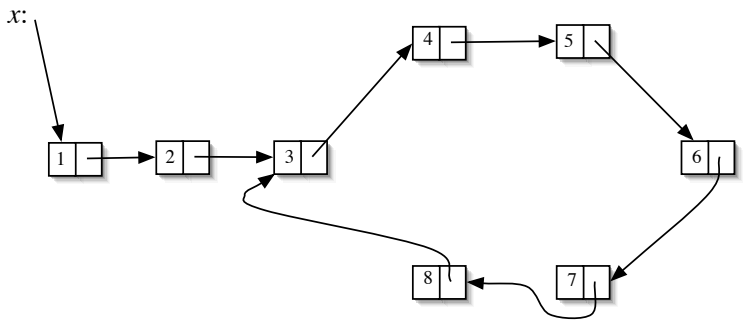


Figure 4: a frying-pan list

$$\begin{array}{rcl} \mathrm{lsp}\, x\ x'\ \langle\,\rangle & \hat{=} & x = x' \wedge \mathbf{emp} \\ \mathrm{lsp}\, x\ x'\ (\langle (n, c)\rangle +\!\!+ xs) & \hat{=} & x \neq x' \wedge x = c \wedge \exists x'' \cdot (x \mapsto n, x'' \star \mathrm{lsp}\, x''\ xs) \end{array}$$

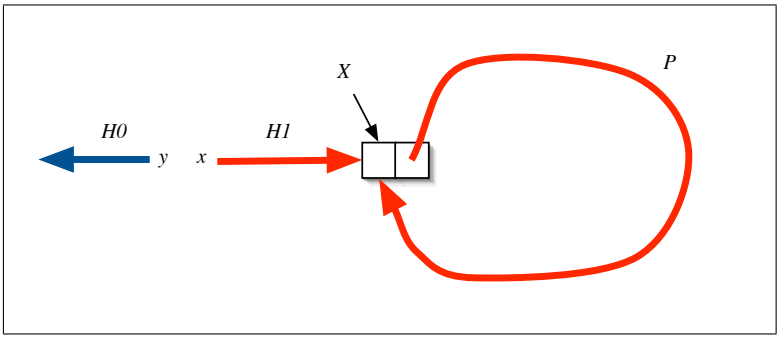
Figure 5: a heap predicate for straight-line list segments

with the frying pan, we need another heap predicate which describes a 'list segment', a straight-line list of cells which doesn't necessarily end with nil. The heap predicate is defined in figure 5: $\mathrm{listp}\, x\ L$ is equivalent to $\mathrm{lsp}\, x$ nil L. It's clear that a frying-pan list is made up of list segments. If the point where the handle joins the circle (the cell containing 3 in figure 4) is X then there's a segment which is the handle – $\mathrm{lsp}\, x\ X\ H$, the cells containing 1 and 2 in figure 4. Then you might suppose that the pan is defined by $\mathrm{lsp}\, X\ X\ P$, but the definition of lsp makes that the empty segment.[1] The pan always contains at least the cell pointed to by X, so there are two possibilities: that cell points to itself ($X \mapsto n, X \wedge P = \langle (n, X)\rangle$) or it points to another cell which is the start of a list segment ending in X ($\exists j \cdot ((X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P') \wedge P = \langle (n, j)\rangle +\!\!+ P')$). And then we can see that the singleton case is subsumed in the general case ($X = j \wedge P = \langle\,\rangle$), and I can write a specification in terms of H, the handle, X, the join point, and P', the rest of the pan (which, from now on, I'm going to call P rather than P'.)

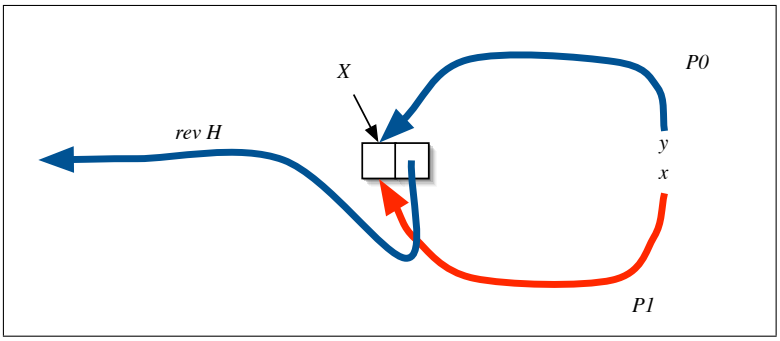
The hard bit in almost every program verification is guessing the invariant. There are three distinct stages in the algorithm, shown in figure 6: it seems inevitable that the invariant will be a three-way disjunction. In figures 7 and 8 H, P, X and n are universally quantified. The invariant (line 6 in each figure) is easily transformed, by three uses of the lsp definition, into

$$\exists j \cdot \left(\begin{array}{l} \exists x', n', H0, H1' \cdot \left(\begin{array}{l} (x \mapsto n', x' \star \mathrm{lsp}\, x'\ X\ H1' \star \mathrm{lsp}\, y\ \mathrm{nil}\ H0 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge \\ \mathrm{rev}(\langle (n', x)\rangle +\!\!+ H1') +\!\!+ H0 = \mathrm{rev}\, H \wedge x \neq X \end{array} \right) \vee \\ \exists H0 \cdot ((\mathrm{lsp}\, y\ \mathrm{nil}\ H0 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge \mathrm{rev}(\langle\,\rangle) +\!\!+ H0 = \mathrm{rev}\, H \wedge x = X) \vee \\ \exists x', n', P0, P1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star x \mapsto n', x' \star \mathrm{lsp}\, x'\ X\ P1' \star X \mapsto n, j \star \mathrm{lsp}\, y\ X\ P0) \wedge \\ \mathrm{rev}(\langle (n', x)\rangle +\!\!+ P1') +\!\!+ P0 = \mathrm{rev}\, P \wedge x \neq X \end{array} \right) \vee \\ \exists P0 \cdot ((\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star \mathrm{lsp}\, y\ X\ P0) \wedge \mathrm{rev}(\langle\,\rangle) +\!\!+ P0 = \mathrm{rev}\, P \wedge x = X) \vee \\ \exists x', n', H0', H1 \cdot \left(\begin{array}{l} (x \mapsto n', x' \star \mathrm{lsp}\, x'\ \mathrm{nil}\ H0' \star \mathrm{lsp}\, y\ X\ H1 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ (\mathrm{rev}\, P)) \wedge \\ \mathrm{rev}(\langle (n', x)\rangle +\!\!+ H0') +\!\!+ H1 = H \wedge x \neq \mathrm{nil} \end{array} \right) \vee \\ \exists H1 \cdot ((\mathrm{lsp}\, y\ X\ H1 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ (\mathrm{rev}\, P)) \wedge \mathrm{rev}(\langle\,\rangle) +\!\!+ H1 = H \wedge x = \mathrm{nil}) \end{array} \right) \quad (1)$$

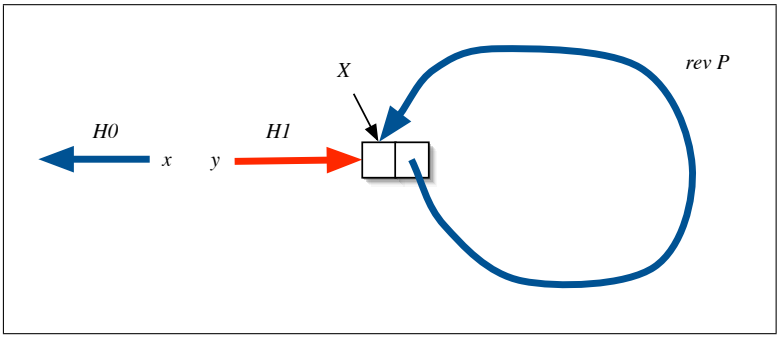
[1] In [3] I allowed list segments to be frying pans, and the third argument of the predicate dealt with ambiguities at the join point. I have learnt something since then: straight-line segments make a longer specification, but they also make a proof possible.



(a) going down the handle



(b) going round the pan



(c) coming back up the handle

Figure 6: stages of frying-pan 'reversal'

1. $\{\exists j \cdot (\ \mathrm{lsp}\, x\ X\ H \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P\ )\}$
2. $\quad y := \mathrm{nil};$
3. $\{\exists j \cdot (\ \mathrm{lsp}\, x\ X\ H \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P\ ) \wedge y = \mathrm{nil}\}\ \therefore$
4. $\{\exists j \cdot (\ \mathrm{lsp}\, x\ X\ H \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P \star \mathrm{lsp}\, y\ \mathrm{nil}\ \langle\rangle\ )\}\ \therefore$
5. $\{\exists j, H0, H1 \cdot (\ (\mathrm{lsp}\, x\ X\ H1 \star \mathrm{lsp}\, y\ \mathrm{nil}\ H0 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge \mathrm{rev}\, H1 +\!\!+ H0 = \mathrm{rev}\, H\ )\}\ \therefore$
6. $\left\{\ \exists j \cdot \left(\begin{array}{l} \exists H0, H1 \cdot ((\mathrm{lsp}\, x\ X\ H1 \star \mathrm{lsp}\, y\ \mathrm{nil}\ H0 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge \mathrm{rev}\, H1 +\!\!+ H0 = \mathrm{rev}\, H) \vee \\ \exists P0, P1 \cdot ((\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star \mathrm{lsp}\, x\ X\ P1 \star X \mapsto n, j \star \mathrm{lsp}\, y\ X\ P0) \wedge \mathrm{rev}\, P1 +\!\!+ P0 = \mathrm{rev}\, P) \vee \\ \exists H0, H1 \cdot ((\mathrm{lsp}\, x\ \mathrm{nil}\ H0 \star \mathrm{lsp}\, y\ X\ H1 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ (\mathrm{rev}\, P)) \wedge \mathrm{rev}\, H0 +\!\!+ H1 = H) \end{array} \right)\ \right\}$

Figure 7: a separation-logic proof of frying-pan 'reversal' (initialisation establishes invariant)

6. $\left\{ \exists j \cdot \left(\begin{array}{l} \exists H0, H1 \cdot ((\mathrm{lsp}\, x\ X\ H1 \star \mathrm{lsp}\, y\ \mathrm{nil}\ H0 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge \mathrm{rev}\, H1 +\!\!+ H0 = \mathrm{rev}\, H) \vee \\ \exists P0, P1 \cdot ((\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star \mathrm{lsp}\, x\ X\ P1 \star X \mapsto n, j \star \mathrm{lsp}\, y\ X\ P0) \wedge \mathrm{rev}\, P1 +\!\!+ P0 = \mathrm{rev}\, P) \vee \\ \exists H0, H1 \cdot ((\mathrm{lsp}\, x\ \mathrm{nil}\ H0 \star \mathrm{lsp}\, y\ X\ H1 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ (\mathrm{rev}\, P)) \wedge \mathrm{rev}\, H0 +\!\!+ H1 = H) \end{array} \right) \right\}$

7. while $x \neq \mathrm{nil}$ do

8. $\left\{ \begin{array}{l} \exists j \cdot \left(\begin{array}{l} \exists x', n', H0, H1' \cdot \left(\begin{array}{l} (x \mapsto n', x' \star \mathrm{lsp}\, x'\ X\ H1' \star \mathrm{lsp}\, y\ \mathrm{nil}\ H0 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge \\ \quad \mathrm{rev}\, H1' +\!\!+ \langle (n', x) \rangle +\!\!+ H0 = \mathrm{rev}\, H \wedge x \neq X \end{array} \right) \vee \\ ((\mathrm{lsp}\, y\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge x = X) \vee \\ \exists x', n', P0, P1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star x \mapsto n', x' \star \mathrm{lsp}\, x'\ X\ P1' \star \mathrm{lsp}\, y\ X\ P0) \wedge \\ \mathrm{rev}\, P1' +\!\!+ \langle (n', x) \rangle +\!\!+ P0 = \mathrm{rev}\, P \wedge x \neq X \end{array} \right) \vee \\ ((\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star \mathrm{lsp}\, y\ X\ (\mathrm{rev}\, P)) \wedge x = X) \vee \\ \exists x', n', H0', H1 \cdot \left(\begin{array}{l} (x \mapsto n', x' \star \mathrm{lsp}\, x'\ \mathrm{nil}\ H0' \star \mathrm{lsp}\, y\ X\ H1 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ (\mathrm{rev}\, P)) \wedge \\ \quad \mathrm{rev}\, H0' +\!\!+ \langle (n', x) \rangle +\!\!+ H1 = H \wedge x \neq \mathrm{nil} \end{array} \right) \end{array} \right) \end{array} \right\}$

9. $z := x$

10. $\left\{ \begin{array}{l} x = z \wedge \exists j \cdot \left(\begin{array}{l} \exists x', n', H0, H1' \cdot \left(\begin{array}{l} (z \mapsto n', x' \star \mathrm{lsp}\, x'\ X\ H1' \star \mathrm{lsp}\, y\ \mathrm{nil}\ H0 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge \\ \quad \mathrm{rev}\, H1' +\!\!+ \langle (n', z) \rangle +\!\!+ H0 = \mathrm{rev}\, H \wedge z \neq X \end{array} \right) \vee \\ ((\mathrm{lsp}\, y\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge z = X) \vee \\ \exists x', n', P0, P1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star z \mapsto n', x' \star \mathrm{lsp}\, x'\ X\ P1' \star \mathrm{lsp}\, y\ X\ P0) \wedge \\ \mathrm{rev}\, P1' +\!\!+ \langle (n', z) \rangle +\!\!+ P0 = \mathrm{rev}\, P \wedge z \neq X \end{array} \right) \vee \\ ((\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star \mathrm{lsp}\, y\ X\ (\mathrm{rev}\, P)) \wedge z = X) \vee \\ \exists x', n', H0', H1 \cdot \left(\begin{array}{l} (z \mapsto n', x' \star \mathrm{lsp}\, x'\ \mathrm{nil}\ H0' \star \mathrm{lsp}\, y\ X\ H1 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ (\mathrm{rev}\, P)) \wedge \\ \quad \mathrm{rev}\, H0' +\!\!+ \langle (n', z) \rangle +\!\!+ H1 = H \wedge z \neq \mathrm{nil} \end{array} \right) \end{array} \right) \end{array} \right\}$

11. $x := [x + 1]$

12. $\left\{ \begin{array}{l} \exists j \cdot \left(\begin{array}{l} \exists n', H0, H1' \cdot \left(\begin{array}{l} (z \mapsto n', x \star \mathrm{lsp}\, x\ X\ H1' \star \mathrm{lsp}\, y\ \mathrm{nil}\ H0 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge \\ \quad \mathrm{rev}\, H1' +\!\!+ \langle (n', z) \rangle +\!\!+ H0 = \mathrm{rev}\, H \wedge z \neq X \end{array} \right) \vee \\ ((\mathrm{lsp}\, y\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge z = X \wedge x = j) \vee \\ \exists n', P0, P1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star z \mapsto n', x \star \mathrm{lsp}\, x\ X\ P1' \star \mathrm{lsp}\, y\ X\ P0) \wedge \\ \mathrm{rev}\, P1' +\!\!+ \langle (n', z) \rangle +\!\!+ P0 = \mathrm{rev}\, P \wedge z \neq X \end{array} \right) \vee \\ ((\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star \mathrm{lsp}\, y\ X\ (\mathrm{rev}\, P)) \wedge z = X \wedge x = j) \vee \\ \exists n', H0', H1 \cdot \left(\begin{array}{l} (z \mapsto n', x \star \mathrm{lsp}\, x\ \mathrm{nil}\ H0' \star \mathrm{lsp}\, y\ X\ H1 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ (\mathrm{rev}\, P)) \wedge \\ \quad \mathrm{rev}\, H0' +\!\!+ \langle (n', z) \rangle +\!\!+ H1 = H \wedge z \neq \mathrm{nil} \end{array} \right) \end{array} \right) \end{array} \right\}$

13. $[z + 1] := y$

14. $\left\{ \begin{array}{l} \exists j \cdot \left(\begin{array}{l} \exists n', H0, H1' \cdot \left(\begin{array}{l} (z \mapsto n', y \star \mathrm{lsp}\, x\ X\ H1' \star \mathrm{lsp}\, y\ \mathrm{nil}\ H0 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge \\ \quad \mathrm{rev}\, H1' +\!\!+ \langle (n', z) \rangle +\!\!+ H0 = \mathrm{rev}\, H \wedge z \neq X \end{array} \right) \vee \\ ((\mathrm{lsp}\, y\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, y \star \mathrm{lsp}\, j\ X\ P) \wedge z = X \wedge x = j) \vee \\ \exists n', P0, P1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star z \mapsto n', y \star \mathrm{lsp}\, x\ X\ P1' \star \mathrm{lsp}\, y\ X\ P0) \wedge \\ \mathrm{rev}\, P1' +\!\!+ \langle (n', z) \rangle +\!\!+ P0 = \mathrm{rev}\, P \wedge z \neq X \end{array} \right) \vee \\ ((\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, y \star \mathrm{lsp}\, y\ X\ (\mathrm{rev}\, P)) \wedge z = X \wedge x = j) \vee \\ \exists n', H0', H1 \cdot \left(\begin{array}{l} (z \mapsto n', y \star \mathrm{lsp}\, x\ \mathrm{nil}\ H0' \star \mathrm{lsp}\, y\ X\ H1 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ (\mathrm{rev}\, P)) \wedge \\ \quad \mathrm{rev}\, H0' +\!\!+ \langle (n', z) \rangle +\!\!+ H1 = H \wedge z \neq \mathrm{nil} \end{array} \right) \end{array} \right) \end{array} \right\} \therefore$

15. $\left\{ \begin{array}{l} \exists j \cdot \left(\begin{array}{l} \exists H0', H1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, x\ X\ H1' \star \mathrm{lsp}\, z\ \mathrm{nil}\ H0' \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge \\ \quad \mathrm{rev}\, H1' +\!\!+ H0' = \mathrm{rev}\, H \end{array} \right) \vee \\ ((\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star \mathrm{lsp}\, x\ X\ P) \wedge z = X) \vee \\ \exists P0', P1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star \mathrm{lsp}\, x\ X\ P1' \star \mathrm{lsp}\, z\ X\ P0') \wedge \\ \mathrm{rev}\, P1' +\!\!+ P0' = \mathrm{rev}\, P \wedge z \neq X \end{array} \right) \vee \\ ((\mathrm{lsp}\, x\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, y \star \mathrm{lsp}\, y\ X\ (\mathrm{rev}\, P)) \wedge z = X) \vee \\ \exists H0', H1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, x\ \mathrm{nil}\ H0' \star \mathrm{lsp}\, z\ X\ H1' \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ (\mathrm{rev}\, P)) \wedge \\ \quad \mathrm{rev}\, H0' +\!\!+ H1' = H \wedge z \neq \mathrm{nil} \end{array} \right) \end{array} \right) \end{array} \right\}$

16. $y := z$

17. $\left\{ \begin{array}{l} \exists j \cdot \left(\begin{array}{l} \exists H0', H1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, x\ X\ H1' \star \mathrm{lsp}\, y\ \mathrm{nil}\ H0' \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge \\ \quad \mathrm{rev}\, H1' +\!\!+ H0' = \mathrm{rev}\, H \end{array} \right) \vee \\ ((\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star \mathrm{lsp}\, x\ X\ P) \wedge y = X) \vee \\ \exists P0', P1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star \mathrm{lsp}\, x\ X\ P1' \star \mathrm{lsp}\, y\ X\ P0') \wedge \\ \mathrm{rev}\, P1' +\!\!+ P0' = \mathrm{rev}\, P \wedge y \neq X \end{array} \right) \vee \\ ((\mathrm{lsp}\, x\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, y \star \mathrm{lsp}\, y\ X\ (\mathrm{rev}\, P)) \wedge y = X) \vee \\ \exists H0', H1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, x\ \mathrm{nil}\ H0' \star \mathrm{lsp}\, y\ X\ H1' \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ (\mathrm{rev}\, P)) \wedge \\ \quad \mathrm{rev}\, H0' +\!\!+ H1' = H \wedge y \neq \mathrm{nil} \end{array} \right) \end{array} \right) \end{array} \right\} \therefore$

18. $\left\{ \begin{array}{l} \exists j \cdot \left(\begin{array}{l} \exists H0', H1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, x\ X\ H1' \star \mathrm{lsp}\, y\ \mathrm{nil}\ H0' \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge \\ \quad \mathrm{rev}\, H1' +\!\!+ H0' = \mathrm{rev}\, H \end{array} \right) \vee \\ \exists P0', P1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star \mathrm{lsp}\, x\ X\ P1' \star \mathrm{lsp}\, y\ X\ P0') \wedge \\ \mathrm{rev}\, P1' +\!\!+ P0' = \mathrm{rev}\, P \wedge y \neq X \end{array} \right) \vee \\ \exists H0', H1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, x\ \mathrm{nil}\ H0' \star \mathrm{lsp}\, y\ X\ H1' \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ (\mathrm{rev}\, P)) \wedge \\ \quad \mathrm{rev}\, H0' +\!\!+ H1' = H \wedge y \neq \mathrm{nil} \end{array} \right) \end{array} \right) \end{array} \right\}$

19. od

20. $\{(\mathrm{lsp}\, y\ X\ H \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ (\mathrm{rev}\, P)) \wedge x = \mathrm{nil}\}$

Figure 8: a separation-logic proof of frying-pan 'reversal' (loop preserves invariant)

$y := \text{nil}; p := 0;$
while $x \neq \text{nil}$ do
  $z := x;\ x := [x+1];\ [z+1] := y;\ y := z;$
  if $y = X$ then $p := p + 1$ else skip fi
od

Figure 9: in-place list reversal with auxiliary phase variable p

which simplifies, using properties of rev, to

$$\exists j \cdot \left(\begin{array}{l} \exists x', n', H0, H1' \cdot \left(\begin{array}{l} (x \mapsto n', x' \star \mathrm{lsp}\, x'\ X\ H1' \star \mathrm{lsp}\, y\ \mathrm{nil}\ H0 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge \\ \quad \mathrm{rev}\, H1' +\!\!+ \langle (n', x) \rangle +\!\!+ H0 = \mathrm{rev}\, H \wedge x \neq X \end{array} \right) \vee \\ ((\mathrm{lsp}\, y\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ P) \wedge x = X) \vee \\ \exists x', n', P0, P1' \cdot \left(\begin{array}{l} (\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star x \mapsto n', x' \star \mathrm{lsp}\, x'\ X\ P1' \star \mathrm{lsp}\, y\ X\ P0) \wedge \\ \mathrm{rev}\, P1' +\!\!+ \langle (n', x) \rangle +\!\!+ P0 = \mathrm{rev}\, P \wedge x \neq X \end{array} \right) \vee \\ ((\mathrm{lsp}\, j\ \mathrm{nil}\ (\mathrm{rev}\, H) \star X \mapsto n, j \star \mathrm{lsp}\, y\ X\ (\mathrm{rev}\, P)) \wedge x = X) \vee \\ \exists x', n', H0', H1 \cdot \left(\begin{array}{l} (x \mapsto n', x' \star \mathrm{lsp}\, x'\ \mathrm{nil}\ H0' \star \mathrm{lsp}\, y\ X\ H1 \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ (\mathrm{rev}\, P)) \wedge \\ \quad \mathrm{rev}\, H0' +\!\!+ \langle (n', x) \rangle +\!\!+ H1 = H \wedge x \neq \mathrm{nil} \end{array} \right) \vee \\ ((\mathrm{lsp}\, y\ X\ H \star X \mapsto n, j \star \mathrm{lsp}\, j\ X\ (\mathrm{rev}\, P)) \wedge x = \mathrm{nil}) \end{array} \right) \quad (2)$$

This equivalence is used to generate lines 8 and 20 of figure 8, and the proof is then simply mechanical.

2 Is that it?

Well no, it isn't quite. To show that the algorithm terminates I need a measure, and the measure can't be a disjunction, but it still has to take account of the stages shown in figure 6. The switch between stages happens when $y = X$, and if I add an auxiliary variable p and manipulate it as in figure 9, the measure is

$$\text{if } p = 0 \text{ then } 2 + \text{length } H1 + \text{length } P \text{ elsf } p = 1 \text{ then } 1 + \text{length } P1 \text{ else length } H0 \text{ fi} \quad (3)$$

– if you see what I mean, of course, because $H1$ and $P1$ are existentially quantified in the invariant. So I'm not going to go through the tedious business of showing that this invariant reduces just yet.

3 A mechanical safety proof

I used Smallfoot [1] to check that my proof wasn't entirely mistaken. In trying to do so I found out how the phase variable works. The Smallfoot control file is shown in figure 10. Note that I had to use case analysis to modify p because Smallfoot (quite reasonably) can't deal with any arithmetic, I had to assign to j because Smallfoot can't yet deal with $\exists$ quantification, and worst of all I had to use the abominable C syntax for assignment and equality and I had to put semicolons in the strange places that C demands them.

Acknowledgements

I didn't get to this point unaided. I knew almost immediately after doing a Hoare-logic proof of in-place reversal for [3] that the algorithm terminated on frying-pan lists, but I had no notion of

```
hd,tl;

fryingpanrev(x) [lseg(x,X) * X|->tl:j * lseg(j,X)] {
  local z;
  p = 0;
  y = NULL;
  while (x!=NULL)
   [if p==0 then (lseg(y,NULL)*lseg(x,X)*X|->tl:j*lseg(j,X))
    else
    if p==1 then (lseg(j,NULL)*X|->tl:j*lseg(y,X)*lseg(x,X))
    else
    (lseg(x,NULL)*lseg(y,X)*X|->tl:j*lseg(j,X))] {
    z = x; x = x->tl; z->tl = y; y = z;
    if (y==X) {
      if (p==0) p = 1; else p = 2;
      j = X->tl;
    }
  }
} [lseg(y,X) * X|->tl:j * lseg(j,X)]
```

Figure 10: a Smallfoot control file for in-place list reversal

how to specify or prove it. A year or so later, Cristiano Calcagno and Josh Berdine were working on SAW, a precursor of Smallfoot [1, 2], struggling to understand what was right and what was wrong with my definition of list segment, and the frying pan came up often in our conversations. They eventually settled on straight-line segments like those in figure 5, but by then I think we'd given up on the frying pan problem. Then, in March 2006, Dino Distefano showed a demo of the separation-logic shape analysis tool SpaceInvader that he and Peter O'Hearn had been developing [5]. It took a description of a frying-pan list and spat out lots of text amongst which was the remark "ok" and an eleven-way disjunction labelled "invariant". I was startled, and asked Dino if he had analysed the invariant. He hadn't, and neither so far have I, but it was those eleven lines that inspired me to try again to make a manual proof, and it was Cris and Josh's straight-line list segments that made it work.

Peter O'Hearn's name 'panhandle list' needed correcting, and I have done so. At the time of writing, Matthew Parkinson has not yet interfered with the ideas in this note, and neither has Hongseok Yang.

References

[1] Josh Berdine and Cristiano Calcagno. Smallfoot software. http://www.dcs.qmul.ac.uk/research/logic/theory/projects/smallfoot/index.html.

[2] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Modular automatic assertion checking with separation logic. submitted, 2005.

[3] Richard Bornat. Proving pointer programs in Hoare logic. In R. C. Backhouse and J. N. Oliveira, editors, *Mathematics of Program Construction, 5th International Conference*, LNCS,

pages 102–126. Springer, 2000.

[4] R.M. Burstall. Some techniques for proving correctness of programs which alter data structures. *Machine Intelligence*, 7:23–50, 1972.

[5] Dino Distefano, Peter O'Hearn, and Hongseok Yang. A local shape analysis based on separation logic. To appear in TACAS'06, 2006.