7.3 Sorting of Lists

7.3.1 Introduction. In this section we will consider the problem of sorting of lists. We will demonstrate the verification of two sorting algorithms: insertion sort and merge sort. We start by introducing into PA some of the specification predicates which are needed to specify and verify sorting algorithms.

7.3.2 Lower bounds of lists. The predicate $a \le x$ holds if the number a is a *lower bound* of the list x, i.e. we have $a \le b$ for every element b of x. The predicate is defined explicitly as primitive recursive by

$$a \le x \leftrightarrow \forall b(b \,\varepsilon\, x \to a \le b).$$

The predicate satisfies

$$\vdash_{\mathrm{PA}} \ a \le 0 \tag{1}$$
$$\vdash_{\mathrm{PA}} \ a \le \langle v, w\rangle \leftrightarrow a \le v \wedge a \le w \tag{2}$$
$$\vdash_{\mathrm{PA}} \ a \le b \wedge b \le x \to a \le x \tag{3}$$
$$\vdash_{\mathrm{PA}} \ a \le x \oplus y \leftrightarrow a \le x \wedge a \le y \tag{4}$$
$$\vdash_{\mathrm{PA}} \ x \sim y \to a \le x \leftrightarrow a \le y. \tag{5}$$

Proof. (1): Obvious. (2): This follows from

$$a \le \langle v, w\rangle \Leftrightarrow \forall b(b \,\varepsilon\, \langle v, w\rangle \to a \le b) \overset{7.1.13(2)}{\Leftrightarrow} \forall b(b = v \vee b \,\varepsilon\, w \to a \le b) \Leftrightarrow$$
$$\Leftrightarrow a \le v \wedge \forall b(b \,\varepsilon\, w \to a \le b) \Leftrightarrow a \le v \wedge a \le w.$$

(3): Obvious. (4): This follows from

$$a \le x \oplus y \Leftrightarrow \forall b(b \,\varepsilon\, x \oplus y \to a \le b) \overset{7.1.13(3)}{\Leftrightarrow} \forall b(b \,\varepsilon\, x \vee b \,\varepsilon\, y \to a \le b) \Leftrightarrow$$
$$\Leftrightarrow \forall b(b \,\varepsilon\, x \to a \le b) \wedge \forall b(b \,\varepsilon\, y \to a \le b) \Leftrightarrow a \le x \wedge a \le y.$$

(5) Suppose that $x \sim y$. We have

$$a \le x \Leftrightarrow \forall b(b \,\varepsilon\, x \to a \le b) \overset{7.3.5(7)}{\Leftrightarrow} \forall b(b \,\varepsilon\, y \to a \le b) \Leftrightarrow a \le y. \qquad \square$$

7.3.3 Ordered lists. The predicate $Ord(x)$ holds if x is an ordered list, i.e. the elements of the list x are stored in x in increasing order. The predicate is explicitly defined as primitive recursive by

$$Ord(x) \leftrightarrow \forall i \forall j(i < j < L(x) \to x[i] \le x[j]).$$

We list here some properties of ordered lists which we will use in sequel:

$$\vdash_{\mathrm{PA}} \ \mathit{Ord}(0) \tag{1}$$
$$\vdash_{\mathrm{PA}} \ \mathit{Ord}\,\langle v,w\rangle \leftrightarrow v \leq w \land \mathit{Ord}(w). \tag{2}$$
$$\vdash_{\mathrm{PA}} \ \mathit{Ord}\,\langle v,w\rangle \rightarrow a \leq \langle v,w\rangle \leftrightarrow a \leq v. \tag{3}$$

Proof. (1): Obvious. (2): This follows from

$$\begin{aligned}
&\mathit{Ord}\,\langle v,w\rangle \Leftrightarrow \forall i \forall j\big(i < j < L\,\langle v,w\rangle \rightarrow \langle v,w\rangle[i] \leq \langle v,w\rangle[j]\big) \overset{(*)}{\Leftrightarrow}\\
&\forall j\big(0 < j < L(w)+1 \rightarrow \langle v,w\rangle[0] \leq \langle v,w\rangle[j]\big) \land\\
&\quad \land \forall i_1 \forall j\big(i_1+1 < j < L(w)+1 \rightarrow \langle v,w\rangle[i_1+1] \leq \langle v,w\rangle[j]\big) \Leftrightarrow\\
&\forall j_1\big(j_1+1 < L(w)+1 \rightarrow v \leq \langle v,w\rangle[j_1+1]\big) \land\\
&\quad \land \forall i_1 \forall j_1\big(i_1+1 < j_1+1 < L(w)+1 \rightarrow w[i_1] \leq \langle v,w\rangle[j_1+1]\big) \Leftrightarrow\\
&\forall j_1\big(j_1 < L(w) \rightarrow v \leq w[j_1]\big) \land\\
&\quad \land \forall i_1 \forall j_1\big(i_1 < j_1 < L(w) \rightarrow w[i_1] \leq w[j_1]\big) \Leftrightarrow v \leq w \land \mathit{Ord}(w).
\end{aligned}$$

The step marked by $(*)$ is by case analysis on whether or not $i = 0$.

(3): If $\mathit{Ord}\,\langle v,w\rangle$ then $v \leq w$ by (2) and thus, by 7.3.2(3), we have

$$a \leq v \rightarrow a \leq w. \tag{$\dagger_1$}$$

We then obtain

$$a \leq \langle v,w\rangle \overset{7.3.2(2)}{\Leftrightarrow} a \leq v \land a \leq w \overset{(\dagger_1)}{\Leftrightarrow} a \leq v. \qquad \square$$

7.3.4 Permutations. We wish to introduce into PA the binary predicate $x \sim y$ holding if the list x is a permutation of the list y. For example:

$$\langle 1,2,3,0\rangle \quad \langle 2,1,3,0\rangle \quad \langle 2,3,1,0\rangle \quad \langle 1,3,2,0\rangle \quad \langle 3,1,2,0\rangle \quad \langle 3,2,1,0\rangle$$

are all permutations of the three-element list $\langle 1,2,3,0\rangle$. The standard mathematical definition uses a second-order concept (bijections over finite sets) which is not expressible directly in first-order arithmetic. Our definition of the predicate in PA is based on the following simple observation:

> two lists are permutations precisely when every number has the same multiplicity in either list.

Thus we can define the predicate explicitly by

$$x \sim y \leftrightarrow \forall a\, \#_a(x) = \#_a(y)\,.$$

Note that from 7.2.10(3) we get

$$\vdash_{\mathrm{PA}} \ x \sim y \leftrightarrow \forall a\big(a \,\varepsilon\, x \rightarrow \#_a(x) = \#_a(y)\big) \land \forall a\big(a \,\varepsilon\, y \rightarrow \#_a(x) = \#_a(y)\big).$$

Consequently, the predicate $x \sim y$ is primitive recursive.

7.3.5 Basic properties of permutations. First note the predicate $x \sim y$ constitutes an equivalence relation which is reflexive, symmetric and transitive. This is expressed in that order by

$$\vdash_{\mathrm{PA}} \; x \sim x \tag{1}$$
$$\vdash_{\mathrm{PA}} \; x \sim y \to y \sim x \tag{2}$$
$$\vdash_{\mathrm{PA}} \; x \sim y \land y \sim z \to x \sim z. \tag{3}$$

Congruence properties of permutations are expressed by

$$\vdash_{\mathrm{PA}} \; x \sim y \to \langle a, x\rangle \sim \langle a, y\rangle \tag{4}$$
$$\vdash_{\mathrm{PA}} \; x \sim y \to L(x) = L(y) \tag{5}$$
$$\vdash_{\mathrm{PA}} \; x_1 \sim y_1 \land x_2 \sim y_2 \to x_1 \oplus x_2 \sim y_1 \oplus y_2 \tag{6}$$
$$\vdash_{\mathrm{PA}} \; x \sim y \land a \mathrel{\varepsilon} x \to a \mathrel{\varepsilon} y. \tag{7}$$

There is one cancellation law, namely:

$$\vdash_{\mathrm{PA}} \; x_1 \oplus \langle a, x_2\rangle \sim y_1 \oplus \langle a, y_2\rangle \leftrightarrow x_1 \oplus x_2 \sim y_1 \oplus y_2. \tag{8}$$

Finally, we have also the following recurrent properties of permutations:

$$\vdash_{\mathrm{PA}} \; x \sim 0 \leftrightarrow x = 0 \tag{9}$$
$$\vdash_{\mathrm{PA}} \; x \sim \langle v, w\rangle \leftrightarrow \exists z_1 \exists z_2 (x = z_1 \oplus \langle v, z_2\rangle \land w \sim z_1 \oplus z_2). \tag{10}$$

In the sequel we will use these properties without explicitly referring to them.

Proof. Properties (1)–(3) hold trivially. Property (4) follows directly from the definition. Properties (6)–(9) follow from the properties of the multiplicity function (see Par. 7.2.10).

(10): In the direction ($\to$) assume $x \sim \langle v, w\rangle$. Then $v \mathrel{\varepsilon} x$ by (7) and thus, by 7.1.13(4), we have $x = z_1 \oplus \langle v, z_2\rangle$ for some z_1, z_2. Now it suffices to apply (8) to get $w \sim z_1 \oplus z_2$. The reverse direction ($\leftarrow$) follows from (8).

(5): This is proved as $\forall y(5)$ by structural induction on the list x. The base case is straightforward. In the induction step, when $x = \langle v, w\rangle$ for some v, w, take any y such that $\langle v, w\rangle \sim y$. By (10), there are lists z_1, z_2 such that $y = z_1 \oplus \langle v, z_2\rangle$ and $w \sim z_1 \oplus z_2$. We then obtain

$$\begin{aligned} L\langle v, w\rangle &= L(w) + 1 \overset{\mathrm{IH}}{=} L(z_1 \oplus z_2) + 1 = L(z_1) + L(z_2) + 1 = \\ &= L(z_1) + L\langle v, z_2\rangle = L(z_1 \oplus \langle v, z_2\rangle). \end{aligned}$$

Note that the induction hypothesis is applied with $z_1 \oplus z_2$ in place of y. $\square$

Insertion Sort

7.3.6 Introduction. The simplest sorting algorithm is *insertion sort* which takes order $\mathcal{O}(L(x)^2)$ time to sort a list x. Insertion sort works on a non-empty list by recursively sorting its tail and then inserts its first element into the sorted list.

7.3.7 Insertion. At the heart of insertion sort algorithm is the insertion function $\mathit{Insert}(a,x)$ which takes an ordered list x and yields a new one by inserting the element a into it. The function satisfies

$$\vdash_{\mathrm{PA}} \mathit{Insert}(a,x) \sim \langle a,x\rangle \tag{1}$$

$$\vdash_{\mathrm{PA}} \mathit{Ord}(x) \to \mathit{Ord}\,\mathit{Insert}(a,x) \tag{2}$$

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

$$\begin{aligned}
&\mathit{Insert}(a,0) = \langle a,0\rangle\\
&\mathit{Insert}(a,\langle v,w\rangle) = \langle a,v,w\rangle \leftarrow a \le v\\
&\mathit{Insert}(a,\langle v,w\rangle) = \big\langle v,\mathit{Insert}(a,w)\big\rangle \leftarrow a > v.
\end{aligned}$$

Verification. (1): By structural induction on the list x. The base case is obvious. In the induction step when $x = \langle v,w\rangle$ we consider two cases. If $a \le v$ then the claim follows directly from the definition. Otherwise $a > v$ and then

$$\mathit{Insert}(a,\langle v,w\rangle) \sim \big\langle v,\mathit{Insert}(a,w)\big\rangle \overset{\mathrm{IH}}{\sim} \langle v,a,w\rangle \sim \langle a,v,w\rangle.$$

As a simple consequence of (1) and 7.3.2(5) we get the following

$$\vdash_{\mathrm{PA}} b \le \mathit{Insert}(a,x) \leftrightarrow b \le a \land b \le x. \tag{$\dagger_1$}$$

(2): By structural induction on the list x. The base case is straightforward. In the induction step, when $x = \big\langle v,w\big\rangle$ for some v,w, assume $\mathit{Ord}\,\langle v,w\rangle$ and consider two cases. If $a \le v$ then we have

$$\mathit{Ord}\,\mathit{Insert}(a,\langle v,w\rangle) \Leftrightarrow \mathit{Ord}\,\langle a,v,w\rangle \overset{7.3.3(2)}{\Leftrightarrow} \mathit{Ord}\,\langle v,w\rangle \land a \le \langle v,w\rangle.$$

The last follows from assumptions by 7.3.3(3). If $a > v$ then we have

$$\begin{aligned}
&\mathit{Ord}\,\mathit{Insert}(a,\langle v,w\rangle) \Leftrightarrow \mathit{Ord}\,\langle v,\mathit{Insert}(a,w)\rangle \overset{7.3.3(2)}{\Leftrightarrow}\\
&\mathit{Ord}\,\mathit{Insert}(a,w) \land v \le \mathit{Insert}(a,w) \overset{(\dagger_1)}{\Leftrightarrow} \mathit{Ord}\,\mathit{Insert}(a,w) \land v \le a \land v \le w.
\end{aligned}$$

The last follows from assumptions and IH. □

7.3.8 Insertion sort. The function $\mathit{Isort}(x)$ recursively sorts the tail of an non-empty list and then inserts its first element into the sorted one. The function satisfies

$$\vdash_{\mathrm{PA}} \; \mathit{Isort}(x) \sim x \tag{1}$$
$$\vdash_{\mathrm{PA}} \; \mathit{Ord}\,\mathit{Isort}(x) \tag{2}$$

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

$\mathit{Isort}(0) = 0$
$\mathit{Isort}\,\langle v, w\rangle = \mathit{Insert}(v, \mathit{Isort}(w))$.

Verification. (1): By structural list induction. The base case is straightforward and the induction step follows from

$$\mathit{Isort}\,\langle v, w\rangle \sim \mathit{Insert}(v, \mathit{Isort}(w)) \overset{7.3.7(1)}{\sim} \big\langle v, \mathit{Isort}(w)\big\rangle \overset{\mathrm{IH}}{\sim} \langle v, w\rangle.$$

(2): By structural list induction. The base case follows from 7.3.3(1). In the induction step, when $x = \big\langle v, w\big\rangle$ for some v, w, assume $\mathit{Ord}\,\langle v, w\rangle$. Then $\mathit{Ord}(w)$ by 7.3.3(2) and we get from IH:

$$\mathit{Ord}\,\mathit{Isort}(w) \overset{7.3.7(2)}{\Rightarrow} \mathit{Ord}\,\mathit{Insert}(v, \mathit{Isort}(w)) \Rightarrow \mathit{Ord}\,\mathit{Isort}\,\langle v, w\rangle. \qquad \square$$

Merge Sort

7.3.9 Introduction. More efficient sorting algorithm than insertion sort is *merge sort* which takes order $\mathcal{O}(L(x) \lg L(x))$ time to sort a list x The algorithm sorts a list by dividing it into two roughly equal parts. Each part is then recursively sorted and the resulting lists are merged into one list.

Our implementation uses the discrimination on whether or not $L(x) \le 1$. As we have

$$\vdash_{\mathrm{PA}} \; L(x) \le 1 \leftrightarrow (\pi_2(x) =_* 0) = 1,$$

the evaluation of the variant $L(x) \le 1$ takes constant time provided the expression $\pi_2(x) =_* 0$ is taken as its characteristic term.

7.3.10 Splitting the list into two halves. The function $\mathit{Split}(x)$ divides a list into two lists: the length of the first one is at most one more than the length of the second. The function satisfies

$$\vdash_{\mathrm{PA}} \; \exists y \exists z\, \mathit{Split}(x) = \langle y, z\rangle \tag{1}$$
$$\vdash_{\mathrm{PA}} \; \mathit{Split}(x) = \langle y, z\rangle \to x \sim y \oplus z \tag{2}$$
$$\vdash_{\mathrm{PA}} \; \mathit{Split}(x) = \langle y, z\rangle \to (L(y) = L(z) \lor L(y) = L(z) + 1) \tag{3}$$

and it is defined by course of values recursion with measure $L(x)$ as a p.r. function by

$\mathit{Split}(x) = \langle x, 0\rangle \leftarrow L(x) \le 1$

$$Split(x) = \big\langle \langle u, y\rangle, \langle v, z\rangle \big\rangle \leftarrow L(x) > 1 \wedge x = \langle u, v, w\rangle \wedge Split(w) = \langle y, z\rangle.$$

Verification. (2): By induction with measure $L(x)$ as $\forall y \forall z(2)$. Take any y, z such that $Split(x) = \langle y, z\rangle$ and consider two cases. The case when $L(x) \leq 1$ is obvious. So suppose that $L(x) > 1$. Then $x = \langle u, v, w\rangle$ for some u, v, w. By (1) there are y_1, z_1 such that $Split(w) = \langle y_1, z_1\rangle$. By definition $\langle u, y_1\rangle = y$ and $\langle v, z_1\rangle = z$. We then obtain

$$\langle u, v, w\rangle \overset{\mathrm{IH}}{\sim} \langle u, v, y_1 \oplus z_1\rangle \sim \langle u, y_1\rangle \oplus \langle v, z_1\rangle \sim y \oplus z.$$

(1),(3): This is proved similarly. □

7.3.11 Merging two ordered lists into one. The function $Merge(x, y)$ merges two ordered lists into one ordered list. The function satisfies

$$\vdash_{\mathrm{PA}} Merge(x, y) \sim x \oplus y \tag{1}$$
$$\vdash_{\mathrm{PA}} Ord(x) \wedge Ord(y) \to Ord\, Merge(x, y) \tag{2}$$

and it is defined by course of values recursion with measure $L(x) + L(y)$ as a p.r. function by

$$\begin{aligned}
&Merge(0, y) = y\\
&Merge(\langle v_1, w_1\rangle, 0) = \langle v_1, w_1\rangle\\
&Merge(\langle v_1, w_1\rangle, \langle v_2, w_2\rangle) = \big\langle v_1, Merge(w_1, \langle v_2, w_2\rangle)\big\rangle \leftarrow v_1 \leq v_2\\
&Merge(\langle v_1, w_1\rangle, \langle v_2, w_2\rangle) = \big\langle v_2, Merge(\langle v_1, w_1\rangle, w_2)\big\rangle \leftarrow v_1 > v_2.
\end{aligned}$$

Verification. (1): By course of values induction with measure $L(x) + L(y)$. We consider two cases. The case when either $x = 0$ or $y = 0$ is straightforward. So suppose $x = \langle v_1, w_1\rangle$ and $y = \langle v_2, w_2\rangle$ for some v_1, w_1, v_2, w_2. If $v_1 \leq v_2$ then we have

$$\begin{aligned}
Merge(\langle v_1, w_1\rangle, \langle v_2, w_2\rangle) \sim \big\langle v_1, Merge(w_1, \langle v_2, w_2\rangle)\big\rangle &\overset{\mathrm{IH}}{\sim}\\
\sim \langle v_1, w_1 \oplus \langle v_2, w_2\rangle\rangle \sim \langle v_1, w_1\rangle \oplus \langle v_2, w_2\rangle&.
\end{aligned}$$

The subcase when $v_1 < v_2$ has a similar proof.

As a simple consequence of (1) and 7.3.2(5) we get

$$\vdash_{\mathrm{PA}} a \leq Merge(x, y) \leftrightarrow a \leq x \wedge a \leq y. \tag{$\dagger_1$}$$

(2): By course of values induction with measure $L(x) + L(y)$. Assume $Ord(x)$ and $Ord(y)$, and consider two cases. If $x = 0$ or $y = 0$ then the property holds trivially. So suppose $x = \langle v_1, w_1\rangle$ and $y = \langle v_2, w_2\rangle$ for some v_1, w_1, v_2, w_2. If $v_1 \leq v_2$ then we have

$$Ord\ Merge(\langle v_1, w_1\rangle, \langle v_2, w_2\rangle) \Leftrightarrow Ord\ \big\langle v_1, Merge(w_1, \langle v_2, w_2\rangle)\big\rangle \overset{7.3.3(2)}{\Leftrightarrow}$$
$$Ord\ Merge(w_1, \langle v_2, w_2\rangle) \wedge v_1 \le Merge(w_1, \langle v_2, w_2\rangle) \overset{(\dagger_1)}{\Leftrightarrow}$$
$$Ord\ Merge(w_1, \langle v_2, w_2\rangle) \wedge v_1 \le w_1 \wedge v_1 \le \langle v_2, w_2\rangle.$$

The last follows from IH and from assumptions by 7.3.3(2) and 7.3.3(3). The subcase when $v_1 < v_2$ is similar. □

7.3.12 Merge sort. The function $Msort(x)$ sorts a list by dividing it into two equal parts. Each part is then recursively sorted and the resulting lists are merged together. The function $Msort(x)$ satisfies

$$\vdash_{PA} Msort(x) \sim x \tag{1}$$
$$\vdash_{PA} Ord\ Msort(x) \tag{2}$$

and it is defined by course of values recursion with measure $L(x)$ as a p.r. function by

$$Msort(x) = x \leftarrow L(x) \le 1$$
$$Msort(x) = Merge(Msort(y), Msort(z)) \leftarrow L(x) > 1 \wedge Split(x) = \langle y, z\rangle.$$

Its conditions of regularity

$$\vdash_{PA} L(x) > 1 \wedge Split(x) = \langle y, z\rangle \to L(y) < L(x) \tag{3}$$
$$\vdash_{PA} L(x) > 1 \wedge Split(x) = \langle y, z\rangle \to L(z) < L(x) \tag{4}$$

follows from 7.3.5(5) and 7.3.10(2)(3).

Verification. (1): By course of values induction with measure $L(x)$. We consider two cases. The case when $L(x) \le 1$ is obvious. So suppose $L(x) > 1$. By 7.3.10(1) there are y, z such that $Split(x) = \langle y, z\rangle$. Note that $L(y) < L(x)$ and $L(z) < L(x)$ by (3),(4). We have

$$Msort(x) \sim Merge(Msort(y), Msort(z)) \overset{7.3.11(1)}{\sim}$$
$$\sim Msort(y) \oplus Msort(z) \overset{IH}{\sim} y \oplus z \overset{7.3.10(2)}{\sim} x.$$

(2): By course of values induction with measure $L(x)$. We consider two cases. The case when $L(x) \le 1$ is obvious. So suppose $L(x) > 1$. By 7.3.10(1) there are y, z such that $Split(x) = \langle y, z\rangle$. Note that $L(y) < L(x)$ and $L(z) < L(x)$ by (3),(4). We have by IH

$$Ord\ Msort(y) \wedge Ord\ Msort(z) \overset{7.3.11(2)}{\Rightarrow} Ord\ Merge(Msort(y), Msort(z)) \Rightarrow$$
$$\Rightarrow \quad Ord\ Msort(x).$$
□