functions

$\underline{empty} \quad = (\emptyset, <p_0>)$

Sei für die restlichen Funktionsdefinitionen $l = (a_1 \ldots a_n, p_0 \ldots p_n)$.

$\underline{front}\ (l) \quad = p_0$

$last\ (l) \quad = \begin{cases} p_n & \text{falls } n > 0 \\ \text{undefiniert} & \text{sonst} \end{cases}$

$\underline{next}\ (l, p) \quad = \begin{cases} p_{i+1} & \text{falls } \exists i \in \{0, \ldots, n-1\}: p = p_i \\ null & \text{sonst} \end{cases}$

$previous\ (l, p) \quad = \begin{cases} p_{i-1} & \text{falls } \exists i \in \{1, \ldots, n\}: p = p_i \\ null & \text{sonst} \end{cases}$

$bol\ (l, p) \quad = (p = p_0)$

$\underline{eol}\ (l, p) \quad = (p = p_n)$

Für *insert* sei $p = p_i \in \{p_0, \ldots, p_n\}$. Sonst ist *insert* undefiniert. Sei $p' \in \text{POS} \setminus \{p_0, \ldots, p_n\}$.

$\underline{insert}\ (l, p, x) \quad = (<a_1, \ldots, a_i, x, a_{i+1}, \ldots, a_n>,$
$<p_0, \ldots, p_i, p', p_{i+1}, \ldots, p_n>)$

Für *delete* sei $p = p_i \in \{p_1, \ldots, p_n\}$. Sonst ist *delete* undefiniert.

$\underline{delete}\ (l, p) \quad = (<a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n>,$
$<p_0, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n>)$

$concat\ ((a_1 \ldots a_n, p_0 \ldots p_n), (b_1 \ldots b_m, q_0 \ldots q_m))$
$= (<a_1, \ldots, a_n, b_1, \ldots, b_m>, <p_0, \ldots, p_n, q_1, \ldots, q_m>)$

$isempty\ (l) \quad = (n = 0)$

$\boxed{find\ (l, f)} \quad = \begin{cases} p_i & \text{falls } \exists\ i: f(a_i) = \text{true} \wedge \\ & \quad \forall\ j \in \{1, \ldots, i-1\}: f(a_j) = \text{false} \\ null & \text{sonst} \end{cases}$

Für *retrieve* sei $p = p_i \in \{p_1, \ldots, p_n\}$. Sonst ist *retrieve* undefiniert.

$retrieve\ (l, p) \quad = a_i$

nd $list_2$.

algebra $list_1$

sorts $list, elem$ {*bool* ist im folgenden implizit immer dabei}

ops

empty	:	$\rightarrow list$
first	: *list*	$\rightarrow elem$
rest	: *list*	$\rightarrow list$
append	: $list \times elem$	$\rightarrow list$
concat	: $list \times list$	$\rightarrow list$
isempty	: *list*	$\rightarrow bool$

sets $list = \{ \langle a_1, ..., a_n \rangle \mid n \geq 0, a_i \in elem \}$

functions

empty	$= \Diamond$	
first $(a_1 \ldots a_n)$	$= \begin{cases} a_1 \\ \text{undefiniert} \end{cases}$	falls $n > 0$ sonst
rest $(a_1 \ldots a_n)$	$= \begin{cases} a_2 \ldots a_n \\ \text{undefiniert} \end{cases}$	falls $n > 0$ sonst
append $(a_1 \ldots a_n, x)$	$= x\, a_1 \ldots a_n$	
concat $(a_1 \ldots a_n, b_1 \ldots b_m)$	$= a_1 \ldots a_n \circ b_1 \ldots b_m$	
isempty $(a_1 \ldots a_n)$	$= (n = 0)$	

end $list_1$.

---

algebra $list_2$

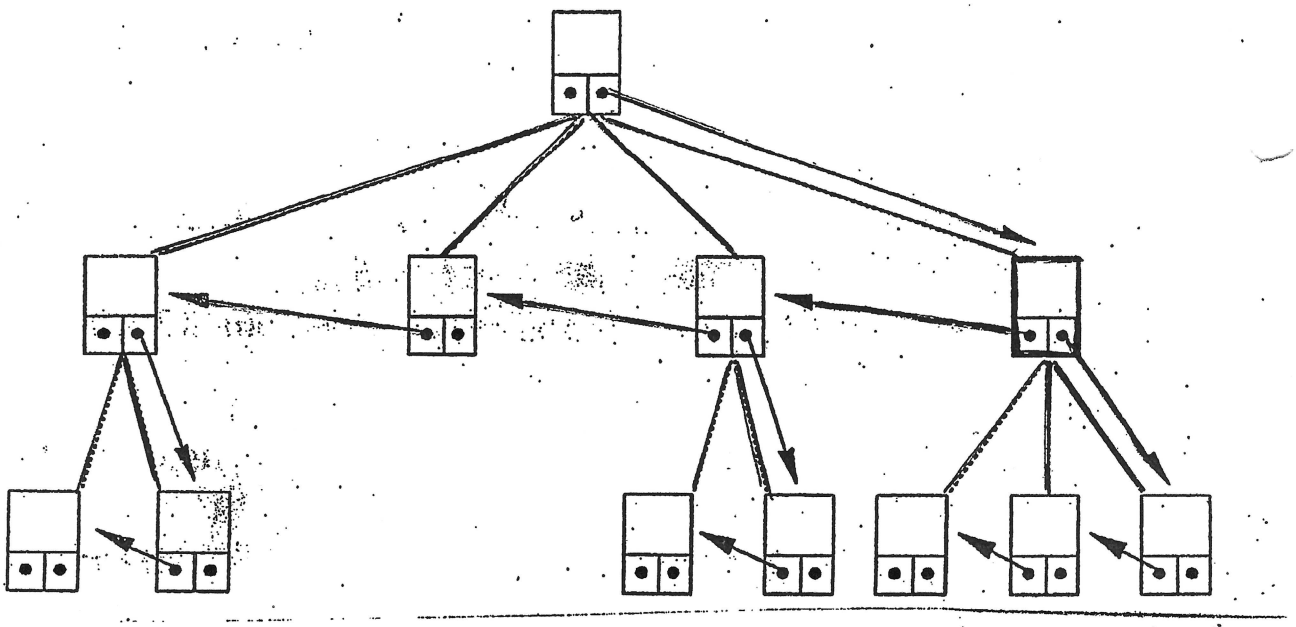
sorts *list, elem, pos*

ops

empty	:	$\rightarrow list$
front, last	: *list*	$\rightarrow pos$
next, previous	: $list \times pos$	$\rightarrow pos \cup \{null\}$
bol, eol	: $list \times pos$	$\rightarrow bool$
insert	: $list \times pos \times elem$	$\rightarrow list$
delete	: $list \times pos$	$\rightarrow list$
concat	: $list \times list$	$\rightarrow list$
isempty	: *list*	$\rightarrow bool$
find	: $list \times (elem \rightarrow bool)$	$\rightarrow pos \cup \{null\}$
retrieve	: $list \times pos$	$\rightarrow elem$

algebra	tree		
sorts	tree, elem, bool		
ops	empty :		$\rightarrow$ tree
	maketree :	tree $\times$ elem $\times$ tree	$\rightarrow$ tree
	key :	tree	$\rightarrow$ elem
	left,right :	tree	$\rightarrow$ tree
	isempty :	tree	$\rightarrow$ bool
sets	bool $= \{$true,false$\}$		
	elem : beliebige Menge		
	tree $= \{\langle\,\rangle\} \cup \{\langle l, x, r\rangle \mid x \in \text{elem},\ l, r \in \text{tree}\}$		
functions	empty()	$= \langle\,\rangle$	
	maketree(l, x, r)	$= \langle l, x, r\rangle$	
	Sei $t = \langle l, x, r\rangle$;	sonst sind key, left, right undefiniert.	
	key(t)	$= x$	
	left(t)	$= l$	
	right(t)	$= r$	
	isempty(t)	$= (t \stackrel{?}{=} \langle\,\rangle)$	

b) Implementierung über Binärbäume



find: list× (elem → bool) → pos∪{nil}

`find`(l, f)
Input: Liste l, berechenbare Funktion f : elem → bool.
Output: Position p mit $f(p\uparrow.\text{value}) = \text{true}$, wenn möglich; ansonsten nil.

```
(1)  p:=front(l)
(2)  while not eol(l,p) do
(3)      p:=next(l,p)
(4)      if f(p↑.value) then return p fi
(5)  od
(6)  return nil
```