Klausur Gdp I Wintersem. 05/06 (Rekonstruktion)

Datum: 19.03.06
Kommentare:
Die Unterstrichenen Wörter sind vermutlich die Schlagwörter auf die es Punkte gab.
In der angehangenen Date **test.sml** befinden sich alle in ML verfassten Lösungen zusammen mit Testeinsetzungen.
Ich erhebe zwar Anspruch aber keine Garantie auf Korrektheit.

Aufgabe 1 (11/11 Punkte)

Beweisen oder widerlegen Sie: die folgenden Funktion f, f ', f '':A* x A* $\rightarrow$ {0.1}, f ''': A* $\rightarrow${0,1} sind berechenbar:

a.)

$$f(P,w) = \begin{cases} 1, \text{ falls P ein Programm, das auf die Eingabe w terminiert} \\ 0, \text{sonst} \end{cases}$$

b.)

$$f'(P,w) = \begin{cases} 1, \text{ falls P ein Programm, das auf die Eingabe w terminiert} \\ \perp, \text{sonst} \end{cases}$$

c.)

$$f''(P,w) = \begin{cases} 1, \text{ falls P ein Programm, das auf die Eingabe w nicht terminiert} \\ \perp, \text{sonst} \end{cases}$$

d.) Q ist ein beliebiges festes Programm

$$f'''(w) = \begin{cases} 1, \text{ falls Q für alle Eingaben terminiert} \\ 0, \text{sonst} \end{cases}$$

Lösung zu Aufgabe 1

a.) Die Funktion f beschreibt genau das Halteproblem.

$$h(\alpha, w) = \begin{cases} Ja, \text{ falls w} \in D(f_\alpha) \\ \text{nein, sonst} \end{cases}$$

Die Funktion f soll „1“ ausgeben, wenn das als Parameter P übergebene Programm für das als Parameter übergebene Wort w terminiert. Andernfalls soll „0“ ausgegeben werden. Der Unterschied zur Definition der Funktion des Halteproblems besteht darin, dass im hier „1“ anstatt „Ja“ und „0“ anstatt „nein“ ausgegeben wird. Da das Halteproblem nicht berechenbar ist, ist damit auch die Funktion f nicht berechenbar.
(2/2 Pkt.)

b.) Das Programm f '(P,w) ist berechenbar. Das Programm gibt „1“ aus, wenn das Programm P für das Wort w terminiert. Andernfalls geht f ' in einen undefinierten Zustand($\perp$), welcher zum Beispiel eine Endlosschleifen sein kann.
Das Programm soll also nicht eindeutig entscheiden ob ein beliebiges Programm P für die Eingabe w terminiert(Halteproblem) sondern ausschließlich dann „1“ ausgeben wenn das der

Fall ist. Andernfalls läuft das Programm zum Beispiel endlos weiter(⊥), gibt also nicht an wenn das Programm für die Eingabe nicht terminiert.
(3/3 Pkt.)

c.) Die Funktion f ''(P,w) soll „1“ ausgeben wenn das Programm P für die Eingabe w nicht terminiert und soll andernfalls in einen undefinierten Zustand(⊥) gehen. Nach dem Halteproblem ist es nicht möglich zu entscheiden ob ein beliebiges Programm P mit beliebigem Wort w terminiert oder nicht. Es kann also nicht „nein“ ausgeben werden wenn das Programm nicht terminiert(Halteproblem). Damit kann also auch nicht „1“ ausgegeben werden falls ein Programm P für die Eingabe w nicht terminiert(f '').
Daraus folgt, dass die Funktion f '' nicht berechenbar ist.
Darüber hinaus ist nicht sinnvoll in einen undefinierten Zustand zu wechseln(⊥) wenn das Programm für die Eingabe hält und dieser Fall wird von dem Sonstzweig umfasst.
(3/3 Pkt.)

d.) Die Funktion f ''' ist berechenbar.
Das Halteproblem besagt nur, dass kein universelles Programm geschrieben werden kann, was prüft ob ein beliebiges Programm(Algorithmus) für ein beliebiges Wort w als Eingabe terminiert oder nicht. Für ein festes Programm Q jedoch kann ein Programm geschrieben werden, welches entscheidet ob das Programm Q terminiert oder nicht. Das ist möglich, weil in dem Programm direkt auf den (bekannten) Algorithmus von Q eingegangen und reagiert werden kann.
(3/3 Pkt.)

Aufgabe 2 (6 Punkte)

a.) Gebe Sie alle Bezeichner an, die in dem folgendem Ausdruck frei vorkommen:

funktion f x:Δ y:Δ' → Δ'' ≡ h(f x) (g y) x

b.) Seine nachf,vorg und mult gegebene Funktionen, die im Bereich der ganzen Zahlen den Nachfolger, Vorgänger und die Multiplikation realisieren. Gegeben sei die folgende Deklaration:

funktion p x:Δ y:Δ' f:Δ'' g:Δ''' h:Δ'''' → Δ''''' ≡ h (f x) (g y)

Welchen Wert liefert p 3 4 vorg nachf mult?
Bestimmen Sie den Typ von p!

Lösung:
a.) h und g
Alle anderen Bezeichner außer h und g sind schon gebunden.

b.) f= vorg
g= nachf
h= mult
x=3
y=4
f 3 =2; g 4=5; h 2 5=10

Es wird der Wert 10 zurückgegeben!

Typ:
$\Delta \rightarrow (\Delta^1 \rightarrow ([\Delta \rightarrow \Delta^6] \rightarrow ([\Delta \rightarrow \Delta^7] \rightarrow ([\Delta^6 \rightarrow (\Delta^7 \rightarrow \Delta^5)] \rightarrow \Delta^5))))$

Aufgabe 3 (? Punkte)
Multiple Choice zum Thema Prozess

Aufgabe 4 (6 Punkte)

Sei A ein endliches Alphabet. Schreiben Sie ein Programm, das an genau drei Stellen aus seinem Eingabebereich A* undefiniert ist, Geben Sie zuvor eine funktionale Spezifikation Ihres Programms an.

Lösung:

Funktionale Spezifikation: Z sei das Z für die Menge der ganzen Zahlen

S_{undef3}:	spec	undef3: $Z \rightarrow Z$ with
		undef3(n)=n where
	pre	$n \in Z_{\geq 4}$ und $n \in Z_{\leq 0}$
	post	$n \in Z_{\geq 4}$ und $n \in Z_{\leq 0}$

Mein Programm undef3:

```
fun undef3 (n:int) =  if(n=3)then n div 0 else
                      if(n=2)then n div 0 else
                      if(n=1)then n div 0 else
                      n;
```

Aufgabe 5 (8 Punkte)

Skizzieren Sie ein Übersetzungsschema von PRO nach ASS für die Komposition von Funktionen.
Nehmen Sie an, Sie hätten bereits für zwei berechenbare Funktionen f,f':$N \rightarrow N$ Funktionsprozeduren in Pro geschrieben und nach ASS übersetzt.
Wie lautet das Ass-Programm für f°f'? Machen Sie ggf. geeignete Annahmen für Ihre beiden Ass-Programme.

Lösung:

c(f) Sei die Ass-Übersetzung von f
c(f') Sei die Ass-Übersetzung von f'

komp(f,g) = fog
f°f'bedeutet f(f')
f bekommt also das Ergebnis der Berechnung von f' als Parameter
Angenommen: f und f' sind Funktionen die jeweils einen Parameter erwarten und eine beliebige Berechnung durchführen und diese als Ergebnis liefern.

Die Komposition würde in pro so aussehen:

x ← f(f'(y)) wobei f und f' 2 pro-Funktionen sind, die einen Parameter (∈IN) erwarten und als Ergebnis „Irgendwas"∈IN ausgeben, dies wird der Variablen x zugewiesen

Das Ass-Programm:
get // Es wird ein Wert vom Benutzer eingelesen und im Register A abgelegt
// Dieser Wert wird in c(f ') als Parameter verwendet
c(f') // In diesem Programmfragment wird die Funktion f' berechnet. Als Parameter nutzt diese Funktion das Register A(Wert wurde vorher eingelesen). Das Ergebnis der Berechnung wird im Register A gespeichert und dient dann wieder als Parameter für das Programmfragment c(f ')

c(f) // In diesem Programmfragment wird die Funktion f berechnet. Der Inhalt des Registers A dient als Parameter für die Funktion f. Das Ergebnis der Berechnung wird wieder im Register A gespeichert und steht damit evntl folgenden Kompositionen als Parameter zur Verfügung

Die Komposition kann auf beliebig viele Funktionen ausgeweitet werden.
… x°y°…°z wobei die Berechnungsreihenfolge mit z beginnt und mit x endet.

Aufgabe 6 (12 Punkte)

Schreiben Sie eine Funktion in FUN oder ML, die

- **a.)** eine Zahlenfolge in Form einer Linkssequenz und eine Zahl n als Parameter besitzt und das n-te Element der Zahlenfolge als Ergebnis liefert, (2 Punkte)
- **b.)** eine Zahlenfolge in Form einer Linkssequenz als Parameter besitzt und das letzte Element der Zahlenfolge liefert, (2 Punkte)
- **c.)** zwei Zahlenfolgen in Form von Linkssequenzen als Parameter besitzt und die Zahl 1 liefert, falls die beiden Folgen gleich sind, und die Zahl 0 sonst. (2 Punkte)

Geben Sie jeweils die Mathematische Funktion an, die das Programm berechnet.
(jeweils 2 Punkte)

Lösung:

```
(* Linkssequenz *)
datatype 'a list= nil | :: of 'a*'a list;

(*beispiel eines Elements vom typ list  Kann aber auch wie folgt dargestellt werden:
(1,(5,(3,nil)))*)
val list1 = 1::5::3::nil;
val list2 = 1::3::3::nil;

(* Die Funktion erstes gibt das erste Element einer Linksseqenz zurück *)
fun erstes(x::_)=x;

(* Die Funktion rest gibt die Elemente einer Linkssequenz ohne das erste Element zurück *)
fun rest(_::x)=x;
```

a.)

(* Die Funktion nelem gibts das n te Element einer Linkssequenz zurück *)
fun nelem t n= if (n=1) then erstes t else nelem (rest t) (n-1);

$$nelem(t,n) = \begin{cases} \text{erstes t}, & \text{falls n} = 1 \\ \text{nelem (rest t, n - 1)}, & \text{sonst} \end{cases}$$

b.)
(* Die Funktion letztes gibt das letzte Element einer Linkssequenz t zurück *)
fun letztes t=if (rest t=nil) then erstes t else letztes (rest t);

$$letztes(t) = \begin{cases} \text{erstes t}, & \text{falls } rest(t) = \text{nil} \\ \text{letztes (rest (t))}, & \text{sonst} \end{cases}$$

c.)
(* die Funktion equal prüft ob zwei polymorphe Elemente gleich sind *)
fun equal (l, t):bool = if (l = t) then true else false;

(* equall überprüft ob zwei Linksequenzen gleich sind *)
fun equall (l:"a list, t:"a list)= if (l=nil andalso t=nil) then true else
if (l= nil andalso t<>nil) then false else
if (l<> nil andalso t=nil) then false else
if (equal (erstes l, erstes t)) then equall (rest l, rest t) else
false;

$$equall(t) = \begin{cases} \text{true}, & \text{falls l} = \text{nil und t} = \text{nil} \\ \text{false}, & \text{falls l} = \text{nil und t != nil} \\ \text{false}, & \text{falls l != nil und t} = \text{nil} \\ \text{equall (rest (l), rest(t))}, & \text{falls erstes (l)} = \text{erstes (t)} \\ \text{false}, & \text{sonst} \end{cases}$$

Aufgabe 7 (12 Punkte)

- **a.)** Definieren Sie einen Datentyp zur Darstellung natürlicher Zahlen durch die Bierdeckelnotation. (2= II, 6 = IIII- I, etc) (3 Punkte)
- **b.)** Wie lautet in Ihrem Datentyp die Repräsentation der Zahlen 0,1,5,12? (1 Punkt)
- **c.)** Gebe Sie zwei Konvertierungsfunktionen in funktionaler Programmierung an, wovon die eine eine natürliche Zahl in die Bierdeckelnotation umwandelt und die andere eine Zahl in Bierdeckelnotation in eine natürliche Zahl umwandelt. (3 Punkte)
- **d.)** Schreiben Sie zwei Funktionen add und sub in funktionaler Programmierung, die (ohne Rückgriff auf Konvertierungen gem. c.) zwei Zahlen in Bierdeckelnotation addieren bzw. subtrahieren. (Die Substraktion sei so definiert: sub x y=x-y, falls x>=y , und 0 sonst) (5 Punkte)

Lösung (in ML):

a.)

```
datatype fuenf = IIIII;
datatype Bnot = leer | I | II | III | IIII | :: of fuenf*Bnot;
```

b.)

0= leer
1=I
5=IIIII
12= IIIII IIIII II

c.)

```
(* Umwandeln einer Zahl in Bierdeckelnotation *)
fun zahlinBiernot (nummer:int):Bnot =
if(nummer=0) then leer                    else
if(nummer=1)  then I                      else
if(nummer=2)  then II                     else
if(nummer=3)  then III                    else
if(nummer=4)  then IIII                   else
IIIII::zahlinBiernot(nummer-5);

(* Eine testvariable vom Typ Bnot *)
val test = IIIII::IIIII::III;

(* Zwei Hilfsfunktionen zum bestimmen vom ersten Element und dem Rest (Bnot ohne erstes
Element) einer Bnot *)
fun erstesb (x::_)=x;
fun restb (_::x)=x;

(* Umwandeln einer Bierdeckelnotation in eine Zahl *)
fun Biernotinzahl (biernotnummer:Bnot):int =

if(biernotnummer=leer)  then 0                    else
if(biernotnummer=I)  then 1                       else
if(biernotnummer=II)  then 2                      else
if(biernotnummer=III)  then 3                     else
if(biernotnummer=IIII)  then 4                    else
5+Biernotinzahl(restb biernotnummer);
```

d.)

```
(* Zwei testvariablen vom Typ Bnot *)
val bn=IIIII::IIIII::I;
val bt=IIIII::IIIII::IIIII::IIIII::leer;

(* Funktion zum Addieren zweier Bierdeckelnotationen *)
fun     Bnotadd ((IIIII::x),y)          = IIIII::Bnotadd(x,y)  |
        Bnotadd (x,(IIIII::y))          = IIIII::Bnotadd(x,y)  |
        Bnotadd (IIII,I)                = IIIII::leer          |
        Bnotadd (IIII,II)               = IIIII::I             |
        Bnotadd (IIII,III)              = IIIII::II            |
        Bnotadd (IIII,IIII)             = IIIII::III           |
        Bnotadd (III,I)                 = IIII                 |
        Bnotadd (III,II)                = IIIII::leer          |
        Bnotadd (III,III)               = IIIII::I             |
        Bnotadd (II,I)                  = III                  |
        Bnotadd (II,II)                 = IIII                 |
        Bnotadd (I,IIII)                = IIIII::leer          |
        Bnotadd (II,IIII)               = IIIII::I             |
        Bnotadd (III,IIII)              = IIIII::II            |
        Bnotadd (I,III)                 = IIII                 |
        Bnotadd (II,III)                = IIIII::leer          |
        Bnotadd (I,II)                  = III                  |
        Bnotadd (I,I)                   = II                   |
        Bnotadd (x,leer)                = x                    |
        Bnotadd (leer,y)                = y                    ;
```

```
(* Funktion zum Subtrahieren zweier Bierdeckelnotationen *)

fun     Bnotsub ((IIIII::x),(IIIII::y)) = Bnotsub(x,y)      |
        Bnotsub (leer,y)                = leer              |
        Bnotsub (x,leer)                = x                 |

        Bnotsub ((IIIII::leer),I)       = IIII              |
        Bnotsub ((IIIII::leer),II)      = III               |
        Bnotsub ((IIIII::leer),III)     = II                |
        Bnotsub ((IIIII::leer),IIII)    = I                 |

        Bnotsub (IIII,IIII)             = leer              |
        Bnotsub (III,III)               = leer              |
        Bnotsub (II,II)                 = leer              |
        Bnotsub (I,I)                   = leer              |

        Bnotsub (IIII,I)                = III               |
        Bnotsub (IIII,II)               = II                |
        Bnotsub (IIII,III)              = I                 |

        Bnotsub (III,I)                 = II                |
        Bnotsub (III,II)                = I                 |

        Bnotsub (II,I)                  = I                 |

        Bnotsub (I,IIII)                = leer              |
        Bnotsub (II,IIII)               = leer              |
        Bnotsub (III,IIII)              = leer              |
        Bnotsub (I,III)                 = leer              |
        Bnotsub (II,III)                = leer              |
        Bnotsub (I,II)                  = leer              |


        Bnotsub ((IIIII::x),y)= IIIII::Bnotsub(x,y)         |
        Bnotsub (x,(IIIII::y))= leer                        ;
```

Aufgabe 8 (8 Punkte)

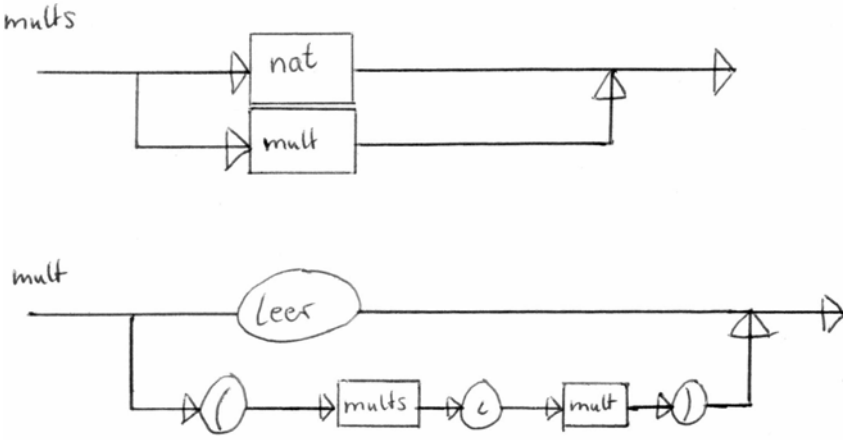
Eine Multilinkssequenz über einem Datentyp D ist eine Linkssequenz, in der jedes Element entweder ein Objekt vom Typ D oder wieder eine Multilinkssequenz über D ist. Entwickeln Sie ein Syntaxdiagramm und eine Grammatik in Backur-Naur-Form für Multilinkssequenzen über dem Datentyp nat. Die Syntaxdiagramme für nat können Sie als gegeben annehmen. Geben Sie für Ihre BNF-Grammatik eine Ableitung für folgende Multilinkssequenz an: ((1,(3,leer)),(2,(leer,leer))).

Lösung:

datatype D =int;
datatype mults = D | mult;
datatype mult = :: of mults*mult | leer;
bzw.:
datatype mult= ::of mult*mult | :: of D*mult | leer;

Kommentar [FMM1]: Dies ist in wird in ML nicht korrekt interpretier. Fehler: Der Konstruktorname wurde doppelt verwendet.

Syntaxdiagramm



BNF-Grammatik

<ZifferohneNull> ::= 1|2|3|4|5|6|7|8|9
<Ziffer>::= 0 | <ZifferohneNull>
<Ziffernfolge>::= eps|<Ziffer>|<Ziffernfolge>
<nat>::=0| <ZifferohneNull><Ziffernfolge>
<mults>::= <nat>|<mult>
<mult>::=leer|(mult,mult)
bzw.
<mult>::=leer|(<nat>,mult) | (mult,mult) (diese wird für Ableitung genutzt)

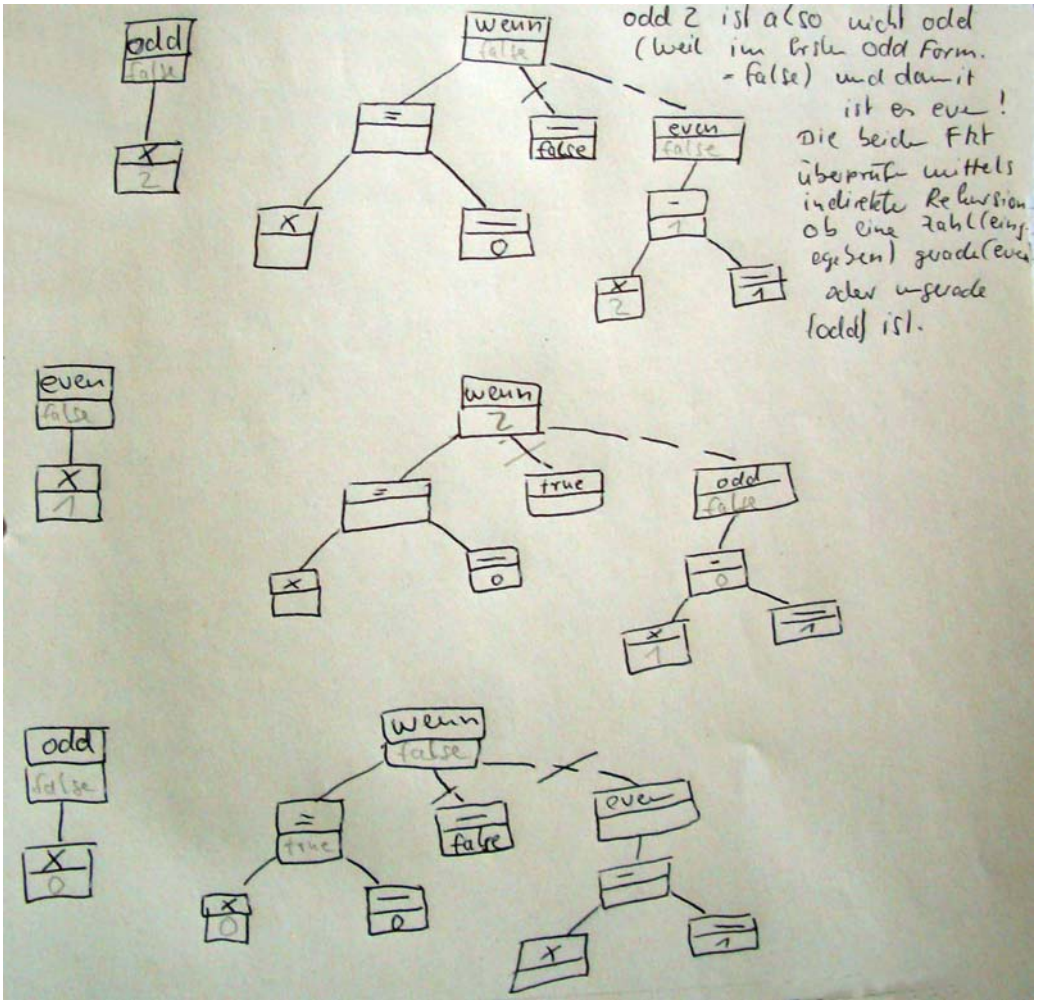
Ableitung für ((1,(3,leer)),(2,(leer,leer,)))

<mult> →(<mult>,<mult>) →((<nat>,<mult>),(<nat>,<mult>))
→((<nat>,(<nat>,<mult>)),(<nat>,(<mult>,<mult>))
→((<ZifferohneNull><Ziffernfolge>,(<ZifferohneNull><Ziffernfolge>,<mult>)),
(<ZifferohneNull><Ziffernfolge>,(<mult>,<mult>))
→((1eps,(3eps,<mult>)),(2eps,(<mult>,<mult>))
→ ((1,(3,leer),(2,(leer,leer)))

Aufgabe 9 (4 Punkte)

Gegeben seien die beiden Funktionen
funktion even x:nat →bool= wenn x=0 dann true sonst odd (x-1) ende.
funktion odd x:nat→bool= wenn x=0 dann false sonst even(x-1) ende.
Vollziehen die Arbeitsweise der Formularmaschine für odd 2 nach.

Lösung: (4/4 Punkte)



Aufgabe 10 (12 Punkte)

Ein ternärer markierter Baum ist ein markierter geordneter Baum, in dem jeder Knoten höchstens drei Söhne besitzt.

- **a.)** Definieren Sie einen polymorphen ternären markierten Baum in einer funktionalen Programmiersprache. (3 Punkte)
- **b.)** Schreiben Sie eine boolesche Funktion in funktionaler Programmierung, die prüft, ob ein vorgegebenes Objekt in einem ternären Baum als Marke vorkommt oder nicht. (4 Punkte)
- **c.)** Schreiben Sie ein Funktional in funktionaler Programmierung auf ternären Bäumen, das die Markierung jedes Knotens mithilfe einer Transformationsfunktion in eine andere Markierung ändert. (5 Punkte)

Lösung:

a.)

```
(* Baum ist die Definition eines ternären polymorphen('mark) Baumes *)
(* Dieser Baum hat folgende Struktur : node(Marke, Baum, Baum, Baum)  wobei die Marke
von beliebigen Typs ist (polymorph) *)
datatype 'mark Baum = leer | node of 'mark*'mark Baum*'mark Baum*'mark Baum ;

(* Diese Funktion gibt die erste Marke eines Baumes zurück *)
fun 'mark getmark(node(x:'mark,_,_,_))=x;

(* Diese Funktion gibt den ersten Sohn eines ternärenBaumes zurück *)
fun 'mark getfirstson(node(_,x:'mark Baum,_,_))=x;

(* Diese Funktion gibt den zweiten Sohn eines Ternären Baumes zurück *)
fun 'mark getsecondson(node(_,_,x:'mark Baum,_))=x;

(* Diese Funktion gibt den dritten Sohn eines ternären Baumes zurück *)
fun 'mark getthirdson(node(_,_,_,x:'mark Baum))=x;

(* Ein Beispiel eines ternären Baumes *)
val baum1:int Baum = (node(4,leer,leer,leer));
```

b.)

```
(* iselemoftree prüft ob ein Element als eine Markierung im Baum vorhanden ist*)
fun iselemoftree (l,n):bool= if(getmark l = n) then
                                 true     else
                                 if(getfirstson l=leer andalso getsecondson l=leer andalso
                                 getthirdson l=leer) then false else
                                 if(getfirstson l <>leer) then iselemoftree (getfirstson l,n) else
                                 if(getsecondson l <>leer)then  iselemoftree (getsecondson l,n)
                                 else
                                 if(getthirdson l <>leer)then iselemoftree (getthirdson l,n) else
                                 false;
```

c.)

```
(* Eine Beispielfunktion zum testen von Changemarks *)
fun change a = a+1 ;

(* Ist eine Funktion die eine andere Funktion auf jede Markierung eines Baumes anwendet *)
fun changemarks (tree, funk)=
if (tree = leer) then leer else
node(funk (getmark tree), changemarks (getfirstson tree, funk), changemarks (getsecondson
tree, funk),changemarks (getthirdson tree, funk));

(* test der Funktion changemarks *)
changemarks (baum1, change);
```