7.1 Lists

7.1.1 Introduction. In this section we will show how to arithmetize *lists of natural numbers*. In most functional programming languages the type Ln of such lists can be defined by a *union type*:

$$Ln = Nil \mid Cons(\mathrm{N}, Ln).$$

A value of type Ln is therefore either the *empty* list Nil or a non-empty list of the form $Cons(v, w)$, where v is its first *element* of of type N and w is a value of type Ln called the *tail* of that non-empty list. The constant Nil and the function $Cons$ are called *constructors*.

7.1.2 Constructors of lists. Arithmetization of lists is done with the help of the following two constructors: the first one is the number 0 and the second is the pairing function $\langle v, w\rangle$. From the properties of the pairing function we obtain

$$\vdash_{\mathrm{PA}} 0 \neq \langle v, w\rangle$$
$$\vdash_{\mathrm{PA}} \langle v_1, w_1\rangle = \langle v_2, w_2\rangle \to v_1 = v_2 \wedge w_1 = w_2.$$

The first property says that the constructors are pairwise disjoint and the second that the functional constructor $\langle v, w\rangle$ is an injective mapping.

We obtain the pattern matching style of definitions of functions operating over lists with conditionals of the form

case
$x = 0 \Rightarrow \beta_1$
$x = \langle v, w\rangle \Rightarrow_{v,w} \beta_2[x, v, w]$.
end

This is called *discrimination on the constructors of lists*.

The above conditional is evaluated as follows. First note that the expression $x =_* 0$ is the characteristic term of its first variant, and the expression $x \neq_* 0$ is the characteristic term of its second variant as we have

$$\vdash_{\mathrm{PA}} \exists v \exists w\, x = \langle v, w\rangle \leftrightarrow x \neq 0.$$

Note also that we have

$$\vdash_{\mathrm{PA}} x = \langle v, w\rangle \to v = \pi_1(x) \wedge w = \pi_2(x)$$

and therefore, the terms $\pi_1(x)$ and $\pi_2(x)$ are the witnessing terms for the output variables v, w of the second variant of the conditional.

7.1.3 List representation of N**.** The pairing function $\langle x, y\rangle$ permits an extremely simple uniform coding of finite sequences over natural numbers. We

assign the code 0 to the empty sequence $\varnothing$. A non-empty sequence $x_1, \ldots, x_n$ is coded by the number $\langle x_1, x_2, \ldots, x_n, 0\rangle$ as shown in Fig. 7.1

The reader will note that the assignment of codes is one to one, every finite sequence of natural numbers is coded by exactly one natural number, and vice versa, every natural number is the code of exactly one finite sequence of natural numbers. This is called *list representation* of numbers. Codes of finite sequences are called *lists* in computer science and this is how we will be calling them from now on.

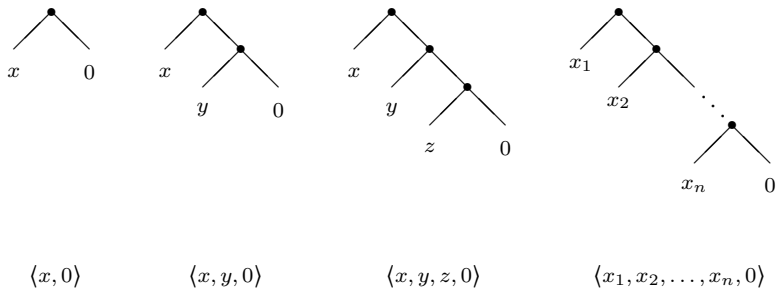

Fig. 7.1 List representation of natural numbers

7.1.4 Case analysis on lists. From properties of the pairing function we can see that every list x is either the empty list 0 or can be uniquely be written in the form $\langle v, w\rangle$, where the number v is called the *head* of the list x and the number w is called the *tail* of the list x. In particular

$$\vdash_{\mathrm{PA}} \; x = 0 \vee \exists v \exists w \, x = \langle v, w\rangle.$$

This is called the principle of *structural case analysis on the constructors of the list* x.

7.1.5 Structural induction on lists. The principle of structural induction over lists can be informally stated as follows. To prove by list induction that a property holds for every list it suffices to prove:

Base case: the property holds for the empty list 0.
Induction step: if the property holds for the list w then it holds also for the list $\langle v, w\rangle$.

This is expressed formally in PA as follows. Let $\varphi[x]$ be a formula of PA with the indicated variable x free. The principle of *list induction on* x *for* $\varphi[x]$ is the following one:

$$\vdash_{\mathrm{PA}} \; \varphi[0] \wedge \forall v \forall w (\varphi[w] \to \varphi[\langle v, w\rangle]) \to \varphi[x].$$

Note that the formula $\varphi[x]$ may contain additional variables as parameters.

Proof. The principle of list induction is proved as follows. Under the assumptions $\varphi[0]$ and $\forall v \forall w(\varphi[w] \to \varphi[\langle v, w\rangle])$ we prove that $\varphi[x]$ holds for every x by complete induction on x. So take any x and consider two cases. If $x = 0$ then the claim follows directly from the first assumption. Otherwise, x is of the form $\langle v, w\rangle$ for some v, w. By **??**(**??**), we have $w < \langle v, w\rangle$ and thus $\varphi[w]$ by IH. We obtain $\varphi[\langle v, w\rangle]$ from the second assumption.

7.1.6 Structural recursion on lists. List induction is used to prove properties of functions defined by the scheme of *list recursion*. In its simplest form, the operator of list recursion introduces a function f from two functions g and h satisfying

$$\begin{aligned} f(x, y) = \mathbf{case}& \\ & x = 0 \Rightarrow g(y) \\ & x = \langle v, w\rangle \Rightarrow h\big(v, w, f(w, y), y\big) \\ \mathbf{end}&. \end{aligned}$$

Note that this is a recursive definition regular in the first argument with discrimination on the constructors of lists (output variables of the second variant are omitted). The following identities form the clausal form of the above definition

$$\begin{aligned} &f(0, y) = g(y) \\ &f(\langle v, w\rangle, y) = h(v, w, f(w, y), y). \end{aligned}$$

Note that this is a recursive definition regular in the first argument. Similar schemes, when we allow terms with arbitrary number of parameters on the right-hand side of the above identities, substitution in parameters, or even nested recursive applications, will be also called definitions by list recursion.

7.1.7 List length. The function $L(x)$ yields the length of the list x:

$$L\,\langle x_1, x_2, \ldots, x_n, 0\rangle = n.$$

It is defined by parameterless structural list recursion as a p.r. function:

$$\begin{aligned} &L(0) = 0 \\ &L\,\langle v, w\rangle = L(w) + 1. \end{aligned}$$

7.1.8 List indexing. The binary function $x[i]$ yields the $(i+1)$-st element of the list x (counting from 0):

$$\langle x_0, \ldots, x_i, \ldots, x_{n-1}, 0\rangle[i] = \begin{cases} x_i & \text{if } i < n, \\ 0 & \text{otherwise.} \end{cases}$$

The function is defined by primitive recursion on i with substitution in parameter as p.r. function:

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

Note that $0[i] = 0$ by default.

Usually we intend to apply the operations $x[i]$ only in cases when $i < L(x)$. We can take the following property as alternative programs for computing the function in such cases:

$$\vdash_{\mathrm{PA}} \; i < L(x) \to x[i] = \textbf{case} \; i = 0 \Rightarrow \textbf{let}\ x = \langle v, w\rangle\ \textbf{in}\ v \; i = j+1 \Rightarrow \textbf{let}\ x = \langle v, w\rangle\ \textbf{in}\ w[j] \; \textbf{end}.$$

Its condition of regularity

$$\vdash_{\mathrm{PA}} \; i < L(x) \wedge i = j+1 \wedge x = \langle v, w\rangle \to w < x \wedge j < L(w)$$

is trivially satisfied.

7.1.9 Remark. The following property can be used as an alternative definition of list indexing:

$$\vdash_{\mathrm{PA}} \; x[i] = \pi_1\, \pi_2^i(x). \tag{1}$$

This is proved by (mathematical) induction on i as $\forall x(1)$ In the base case take any x and consider two cases. If $x = 0$ then $0[0] = 0 = \pi_1(0) = \pi_1\, \pi_2^0(0)$; if $x = \langle v, w\rangle$ for some v, w then $\langle v, w\rangle[0] = v = \pi_1\langle v, w\rangle = \pi_1\, \pi_2^0\langle v, w\rangle$. In the induction step take any x and consider the same two cases. If $x = 0$ then

$$0[i+1] = 0 = \pi_1(0) \overset{??(??)}{=} \pi_1\, \pi_2^{i+1}(0).$$

Otherwise $x = \langle v, w\rangle$ for some v, w and we obtain

$$\langle v, w\rangle[i+1] = w[i] \overset{\mathrm{IH}}{=} \pi_1\, \pi_2^i(w) = \pi_1\, \pi_2^i\, \pi_2\langle v, w\rangle = \pi_1\, \pi_2^{i+1}\langle v, w\rangle.$$

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

7.1.10 List concatenation. The binary function $x \oplus y$ concatenates two lists together to form a new one:

$$\langle x_1, \ldots, x_n, 0\rangle \oplus \langle y_1, \ldots, y_m, 0\rangle = \langle x_1, \ldots, x_n, y_1, \ldots, y_m, 0\rangle.$$

The function is defined by structural list recursion on x as a p.r. function by

$0 \oplus y = y$
$\langle v, w\rangle \oplus y = \langle v, w \oplus y\rangle$.

We can use the recurrences directly for computation. For example:

$$\langle 1,2,3,0\rangle \oplus \langle 4,5,0\rangle = \big\langle 1, \langle 2,3,0\rangle \oplus \langle 4,5,0\rangle\big\rangle = \big\langle 1,2, \langle 3,0\rangle \oplus \langle 4,5,0\rangle\big\rangle = \\ = \big\langle 1,2,3,0 \oplus \langle 4,5,0\rangle\big\rangle = \langle 1,2,3,4,5,0\rangle$$

Note that during the computation there is no need to convert the values into monadic (or binary) notation.

7.1.11 Basic properties of list concatenation. We have

$$\vdash_{\mathrm{PA}} x \oplus y = 0 \leftrightarrow x = 0 \wedge y = 0 \tag{1}$$
$$\vdash_{\mathrm{PA}} x \oplus 0 = x \tag{2}$$
$$\vdash_{\mathrm{PA}} x \oplus (y \oplus z) = (x \oplus y) \oplus z \tag{3}$$
$$\vdash_{\mathrm{PA}} x \oplus y = x \oplus z \to y = z \tag{4}$$
$$\vdash_{\mathrm{PA}} x \oplus \langle a,0\rangle = y \oplus \langle b,0\rangle \to x = y \wedge a = b \tag{5}$$
$$\vdash_{\mathrm{PA}} x \oplus z = y \oplus z \to x = y \tag{6}$$
$$\vdash_{\mathrm{PA}} L(x \oplus y) = L(x) + L(y) \tag{7}$$
$$\vdash_{\mathrm{PA}} i < L(x) \to (x \oplus y)[i] = x[i] \tag{8}$$
$$\vdash_{\mathrm{PA}} i < L(y) \to (x \oplus y)[L(x) + i] = y[i]. \tag{9}$$

In the sequel we will use these properties without explicitly referring to them. Note that (3) says that list concatenation is an associative operation. For this reason we will not be using any parentheses in expressions like $\tau_1 \oplus \tau_2 \oplus \tau_3$.

Proof. (1): By case analysis on whether or not the list x is empty.

(2): By a straightforward structural list induction.

(3): This is proved by structural induction on the list x. The base case follows from $0 \oplus (y \oplus z) = y \oplus z = (0 \oplus y) \oplus z$. In the induction step we have

$$\langle v,w\rangle \oplus (y \oplus z) = \langle v, w \oplus (y \oplus z)\rangle \overset{\text{IH}}{=} \langle v, (w \oplus y) \oplus z\rangle = \\ = \langle v, w \oplus y\rangle \oplus z = (\langle v,w\rangle \oplus y) \oplus z.$$

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

$$\langle v,w\rangle \oplus y = \langle v,w\rangle \oplus z \Rightarrow \langle v, w \oplus y\rangle = \langle v, w \oplus z\rangle \Rightarrow w \oplus y = w \oplus z \overset{\text{IH}}{\Rightarrow} y = z.$$

(5): By structural induction on the list x as $\forall y(5)$. In the base case take any y and consider two cases. If $y = 0$ then

$$0 \oplus \langle a,0\rangle = 0 \oplus \langle b,0\rangle \Rightarrow \langle a,0\rangle = \langle b,0\rangle \Rightarrow a = b \Rightarrow 0 = 0 \wedge a = b.$$

The case when $y = \langle v_2, w_2\rangle$ for some v_2, w_2 leads to contradiction:

$$0 \oplus \langle a, 0\rangle = \langle v_2, w_2\rangle \oplus \langle b, 0\rangle \Rightarrow \langle a, 0\rangle = \left\langle v_2, w_2 \oplus \langle b, 0\rangle\right\rangle \Rightarrow$$
$$\Rightarrow 0 = w_2 \oplus \langle b, 0\rangle \overset{(1)}{\Rightarrow} 0 = \langle b, 0\rangle.$$

In the induction step, when $x = \langle v_1, w_1\rangle$ for some v_1, w_1, take any y and consider two cases. The case $y = 0$ leads to contradiction by similar arguments as above. So it must be $y = \langle v_2, w_2\rangle$ for some v_2, w_2. We then have

$$\langle v_1, w_1\rangle \oplus \langle a, 0\rangle = \langle v_2, w_2\rangle \oplus \langle b, 0\rangle \Rightarrow \left\langle v_1, w_1 \oplus \langle a, 0\rangle\right\rangle = \left\langle v_2, w_2 \oplus \langle b, 0\rangle\right\rangle \Rightarrow$$
$$\Rightarrow v_1 = v_2 \wedge w_1 \oplus \langle a, 0\rangle = w_2 \oplus \langle b, 0\rangle \overset{\text{IH}}{\Rightarrow} v_1 = v_2 \wedge w_1 = w_2 \wedge a = b \Rightarrow$$
$$\Rightarrow \langle v_1, w_1\rangle = \langle v_2, w_2\rangle \wedge a = b.$$

Note that the induction hypothesis is applied with w_2 in place of y.

(6): By structural induction on the list z as $\forall x \forall y(6)$. The base case is follows from (2). In the induction step take any x, y and we have

$$x \oplus \langle v, w\rangle = y \oplus \langle v, w\rangle \Rightarrow x \oplus \langle v, 0 \oplus w\rangle = y \oplus \langle v, 0 \oplus w\rangle \Rightarrow$$
$$\Rightarrow x \oplus (\langle v, 0\rangle \oplus w) = y \oplus (\langle v, 0\rangle \oplus w) \overset{(3)}{\Rightarrow}$$
$$\Rightarrow (x \oplus \langle v, 0\rangle) \oplus w = (y \oplus \langle v, 0\rangle) \oplus w \overset{\text{IH}}{\Rightarrow}$$
$$\Rightarrow x \oplus \langle v, 0\rangle = y \oplus \langle v, 0\rangle \overset{(5)}{\Rightarrow} x = y.$$

Note that the induction hypothesis is applied with $x \oplus \langle v, 0\rangle$ in place of x and with $y \oplus \langle v, 0\rangle$ in place of y.

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

(8): By structural induction on the list x as $\forall i(8)$. 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

$$(\langle v, w\rangle \oplus y)[0] = \langle v, w \oplus y\rangle[0] = v = \langle v, w\rangle[0].$$

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

$$(\langle v, w\rangle \oplus y)[j + 1] = \langle v, w \oplus y\rangle[j + 1] = (w \oplus y)[j] \overset{\text{IH}}{=} w[j] = \langle v, w\rangle[j + 1].$$

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

(9): By a straightforward structural induction on the list x. □

7.1.12 List membership. The binary predicate $x \varepsilon y$ holds if the number x is an element of the list y:

$$x \varepsilon \langle y_1, \ldots, y_n, 0\rangle \quad \text{if } x = y_i \text{ for some } 1 \le i \le n.$$

The list membership predicate is defined explicitly as primitive recursive by

$$x \varepsilon y \leftrightarrow \exists i(i < L(y) \wedge x = y[i]).$$

Note that from the property **??**(**??**) of the pairing function we get

$$\vdash_{\mathrm{PA}} \; x \varepsilon y \to x < y$$

and therefore

$$\vdash_{\mathrm{PA}} \; \forall x(x \varepsilon y \to \varphi[x]) \leftrightarrow \forall x \leq y(x \varepsilon y \to \varphi[x])$$

for every formula $\varphi[x]$ of PA. The universal quantifier $\forall x$ in the contexts like $\forall x(x \varepsilon \ldots \to \cdots)$ can be bounded and thus it can be used in explicit definitions of primitive recursive predicates. Similarly for existential quantifiers.

7.1.13 Basic properties of list membership. We have

$$\vdash_{\mathrm{PA}} \; x \notin 0 \tag{1}$$
$$\vdash_{\mathrm{PA}} \; x \varepsilon \langle v, w \rangle \leftrightarrow x = v \vee x \varepsilon w \tag{2}$$
$$\vdash_{\mathrm{PA}} \; x \varepsilon y \oplus z \leftrightarrow x \varepsilon y \vee x \varepsilon z \tag{3}$$
$$\vdash_{\mathrm{PA}} \; x \varepsilon y \leftrightarrow \exists z_1 \exists z_2 \, y = z_1 \oplus \langle x, z_2 \rangle. \tag{4}$$

In the sequel we will use the properties (1)–(3) without explicitly referring to them. Note also that the last property (4) can be used as alternative definition of the list membership predicate.

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

$$\begin{aligned} &x \varepsilon \langle v, w \rangle \Leftrightarrow \exists i\big(i < L\langle v, w \rangle \wedge x = \langle v, w \rangle[i]\big) \overset{(*_1)}{\Leftrightarrow} \\ &0 < L(w) + 1 \wedge x = \langle v, w \rangle[0] \vee \exists j\big(j + 1 < L(w) + 1 \wedge x = \langle v, w \rangle[j + 1]\big) \Leftrightarrow \\ &x = v \vee \exists j(j < L(w) \wedge x = w[j]) \Leftrightarrow x = v \vee x \varepsilon w. \end{aligned}$$

The step marked by $(*_1)$ is by case analysis on whether or not $i = 0$.

(3): By structural induction on the list y. The base case is trivial and the induction step follows from

$$\begin{aligned} x \varepsilon \langle v, w \rangle \oplus z \Leftrightarrow x \varepsilon \langle v, w \oplus z \rangle \overset{(2)}{\Leftrightarrow} x = v \vee x \varepsilon w \oplus z \overset{\mathrm{IH}}{\Leftrightarrow} \\ \Leftrightarrow x = v \vee x \varepsilon w \vee x \varepsilon z \overset{(2)}{\Leftrightarrow} x \varepsilon \langle v, w \rangle \vee x \varepsilon z. \end{aligned}$$

(4): By structural induction on the list y. The base case follows from (1) and 7.1.11(1). In the induction step we have

$$x \varepsilon \langle v, w\rangle \overset{(2)}{\Leftrightarrow} x = v \vee x \varepsilon w \overset{\text{IH}}{\Leftrightarrow} x = v \vee \exists z_1 \exists z_2\, w = z_1 \oplus \langle x, z_2\rangle \Leftrightarrow$$
$$\Leftrightarrow \langle v, w\rangle = 0 \oplus \langle x, w\rangle \vee \exists z_1 \exists z_2\, \langle v, w\rangle = \langle v, z_1\rangle \oplus \langle x, z_2\rangle \overset{(*_2)}{\Leftrightarrow}$$
$$\Leftrightarrow \exists z_1 \exists z_2\, \langle v, w\rangle = z_1 \oplus \langle x, z_2\rangle.$$

The step $(*_2)$ is by case analysis on whether or not the list z_1 is empty. □

7.1.14 List reversal. We wish to introduce into PA the function $Rev(x)$ which reverses the elements of the list x:

$$Rev\,\langle x_1, x_2, \ldots, x_n, 0\rangle = \langle x_n, \ldots, x_2, x_1, 0\rangle.$$

The list reversal is defined by structural list recursion as a p.r. function:

$Rev(0) = 0$
$Rev\,\langle v, w\rangle = Rev(w) \oplus \langle v, 0\rangle$.

7.1.15 Basic properties of list reversal. We have

$$\vdash_{\text{PA}} Rev(x) = 0 \leftrightarrow x = 0 \tag{1}$$
$$\vdash_{\text{PA}} Rev(x \oplus y) = Rev(y) \oplus Rev(x) \tag{2}$$
$$\vdash_{\text{PA}} Rev\,Rev(x) = x \tag{3}$$
$$\vdash_{\text{PA}} Rev(x) = Rev(y) \to x = y \tag{4}$$
$$\vdash_{\text{PA}} \exists y\, x = Rev(y) \tag{5}$$
$$\vdash_{\text{PA}} L\,Rev(x) = L(x) \tag{6}$$
$$\vdash_{\text{PA}} y \varepsilon Rev(x) \leftrightarrow y \varepsilon x. \tag{7}$$

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

Proof. (1): By case analysis on whether or not the list x is empty.

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

$$Rev(\langle v, w\rangle \oplus y) = Rev\,\langle v, w \oplus y\rangle = Rev(w \oplus y) \oplus \langle v, 0\rangle \overset{\text{IH}}{=}$$
$$= Rev(y) \oplus Rev(w) \oplus \langle v, 0\rangle = Rev(y) \oplus Rev\,\langle v, w\rangle.$$

(3): By structural list induction. The base case is obvious and the induction step follows from

$$Rev\,Rev(\langle v, w\rangle \oplus y) = Rev\,Rev\,\langle v, w \oplus y\rangle = Rev\big(Rev(w \oplus y) \oplus \langle v, 0\rangle\big) \overset{(2)}{=}$$
$$= Rev\,\langle v, 0\rangle \oplus Rev\,Rev(w \oplus y) \overset{\text{IH}}{=} \langle v, 0\rangle \oplus w \oplus y = \langle v, w\rangle \oplus y.$$

(4): This follows from

$$Rev(x) = Rev(y) \Rightarrow Rev\, Rev(x) = Rev\, Rev(y) \overset{(3)}{\Rightarrow} x = y.$$

(5): This follows from (3) by setting $y := Rev(x)$.
(6),(7): By a straightforward structural induction on the list x. □

7.1.16 Fast reversal. The application $Rev(x)$ repeatedly invokes list concatenation to append an element to the end of a list. Consequently, it takes $\mathcal{O}(L(x)^2)$ operations to compute $Rev(x)$. This is clearly wasteful and we can ask the question whether $Rev(x)$ cannot be computed in $\mathcal{O}(L(x))$ steps. By accumulating the reversed list into an *accumulator* a we can perform the reversal of x in $\mathcal{O}(L(x))$ operations with the help of the binary accumulator function $f(x, a)$ defined by

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

The reader will note that this is a structural recursion on the list x with substitution in the parameter a.

The auxiliary function f satisfies the property

$$\vdash_{\mathrm{PA}} \ \forall a\, f(x, a) = Rev(x) \oplus a, \tag{1}$$

from which, by instantiating $a := 0$, we get the relation between Rev and its accumulator version:

$$\vdash_{\mathrm{PA}} \ Rev(x) = f(x, 0).$$

Now we can take the last identity as a program computing $Rev(x)$ with a number of reduction steps proportional to the length of x.

It remains to show that (1) holds. The proof is by structural induction on the list x. The base case is trivial. In the induction step take any a and we obtain

$$\begin{aligned} f(\langle v, w\rangle, a) = f(w, \langle v, a\rangle) &\overset{\mathrm{IH}}{=} Rev(w) \oplus \langle v, a\rangle = \\ &= Rev(w) \oplus \langle v, 0\rangle \oplus a = Rev\, \langle v, w\rangle \oplus a. \end{aligned}$$