## 7.2 Operations on Lists

**7.2.1 Introduction.** In this section we are concerned with the problem of specification and verification of various useful simple operations over lists. We will show how the algorithms can be implemented by using structural recursion and how their specification properties can be proved by the corresponding induction principles.

**7.2.2 Map.** The operation $Map_f(x)$ applies an unary function $f$ to each element of the list $x$:

$$\vdash_{\mathrm{PA}} L\,Map_f(x) = L(x) \tag{1}$$
$$\vdash_{\mathrm{PA}} i < L(x) \to Map_f(x)[i] = f(x[i]) \tag{2}$$

The mapping is defined by structural list recursion as a p.r. function in $f$ by

$Map_f(0) = 0$
$Map_f\langle v, w\rangle = \big\langle f(v), Map_f(w)\big\rangle$.

*Verification.* (1): By a straightforward structural list induction.

(2): By structural induction on the list $x$ as $\forall i(2)$. In the base case there is nothing to prove. In the induction step, when $x = \langle v, w\rangle$ for some $v, w$, take any $i$ s.t. $i < L\langle v, w\rangle = L(w) + 1$ and consider two cases. If $i = 0$ then

$$Map_f\langle v, w\rangle[0] = \big\langle f(v), Map_f(w)\big\rangle[0] = f(v) = f(\langle v, w\rangle[0]).$$

If $i = j + 1$ for some $j$ then $j < L(w)$ and we thus obtain

$$\begin{aligned} Map_f\langle v, w\rangle[j+1] = \big\langle f(v), Map_f(w)\big\rangle[j+1] = Map_f(w)[j] &\overset{\mathrm{IH}}{=} \\ = f(w[j]) = f(\langle v, w\rangle[j+1]). & \end{aligned}$$

Note that the induction hypothesis is applied with $j$ in place of $i$. □

**7.2.3 List modification.** The ternary function $x[i := a]$ takes a list $x$ and yields a new list with the $i$-th element replaced by $a$ (starting from 0):

$$\vdash_{\mathrm{PA}} i < L(x) \to L(x[i := a]) = L(x) \tag{1}$$
$$\vdash_{\mathrm{PA}} i < L(x) \to x[i := a][i] = a \tag{2}$$
$$\vdash_{\mathrm{PA}} i < L(x) \wedge j < L(x) \wedge j \neq i \to x[i := a][j] = x[j]. \tag{3}$$

Note that the properties do not specify the value $x[i := a]$ for $i \geq L(x)$.

The list modification $x[i := a]$ is defined by primitive recursion on $i$ with substitution in the parameter $x$ as a p.r. function:

$x[0 := a] = \langle a, w\rangle \leftarrow x = \langle v, w\rangle$
$x[i + 1 := a] = \big\langle v, w[i := a]\big\rangle \leftarrow x = \langle v, w\rangle$.

Note that the second parameter does not change in the recursion.

*Verification.* We show here only the proof of the last property for the previous ones can be proved similarly.

Property $\forall x \forall j(3)$ is proved by induction on the index $i$. In the base case take any $x, j$ such that $0 < L(x)$, $j < L(x)$ and $j \neq i$. Then it must be $x = \langle v, w\rangle$ and $j = j_1 + 1$ for some $v, w, j_1$. We then have

$$\langle v, w\rangle[0 := a][j_1 + 1] = \langle a, w\rangle[j_1 + 1] = w[j_1] = \langle v, w\rangle[j_1 + 1].$$

In induction step take any $x, j$ such that $i + 1 < L(x)$, $j < L(x)$ and $j \neq i + 1$. Then it must be $x = \langle v, w\rangle$ for some $v, w$. From $i + 1 < L\langle v, w\rangle = L(w) + 1$ we obtain $i < L(w)$. We consider two subcases. It $j = 0$ then

$$\langle v, w\rangle[i + 1 := a][0] = \big\langle v, w[i := a]\big\rangle[0] = v = \langle v, w\rangle[0].$$

It $j = j_1 + 1$ for some $j_1$, then $j_1 \neq i$ and $j_1 < L(w)$. We now obtain

$$\begin{aligned}\langle v, w\rangle[i + 1 := a][j_1 + 1] = \big\langle v, w[i := a]\big\rangle[j_1 + 1] = w[i := a][j_1] &\stackrel{\text{IH}}{=} \\ = w[j_1] = \langle v, w\rangle[j_1 + 1].&\end{aligned}$$

Note that IH is applied respectively with $w$ or $j_1$ in place of $x$ or $j$. □

**7.2.4 Take and drop.** The function $\mathit{Take}(n, x)$ yields the initial segment of a list $x$ of the length $n$ provided $n \leq L(x)$. The function satisfies

$$\vdash_{\mathrm{PA}} n \leq L(x) \to L\,\mathit{Take}(n, x) = n \tag{1}$$
$$\vdash_{\mathrm{PA}} n \leq L(x) \to \exists y\, x = \mathit{Take}(n, x) \oplus y \tag{2}$$

and it is defined by primitive recursion on $n$ with substitution in parameter as primitive recursive by

$\mathit{Take}(0, x) = 0$
$\mathit{Take}(n + 1, \langle v, w\rangle) = \big\langle v, \mathit{Take}(n, w)\big\rangle$.

Note the default $\mathit{Take}(n + 1, 0) = 0$.

The function $\mathit{Drop}(n, x)$ removes the initial segment of a list $x$ of the length $n$ provided $n \leq L(x)$. The function satisfies

$$\vdash_{\mathrm{PA}} n \leq L(x) \to L\,\mathit{Drop}(n, x) = L(x) \dot{-} n \tag{3}$$
$$\vdash_{\mathrm{PA}} n \leq L(x) \to \exists y\, x = y \oplus \mathit{Drop}(n, x) \tag{4}$$

and it is defined by primitive recursion on $n$ with substitution in parameter as primitive recursive by

$\mathit{Drop}(0, x) = x$
$\mathit{Drop}(n + 1, \langle v, w\rangle) = \mathit{Drop}(n, w)$.

Note the default $Drop(n+1,0) = 0$.

Usually we intend to apply both operations $Take(n,x)$ and $Drop(n,x)$ only in cases when $n \le L(x)$. We can take the following properties as alternative programs for computing the functions in such cases:

$$\vdash_{\mathrm{PA}} n \le L(x) \to Take(n,x) = \begin{array}{l}\textbf{case}\\ \quad n = 0 \Rightarrow\ 0\\ \quad n = m+1 \Rightarrow \textbf{let}\ x = \langle v,w\rangle\ \textbf{in}\ \big\langle v, Take(n,w)\big\rangle\\ \textbf{end}.\end{array}$$

$$\vdash_{\mathrm{PA}} n \le L(x) \to Drop(n,x) = \begin{array}{l}\textbf{case}\\ \quad n = 0 \Rightarrow\ x\\ \quad n = m+1 \Rightarrow \textbf{let}\ x = \langle v,w\rangle\ \textbf{in}\ Drop(n,w)\\ \textbf{end}.\end{array}$$

Note that both programs share the same condition of regularity

$$\vdash_{\mathrm{PA}} n \le L(x) \wedge n = m+1 \wedge x = \langle v,w\rangle \to w < x \wedge m \le L(w)$$

which is trivially satisfied.

*Verification.* (1): This is proved by induction on $n$ as $\forall x(1)$. The base case is obvious. In the induction step take any $x$ such that $n+1 \le L(x)$. Then $x = \langle v,w\rangle$ for some $v,w$, where $n \le L(w)$. We obtain

$$L\ Take(n+1,\langle v,w\rangle) = L\big\langle v, Take(n,w)\big\rangle = L\ Take(n,w) + 1 \stackrel{\mathrm{IH}}{=} n+1.$$

Note that the induction hypothesis is applied with $w$ in place of $x$.

(2): By induction on $n$ as $\forall x(2)$. In the base case it suffices to take $y := x$ since $Take(0,x) \oplus x = 0 \oplus x = x$. In the induction step assume $n+1 \le L(x)$. Then $x = \langle v,w\rangle$ for some $v,w$. Since $n \le L(w)$ we get from IH applied with $w$ in place of $x$ that $w = Take(n,w) \oplus y$ for some $y$. We then have

$$\langle v,w\rangle = \langle v, Take(n,w) \oplus y\rangle = \langle v, Take(n,w)\rangle \oplus y = Take(n+1,\langle v,w\rangle) \oplus y.$$

The remaining properties (3) and (4) are proved similarly. □

**7.2.5 Zip and unzip.** The function $Zip(x,y)$ takes two lists of the same length and yields a list of pairs of the corresponding elements. The function satisfies

$$\vdash_{\mathrm{PA}} L(x) = L(y) \to L\ Zip(x,y) = L(x) \tag{1}$$

$$\vdash_{\mathrm{PA}} L(x) = L(y) \wedge i < L(x) \to Zip(x,y)[i] = \big\langle x[i], y[i]\big\rangle \tag{2}$$

and it is defined by list recursion with substitution in parameter as a primitive recursive function:

$Zip(0,0) = 0$
$Zip(\langle v_1,w_1\rangle,\langle v_2,w_2\rangle) = \big\langle\langle v_1,v_2\rangle, Zip(w_1,w_2)\big\rangle$.

The function $Unzip(z)$, the inverse of $Zip$, takes a list of pairs to a pair of lists. The function satisfies

$$\vdash_{\mathrm{PA}} \; L(x) = L(y) \to Unzip\, Zip(x,y) = \langle x, y\rangle \tag{3}$$

and it is defined by list recursion as a primitive recursive function:

$Unzip(0) = \langle 0, 0\rangle$
$Unzip(\langle v_1, v_2\rangle, w) = \big\langle \langle v_1, w_1\rangle, \langle v_2, w_2\rangle\big\rangle \leftarrow Unzip(w) = \langle w_1, w_2\rangle.$

Property $\forall y(1)$ is proved by list induction on $x$. In the base case take any $y$ and assume $L(0) = L(y)$. Then $y = 0$ and we are done since $L\, Zip(0,0) = L(0)$ from the definition. In the inductive case when $x = \langle v_1, w_1\rangle$ take any $y$ and assume $L\langle v_1, w_1\rangle = L(w_1) + 1 = L(y)$. Then $y = \langle v_2, w_2\rangle$ for some $v_2$ and $w_2$, and thus $L(w_1) = L(w_2)$. We have

$$L\, Zip(\langle v_1, w_1\rangle, \langle v_2, w_2\rangle) = L\big\langle \langle v_1, v_2\rangle, Zip(w_1, w_2)\big\rangle = L\, Zip(w_1, w_2) + 1 = \\ = L(w_1) + 1 = L\langle v_1, w_1\rangle.$$

Property $\forall x \forall y(2)$ is proved by induction on $i$. In the base case take any $x$ and $y$ such that $0 < L(x) = L(y)$. Then $x = \langle v_1, w_1\rangle$ and $y = \langle v_2, w_2\rangle$ for some $v_1, w_1, v_2, w_2$. Clearly $L(w_1) = L(w_2)$ and thus we obtain

$$Zip(\langle v_1, w_1\rangle, \langle v_2, w_2\rangle)[0] = \big(\big\langle \langle v_1, v_2\rangle, Zip(w_1, w_2)\big\rangle\big)[0] = \\ = \langle v_1, v_2\rangle = \big\langle \langle v_1, w_1\rangle[0], \langle v_2, w_2\rangle[0]\big\rangle.$$

In the inductive case take any $x$ and $y$ such that $i + 1 < L(x) = L(y)$. Then $x = \langle v_1, w_1\rangle$ and $y = \langle v_2, w_2\rangle$ for some $v_1, w_1, v_2, w_2$. Clearly $i < L(w_1) = L(w_2)$ and thus we obtain

$$Zip(\langle v_1, w_1\rangle, \langle v_2, w_2\rangle)[i+1] = \big(\big\langle \langle v_1, v_2\rangle, Zip(w_1, w_2)\big\rangle\big)[i+1] = Zip(w_1, w_2)[i] = \\ = \langle w_1[i], w_2[i]\rangle = \big\langle \langle v_1, w_1\rangle[i+1], \langle v_2, w_2\rangle[i+1]\big\rangle.$$

Property $\forall y(3)$ is proved by list induction on $x$ and is left to the reader.

**7.2.6 Interval.** The binary function $[m\,..\,n)$ returns the list of numbers from $m$ to $n-1$ if $m < n$; the list is empty if $m \geq n$. The function satisfies

$$\vdash_{\mathrm{PA}} \; L\,[m\,..\,n) = n \dot{-} m \tag{1}$$
$$\vdash_{\mathrm{PA}} \; i + m < n \to [m\,..\,n)[i] = m + i \tag{2}$$

and it is defined by recursion with measure $n \dot{-} m$ as a p.r. function by

$[m\,..\,n) = 0 \leftarrow m \geq n$
$[m\,..\,n) = \big\langle m, [m+1\,..\,n)\big\rangle \leftarrow m < n.$

Note that this is an example of function definition by backward recursion.

We usually intend to apply the operation $[m\,..\,n)$ only in cases when $m \le n$. For that we can take the following property as an alternative (conditional) program for computing the function:

$$\begin{aligned}\vdash_{\mathrm{PA}}\ m \le n \to [m\,..\,n) = \ & \mathbf{case}\\ & m = n \Rightarrow 0\\ & m \ne n \Rightarrow \langle m, [m+1\,..\,n)\rangle\\ & \mathbf{end}\end{aligned}$$

Its condition of regularity

$$\vdash_{\mathrm{PA}}\ m \le n \wedge m \ne n \to n \dot{-} (m+1) < n \dot{-} m \wedge m + 1 \le n$$

is trivially satisfied. Note that the program does not terminate for $m > n$.

*Verification.* (1): By induction with measure $n \dot{-} m$. Take any $m, n$ and consider two cases. If $m \ge n$ then $L\,[m\,..\,n) = L(0) = 0 = n \dot{-} m$. If $m < n$ then $m < m + 1 \le n$ and we obtain

$$L\,[m\,..\,n) = L\,\langle m, [m+1\,..\,n)\rangle = L\,[m+1\,..\,n) + 1 \stackrel{\mathrm{IH}}{=} n \dot{-} (m+1) + 1 = n \dot{-} m.$$

(2): This is proved by induction with measure $n \dot{-} m$ as $\forall i(2)$. Take any $m, n, i$ such that $i + m < n$ and consider two cases. If $i = 0$ then we have

$$[m\,..\,n)[0] = \langle m, [m+1\,..\,n)\rangle[0] = m = m + 0.$$

If $i = j + 1$ for some $j$ then $j + (m+1) = j + 1 + m < n$ and thus

$$\begin{aligned}[m\,..\,n)[j+1] = \langle m, [m+1\,..\,n)\rangle[j+1] = [m+1\,..\,n)[j] \stackrel{\mathrm{IH}}{=}\\ = m + 1 + j = m + (j+1).\end{aligned}$$

Note that the induction hypothesis is applied with $j$ in place of $i$. □

**7.2.7 List minimum.** The function $Minl(x)$ yields the minimal element of a non-empty list. The function satisfies

$$\vdash_{\mathrm{PA}}\ x \ne 0 \to Minl(x)\ \varepsilon\ x \tag{1}$$

$$\vdash_{\mathrm{PA}}\ x \ne 0 \wedge a\ \varepsilon\ x \to Minl(x) \le a \tag{2}$$

and it is defined by list recursion as a p.r. function:

$Minl\,\langle v, 0\rangle = v$
$Minl\,\langle v, w\rangle = \min(v, Minl(w)) \leftarrow w \ne 0.$

Note the default $Minl(0) = 0$.

We usually intend to apply the operation $Minl(\mathrm{x})$ only in cases when input lists are non-empty. For that we can take the following property as an alternative (conditional) program for computing list minimum:

$$\vdash_{\mathrm{PA}} \; x \neq 0 \to \mathit{Minl}(x) = \textbf{let } x = \langle v, w\rangle \textbf{ in}$$
$$\textbf{case}$$
$$w = 0 \Rightarrow v$$
$$w \neq 0 \Rightarrow \min(v, \mathit{Minl}(w))$$
$$\textbf{end}.$$

Its condition of regularity

$$\vdash_{\mathrm{PA}} \; x \neq 0 \wedge x = \langle v, w\rangle \wedge w \neq 0 \to w < x \wedge w \neq 0$$

is trivially satisfied.

*Verification.* (1),(2): By a straightforward structural list induction on $x$. □

**7.2.8 Deleting elements from lists.** The function $\mathit{Delall}(a, x)$ removes all occurrences of $a$ in the list $x$. The function satisfies

$$\vdash_{\mathrm{PA}} \; b \varepsilon \mathit{Delall}(a, x) \leftrightarrow b \varepsilon x \wedge b \neq a \tag{1}$$

and it is defined by list recursion on $x$ as a primitive recursive function:

$\mathit{Delall}(a, 0) = 0$
$\mathit{Delall}(a, \langle v, w\rangle) = \mathit{Delall}(a, w) \leftarrow a = v$
$\mathit{Delall}(a, \langle v, w\rangle) = \langle v, \mathit{Delall}(a, w)\rangle \leftarrow a \neq v.$

Property (1) is proved by list induction on $x$. The base case when $x = 0$ is straightforward since $\mathit{Delall}(a, 0) = 0$. In the inductive case when $x = \langle v, w\rangle$ consider two case. If $a = v$ then we have

$$b \varepsilon \mathit{Delall}(a, \langle v, w\rangle) \Leftrightarrow b \varepsilon \mathit{Delall}(a, w) \Leftrightarrow b \varepsilon w \wedge b \neq a \Leftrightarrow$$
$$(b = v \vee b \varepsilon w) \wedge b \neq a \Leftrightarrow b \varepsilon \langle v, w\rangle \wedge b \neq a.$$

If $a \neq v$ then we have

$$b \varepsilon \mathit{Delall}(a, \langle v, w\rangle) \Leftrightarrow b \varepsilon v, \mathit{Delall}(a, w) \Leftrightarrow b = v \vee b \varepsilon \mathit{Delall}(a, w) \Leftrightarrow$$
$$b = v \vee b \varepsilon w \wedge b \neq a \Leftrightarrow (b = v \vee b \varepsilon w) \wedge b \neq a \Leftrightarrow b \varepsilon \langle v, w\rangle \wedge b \neq a.$$

**7.2.9 Filter.** Let $A(x)$ be arbitrary but fixed unary predicate. The function $\mathit{Filter}_A(x)$ removes all elements from a list which do not satisfy the predicate. The function satisfies

$$\vdash_{\mathrm{PA}} \; a \varepsilon \mathit{Filter}_A(x) \leftrightarrow a \varepsilon x \wedge A(a) \tag{1}$$

and it is defined by structural list recursion as a p.r. predicate in $A$:

$\mathit{Filter}_A(0) = 0$
$\mathit{Filter}_A \langle v, w\rangle = \langle v, \mathit{Filter}_A(w)\rangle \leftarrow A(v)$
$\mathit{Filter}_A \langle v, w\rangle = \mathit{Filter}_A(w) \leftarrow \neg A(v).$

*Verification.* Property (1) is proved by structural induction on the list $x$. The base case is obvious. In the induction step, when $x = \langle v, w \rangle$ for some $v, w$, we consider two cases. If $A(v)$ then we have

$$a \,\varepsilon\, \mathit{Filter}_A \langle v, w \rangle \Leftrightarrow a \,\varepsilon\, \left\langle v, \mathit{Filter}_A(w) \right\rangle \Leftrightarrow a = v \vee a \,\varepsilon\, \mathit{Filter}_A(w) \overset{\text{IH}}{\Leftrightarrow}$$
$$a = v \vee a \,\varepsilon\, w \wedge A(a) \overset{(*)}{\Leftrightarrow} (a = v \vee a \,\varepsilon\, w) \wedge A(a) \Leftrightarrow a \,\varepsilon\, \langle v, w \rangle \wedge A(a).$$

The equivalence marked by $(*)$ is by case analysis on whether or not $a = v$. The case when $A(v)$ does not hold is similar. □

**7.2.10 Multiplicity.** The binary function $\#_a(x)$ counts the number of occurrences of the element $a$ in the list $x$. This is called the multiplicity of $a$ in $x$. The function satisfies

$$\vdash_{\mathrm{PA}} \#_a\langle b, 0 \rangle = (a =_* b) \tag{1}$$
$$\vdash_{\mathrm{PA}} \#_a(x \oplus y) = \#_a(x) + \#_a(y) \tag{2}$$

and it is defined by structural list recursion as a p.r. function:

$$\#_a(0) = 0$$
$$\#_a\langle v, w \rangle = (a =_* v) + \#_a(w).$$

We have also

$$\vdash_{\mathrm{PA}} a \,\varepsilon\, x \leftrightarrow \#_a(x) \neq 0. \tag{3}$$
$$\vdash_{\mathrm{PA}} x = 0 \leftrightarrow \forall a \, \#_a(x) = 0. \tag{4}$$

*Verification.* (1): Directly from definition.

(2): By structural induction on the list $x$. The base case is obvious. The induction step follows from

$$\#_a(\langle v, w \rangle \oplus y) = \#_a\langle v, w \oplus y \rangle = (a =_* v) + \#_a(w \oplus y) \overset{\text{IH}}{=}$$
$$= (a =_* v) + \#_a(w) + \#_a(y) = \#_a\langle v, w \rangle + \#_a(y).$$

(3): By a straightforward structural induction on the list $x$.

(4): By a simple case analysis on whether or not the list $x$ is empty. □