2.3. Visiting Trees While Looking for Good Nodes

In this section we will present some algorithms for visiting trees while looking for nodes which satisfy a given predicate, say p. We assume that the trees are generated in a dynamic fashion, that is, every tree is given by its root node and a function, say f, which for each node returns the list of its son-nodes. Any such function f is also called a *tree-generating* function.

If we assume that every node of a tree is of type α we have that:
(i) the predicate p is a function from α to *bool*, where $bool = \{true, false\}$, and
(ii) the tree-generating function f is a function from α to $\alpha\ list$, where $\alpha\ list$ denotes the type of the lists whose elements are of type α.

In what follows the infix append function between two lists is also denoted by $<>$. Thus, $<>: (\alpha\ list) \times (\alpha\ list) \rightarrow (\alpha\ list)$.

2.3.1. Depth-first Visit. Version 1. In this section we present a function, called existsev (short for *exists eventually*), which given: (i) a predicate p, (ii) a tree-generating function f, and (iii) a list L of nodes, returns *true* if there exists a node n in L which is the root of a tree t_n generated by the function f, such that in t_n there exists a node m such that $p(m) = true$, otherwise it returns *false*.

The correctness of existsev is based on the assumption that for any node n the tree which is rooted in n is finite. This finiteness assumption is required because given any node n, the tree rooted in n is generated and visited in a *depth-first* manner (see the line marked with (†)).

Thus, if a node is the root of an infinite tree (because of the particular tree-generating function which is given), then the evaluation of existsev may not terminate even if in that infinite tree there exists a node which satisfies p.

The existsev function is as follows. In the first line we give first the type of the function and in the following lines we give its definition.

```
existsev : (α → bool) × (α → α list) × (α list) → bool
existsev p f L = if L=[] then false
                 else if p(hd(L)) then true
                 else existsev p f (f(hd(L)) <> tl(L))     (†)
```

This function existsev can be used for visiting trees as follows. Suppose we are given a predicate p, a tree-generating function f, and a node n. Suppose also that f generates a finite tree rooted in n. Then, there exists a node m in the tree rooted in n generated by f such that $p(m)$ is *true* iff $\text{existsev}(p, f, [n]) = true$.

The inductive proof of this statement is left to the reader.

2.3.2. Depth-first Visit. Version 2. In this section we present a different algorithm for visiting trees in a depth-first manner. It has been suggested to us by Prof. R. M. Burstall.

As in the previous Section 2.3.1, we are given a predicate p, a tree-generating function f, and a node n. The following function $\text{existsev1}(p, f, n)$ returns *true* if in the tree rooted in n generated from n by the function f there exists a node m such that $p(m) = true$, otherwise it returns *false*.

As for the function existsev of the previous section, the correctness of the existsev1 function is based on the assumption that for any node n, the tree rooted in n, is finite. This assumption is necessary because the tree rooted in n is generated and visited in a *depth-first* manner (see the line marked with (††)).

Given a predicate p and a list L of nodes, the function exists(p, L) returns *true* if in the list L there exists a node m such that $p(m)$ is *true*, otherwise it returns *false*.

$$
\begin{array}{l}
\text{exists} : (\alpha \rightarrow \mathit{bool}) \times (\alpha\ \mathit{list}) \ \rightarrow\ \mathit{bool} \\
\text{exists}\ p\ L = \text{if } L\!=\![\,]\text{ then } \mathit{false} \\
\qquad\qquad\qquad \text{else if } p(hd(L)) \text{ then } \mathit{true} \\
\qquad\qquad\qquad \text{else exists } p\ \ tl(L) \\
\\
\text{existsev1} : (\alpha \rightarrow \mathit{bool}) \times (\alpha \rightarrow \alpha\ \mathit{list}) \times \alpha \ \rightarrow\ \mathit{bool} \\
\text{existsev1}\ p\ f\ x = \text{if } p(x) \text{ then } \mathit{true} \\
\qquad\qquad\qquad\quad \text{else exists (existsev1 } p\ f)\ \ f(x) \qquad (\dagger\dagger)
\end{array}
$$

Note that in the above function we use the partial application technique: the predicate which is the first argument of exists in the line marked with (††), is the result of the application of the predicate p and the function f to the function existsev1 which is of arity 3. Thus, at line (††) the type of existsev $p\,f$ is $\alpha \rightarrow \mathit{bool}$, which is the type of a predicate on nodes, and the type of $f(x)$ is that of a list of α's, that is, α *list*.

NOTE 2.3.1. The functions existsev and existsev1 are equivalent in the sense that for any given predicate p from α to *bool*, any tree generating function f from α to an α list, and any given initial node n of type α, we have that:
existsev $p\ f\ [n]$ = existsev1 $p\ f\ n$. □

As we will see in more detail in Chapter 3, one can use this depth-first visit algorithm for parsing. Here is an example where we consider the regular grammar G whose productions are:

$$
\begin{array}{lcl}
P & \rightarrow & b \\
P & \rightarrow & bQ \\
Q & \rightarrow & a \\
Q & \rightarrow & aQ
\end{array}
$$

and whose axiom is P. We want to check whether or not the string ba belongs to the language of G. We can do so by constructing a tree whose root node is the pair (P, ba) of the axiom P and the string to parse ba. The tree-generating function f generates the list of the son-nodes of any given node (*sentential-form*, *string-to-parse*) by applying the productions of the first symbol of the *sentential-form*. For instance, given the node (P, ba), the list of the son-nodes is $[(b, ba),\ (bQ, ba)]$. The predicate p is the one which is *true* for a node of the form (ε, ε). The following Figure 2.3.1 shows a tree of nodes with root (P, ba). It indicates that the string ba belongs to the language of G because there is a node (ε, ε).

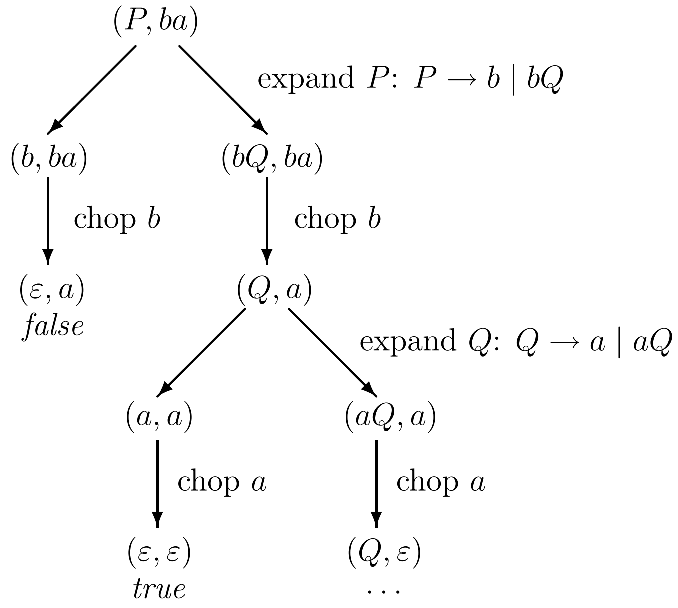

FIGURE 2.3.1. The tree of nodes which shows that the string ba is generated by the grammar G whose productions are: $P \to b \mid bQ$, $Q \to a \mid aQ$.

2.3.3. Breadth-first Visit. In this section we present a third algorithm for visiting trees. The visit is performed in a *breadth-first* manner so that we do not need for this algorithm the hypothesis that the tree generated from any given node by the given tree-generating function, is finite.

Given a function f from an element of type α to a list of α's and a list L of elements of type α, the function flatmap(f, L) returns the concatenation of the lists produced by applying the function f to every element of L.

The function exists is the one we have presented in the previous Section 2.3.2.

The function bf-existsev (short for *breadth-first exists eventually*) is very similar to the function existsev of Section 2.3.1. They have the same type. However, in the case of the function bf-existsev, the tree is generated and visited in a breadth-first manner (see line marked with (†††)). Indeed, at each recursive call of the function bf-existsev, we construct the list of son-nodes of all nodes in the list of the previous call of bf-existsev. Thus, the tree is generated *level-by-level*, by considering the nodes at distance $k+1$ from the root, only after all nodes at distance k, for any $k \geq 0$.

The functions flatmap, exists, and bf-existsev are defined as follows.

flatmap : $(\alpha \to \alpha\ list) \times (\alpha\ list) \ \to\ (\alpha\ list)$
flatmap $f\ L =$ if $L = [\,]$ then $[\,]$
else $(f(hd(L)) <> (\text{flatmap}\ f\ \ tl(L))$

exists : $(\alpha \to bool) \times (\alpha\ list) \ \to\ bool$
exists $p\ \ L =$ if $L = [\,]$ then $false$
else if $p(hd(L))$ then $true$
else exists $p\ \ tl(L)$

bf-existsev : $(\alpha \to bool) \times (\alpha \to \alpha\ list) \times (\alpha\ list) \ \to\ bool$
bf-existsev $p\ f\ L =$ if $L = [\,]$ then $false$
else if exists $p\ \ L$ then $true$
else bf-existsev $p\ f$ (flatmap $f\ L$) (†††)

Note that the functions existsev and bf-existsev may not terminate, if starting from a given node, the iterated applications of the function f produce an infinite tree.

EXERCISE 2.3.2. Show that the following function definition of existsev is not correct:
existsev $p\ f\ L =$ if exists $p\ \ L$ then $true$
else existsev $p\ f\ \ (f(hd(L)) <> tl(L))$
Hint: Consider the case when $L = [\,]$.
□

EXERCISE 2.3.3. Show that the following function definition for bf-existsev is not correct:
bf-existsev $p\ f\ L =$ if $L = [\,]$ then $false$
else if $p(hd(L))$ then $true$
else bf-existsev $p\ f$ (flatmap $f\ L$)
Hint: Some nodes are used for generating their son-nodes, but never tested by p.
□

EXERCISE 2.3.4. Show that the following function definition for bf-existsev is not correct:
bf-existsev $p\ f\ L =$ if exists $p\ \ L$ then $true$
else bf-existsev $p\ f$ (flatmap $f\ L$)
Hint: Consider the case when $L = [\,]$.
□