An Introduction
to Separation Logic
©2007 John C. Reynolds
February 28, 2007

Chapter 6

Iterated Separating Conjunction

In this chapter, we introduce an iterative version of the separating conjunction that is useful in describing arrays, as well as certain properties of list structures.

6.1 A New Form of Assertion

We extend the language of assertions with an binding operator $\bigodot$, which is used to construct an assertion of the form

$$\bigodot_{v=e}^{e'} p,$$

where the occurrence of v in the subscript is a binder whose scope is p. Roughly speaking, this assertion describes the separating conjunction

$$(p/v \to e) \ast (p/v \to e+1) \ast \cdots \ast (p/v \to e').$$

More precisely, for a state s, h, let $m = [\![e]\!]_{\mathrm{exp}} s$ and $n = [\![e']\!]_{\mathrm{exp}} s$ be the *lower* and *upper bounds*, and $I = \{\, i \mid m \le i \le n \,\}$ be the set of *indices*. Then $s, h \models \bigodot_{v=e}^{e'} p$ iff there is a function $H \in I \to \mathrm{Heaps}$ that partitions h into an indexed set of heaps,

$$h = \bigcup \{\, Hi \mid i \in I \,\} \text{ and } \forall i, j \in I.\ i \neq j \text{ implies } Hi \perp Hj,$$

such that, for all indices $i \in I$, $[\, s \mid v\colon i \,], Hi \models p$.

This new form satisfies the following axiom schemata, in which, for readability, we have written $p(e)$ for $p/i \to e$:

$$m > n \;\Rightarrow\; \left(\bigodot_{i=m}^{n} p(i) \Leftrightarrow \mathbf{emp}\right) \tag{6.1}$$

$$m = n \;\Rightarrow\; \left(\bigodot_{i=m}^{n} p(i) \Leftrightarrow p(m)\right) \tag{6.2}$$

$$k \le m \le n+1 \;\Rightarrow\; \left(\bigodot_{i=k}^{n} p(i) \Leftrightarrow \left(\bigodot_{i=k}^{m-1} p(i) \;*\; \bigodot_{i=m}^{n} p(i)\right)\right) \tag{6.3}$$

$$\bigodot_{i=m}^{n} p(i) \Leftrightarrow \bigodot_{i=m-k}^{n-k} p(i+k) \tag{6.4}$$

$$m \le n \;\Rightarrow\; \left(\left(\bigodot_{i=m}^{n} p(i)\right) \;*\; q \Leftrightarrow \bigodot_{i=m}^{n}(p(i) \;*\; q)\right) \tag{6.5}$$
$$\text{when } q \text{ is pure and } i \notin \mathrm{FV}(q)$$

$$m \le n \;\Rightarrow\; \left(\left(\bigodot_{i=m}^{n} p(i)\right) \wedge q \Leftrightarrow \bigodot_{i=m}^{n}(p(i) \wedge q)\right) \tag{6.6}$$
$$\text{when } q \text{ is pure and } i \notin \mathrm{FV}(q)$$

$$m \le j \le n \;\Rightarrow\; \left(\left(\bigodot_{i=m}^{n} p(i)\right) \Rightarrow (p(j) \;*\; \mathbf{true})\right). \tag{6.7}$$

6.2 Arrays

The most obvious use of the iterated separating conjunction is to describe arrays that occur in the heap. To allocate such arrays, as discussed in Section 1.8, we introduce the command

$$\langle\text{comm}\rangle ::= \cdots \mid \langle\text{var}\rangle := \mathbf{allocate}\ \langle\text{exp}\rangle$$

The effect of $v := \mathbf{allocate}\ e$ is to allocate a block of size e, and to assign the address of the first element to v. The initialization of the array elements is not specified.

The inference rules for this new command are similar to those for the ordinary allocation of records:

- The local nonoverwriting form (ALLOCNOL)

$$\frac{}{\{\mathbf{emp}\}\ v := \mathbf{allocate}\ e\ \{\bigodot_{i=v}^{v+e-1} i \mapsto -\},}$$

where $v \notin \mathrm{FV}(e)$.

- The global nonoverwriting form (ALLOCNOG)

$$\frac{}{\{r\}\ v := \mathbf{allocate}\ e\ \{(\bigodot_{i=v}^{v+e-1} i \mapsto -) \ast r\},}$$

where $v \notin \mathrm{FV}(e, r)$.

- The local form (ALLOCL)

$$\frac{}{\{v = v' \wedge \mathbf{emp}\}\ v := \mathbf{allocate}\ e\ \{\bigodot_{i=v}^{v+e'-1} i \mapsto -\},}$$

where v' is distinct from v, and e' denotes $e/v \to v'$.

- The global form (ALLOCG)

$$\frac{}{\{r\}\ v := \mathbf{allocate}\ e\ \{\exists v'.\ (\bigodot_{i=v}^{v+e'-1} i \mapsto -) \ast r'\},}$$

where v' is distinct from v, $v' \notin \mathrm{FV}(e, r)$, e' denotes $e/v \to v'$, and r' denotes $r/v \to v'$.

- The backward-reasoning form (ALLOCBR)

$$\frac{}{\{\forall v''.\ (\bigodot_{i=v''}^{v''+e-1} i \mapsto -) -\!\!\ast\ p''\}\ v := \mathbf{allocate}\ e\ \{p\},}$$

where v'' is distinct from v, $v'' \notin \mathrm{FV}(e, p)$, and p'' denotes $p/v \to v''$.

Usually, (one-dimensional) arrays are used to represent sequences. We define

$$\mathsf{array}\ \alpha\,(\mathsf{a}, \mathsf{b}) \stackrel{\mathrm{def}}{=} \#\alpha = \mathsf{b} - \mathsf{a} + 1 \wedge \bigodot_{\mathsf{i}=\mathsf{a}}^{\mathsf{b}} \mathsf{i} \mapsto \alpha_{\mathsf{i}-\mathsf{a}+1}.$$

When $\mathsf{array}\ \alpha\,(\mathsf{a}, \mathsf{b})$ holds, we say that a to b (more precisely, the heap from a to b) represents the sequence α.

Notice that, since the length of a sequence is never negative, the assertion $\mathsf{array}\ \alpha\,(\mathsf{a}, \mathsf{b})$ implies that $\mathsf{a} \leq \mathsf{b} + 1$. In fact, it would be consistent to define a to b to represent the empty sequence when $\mathsf{a} > \mathsf{b} + 1$ (as well as $\mathsf{a} = \mathsf{b} + 1$), but we will not use such "irregular" representations in this book. (An integrated approach to regular and irregular represetations is discussed in Reference [33, Chapter 2].)

This new form of assertion satisfies the following axioms:

$$
\begin{aligned}
\mathsf{array}\ \alpha\,(\mathsf{a},\mathsf{b}) &\Rightarrow \#\alpha = \mathsf{b} - \mathsf{a} + 1 \\
\mathsf{array}\ \alpha\,(\mathsf{a},\mathsf{b}) &\Rightarrow \mathsf{i} \hookrightarrow \alpha_{\mathsf{i}-\mathsf{a}+1} \quad \text{when } \mathsf{a} \le \mathsf{i} \le \mathsf{b} \\
\mathsf{array}\ \epsilon\,(\mathsf{a},\mathsf{b}) &\Leftrightarrow \mathsf{b} = \mathsf{a} - 1 \wedge \mathbf{emp} \\
\mathsf{array}\ \mathsf{x}\,(\mathsf{a},\mathsf{b}) &\Leftrightarrow \mathsf{b} = \mathsf{a} \wedge \mathsf{a} \mapsto \mathsf{x} \\
\mathsf{array}\ \mathsf{x}{\cdot}\alpha\,(\mathsf{a},\mathsf{b}) &\Leftrightarrow \mathsf{a} \mapsto \mathsf{x} \ast \mathsf{array}\ \alpha\,(\mathsf{a}+1,\mathsf{b}) \\
\mathsf{array}\ \alpha{\cdot}\mathsf{x}\,(\mathsf{a},\mathsf{b}) &\Leftrightarrow \mathsf{array}\ \alpha\,(\mathsf{a},\mathsf{b}-1) \ast \mathsf{b} \mapsto \mathsf{x} \\
\mathsf{array}\ \alpha\,(\mathsf{a},\mathsf{c}) \ast \mathsf{array}\ \beta\,(\mathsf{c}+1,\mathsf{b}) & \\
&\Leftrightarrow \mathsf{array}\ \alpha{\cdot}\beta\,(\mathsf{a},\mathsf{b}) \wedge \mathsf{c} = \mathsf{a} + \#\alpha - 1 \\
&\Leftrightarrow \mathsf{array}\ \alpha{\cdot}\beta\,(\mathsf{a},\mathsf{b}) \wedge \mathsf{c} = \mathsf{b} - \#\beta
\end{aligned}
$$

6.3 Partition

As an example, we present a program that, given an array representing a sequence, and a *pivot* value r, rearranges the sequence so that it splits into two contiguous parts, containing values smaller or equal to r and values larger than r, respectively. (This is a variant of the well-known program "Partition" by C. A. R. Hoare [44].)

$$
\begin{array}{l}
\{\mathsf{array}\ \alpha(\mathsf{a},\mathsf{b})\}\\
\textbf{newvar}\ \mathsf{d},\mathsf{x},\mathsf{y}\ \textbf{in}\ \big(\mathsf{c} := \mathsf{a}-1\ ;\ \mathsf{d} := \mathsf{b}+1\ ;\\
\{\exists\alpha_1,\alpha_2,\alpha_3.\ (\mathsf{array}\ \alpha_1\,(\mathsf{a},\mathsf{c})\ *\ \mathsf{array}\ \alpha_2\,(\mathsf{c}+1,\mathsf{d}-1)\ *\ \mathsf{array}\ \alpha_3\,(\mathsf{d},\mathsf{b}))\\
\quad \wedge\ \alpha_1{\cdot}\alpha_2{\cdot}\alpha_3 \sim \alpha \wedge \{\alpha_1\} \le^* \mathsf{r} \wedge \{\alpha_3\} >^* \mathsf{r}\}\\
\textbf{while}\ \mathsf{d} > \mathsf{c}+1\ \textbf{do}\ \big(\mathsf{x} := [\mathsf{c}+1];\\
\quad \textbf{if}\ \mathsf{x} \le \mathsf{r}\ \textbf{then}\\
\qquad \{\exists\alpha_1,\alpha_2,\alpha_3.\ (\mathsf{array}\ \alpha_1\,(\mathsf{a},\mathsf{c})\ *\ \mathsf{c}+1 \mapsto \mathsf{x}\ *\ \mathsf{array}\ \alpha_2\,(\mathsf{c}+2,\mathsf{d}-1)\\
\qquad\quad *\ \mathsf{array}\ \alpha_3\,(\mathsf{d},\mathsf{b})) \wedge \alpha_1{\cdot}\mathsf{x}{\cdot}\alpha_2{\cdot}\alpha_3 \sim \alpha \wedge \{\alpha_1{\cdot}\mathsf{x}\} \le^* \mathsf{r} \wedge \{\alpha_3\} >^* \mathsf{r}\}\\
\qquad \mathsf{c} := \mathsf{c}+1\\
\quad \textbf{else}\ \big(\mathsf{y} := [\mathsf{d}-1];\\
\qquad \textbf{if}\ \mathsf{y} > \mathsf{r}\ \textbf{then}\\
\qquad\quad \{\exists\alpha_1,\alpha_2,\alpha_3.\ (\mathsf{array}\ \alpha_1\,(\mathsf{a},\mathsf{c})\ *\ \mathsf{array}\ \alpha_2\,(\mathsf{c}+1,\mathsf{d}-2)\ *\ \mathsf{d}-1 \mapsto \mathsf{y}\\
\qquad\qquad *\ \mathsf{array}\ \alpha_3\,(\mathsf{d},\mathsf{b})) \wedge \alpha_1{\cdot}\alpha_2{\cdot}\mathsf{y}{\cdot}\alpha_3 \sim \alpha \wedge \{\alpha_1\} \le^* \mathsf{r} \wedge \{\mathsf{y}{\cdot}\alpha_3\} >^* \mathsf{r}\}\\
\qquad\quad \mathsf{d} := \mathsf{d}-1\\
\qquad \textbf{else}\\
\qquad\quad \{\exists\alpha_1,\alpha_2,\alpha_3.\ (\mathsf{array}\ \alpha_1\,(\mathsf{a},\mathsf{c})\ *\ \mathsf{c}+1 \mapsto \mathsf{x}\\
\qquad\qquad *\ \mathsf{array}\ \alpha_2\,(\mathsf{c}+2,\mathsf{d}-2)\ *\ \mathsf{d}-1 \mapsto \mathsf{y}\ *\ \mathsf{array}\ \alpha_3\,(\mathsf{d},\mathsf{b}))\\
\qquad\quad\ \wedge\ \alpha_1{\cdot}\mathsf{x}{\cdot}\alpha_2{\cdot}\mathsf{y}{\cdot}\alpha_3 \sim \alpha \wedge \{\alpha_1\} \le^* \mathsf{r} \wedge \{\alpha_3\} >^* \mathsf{r} \wedge \mathsf{x} > \mathsf{r} \wedge \mathsf{y} \le \mathsf{r}\}\\
\qquad\quad ([\mathsf{c}+1] := \mathsf{y}\ ;\ [\mathsf{d}-1] := \mathsf{x}\ ;\ \mathsf{c} := \mathsf{c}+1\ ;\ \mathsf{d} := \mathsf{d}-1)\big)\big)\big)\\
\{\exists\alpha_1,\alpha_2.\ (\mathsf{array}\ \alpha_1(\mathsf{a},\mathsf{c})\ *\ \mathsf{array}\ \alpha_2(\mathsf{c}+1,\mathsf{b}))\\
\quad \wedge\ \alpha_1{\cdot}\alpha_2 \sim \alpha \wedge \{\alpha_1\} \le^* \mathsf{r} \wedge \mathsf{r} <^* \{\alpha_2\}\}.
\end{array}
$$

For the most part, the reasoning here is straightforward. It should be noticed, however, that the assertion following the second **else** depends upon the validity of the implication

$$\mathsf{c}+1 \hookrightarrow \mathsf{x} \wedge \mathsf{d}-1 \hookrightarrow \mathsf{y} \wedge \mathsf{x} > \mathsf{r} \wedge \mathsf{y} \le \mathsf{r} \Rightarrow \mathsf{c}+1 \ne \mathsf{d}-1.$$

It is also straightforward to encapsulate the above program as a nonrecursive procedure. If we define

$$
\begin{array}{l}
\mathsf{partition}(\mathsf{a},\mathsf{b},\mathsf{r};\mathsf{c}) = \\
\quad \mathbf{newvar}\ \mathsf{d},\mathsf{x},\mathsf{y}\ \mathbf{in}\ \big(\mathsf{c} := \mathsf{a} - 1\ ;\ \mathsf{d} := \mathsf{b} + 1\ ; \\
\quad \mathbf{while}\ \mathsf{d} > \mathsf{c} + 1\ \mathbf{do} \\
\quad\quad \big(\mathsf{x} := [\mathsf{c}+1]\ ;\ \mathbf{if}\ \mathsf{x} \leq \mathsf{r}\ \mathbf{then}\ \mathsf{c} := \mathsf{c}+1\ \mathbf{else} \\
\quad\quad\quad \big(\mathsf{y} := [\mathsf{d}-1]\ ;\ \mathbf{if}\ \mathsf{y} > \mathsf{r}\ \mathbf{then}\ \mathsf{d} := \mathsf{d}-1\ \mathbf{else} \\
\quad\quad\quad\quad ([\mathsf{c}+1] := \mathsf{y}\ ;\ [\mathsf{d}-1] := \mathsf{x}\ ;\ \mathsf{c} := \mathsf{c}+1\ ;\ \mathsf{d} := \mathsf{d}-1)\big)\big)\big),
\end{array}
$$

then $\mathsf{partition}$ satisfies

$$
\begin{array}{l}
\{\mathsf{array}\ \alpha(\mathsf{a},\mathsf{b})\} \\
\mathsf{partition}(\mathsf{a},\mathsf{b},\mathsf{r};\mathsf{c}) \\
\{\exists \alpha_1, \alpha_2.\ (\mathsf{array}\ \alpha_1(\mathsf{a},\mathsf{c})\ *\ \mathsf{array}\ \alpha_2(\mathsf{c}+1,\mathsf{b})) \\
\quad \wedge\ \alpha_1{\cdot}\alpha_2 \sim \alpha \wedge \{\alpha_1\} \leq^* \mathsf{r} \wedge \mathsf{r} <^* \{\alpha_2\}\}.
\end{array}
$$

6.4 From Partition to Quicksort

We can use the procedure partition to define a version of the recursive sorting procedure called quicksort (which is again a variant of a well-known algorithm by Hoare [45]).

Since quicksort is recursive, we must state the specification of the procedure before we prove the correctness of its body, in order to reason about the recursive calls within the body. Fortunately, the specification is an obvious formalization of the requirements for a sorting procedure: We assume the specification

$$
\begin{array}{l}
\{\mathsf{array}\ \alpha\ (\mathsf{a},\mathsf{b})\} \\
\mathsf{quicksort}(\mathsf{a},\mathsf{b}) \\
\{\exists \beta.\ \mathsf{array}\ \beta\ (\mathsf{a},\mathsf{b}) \wedge \beta \sim \alpha \wedge \mathbf{ord}\ \beta\}.
\end{array}
\tag{6.8}
$$

(Notice that quicksort does not modify any variables.)

The basic idea behind quicksort is straightforward: One chooses a pivot value, partitions the array to be sorted into segments that are smaller or equal to the pivot and larger or equal to the pivot. Then one uses recursive calls to sort the two segments.

In our version, there is a complication because it is possible that one of the segments produced by the procedure partition will be empty while the other is the entire array to be sorted, in which case a naive version of quicksort will never terminate. (Consider, for example, the case where all elements of the array have the same value.) To circumvent this problem, we sort the end elements of the array separately, use their mean as the pivot value, and apply partition only to the interior of the array, so that the division of the entire array always has at least one element in each segment.

Then the following is an annotated specification of the body of quicksort:

$$
\begin{array}{l}
\{\mathsf{array}\ \alpha\,(\mathsf{a},\mathsf{b})\}\\
\textbf{if}\ \mathsf{a} < \mathsf{b}\ \textbf{then newvar}\ \mathsf{c}\ \textbf{in}\\
\quad\big(\{\exists \mathsf{x}_1, \alpha_0, \mathsf{x}_2.\ (\mathsf{a} \mapsto \mathsf{x}_1 \ * \ \mathsf{array}\ \alpha_0(\mathsf{a}+1, \mathsf{b}-1) \ * \ \mathsf{b} \mapsto \mathsf{x}_2)\\
\qquad \wedge\, \mathsf{x}_1{\cdot}\alpha_0{\cdot}\mathsf{x}_2 \sim \alpha\}\\
\quad \textbf{newvar}\ \mathsf{x1}, \mathsf{x2}, \mathsf{r}\ \textbf{in}\\
\qquad\big(\mathsf{x1} := [\mathsf{a}]\ ;\ \mathsf{x2} := [\mathsf{b}]\ ;\\
\qquad \textbf{if}\ \mathsf{x1} > \mathsf{x2}\ \textbf{then}\ ([\mathsf{a}] := \mathsf{x2}\ ;\ [\mathsf{b}] := \mathsf{x1})\ \textbf{else skip}\ ;\\
\qquad \mathsf{r} := (\mathsf{x1} + \mathsf{x2}) \div 2\ ;\\
\qquad \{\exists \mathsf{x}_1, \alpha_0, \mathsf{x}_2.\ (\mathsf{a} \mapsto \mathsf{x}_1 \ * \ \mathsf{array}\ \alpha_0(\mathsf{a}+1, \mathsf{b}-1) \ * \ \mathsf{b} \mapsto \mathsf{x}_2)\\
\qquad\quad \wedge\, \mathsf{x}_1{\cdot}\alpha_0{\cdot}\mathsf{x}_2 \sim \alpha \wedge \mathsf{x}_1 \le \mathsf{r} \le \mathsf{x}_2\}\\
\qquad \left.\left.\left.\begin{array}{l}
\{\mathsf{array}\ \alpha(\mathsf{a},\mathsf{b})\}\\
\mathsf{partition}(\mathsf{a},\mathsf{b},\mathsf{r};\mathsf{c})\\
\{\exists \alpha_1, \alpha_2.\ (\mathsf{array}\ \alpha_1(\mathsf{a},\mathsf{c})\\
\quad *\ \mathsf{array}\ \alpha_2(\mathsf{c}+1,\mathsf{b}))\\
\ \wedge\, \alpha_1{\cdot}\alpha_2 \sim \alpha\\
\ \wedge\, \{\alpha_1\} \le^* \mathsf{r} \wedge \mathsf{r} <^* \{\alpha_2\}\}
\end{array}\right\} \Big/ \left[\begin{array}{lcl}
\mathsf{a} & \to & \mathsf{a}{+}1\\
\mathsf{b} & \to & \mathsf{b}{-}1\\
\alpha & \to & \alpha_0
\end{array}\right.\right\} * \left[\begin{array}{c}
(\mathsf{a} \mapsto \mathsf{x}_1 * \mathsf{b} \mapsto \mathsf{x}_2)\\
\wedge\, \mathsf{x}_1{\cdot}\alpha_0{\cdot}\mathsf{x}_2 \sim \alpha\\
\wedge\, \mathsf{x}_1 \le \mathsf{r} \le \mathsf{x}_2
\end{array}\right.\right\} \exists \mathsf{x}_1, \alpha_0, \mathsf{x}_2\\
\qquad \{\exists \mathsf{x}_1, \alpha_1, \alpha_2, \mathsf{x}_2.\\
\qquad\quad (\mathsf{a} \mapsto \mathsf{x}_1 * \mathsf{array}\ \alpha_1(\mathsf{a}+1,\mathsf{c}) * \mathsf{array}\ \alpha_2(\mathsf{c}+1,\mathsf{b}-1) * \mathsf{b} \mapsto \mathsf{x}_2)\\
\qquad\quad \wedge\, \mathsf{x}_1{\cdot}\alpha_1{\cdot}\alpha_2{\cdot}\mathsf{x}_2 \sim \alpha \wedge \mathsf{x}_1 \le \mathsf{r} \le \mathsf{x}_2 \wedge \{\alpha_1\} \le^* \mathsf{r} \wedge \mathsf{r} <^* \{\alpha_2\}\}\big)\ ;\\
\quad \{\exists \alpha_1, \alpha_2.\ (\mathsf{array}\ \alpha_1\,(\mathsf{a},\mathsf{c}) \ * \ \mathsf{array}\ \alpha_2\,(\mathsf{c}+1,\mathsf{b}))\\
\qquad \wedge\, \alpha_1{\cdot}\alpha_2 \sim \alpha \wedge \{\alpha_1\} \le^* \{\alpha_2\}\}\\
\qquad \vdots
\end{array}
$$

$$
\begin{array}{l}
\quad\vdots \\
\quad\{\exists\alpha_1,\alpha_2.\ (\mathsf{array}\ \alpha_1\,(\mathsf{a},\mathsf{c})\ *\ \mathsf{array}\ \alpha_2\,(\mathsf{c}+1,\mathsf{b})) \\
\qquad \wedge\ \alpha_1{\cdot}\alpha_2 \sim \alpha \wedge \{\alpha_1\} \leq^* \{\alpha_2\}\} \\
\quad \left.\left.\begin{array}{l}
\{\mathsf{array}\ \alpha\,(\mathsf{a},\mathsf{b})\} \\
\mathsf{quicksort}(\mathsf{a},\mathsf{b}) \\
\{\exists\beta.\ \mathsf{array}\ \beta\,(\mathsf{a},\mathsf{b}) \\
\quad \wedge\ \beta \sim \alpha \wedge \mathbf{ord}\ \beta\}
\end{array}\right\} / \left[\begin{array}{lcl} \mathsf{b} & \rightarrow & \mathsf{c} \\ \alpha & \rightarrow & \alpha_1 \end{array}\right.\right\} * \left[\begin{array}{l} \mathsf{array}\ \alpha_2(\mathsf{c}+1,\mathsf{b}) \\ \quad\wedge\ \alpha_1{\cdot}\alpha_2 \sim \alpha \\ \quad\wedge\ \{\alpha_1\} \leq^* \{\alpha_2\} \end{array}\right.\left.\vphantom{\begin{array}{l}a\\a\\a\\a\end{array}}\right\}\exists\alpha_1,\exists\alpha_2 \\
\quad\{\exists\beta_1,\alpha_2.\ (\mathsf{array}\ \beta_1\,(\mathsf{a},\mathsf{c})\ *\ \mathsf{array}\ \alpha_2\,(\mathsf{c}+1,\mathsf{b})) \\
\qquad \wedge\ \beta_1{\cdot}\alpha_2 \sim \alpha \wedge \{\beta_1\} \leq^* \{\alpha_2\} \wedge \mathbf{ord}\ \beta_1\} \\
\quad \left.\left.\begin{array}{l}
\{\mathsf{array}\ \alpha\,(\mathsf{a},\mathsf{b})\} \\
\mathsf{quicksort}(\mathsf{a},\mathsf{b}) \\
\{\exists\beta.\ \mathsf{array}\ \beta\,(\mathsf{a},\mathsf{b}) \\
\quad \wedge\ \beta \sim \alpha \wedge \mathbf{ord}\ \beta\}
\end{array}\right\} / \left[\begin{array}{lcl} \mathsf{a} & \rightarrow & \mathsf{c}{+}1 \\ \alpha & \rightarrow & \alpha_2 \end{array}\right.\right\} * \left[\begin{array}{l} \mathsf{array}\ \beta_1(\mathsf{a},\mathsf{c}) \\ \quad\wedge\ \beta_1{\cdot}\alpha_2 \sim \alpha \\ \quad\wedge\ \{\beta_1\} \leq^* \{\alpha_2\} \end{array}\right.\left.\vphantom{\begin{array}{l}a\\a\\a\\a\end{array}}\right\}\exists\beta_1,\exists\alpha_2 \\
\quad\{\exists\beta_1,\beta_2.\ (\mathsf{array}\ \beta_1\,(\mathsf{a},\mathsf{c})\ *\ \mathsf{array}\ \beta_2\,(\mathsf{c}+1,\mathsf{b})) \\
\qquad \wedge\ \beta_1{\cdot}\beta_2 \sim \alpha \wedge \{\beta_1\} \leq^* \{\beta_2\} \wedge \mathbf{ord}\ \beta_1 \wedge \mathbf{ord}\ \beta_2\}\big) \\
\mathbf{else\ skip} \\
\{\exists\beta.\ \mathsf{array}\ \beta\,(\mathsf{a},\mathsf{b}) \wedge \beta \sim \alpha \wedge \mathbf{ord}\ \beta\}
\end{array}
$$

The pre- and postconditions of the above specification match those of the assumed specification 6.8. Moreover, the only free variables of the specified command are a and b, neither of which is modified. Thus we may satisfy 6.8

by using the command as the body of the procedure declaration:

$$
\begin{aligned}
&\mathsf{quicksort}(\mathsf{a},\mathsf{b}) = \\
&\quad \textbf{if } \mathsf{a} < \mathsf{b} \textbf{ then newvar } \mathsf{c} \textbf{ in} \\
&\quad\quad \big(\textbf{newvar } \mathsf{x1}, \mathsf{x2}, \mathsf{r} \textbf{ in} \\
&\quad\quad\quad \big(\mathsf{x1} := [\mathsf{a}]\, ;\, \mathsf{x2} := [\mathsf{b}]\, ; \\
&\quad\quad\quad \textbf{if } \mathsf{x1} > \mathsf{x2} \textbf{ then } ([\mathsf{a}] := \mathsf{x2}\, ;\, [\mathsf{b}] := \mathsf{x1}) \textbf{ else skip}\, ; \\
&\quad\quad\quad \mathsf{r} := (\mathsf{x1} + \mathsf{x2}) \div 2\, ;\, \mathsf{partition}(\mathsf{a}+1, \mathsf{b}-1, \mathsf{r}; \mathsf{c})\big)\, ; \\
&\quad\quad \mathsf{quicksort}(\mathsf{a},\mathsf{c})\, ;\, \mathsf{quicksort}(\mathsf{c}+1,\mathsf{b})\big) \\
&\quad \textbf{else skip}.
\end{aligned}
$$

6.5 Another Cyclic Buffer

When an array is used as a cyclic buffer, it represents a sequence in a more complex way than is described by the predicate array: The array location holding a sequence element is determined by modular arithmetic.

To illustrate, we assume that an n-element array has been allocated at location l, and we write $x \oplus y$ for the integer such that

$$x \oplus y = x + y \text{ modulo } \mathsf{n} \quad \text{and} \quad \mathsf{l} \leq \mathsf{j} < \mathsf{l} + \mathsf{n}.$$

We will also use the following variables:

$$
\begin{aligned}
\mathsf{m} &: \text{ number of active elements} \\
\mathsf{i} &: \text{ pointer to first active element} \\
\mathsf{j} &: \text{ pointer to first inactive element.}
\end{aligned}
$$

Let R abbreviate the assertion

$$R \stackrel{\text{def}}{=} 0 \leq \mathsf{m} \leq \mathsf{n} \wedge \mathsf{l} \leq \mathsf{i} < \mathsf{l} + \mathsf{n} \wedge \mathsf{l} \leq \mathsf{j} < \mathsf{l} + \mathsf{n} \wedge \mathsf{j} = \mathsf{i} \oplus \mathsf{m}$$

It is easy to show (using ordinary Hoare logic) that

$$\{R \wedge \mathsf{m} < \mathsf{n}\}\ \mathsf{m} := \mathsf{m} + 1\, ;\, \textbf{if } \mathsf{j} = \mathsf{l} + \mathsf{n} - 1 \textbf{ then } \mathsf{j} := \mathsf{l} \textbf{ else } \mathsf{j} := \mathsf{j} + 1\ \{R\}$$

and

$$\{R \wedge \mathsf{m} > 0\}\ \mathsf{m} := \mathsf{m} - 1\, ;\, \textbf{if } \mathsf{i} = \mathsf{l} + \mathsf{n} - 1 \textbf{ then } \mathsf{i} := \mathsf{l} \textbf{ else } \mathsf{i} := \mathsf{i} + 1\ \{R\}$$

Then the following invariant describes the situation where the cyclic buffer contains the sequence α:

$$((\bigodot_{\mathsf{k}=0}^{\mathsf{m}-1} \mathsf{i} \oplus \mathsf{k} \mapsto \alpha_{\mathsf{k}+1}) \ * \ (\bigodot_{\mathsf{k}=0}^{\mathsf{n}-\mathsf{m}-1} \mathsf{j} \oplus \mathsf{k} \mapsto -)) \wedge \mathsf{m} = \#\alpha \wedge R,$$

and the following is an annotated specification of a command that inserts the value x at the end of the sequence α. (The indications on the right refer to axiom schema in Section 6.1.)

$$\{((\bigodot_{\mathsf{k}=0}^{\mathsf{m}-1} \mathsf{i} \oplus \mathsf{k} \mapsto \alpha_{\mathsf{k}+1}) \ * \ (\bigodot_{\mathsf{k}=0}^{\mathsf{n}-\mathsf{m}-1} \mathsf{j} \oplus \mathsf{k} \mapsto -)) \\ \wedge \mathsf{m} = \#\alpha \wedge R \wedge \mathsf{m} < \mathsf{n}\}$$

$$\{((\bigodot_{\mathsf{k}=0}^{\mathsf{m}-1} \mathsf{i} \oplus \mathsf{k} \mapsto \alpha_{\mathsf{k}+1}) \ * \ (\bigodot_{\mathsf{k}=0}^{0} \mathsf{j} \oplus \mathsf{k} \mapsto -) \ * \\ (\bigodot_{\mathsf{k}=1}^{\mathsf{n}-\mathsf{m}-1} \mathsf{j} \oplus \mathsf{k} \mapsto -)) \wedge \mathsf{m} = \#\alpha \wedge R \wedge \mathsf{m} < \mathsf{n}\} \qquad (6.3)$$

$$\{((\bigodot_{\mathsf{k}=0}^{\mathsf{m}-1} \mathsf{i} \oplus \mathsf{k} \mapsto \alpha_{\mathsf{k}+1}) \ * \ \mathsf{j} \oplus 0 \mapsto - \ * \\ (\bigodot_{\mathsf{k}=1}^{\mathsf{n}-\mathsf{m}-1} \mathsf{j} \oplus \mathsf{k} \mapsto -)) \wedge \mathsf{m} = \#\alpha \wedge R \wedge \mathsf{m} < \mathsf{n}\} \qquad (6.2)$$

$$[\mathsf{j}] := \mathsf{x}\,;$$

$$\{((\bigodot_{\mathsf{k}=0}^{\mathsf{m}-1} \mathsf{i} \oplus \mathsf{k} \mapsto \alpha_{\mathsf{k}+1}) \ * \ \mathsf{j} \oplus 0 \mapsto \mathsf{x} \ * \\ (\bigodot_{\mathsf{k}=1}^{\mathsf{n}-\mathsf{m}-1} \mathsf{j} \oplus \mathsf{k} \mapsto -)) \wedge \mathsf{m} = \#\alpha \wedge R \wedge \mathsf{m} < \mathsf{n}\}$$

$$\{((\bigodot_{\mathsf{k}=0}^{\mathsf{m}-1} \mathsf{i} \oplus \mathsf{k} \mapsto \alpha_{\mathsf{k}+1}) \ * \ \mathsf{i} \oplus \mathsf{m} \mapsto \mathsf{x} \ * \\ (\bigodot_{\mathsf{k}=1}^{\mathsf{n}-\mathsf{m}-1} \mathsf{j} \oplus \mathsf{k} \mapsto -)) \wedge \mathsf{m} = \#\alpha \wedge R \wedge \mathsf{m} < \mathsf{n}\}$$

$$\{((\bigodot_{\mathsf{k}=0}^{\mathsf{m}-1} \mathsf{i} \oplus \mathsf{k} \mapsto (\alpha{\cdot}\mathsf{x})_{\mathsf{k}+1}) \ * \ \mathsf{i} \oplus \mathsf{m} \mapsto (\alpha{\cdot}\mathsf{x})_{\mathsf{m}+1} \ * \\ (\bigodot_{\mathsf{k}=1}^{\mathsf{n}-\mathsf{m}-1} \mathsf{j} \oplus \mathsf{k} \mapsto -)) \wedge \mathsf{m} = \#\alpha \wedge R \wedge \mathsf{m} < \mathsf{n}\}$$

$$\{((\bigodot_{\mathsf{k}=0}^{\mathsf{m}-1} \mathsf{i} \oplus \mathsf{k} \mapsto (\alpha{\cdot}\mathsf{x})_{\mathsf{k}+1}) \ * \ (\bigodot_{\mathsf{k}=\mathsf{m}}^{\mathsf{m}} \mathsf{i} \oplus \mathsf{k} \mapsto (\alpha{\cdot}\mathsf{x})_{\mathsf{k}+1}) \ * \\ (\bigodot_{\mathsf{k}=1}^{\mathsf{n}-\mathsf{m}-1} \mathsf{j} \oplus \mathsf{k} \mapsto -)) \wedge \mathsf{m} = \#\alpha \wedge R \wedge \mathsf{m} < \mathsf{n}\} \qquad (6.2)$$

$$\{((\bigodot_{\mathsf{k}=0}^{\mathsf{m}} \mathsf{i} \oplus \mathsf{k} \mapsto (\alpha{\cdot}\mathsf{x})_{\mathsf{k}+1}) \ * \ (\bigodot_{\mathsf{k}=1}^{\mathsf{n}-\mathsf{m}-1} \mathsf{j} \oplus \mathsf{k} \mapsto -)) \\ \wedge \mathsf{m}+1 = \#(\alpha{\cdot}\mathsf{x}) \wedge R \wedge \mathsf{m} < \mathsf{n}\} \qquad (6.3)$$

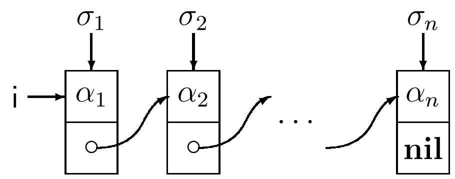
$$\{((\bigodot_{\mathsf{k}=0}^{\mathsf{m}} \mathsf{i} \oplus \mathsf{k} \mapsto (\alpha{\cdot}\mathsf{x})_{\mathsf{k}+1}) \ * \ (\bigodot_{\mathsf{k}=0}^{\mathsf{n}-\mathsf{m}-2} \mathsf{j} \oplus \mathsf{k} \oplus 1 \mapsto -)) \\ \wedge \mathsf{m}+1 = \#(\alpha{\cdot}\mathsf{x}) \wedge R \wedge \mathsf{m} < \mathsf{n}\} \qquad (6.4)$$

$$\mathsf{m} := \mathsf{m}+1\,;\ \mathbf{if}\ \mathsf{j} = \mathsf{l}+\mathsf{n}-1\ \mathbf{then}\ \mathsf{j} := \mathsf{l}\ \mathbf{else}\ \mathsf{j} := \mathsf{j}+1$$

$$\{((\bigodot_{\mathsf{k}=0}^{\mathsf{m}-1} \mathsf{i} \oplus \mathsf{k} \mapsto (\alpha{\cdot}\mathsf{x})_{\mathsf{k}+1}) \ * \ (\bigodot_{\mathsf{k}=0}^{\mathsf{n}-\mathsf{m}-1} \mathsf{j} \oplus \mathsf{k} \mapsto -)) \\ \wedge \mathsf{m} = \#(\alpha{\cdot}\mathsf{x}) \wedge R\}$$

6.6 Connecting Two Views of Simple Lists

Somewhat surprisingly, the iterated separating conjunction can be used profitably to describe lists as well as arrays. A simple example is the connection between ordinary simple lists and Bornat lists:



From the definitions

$$\mathsf{list}\,\epsilon\,\mathsf{i} \stackrel{\text{def}}{=} \mathbf{emp} \wedge \mathsf{i} = \mathbf{nil}$$

$$\mathsf{list}\,(\mathsf{a}{\cdot}\alpha)\,\mathsf{i} \stackrel{\text{def}}{=} \exists \mathsf{j}.\ \mathsf{i} \mapsto \mathsf{a}, \mathsf{j} \ * \ \mathsf{list}\,\alpha\,\mathsf{j}$$

and

$$\mathsf{listN}\,\epsilon\,\mathsf{i} \stackrel{\text{def}}{=} \mathbf{emp} \wedge \mathsf{i} = \mathbf{nil}$$

$$\mathsf{listN}\,(\mathsf{b}{\cdot}\sigma)\,\mathsf{i} \stackrel{\text{def}}{=} \mathsf{b} = \mathsf{i} \wedge \exists \mathsf{j}.\ \mathsf{i} + 1 \mapsto \mathsf{j} \ * \ \mathsf{listN}\,\sigma\,\mathsf{j},$$

one can show that list can be described in terms of listN and the separating conjunction by

$$\mathsf{list}\,\alpha\,\mathsf{i} \Leftrightarrow \exists \sigma.\ \#\sigma = \#\alpha \wedge (\mathsf{listN}\,\sigma\,\mathsf{i} \ * \ \bigodot_{\mathsf{k}=1}^{\#\alpha} \sigma_k \mapsto \alpha_k).$$

The proof is by structural induction on α.

6.7 Specifying a Program for Subset Lists

A more spectacular example of the use of separated iterative conjunction with lists is provided by an early LISP program that, given a list representing a finite set, computes a list of lists representing all subsets of the input. (More generally, the program maps a list representing a finite multiset into a list of lists representing all sub-multisets; sets are the special case where the lists contain no duplicate elements.)

This algorithm was historically important since it created sublists that shared storage extensively, to an extent that reduced the use of storage to a lower order of complexity compared with unshared sublists.

Indeed, the resulting sharing structure is far more complex than anything produced by the other algorithms in this book. Thus, it is significant that the program can be proved in separation logic, and even more so that, with the use of the iterated separating conjunction, one can prove enough about the result to determine its size precisely.

In this section, we reason about an iterative version of the algorithm.

We use the following variables to denote various kinds of sequences:

$$\begin{aligned}\alpha :&\quad \text{sequences of integers}\\ \beta,\gamma :&\quad \text{sequences of addresses}\\ \sigma :&\quad \text{sequences of sequence of integers.}\end{aligned}$$

Then our program will satisfy the specification

$$\begin{array}{l}\{\mathsf{list}\,\alpha\,\mathsf{i}\}\\ \text{“Set } \mathsf{j} \text{ to list of lists of subsets of } \mathsf{i}\text{”}\\ \{\exists\sigma,\beta.\ \ \mathsf{ss}(\alpha^{\dagger},\sigma)\wedge(\mathsf{list}\,\beta\,\mathsf{j}\ *\ (Q(\sigma,\beta)\wedge R(\beta)))\}.\end{array}\tag{6.9}$$

Here $\mathsf{ss}(\alpha,\sigma)$ asserts that σ is a sequence of the sequences that represent the subsets of (the set represented by) α. The inductive definition also specifies the order of elements in σ and its elements (to the extent that the value of σ is uniquely determined by α):

$$\begin{aligned}\mathsf{ss}(\epsilon,\sigma) &\stackrel{\text{def}}{=} \sigma=[\epsilon]\\ \mathsf{ss}(\mathsf{a}{\cdot}\alpha,\sigma) &\stackrel{\text{def}}{=} \exists\sigma'.\ \Big(\mathsf{ss}(\alpha,\sigma')\wedge\sigma=(\mathsf{ext}_{\mathsf{a}}\sigma')^{\dagger}{\cdot}\sigma'\Big),\end{aligned}$$

where $\mathsf{ext}_{\mathsf{a}}$ is a function that prefixes a to every element of its argument:

$$\begin{aligned}\#\mathsf{ext}_{\mathsf{a}}\sigma &\stackrel{\text{def}}{=} \#\sigma\\ \forall_{\mathsf{i}=1}^{\#\sigma}\,(\mathsf{ext}_{\mathsf{a}}\sigma)_i &\stackrel{\text{def}}{=} \mathsf{a}{\cdot}\sigma_i.\end{aligned}$$

(Here $\forall_{\mathsf{i}=1}^{\#\sigma}\,p$ abbreviates $\forall\mathsf{i}.\ (1\le\mathsf{i}\le\#\sigma)\Rightarrow p$.)

The formula $Q(\sigma,\beta)$ asserts that the elements of β are lists representing the elements of σ:

$$Q(\sigma,\beta)\stackrel{\text{def}}{=}\#\beta=\#\sigma\wedge\forall_{\mathsf{i}=1}^{\#\beta}\,(\mathsf{list}\,\sigma_{\mathsf{i}}\,\beta_{\mathsf{i}}\ *\ \mathbf{true}).$$

Finally, $R(\beta)$ uses the iterated separating conjunction to assert that the final element of β is the empty list, and that every previous element is a list consisting of a single record followed by a list occurring later in β:

$$\begin{aligned} R(\beta) \stackrel{\text{def}}{=} (\beta_{\#\beta} = \mathbf{nil} \wedge \mathbf{emp}) \ * \\ \textstyle\bigodot_{\mathsf{i}=1}^{\#\beta-1}(\exists \mathsf{a}, \mathsf{k}.\ \ \mathsf{i} < \mathsf{k} \leq \#\beta \wedge \beta_{\mathsf{i}} \mapsto \mathsf{a}, \beta_{\mathsf{k}}). \end{aligned}$$

It is immediately evident that:

$$Q([\epsilon], [\mathbf{nil}]) \Leftrightarrow \mathbf{true}$$

$$R([\mathbf{nil}]) \Leftrightarrow \mathbf{emp}.$$

Less trivial are the following:

Proposition 16

$$W(\beta, \gamma, \mathsf{a}) \ * \ \mathsf{g} \mapsto \mathsf{a}, \mathsf{b} \Leftrightarrow W(\beta{\cdot}\mathsf{b}, \mathsf{g}{\cdot}\gamma, \mathsf{a}).$$

PROOF

$$\begin{aligned} & W(\beta, \gamma, \mathsf{a}) \ * \ \mathsf{g} \mapsto \mathsf{a}, \mathsf{b} \\ & \quad \Leftrightarrow \#\gamma = \#\beta \wedge \left(\mathsf{g} \mapsto \mathsf{a}, \mathsf{b} \ * \ \textstyle\bigodot_{\mathsf{i}=1}^{\#\gamma} \gamma_{\mathsf{i}} \mapsto \mathsf{a}, (\beta^{\dagger})_{\mathsf{i}}\right) \\ & \quad \Leftrightarrow \#\mathsf{g}{\cdot}\gamma = \#\beta{\cdot}\mathsf{b} \wedge \left(\left(\textstyle\bigodot_{\mathsf{i}=1}^{1}(\mathsf{g}{\cdot}\gamma)_{\mathsf{i}} \mapsto \mathsf{a}, ((\beta{\cdot}\mathsf{b})^{\dagger})_{\mathsf{i}}\right) \ * \right. \\ & \qquad \left. \left(\textstyle\bigodot_{\mathsf{i}=1}^{\#\mathsf{g}\cdot\gamma-1}(\mathsf{g}{\cdot}\gamma)_{\mathsf{i}+1} \mapsto \mathsf{a}, ((\beta{\cdot}\mathsf{b})^{\dagger})_{\mathsf{i}+1}\right)\right) \\ & \quad \Leftrightarrow \#\mathsf{g}{\cdot}\gamma = \#\beta{\cdot}\mathsf{b} \wedge \textstyle\bigodot_{\mathsf{i}=1}^{\#\mathsf{g}\cdot\gamma}(\mathsf{g}{\cdot}\gamma)_{\mathsf{i}} \mapsto \mathsf{a}, ((\beta{\cdot}\mathsf{b})^{\dagger})_{\mathsf{i}} \\ & \quad \Leftrightarrow W(\beta{\cdot}\mathsf{b}, \mathsf{g}{\cdot}\gamma, \mathsf{a}). \end{aligned}$$

END OF PROOF

Proposition 17

$$Q(\sigma, \beta) \ * \ W(\beta, \gamma, \mathsf{a}) \Rightarrow Q((\mathsf{ext}_{\mathsf{a}}\sigma)^{\dagger}{\cdot}\sigma, \gamma{\cdot}\beta).$$

PROOF Let

$$\begin{aligned} p(\mathsf{i}) &\stackrel{\text{def}}{=} \mathsf{list}\ \sigma_{\mathsf{i}}\ \beta_{\mathsf{i}} \\ q(\mathsf{i}) &\stackrel{\text{def}}{=} \gamma_{\mathsf{i}} \mapsto \mathsf{a}, (\beta^{\dagger})_{\mathsf{i}} \\ n &\stackrel{\text{def}}{=} \#\sigma. \end{aligned}$$

Then

$$
\begin{aligned}
& Q(\sigma, \beta) \ * \ W(\beta, \gamma, \mathsf{a}) \\
& \quad \Rightarrow \Big(\#\beta = n \wedge \forall_{\mathsf{i}=1}^{\#\beta}(p(\mathsf{i}) \ * \ \mathbf{true})\Big) \ * \ \Big(\#\gamma = \#\beta \wedge \bigodot_{\mathsf{i}=1}^{\#\gamma} q(\mathsf{i})\Big) \\
& \quad \Rightarrow \#\beta = n \wedge \#\gamma = n \wedge \Big(\big(\forall_{\mathsf{i}=1}^{n}(p(\mathsf{i}) \ * \ \mathbf{true})\big) \ * \ \bigodot_{\mathsf{i}=1}^{n} q(\mathsf{i})\Big) \\
& \quad \Rightarrow \#\beta = n \wedge \#\gamma = n \wedge \forall_{\mathsf{i}=1}^{n}\Big(p(\mathsf{i}) \ * \ \mathbf{true} \ * \ \bigodot_{\mathsf{j}=1}^{n} q(\mathsf{j})\Big) \\
& \quad \Rightarrow \#\beta = n \wedge \#\gamma = n \wedge \forall_{\mathsf{i}=1}^{n}\Big(p(\mathsf{i}) \ * \ \mathbf{true}\Big) \wedge \\
& \qquad \forall_{\mathsf{i}=1}^{n}\Big(p(n+1-\mathsf{i}) \ * \ \mathbf{true} \ * \ \bigodot_{\mathsf{j}=1}^{n} q(\mathsf{j})\Big) \\
& \quad \Rightarrow \#\beta = n \wedge \#\gamma = n \wedge \forall_{\mathsf{i}=1}^{n}\Big(p(\mathsf{i}) \ * \ \mathbf{true}\Big) \wedge \\
& \qquad \forall_{\mathsf{i}=1}^{n}\Big(p(n+1-\mathsf{i}) \ * \ \mathbf{true} \ * \ q(\mathsf{i})\Big) \\
& \quad \Rightarrow \#\beta = n \wedge \#\gamma = n \wedge \forall_{\mathsf{i}=1}^{n}\Big(p(\mathsf{i}) \ * \ \mathbf{true}\Big) \wedge \\
& \qquad \forall_{\mathsf{i}=1}^{n}\Big(\mathsf{list}\ (\sigma^{\dagger})_{\mathsf{i}}\ (\beta^{\dagger})_{\mathsf{i}} \ * \ \mathbf{true} \ * \ \gamma_{\mathsf{i}} \mapsto \mathsf{a}, (\beta^{\dagger})_{\mathsf{i}}\Big) \\
& \quad \Rightarrow \#\gamma = n \wedge \#\beta = n \wedge \forall_{\mathsf{i}=1}^{n}\Big(\mathsf{list}\ \sigma_{\mathsf{i}}\ \beta_{\mathsf{i}} \ * \ \mathbf{true}\Big) \wedge \\
& \qquad \forall_{\mathsf{i}=1}^{n}\Big(\mathsf{list}\ ((\mathsf{ext}_{\mathsf{a}}\sigma)^{\dagger})_{\mathsf{i}}\ \gamma_{\mathsf{i}} \ * \ \mathbf{true}\Big) \\
& \quad \Rightarrow \#\gamma{\cdot}\beta = \#(\mathsf{ext}_{\mathsf{a}}\sigma)^{\dagger}{\cdot}\sigma \wedge \\
& \qquad \forall_{\mathsf{i}=1}^{\#\gamma\cdot\beta}\Big(\mathsf{list}\ ((\mathsf{ext}_{\mathsf{a}}\sigma)^{\dagger}{\cdot}\sigma)_{\mathsf{i}}\ (\gamma{\cdot}\beta)_{\mathsf{i}} \ * \ \mathbf{true}\Big) \\
& \quad \Rightarrow Q((\mathsf{ext}_{\mathsf{a}}\sigma)^{\dagger}{\cdot}\sigma, \gamma{\cdot}\beta).
\end{aligned}
$$

END OF PROOF

Proposition 18

$$R(\beta) \ * \ W(\beta, \gamma, \mathsf{a}) \Rightarrow R(\gamma{\cdot}\beta).$$

$$
\begin{aligned}
&R(\beta) \ * \ W(\beta, \gamma, \mathsf{a}) \\
&\quad \Rightarrow (\beta_{\#\beta} = \mathbf{nil} \wedge \mathbf{emp}) \ * \\
&\qquad \bigodot_{\mathsf{i}=1}^{\#\gamma} \gamma_{\mathsf{i}} \mapsto \mathsf{a}, (\beta^{\dagger})_{\mathsf{i}} \ * \\
&\qquad \bigodot_{\mathsf{i}=1}^{\#\beta-1} (\exists \mathsf{a}, \mathsf{k}.\ \ \mathsf{i} < \mathsf{k} \leq \#\beta \wedge \beta_{\mathsf{i}} \mapsto \mathsf{a}, \beta_{\mathsf{k}}) \\
&\quad \Rightarrow ((\gamma{\cdot}\beta)_{\#\gamma\cdot\beta} = \mathbf{nil} \wedge \mathbf{emp}) \ * \\
&\qquad \bigodot_{\mathsf{i}=1}^{\#\gamma} (\exists \mathsf{a}, \mathsf{k}.\ \ \#\gamma < \mathsf{k} \leq \#\gamma{\cdot}\beta \wedge (\gamma{\cdot}\beta)_{\mathsf{i}} \mapsto \mathsf{a}, (\gamma{\cdot}\beta)_{\mathsf{k}}) \ * \\
&\qquad \bigodot_{\mathsf{i}=\#\gamma+1}^{\#\gamma\cdot\beta-1} (\exists \mathsf{a}, \mathsf{k}.\ \ \mathsf{i} < \mathsf{k} \leq \#\gamma{\cdot}\beta \wedge (\gamma{\cdot}\beta)_{\mathsf{i}} \mapsto \mathsf{a}, (\gamma{\cdot}\beta)_{\mathsf{k}}) \\
&\quad \Rightarrow ((\gamma{\cdot}\beta)_{\#\gamma\cdot\beta} = \mathbf{nil} \wedge \mathbf{emp}) \ * \\
&\qquad \bigodot_{\mathsf{i}=1}^{\#\gamma\cdot\beta-1} (\exists \mathsf{a}, \mathsf{k}.\ \ \mathsf{i} < \mathsf{k} \leq \#\gamma{\cdot}\beta \wedge (\gamma{\cdot}\beta)_{\mathsf{i}} \mapsto \mathsf{a}, (\gamma{\cdot}\beta)_{\mathsf{k}}) \\
&\quad \Rightarrow R(\gamma{\cdot}\beta).
\end{aligned}
$$

END OF PROOF

From the last two propositions, we have

$$
\begin{aligned}
&(Q(\sigma, \beta) \wedge R(\beta)) \ * \ W(\beta, \gamma, \mathsf{a}) \\
&\quad \Rightarrow (Q(\sigma, \beta) \ * \ W(\beta, \gamma, \mathsf{a})) \wedge (R(\beta) \ * \ W(\beta, \gamma, \mathsf{a})) \\
&\quad \Rightarrow Q((\mathsf{ext}_{\mathsf{a}}\sigma)^{\dagger}{\cdot}\sigma, \gamma{\cdot}\beta) \wedge R(\gamma{\cdot}\beta).
\end{aligned}
$$

Using these results, we can verfy a program satisfying (6.9). The program contains two nested **while** commands. The invariant of the outer **while** is

$$
\begin{aligned}
&\exists \alpha', \alpha'', \sigma, \beta.\ \ \alpha'^{\dagger}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge \\
&\qquad (\mathsf{list}\ \alpha''\ \mathsf{i} \ * \ \mathsf{list}\ \beta\ \mathsf{j} \ * \ (Q(\sigma, \beta) \wedge R(\beta))),
\end{aligned}
$$

and the invariant of the inner **while** is

$$
\begin{aligned}
&\exists \alpha', \alpha'', \sigma, \beta', \beta'', \gamma.\ \ \alpha'^{\dagger}{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge \\
&\qquad (\mathsf{list}\ \alpha''\ \mathsf{i} \ * \ \mathsf{lseg}\ \gamma\ (\mathsf{l}, \mathsf{j}) \ * \ \mathsf{lseg}\ \beta'\ (\mathsf{j}, \mathsf{m}) \ * \ \mathsf{list}\ \beta''\ \mathsf{m} \ * \\
&\qquad\quad (Q(\sigma, \beta'{\cdot}\beta'') \wedge R(\beta'{\cdot}\beta'')) \ * \ W(\beta', \gamma, \mathsf{a})).
\end{aligned}
$$

At the completion of the inner **while**, we will have

$$
\begin{aligned}
&\exists \alpha', \alpha'', \sigma, \beta', \gamma.\ \ \alpha'^{\dagger}{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge \\
&\qquad (\mathsf{list}\ \alpha''\ \mathsf{i} \ * \ \mathsf{lseg}\ \gamma\ (\mathsf{l}, \mathsf{j}) \ * \ \mathsf{list}\ \beta'\ \mathsf{j} \ * \\
&\qquad\quad (Q(\sigma, \beta') \wedge R(\beta')) \ * \ W(\beta', \gamma, \mathsf{a})).
\end{aligned}
$$

In full detail, the annotated specification is:

$$\{\mathsf{list}\,\alpha\,\mathsf{i}\}$$
$$\mathsf{j} := \mathbf{cons}(\mathbf{nil}, \mathbf{nil})\;;$$
$$\{\mathsf{list}\,\alpha\,\mathsf{i} \;*\; \mathsf{j} \mapsto \mathbf{nil}, \mathbf{nil}\}$$
$$\{\mathsf{list}\,\alpha\,\mathsf{i} \;*\; \mathsf{list}\,[\mathbf{nil}]\,\mathsf{j}\}$$
$$\{\mathsf{ss}(\epsilon, [\epsilon]) \wedge (\mathsf{list}\,\alpha\,\mathsf{i} \;*\; \mathsf{list}\,[\mathbf{nil}]\,\mathsf{j} \;*\; (Q([\epsilon], [\mathbf{nil}]) \wedge R([\mathbf{nil}])))\}$$
$$\{\exists \alpha', \alpha'', \sigma, \beta.\;\; \alpha'^{\dagger}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge$$
$$\quad (\mathsf{list}\,\alpha''\,\mathsf{i} \;*\; \mathsf{list}\,\beta\,\mathsf{j} \;*\; (Q(\sigma, \beta) \wedge R(\beta)))\}$$
$$\mathbf{while}\ \mathsf{i} \neq \mathbf{nil}\ \mathbf{do}\ \textit{Body of the Outer}\ \mathbf{while}\;;$$
$$\{\exists \sigma, \beta.\;\; \mathsf{ss}(\alpha^{\dagger}, \sigma) \wedge (\mathsf{list}\,\beta\,\mathsf{j} \;*\; (Q(\sigma, \beta) \wedge R(\beta)))\},$$

where the body of the outer **while** is:

$$\{\exists \alpha', \alpha'', \sigma, \beta, \mathsf{a}, \mathsf{k}.\;\; \alpha'^{\dagger}{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge$$
$$\quad (\mathsf{i} \mapsto \mathsf{a}, \mathsf{k} \;*\; \mathsf{list}\,\alpha''\,\mathsf{k} \;*\; \mathsf{list}\,\beta\,\mathsf{j} \;*\; (Q(\sigma, \beta) \wedge R(\beta)))\}$$
$$\big(\mathsf{a} := [\mathsf{i}]\;;\; \mathsf{k} := [\mathsf{i}+1]\;;\; \mathbf{dispose}\ \mathsf{i}\;;\; \mathbf{dispose}\ \mathsf{i}+1\;;\; \mathsf{i} := \mathsf{k}\;;$$
$$\{\exists \alpha', \alpha'', \sigma, \beta.\;\; \alpha'^{\dagger}{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge$$
$$\quad (\mathsf{list}\,\alpha''\,\mathsf{i} \;*\; \mathsf{list}\,\beta\,\mathsf{j} \;*\; (Q(\sigma, \beta) \wedge R(\beta)))\}$$
$$\{\exists \alpha', \alpha'', \sigma, \beta.\;\; \alpha'^{\dagger}{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge$$
$$\quad (\mathsf{list}\,\alpha''\,\mathsf{i} \;*\; \mathsf{lseg}\,\epsilon\,(\mathsf{j}, \mathsf{j}) \;*\; \mathsf{lseg}\,\epsilon\,(\mathsf{j}, \mathsf{j}) \;*\; \mathsf{list}\,\beta\,\mathsf{j} \;*$$
$$\qquad (Q(\sigma, \epsilon{\cdot}\beta) \wedge R(\epsilon{\cdot}\beta)) \;*\; W(\epsilon, \epsilon, \mathsf{a}))\}$$
$$\mathsf{l} := \mathsf{j}\;;\; \mathsf{m} := \mathsf{j}\;;$$
$$\{\exists \alpha', \alpha'', \sigma, \beta.\;\; \alpha'^{\dagger}{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge$$
$$\quad (\mathsf{list}\,\alpha''\,\mathsf{i} \;*\; \mathsf{lseg}\,\epsilon\,(\mathsf{l}, \mathsf{j}) \;*\; \mathsf{lseg}\,\epsilon\,(\mathsf{j}, \mathsf{m}) \;*\; \mathsf{list}\,\beta\,\mathsf{m} \;*$$
$$\qquad (Q(\sigma, \epsilon{\cdot}\beta) \wedge R(\epsilon{\cdot}\beta)) \;*\; W(\epsilon, \epsilon, \mathsf{a}))\}$$
$$\{\exists \alpha', \alpha'', \sigma, \beta', \beta'', \gamma.\;\; \alpha'^{\dagger}{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge$$
$$\quad (\mathsf{list}\,\alpha''\,\mathsf{i} \;*\; \mathsf{lseg}\,\gamma\,(\mathsf{l}, \mathsf{j}) \;*\; \mathsf{lseg}\,\beta'\,(\mathsf{j}, \mathsf{m}) \;*\; \mathsf{list}\,\beta''\,\mathsf{m} \;*$$
$$\qquad (Q(\sigma, \beta'{\cdot}\beta'') \wedge R(\beta'{\cdot}\beta'')) \;*\; W(\beta', \gamma, \mathsf{a}))\}$$
$$\mathbf{while}\ \mathsf{m} \neq \mathbf{nil}\ \mathbf{do}\ \textit{Body of the Inner}\ \mathbf{while}\;;$$
$$\{\exists \alpha', \alpha'', \sigma, \beta', \gamma.\;\; \alpha'^{\dagger}{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge$$
$$\quad (\mathsf{list}\,\alpha''\,\mathsf{i} \;*\; \mathsf{lseg}\,\gamma\,(\mathsf{l}, \mathsf{j}) \;*\; \mathsf{list}\,\beta'\,\mathsf{j} \;*$$

$$
\begin{array}{l}
\qquad\quad (Q(\sigma, \beta') \wedge R(\beta')) \ast W(\beta', \gamma, \mathsf{a}))\} \\
\{\exists \alpha', \alpha'', \sigma, \beta', \gamma.\ (\mathsf{a}{\cdot}\alpha')^\dagger{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\mathsf{a}{\cdot}\alpha', (\mathsf{ext_a}\sigma)^\dagger{\cdot}\sigma) \wedge \\
\quad (\mathsf{list}\ \alpha''\ \mathsf{i} \ast \mathsf{list}\ (\gamma{\cdot}\beta')\ \mathsf{l} \ast (Q((\mathsf{ext_a}\sigma)^\dagger{\cdot}\sigma, \gamma{\cdot}\beta') \wedge R(\gamma{\cdot}\beta')))\} \\
\{\exists \alpha', \alpha'', \sigma, \beta.\ \alpha'^\dagger{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge \\
\quad (\mathsf{list}\ \alpha''\ \mathsf{i} \ast \mathsf{list}\ \beta\ \mathsf{l} \ast (Q(\sigma, \beta) \wedge R(\beta)))\} \\
\mathsf{j} := \mathsf{l}\big),
\end{array}
$$

and the body of the inner **while** is:

$$
\begin{array}{l}
\{\exists \alpha', \alpha'', \sigma, \beta', \beta'', \gamma, \mathsf{b}, \mathsf{m}''.\ \alpha'^\dagger{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge \\
\quad (\mathsf{list}\ \alpha''\ \mathsf{i} \ast \mathsf{lseg}\ \gamma\ (\mathsf{l}, \mathsf{j}) \ast \mathsf{lseg}\ \beta'\ (\mathsf{j}, \mathsf{m}) \ast \mathsf{m} \mapsto \mathsf{b}, \mathsf{m}'' \ast \\
\qquad \mathsf{list}\ \beta''\ \mathsf{m}'' \ast (Q(\sigma, \beta'{\cdot}\mathsf{b}{\cdot}\beta'') \wedge R(\beta'{\cdot}\mathsf{b}{\cdot}\beta'')) \ast W(\beta', \gamma, \mathsf{a}))\} \\
\big(\mathsf{b} := [\mathsf{m}]\ ; \\
\{\exists \alpha', \alpha'', \sigma, \beta', \beta'', \gamma, \mathsf{m}''.\ \alpha'^\dagger{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge \\
\quad (\mathsf{list}\ \alpha''\ \mathsf{i} \ast \mathsf{lseg}\ \gamma\ (\mathsf{l}, \mathsf{j}) \ast \mathsf{lseg}\ \beta'\ (\mathsf{j}, \mathsf{m}) \ast \mathsf{m} \mapsto \mathsf{b}, \mathsf{m}'' \ast \\
\qquad \mathsf{list}\ \beta''\ \mathsf{m}'' \ast (Q(\sigma, \beta'{\cdot}\mathsf{b}{\cdot}\beta'') \wedge R(\beta'{\cdot}\mathsf{b}{\cdot}\beta'')) \ast W(\beta', \gamma, \mathsf{a}))\} \\
\mathsf{m} := [\mathsf{m} + 1]\ ; \\
\{\exists \alpha', \alpha'', \sigma, \beta', \beta'', \gamma, \mathsf{m}'.\ \alpha'^\dagger{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge \\
\quad (\mathsf{list}\ \alpha''\ \mathsf{i} \ast \mathsf{lseg}\ \gamma\ (\mathsf{l}, \mathsf{j}) \ast \mathsf{lseg}\ \beta'\ (\mathsf{j}, \mathsf{m}') \ast \mathsf{m}' \mapsto \mathsf{b}, \mathsf{m} \ast \\
\qquad \mathsf{list}\ \beta''\ \mathsf{m} \ast (Q(\sigma, \beta'{\cdot}\mathsf{b}{\cdot}\beta'') \wedge R(\beta'{\cdot}\mathsf{b}{\cdot}\beta'')) \ast W(\beta', \gamma, \mathsf{a}))\} \\
\{\exists \alpha', \alpha'', \sigma, \beta', \beta'', \gamma.\ \alpha'^\dagger{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge \\
\quad (\mathsf{list}\ \alpha''\ \mathsf{i} \ast \mathsf{lseg}\ \gamma\ (\mathsf{l}, \mathsf{j}) \ast \mathsf{lseg}\ \beta'{\cdot}\mathsf{b}\ (\mathsf{j}, \mathsf{m}) \ast \mathsf{list}\ \beta''\ \mathsf{m} \ast \\
\qquad (Q(\sigma, \beta'{\cdot}\mathsf{b}{\cdot}\beta'') \wedge R(\beta'{\cdot}\mathsf{b}{\cdot}\beta'')) \ast W(\beta', \gamma, \mathsf{a}))\} \\
\mathsf{g} := \mathbf{cons}(\mathsf{a}, \mathsf{b})\ ; \\
\{\exists \alpha', \alpha'', \sigma, \beta', \beta'', \gamma.\ \alpha'^\dagger{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge \\
\quad (\mathsf{list}\ \alpha''\ \mathsf{i} \ast \mathsf{lseg}\ \gamma\ (\mathsf{l}, \mathsf{j}) \ast \mathsf{lseg}\ \beta'{\cdot}\mathsf{b}\ (\mathsf{j}, \mathsf{m}) \ast \mathsf{list}\ \beta''\ \mathsf{m} \ast \\
\qquad (Q(\sigma, \beta'{\cdot}\mathsf{b}{\cdot}\beta'') \wedge R(\beta'{\cdot}\mathsf{b}{\cdot}\beta'')) \ast W(\beta', \gamma, \mathsf{a}) \ast \mathsf{g} \mapsto \mathsf{a}, \mathsf{b})\} \\
\mathsf{l} := \mathbf{cons}(\mathsf{g}, \mathsf{l}) \\
\{\exists \alpha', \alpha'', \sigma, \beta', \beta'', \gamma, \mathsf{l}'.\ \alpha'^\dagger{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge \\
\quad (\mathsf{list}\ \alpha''\ \mathsf{i} \ast \mathsf{l} \mapsto \mathsf{g}, \mathsf{l}' \ast \mathsf{lseg}\ \gamma\ (\mathsf{l}', \mathsf{j}) \ast \mathsf{lseg}\ \beta'{\cdot}\mathsf{b}\ (\mathsf{j}, \mathsf{m}) \ast \mathsf{list}\ \beta''\ \mathsf{m} \ast \\
\qquad (Q(\sigma, \beta'{\cdot}\mathsf{b}{\cdot}\beta'') \wedge R(\beta'{\cdot}\mathsf{b}{\cdot}\beta'')) \ast W(\beta', \gamma, \mathsf{a}) \ast \mathsf{g} \mapsto \mathsf{a}, \mathsf{b})\}
\end{array}
$$

$$
\begin{aligned}
&\{\exists \alpha', \alpha'', \sigma, \beta', \beta'', \gamma.\ \ \alpha'^{\dagger}{\cdot}\mathsf{a}{\cdot}\alpha'' = \alpha \wedge \mathsf{ss}(\alpha', \sigma) \wedge \\
&\quad (\mathsf{list}\ \alpha''\ \mathsf{i}\ *\ \mathsf{lseg}\ \mathsf{g}{\cdot}\gamma\ (\mathsf{l}, \mathsf{j})\ *\ \mathsf{lseg}\ \beta'{\cdot}\mathsf{b}\ (\mathsf{j}, \mathsf{m})\ *\ \mathsf{list}\ \beta''\ \mathsf{m}\ * \\
&\qquad (Q(\sigma, \beta'{\cdot}\mathsf{b}{\cdot}\beta'') \wedge R(\beta'{\cdot}\mathsf{b}{\cdot}\beta''))\ *\ W(\beta'{\cdot}\mathsf{b}, \mathsf{g}{\cdot}\gamma, \mathsf{a}))\}\big).
\end{aligned}
$$

Exercise 16

Derive the axiom scheme

$$
m \le j \le n \Rightarrow \Big(\big(\bigodot_{i=m}^{n} p(i)\big) \Rightarrow (p(j)\ *\ \mathbf{true})\Big)
$$

from the other axiom schemata for iterating separating conjunction given in Section 6.1.

Exercise 17

The following is an alternative global rule for allocation that uses a ghost variable (v'):

- The ghost-variable global form (ALLOCGG)

$$
\frac{}{\{v = v' \wedge r\}\ v := \mathbf{allocate}\ e\ \{(\bigodot_{i=v}^{v+e'-1} i \mapsto -)\ *\ r'\},}
$$

  where v' is distinct from v, e' denotes $e/v \to v'$, and r' denotes $r/v \to v'$.

Derive (ALLOCGG) from (ALLOCG) and (ALLOCL) from (ALLOCGG).