8.1 Binary Trees

8.1.1 Introduction. In this section we will show how to arithmetize *binary trees labelled by natural numbers* (see Fig. 8.1). In most functional programming languages the type Bt of binary trees can be defined by a *union type*:

$$Bt = E \mid Nd(\mathrm{N}, Bt, Bt).$$

A value of type Bt is therefore either the *empty* tree E or a non-empty tree of the form $Nd(x, l, r)$, where x is the *label* of its root node and l, r are values of type Bt called the *left* and *right* subtrees of that non-empty tree. The constant E and the function Nd are called *constructors*.

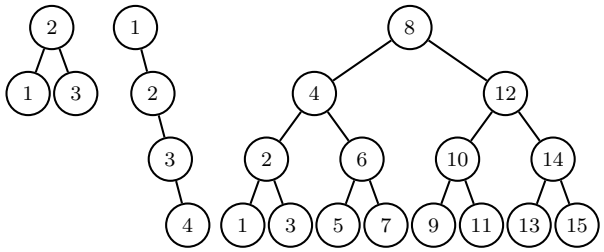

Fig. 8.1 Examples of binary trees

8.1.2 Constructors of binary trees. Arithmetization of binary trees is done with the help of the following two pair constructors with pairwise different tags:

$$\langle\rangle = \langle 0, 0\rangle \qquad \text{(empty tree)}$$
$$\langle l \mid x \mid r\rangle = \langle 1, x, l, r\rangle. \qquad \text{(node)}$$

From the properties of the pairing function we obtain

$$\vdash_{\mathrm{PA}} \langle\rangle \neq \langle l \mid x \mid r\rangle$$
$$\vdash_{\mathrm{PA}} \langle l_1 \mid x_1 \mid r_1\rangle = \langle l_2 \mid x_2 \mid r_2\rangle \to x_1 = x_2 \wedge l_1 = l_2 \wedge r_1 = r_2$$

The first property says that the constructors are pairwise disjoint and the second that the functional constructor $\langle l \mid x \mid r\rangle$ is an injective mapping.

We obtain the pattern matching style of definitions of functions operating over the codes of binary trees with the conditionals of the form

case
$t = \langle\rangle \Rightarrow \beta_1$
$t = \langle l \mid^x \mid r\rangle \Rightarrow_{x,l,r} \beta_2[x,l,r]$
otherwise $\Rightarrow \beta_3$.
end

This is called *discrimination on the constructors of binary trees.*

The above conditional is evaluated as follows. Note that the expression

$$\mathit{Tuple}_*(2,t) \wedge_* [t]_1^2 =_* 0$$

is the characteristic term of its first variant since

$$\vdash_{\mathrm{PA}} \ t = \langle\rangle \leftrightarrow \mathit{Tuple}(2,t) \wedge [t]_1^2 = 0.$$

Similarly, the expression

$$\mathit{Tuple}_*(4,t) \wedge_* [t]_1^4 =_* 1$$

is the characteristic term of its second variant as we have

$$\vdash_{\mathrm{PA}} \ \exists x \exists l \exists r t = \langle l \mid^x \mid r\rangle \leftrightarrow \mathit{Tuple}(4,t) \wedge [t]_1^4 = 1.$$

Finally note that we have

$$\vdash_{\mathrm{PA}} \ t = \langle l \mid^x \mid r\rangle \to x = [t]_2^4 \wedge l = [t]_3^4 \wedge r = [t]_4^4$$

and therefore, the terms $[t]_2^4$, $[t]_3^4$ and $[t]_4^4$ are the witnessing terms for the output variables x, l, r of the second variant of the conditional.

8.1.3 Arithmetization of binary trees. We wish to assign to every binary tree T a unique number $\ulcorner T \urcorner$, called *the code of* T. The mapping is defined inductively on the structure of binary trees:

- If T is the empty tree then $\ulcorner T \urcorner$ is the number $\langle\rangle$.
- If T is a non-empty tree with the root label x, the left son T_1, and the right son T_2, then $\ulcorner T \urcorner$ is the number $\langle \ulcorner T_1 \urcorner \mid^x \mid \ulcorner T_2 \urcorner \rangle$.

For instance, the code of the first binary tree from Fig. 8.1 is the number

$$\big\langle \langle\langle\rangle \mid^1 \mid \langle\rangle\rangle \mid^2 \mid \langle\langle\rangle \mid^3 \mid \langle\rangle\rangle\big\rangle = 26\,646\,277\,093\,331\,868\,372\,637.$$

Clearly, the mapping $\ulcorner T \urcorner$ is injective.

We will use the discrimination on the constructors of binary trees in the definition of the predicate $\mathit{Bt}(t)$ holding of the codes of binary trees. The predicate is defined by course of values recursion as primitive recursive by

$\mathit{Bt}\,\langle\rangle$
$\mathit{Bt}\,\langle l \mid^x \mid r\rangle \leftarrow \mathit{Bt}(l) \wedge \mathit{Bt}(r)$.

Note that the following are the omitted default clauses of the definition:

$$\neg Bt\,\langle l \mid {}^{x} \mid r\rangle \leftarrow \neg Bt(l) \vee \neg Bt(r)$$
$$\neg Bt(t) \leftarrow t \neq \langle\rangle \wedge \neg\exists x \exists l \exists r\, t = \langle l \mid {}^{x} \mid r\rangle.$$

Consequently, the clausal definition is equivalent to

$$\vdash_{\mathrm{PA}} \; Bt(t) \leftrightarrow t = \langle\rangle \vee \exists x \exists l \exists r \big(t = \langle l \mid {}^{x} \mid r\rangle \wedge Bt(l) \wedge Bt(r)\big).$$

The embedding of binary trees into natural numbers is so straightforward that we will henceforth identify binary trees with their codes, i.e. with the subset Bt of natural numbers. In our case we will say *the binary tree* t instead of *the code* t *of a binary tree.*

8.1.4 Case analysis on binary trees. Directly from the definition of the predicate Bt we obtain the following property

$$\vdash_{\mathrm{PA}} \; Bt(t) \rightarrow t = \langle\rangle \vee \exists x \exists l \exists r\, t = \langle l \mid {}^{x} \mid r\rangle.$$

This is called the principle of *structural case analysis on the constructors of the binary tree* t.

We can use the above principle of structural case analysis in order to establish the admissibility of a certain kind of conditional discriminations on the constructors of binary trees. These are of the form

$$\begin{aligned} Bt(t) \rightarrow \; & \mathbf{case} \\ & t = \langle\rangle \Rightarrow \beta_1 \\ & t = \langle l \mid {}^{x} \mid r\rangle \Rightarrow_{x,l,r} \beta_2[x,l,r] \\ & \mathbf{end} \end{aligned}$$

Because of the precondition $Bt(t)$ we have to evaluate only two alternatives instead of three. Moreover, as we have

$$\vdash_{\mathrm{PA}} \; Bt(t) \rightarrow t = \langle\rangle \leftrightarrow [t]_1^2 = 0$$
$$\vdash_{\mathrm{PA}} \; Bt(t) \rightarrow \exists x \exists l \exists r t = \langle l \mid {}^{x} \mid r\rangle \leftrightarrow [t]_1^4 = 1,$$

we can use $[t]_1^2 =_* 0$ and $[t]_1^4 =_* 1$ as the characteristic terms of its two variants which are much simpler expressions than those from Par. 8.1.2.

8.1.5 Structural induction for binary trees. The principle of structural induction for binary trees can be informally stated as follows. To prove by tree induction that a property holds for every binary tree it suffices to prove:

Base case: the property holds for the empty tree $\langle\rangle$.
Induction step: if the property holds for the subtrees l, r then it holds also for the whole tree $\langle l \mid {}^{x} \mid r\rangle$.

This is expressed formally in PA by

$$\vdash_{\mathrm{PA}} \varphi[\langle\rangle] \wedge \forall x \forall l \forall r(\varphi[l] \wedge \varphi[r] \to \varphi[\langle l \mid x \mid r\rangle]) \to Bt(t) \to \varphi[t],$$

where $\varphi[t]$ is a formula of PA. The property is called the principle of *structural induction on the binary tree t for $\varphi[t]$*.

Proof. The principle of structural induction for binary trees is derived in PA as follows. Assume $\varphi[\langle\rangle]$ and $\forall x \forall l \forall r(\varphi[l] \wedge \varphi[r] \to \varphi[\langle l \mid x \mid r\rangle])$ take any binary tree t and prove that $\varphi[t]$ holds by complete induction on t. We consider two cases. If t is the empty tree then the claim follows directly from the first assumption. Otherwise, t is a non-empty tree of a form $\langle l \mid x \mid r\rangle$ for some x, l, r. By the properties of the pairing function we have $l < \langle l \mid x \mid r\rangle$ and $r < \langle l \mid x \mid r\rangle$. By applying two IH's we get $\varphi[l]$ and $\varphi[r]$, and thus $\varphi[\langle l \mid x \mid r\rangle]$ by the second assumption.

8.1.6 Structural recursion on binary trees. Structural induction over binary trees is used to prove properties of functions defined by the scheme of *structural recursion on binary trees*. In its simplest form, the operator of structural recursion introduces a function f from two functions g and h satisfying

$$\begin{aligned} f(t,y) = \mathbf{case}\ & \\ & t = \langle\rangle \Rightarrow g(y) \\ & t = \langle l \mid x \mid r\rangle \Rightarrow h\big(x, l, r, f(l,y), f(r,y), y\big) \\ & \mathbf{otherwise} \Rightarrow 0 \\ \mathbf{end}. & \end{aligned}$$

Note that this is a recursive definition regular in the first argument with discrimination on the constructors of binary tree (output variables of the second variant are omitted).

The following identities form the clausal form of the above definition

$$\begin{aligned} & f(\langle\rangle, y) = g(y) \\ & f(\langle l \mid x \mid r\rangle, y) = h\big(x, l, r, f(l,y), f(r,y), y\big). \end{aligned}$$

Note here that this is a typical example where we wish to use the default clauses – in this case

$$f(t,y) = 0 \leftarrow t \neq \langle\rangle \wedge \neg\exists x \exists l \exists r\, t = \langle l \mid x \mid r\rangle,$$

in order not to clutter the definition. We do not care what value is yielded by the application $f(t,y)$ if t is not the code of a binary tree.

The above definition for the function f can be easily rewritten to a conditional program for the same function as we have

$$\begin{aligned} \vdash_{\mathrm{PA}}\ Bt(t) \to f(t,y) = \mathbf{case}\ & \\ & t = \langle\rangle \Rightarrow g(y) \\ & t = \langle l \mid x \mid r\rangle \Rightarrow h\big(x, l, r, f(l,y), f(r,y), y\big) \\ & \mathbf{end} \end{aligned}$$

Its conditions of regularity

$$\vdash_{\mathrm{PA}} \ Bt(t) \wedge t = \langle l \mid x \mid r \rangle \to l < t \wedge Bt(l)$$
$$\vdash_{\mathrm{PA}} \ Bt(t) \wedge t = \langle l \mid x \mid r \rangle \to r < t \wedge Bt(r)$$

are trivially satisfied.

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/programs by structural recursion on binary trees.

8.1.7 Depth and size of binary trees. The *depth* function $d(t)$ yields the length of the longest path from the root to a leaf in the binary tree t. The function is defined by *parameterless* structural recursion on the binary tree t as a primitive recursive function:

$$d\langle\rangle = 0$$
$$d\langle l \mid x \mid r \rangle = \max\big(d(l), d(r)\big) + 1.$$

The *size* function $|t|$ counts the number of labels in the binary tree t. The function is defined by parameterless structural recursion on the binary tree t as a primitive recursive function:

$$|\langle\rangle| = 0$$
$$|\langle l \mid x \mid r \rangle| = |l| + |r| + 1.$$

The next property relates the number of labels in a binary tree to its depth:

$$\vdash_{\mathrm{PA}} \ Bt(t) \to d(t) \le |t| < 2^{d(t)}. \tag{1}$$

There are trees for which the second inequality is tight, i.e. $|t| + 1 = 2^{d(t)}$. Such trees are called *full binary trees*.

Proof. The property is proved by structural induction on the binary tree t. The base case is obvious. The induction step when $t = \langle l \mid x \mid r \rangle$ follows from

$$\begin{aligned} d\langle l \mid x \mid r \rangle &= \max(d(l), d(r)) + 1 \overset{\mathrm{IH}}{\le} \max(|l|, |r|) + 1 \le \\ &\le |l| + |r| + 1 = |\langle l \mid x \mid r \rangle| \end{aligned}$$

and

$$\begin{aligned} |\langle l \mid x \mid r \rangle| &= |l| + |r| + 1 \overset{\mathrm{IH}}{<} 2^{d(l)} + 2^{d(r)} \le 2 \times 2^{\max(d(l), d(r))} = \\ &= 2^{\max(d(l), d(r)) + 1} = 2^{d\langle l \mid x \mid r \rangle}. \end{aligned}$$

□

8.1.8 Membership in binary trees. The predicate $x \in t$ holds if x is a label of the binary tree t. The predicate is defined by structural recursion on the binary tree t as a primitive recursive predicate:

$x \in \langle l \mid y \mid r \rangle \leftarrow x = y$
$x \in \langle l \mid y \mid r \rangle \leftarrow x \neq y \wedge x \in l$
$x \in \langle l \mid y \mid r \rangle \leftarrow x \neq y \wedge x \notin l \wedge x \in r.$

The following are the basic properties of the tree membership predicate:

$$\vdash_{\mathrm{PA}} x \notin \langle\rangle$$
$$\vdash_{\mathrm{PA}} x \in \langle l \mid y \mid r \rangle \leftrightarrow x = y \vee x \in l \vee x \in r.$$

From the properties of the pairing function we get $\vdash_{\mathrm{PA}} x \in t \rightarrow x < t$. Consequently, the universal quantifier $\forall x$ in a context like $\forall x(x \in \ldots \rightarrow \cdots)$ can be bounded. Similarly for existential quantifiers.

8.1.9 Subtree relation. Given the binary trees t_1 and t_2, the predicate $t_1 \trianglelefteq t_2$ holds if the tree t_1 is a subtree of the tree t_2. The predicate is defined by structural recursion on the binary tree t_2 as primitive recursive by

$t_1 \trianglelefteq t_2 \leftarrow t_1 = t_2$
$t_1 \trianglelefteq t_2 \leftarrow t_1 \neq t_2 \wedge t_2 = \langle l_2 \mid x_2 \mid r_2 \rangle \wedge t_1 \trianglelefteq l_2$
$t_1 \trianglelefteq t_2 \leftarrow t_1 \neq t_2 \wedge t_2 = \langle l_2 \mid x_2 \mid r_2 \rangle \wedge t_1 \not\trianglelefteq l_2 \wedge t_1 \trianglelefteq r_2.$

The following is the basic property of the subtree predicate:

$$\vdash_{\mathrm{PA}} t_1 \trianglelefteq t_2 \leftrightarrow t_1 = t_2 \vee \exists x_2 \exists l_2 \exists r_2 \big(t_2 = \langle l_2 \mid x_2 \mid r_2 \rangle \wedge (t_1 \trianglelefteq l_2 \vee t_1 \trianglelefteq r_2) \big). \quad (1)$$

As a straightforward consequence we obtain that

$$\begin{aligned} \vdash_{\mathrm{PA}} \langle l_1 \mid x_1 \mid r_1 \rangle \trianglelefteq \langle l_2 \mid x_2 \mid r_2 \rangle \leftrightarrow x_1 = x_2 \wedge l_1 = l_2 \wedge r_2 = r_2 \vee \\ \vee \langle l_1 \mid x_1 \mid r_1 \rangle \trianglelefteq l_2 \vee \langle l_1 \mid x_1 \mid r_1 \rangle \trianglelefteq r_2. \end{aligned} \quad (2)$$

Finally note that $\vdash_{\mathrm{PA}} t_1 \trianglelefteq t_2 \rightarrow t_1 \leq t_2$ by the properties of the pairing function. Therefore universal quantifiers in contexts like $\forall t_1(t_1 \trianglelefteq \ldots \rightarrow \cdots)$ can be bounded. Similarly for existential quantifiers.

8.1.10 Subsorts of binary trees. In the following sections we will study various kinds of binary trees where the sorted predicate $R(t)$ for each particular variety of binary trees has the following explicit definition

$$P(t) \leftrightarrow Bt(t) \wedge \forall x \forall l \forall r(\langle l \mid x \mid r \rangle \trianglelefteq t \rightarrow \varphi[x, l, r]) \quad (1)$$

for a suitable $\varphi[x, l, r]$. For instance:

- perfectly size-balanced trees are defined by (1) with $\varphi \equiv |l| = |r|$;
- perfectly depth-balanced trees are defined by (1) with $\varphi \equiv d(l) = d(r)$.

The predicate P defined by (1) has the following basic properties:

$$\vdash_{\mathrm{PA}} P\langle\rangle \quad (2)$$
$$\vdash_{\mathrm{PA}} P\langle l \mid x \mid r \rangle \leftrightarrow \varphi[x, l, r] \wedge P(l) \wedge P(r). \quad (3)$$

Proof. (2): This is obvious. (3): It follows from

$$
\begin{aligned}
&P\,\langle l \mid x \mid r\rangle \Leftrightarrow \\
&Bt\,\langle l \mid x \mid r\rangle \wedge \forall x_1 \forall l_1 \forall r_1 \big(\langle l_1 \mid x_1 \mid r_1\rangle \trianglelefteq \langle l \mid x \mid r\rangle \to \varphi[x_1, l_1, r_1]\big) \overset{8.1.9(2)}{\Leftrightarrow} \\
&Bt(l) \wedge Bt(r) \wedge \varphi[x, l, r] \wedge \forall x_1 \forall l_1 \forall r_1 \big(\langle l_1 \mid x_1 \mid r_1\rangle \trianglelefteq l \to \varphi[x_1, l_1, r_1]\big) \wedge \\
&\qquad\qquad \wedge \forall x_1 \forall l_1 \forall r_1 \big(\langle l_1 \mid x_1 \mid r_1\rangle \trianglelefteq r \to \varphi[x_1, l_1, r_1]\big) \Leftrightarrow \\
&\varphi[x, l, r] \wedge P(l) \wedge P(r).
\end{aligned}
$$

□

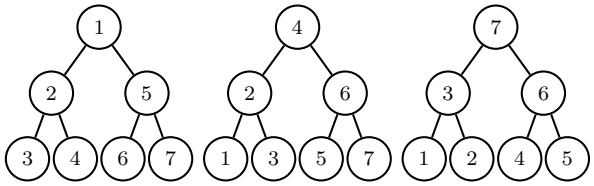

Fig. 8.2 Three basic traversals of binary trees – preorder, inorder and postorder

8.1.11 Depth-first traversal of binary trees. Consider the function $Preorder(t)$ collecting the labels of a binary tree into a list in the order which corresponds to depth-first traversal of binary trees (see Fig. 8.2). The function is defined by structural recursion as primitive recursive by

$$
\begin{aligned}
&Preorder\,\langle\rangle = 0 \\
&Preorder\,\langle l \mid x \mid r\rangle = \big\langle x, 0\big\rangle \oplus Preorder(l) \oplus Preorder(r).
\end{aligned}
$$

Note that the program runs in time $\mathcal{O}(|t|^2)$ due to repeated concatenation.

We obtain more efficient algorithm by keeping the labels of the visiting tree in an accumulator. For that we need a binary *accumulator* function $f(t, a)$ defined by *nested* structural recursion on the binary tree t:

$$
\begin{aligned}
&f(\langle\rangle, a) = a \\
&f(\langle l \mid x \mid r\rangle, a) = \big\langle x, f\big(l, f(r, a)\big)\big\rangle
\end{aligned}
$$

Then we can take the following property

$$\vdash_{\mathrm{PA}} \; Bt(t) \to Preorder(t) = f(t, 0).$$

as an alternative (conditional) program of the preorder traversal function with time complexity $\mathcal{O}(|t|)$.

The property is a straightforward consequence of a more general property of the accumulator function:

$$\vdash_{\mathrm{PA}} \; Bt(t) \to \forall a\, f(t, a) = Preorder(t) \oplus a,$$

which is proved structural induction on the binary tree t. The base case is trivial. In the induction step when $t = \langle l \mid x \mid r \rangle$ take any a and we get

$$\begin{aligned} f(\langle l \mid x \mid r \rangle, a) &= \big\langle x, f\big(l, f(r, a)\big)\big\rangle \overset{\text{IH}}{=} \big\langle x, f(l, \mathit{Preorder}(r) \oplus a)\big\rangle \overset{\text{IH}}{=} \\ &= \big\langle x, \mathit{Preorder}(l) \oplus \mathit{Preorder}(r) \oplus a\big\rangle = \mathit{Preorder}\,\langle l \mid x \mid r \rangle \oplus a. \end{aligned}$$

Note that the second application of IH is with $\mathit{Preorder}(r) \oplus a$ in place of a.