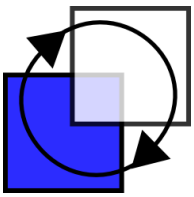
REACTIVE SYSTEMS GROUP

Universität des Saarlandes

Prof. Bernd Finkbeiner, Ph.D.
Markus Rabe, M.Sc.



Programmierung 1 (SS 2010) - 7. Übungsblatt

http://react.cs.uni-saarland.de/prog1/

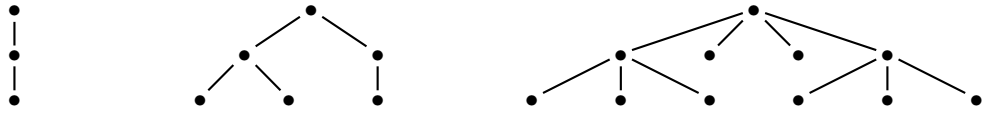
Lesen Sie im Buch bis zum Ende von Kapitel 7

Bei Problemen gehen Sie in die Office Hours: Mo-Mi von 13 bis 14 Uhr.

Aufgabe 7.1

a) Zeichnen Sie die Darstellung des Baums $T[t2, t1, t2]$. Beziehen Sie sich auf die Bäume in §7.1

b) Geben Sie Ausdrücke an, die Bäume mit den unten gezeigten grafischen Darstellungen beschreiben. *t2* verwenden.



Aufgabe 7.2

Sei die grafische Darstellung eines Baums gegeben. Nehmen Sie an, dass die Darstellung $n \geq 1$ Kanten enthält und beantworten Sie folgende Fragen:

a) Wie viele Knoten enthält die Darstellung mindestens/höchstens?

b) Wie viele Blätter enthält die Darstellung mindestens/höchstens?

c) Wie viele innere Knoten enthält die Darstellung mindestens/höchstnes?

Aufgabe 7.3 (Lexikalische Ordnung auf Bäumen)

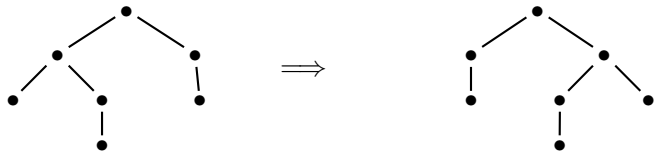
Leider bietet *MosML* die Prozedur List.collate nicht an, sodass Sie die Ordnung auf Bäumen nur durch einen Blick in die "Standard ML Basis Library" verstehen können. Wir geben hier eine alternative, aber semantisch äquivalente Definition der Ordnung auf Bäumen:

```
fun compareTree (T nil, T nil) = EQUAL
  | compareTree (T nil,  T tr) = LESS
  | compareTree (T  ts, T nil) = GREATER
  | compareTree (T (x::xr),T (y::yr)) = case compareTree (x,y) of
                        EQUAL => compareTree (T xr, T yr)
                      |   x   => x
```

Ordnen Sie die folgenden Bäume gemäß der lexikalischen Ordnung an :
$t1$, $t2$, $t3$, $T[T[t1]\,]$, $T[t1, T[T[t1]\,]\,]$, $T[T[T[T[t1]\,]\,]\,]$. Wobei $t1$, $t2$ und $t3$ wie in Kapitel 7.1 definiert sind.

Aufgabe 7.4 (Spiegeln)

Spiegeln reversiert die Ordnung der Unterbäumen der Teilbäume eines Baums.



Schreiben Sie eine Prozedur *mirror* : *tree* → *tree*, die Bäume spiegelt.

Aufgabe 7.5 (Node, Root, Leaf, Inner, Succ, Pred)

Schreiben Sie eine Prozedur

a) *node* : *tree* → *int list* → *bool*, die testet, ob es sich bei einer Adresse um einen Knoten eines Baums handelt.

b) *root* : *tree* → *int list* → *bool* die testet, ob es sich bei einer Adresse um die Wurzel handelt.

c) *leaf* : *tree* → *int list* → *bool*, die testet, ob es sich bei einer Adresse um ein Blatt eines Baums handelt.

d) *inner* : *tree* → *int list* → *bool* die testet, ob es sich bei einer Adresse um einen inneren Knoten eines Baums handelt.

e) *pred* : *int list* → *int list*, die den Vorgänger eines Knotens liefert. Wenn es sich bei dem Knoten um die Wurzel handelt, soll die Ausnahme *Subscript* geworfen werden.

f) *succ* : *tree* → *int list* → *int* → *int list*, die den k-ten Nachfolger eines Knotens liefert. Wenn der Knoten keinen k-ten Nachfolger hat, soll die Ausnahme *Subscript* geworfen werden.

Aufgabe 7.6

Schreiben Sie eine Prozedur *isPred* : *int list* → *int list* → *bool*, die für zwei Adressen a und b testet, ob es einen Baum gibt, in dem a den Vorgänger des durch b bezeichneten Knoten bezeichnet.

Aufgabe 7.7

Schreiben Sie die Prozeduren *size* und *depth* in §7.4 so um, dass sie ohne *map* auskommen. Hilfe: Erledigen Sie die rekursive Anwendung von *size* und *depth* in der Verknüpfungsprozedur für *foldl*.

Aufgabe 7.8

Ein Baum heißt ternär wenn jeder seiner inneren Knoten drei Unterbäume hat.

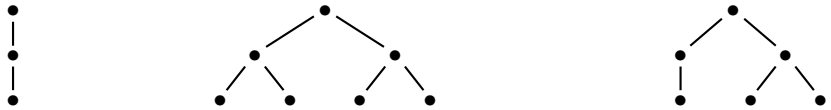
a) Entwickeln Sie Konstruktortypen zur Darstellung ternärer Bäume.

```
datatype terntree =
```

b) Schreiben Sie eine Prozedur *size3* : *terntree* → *int* , die die Größe eines ternären Baumes bestimmt.

Aufgabe 7.9

Geben Sie für die folgenden Bäume jeweils die Prä- und die Postnummerierung an:



Aufgabe 7.10

Die arithmetischen Ausdrücke aus Kapitel 6.4 kann man als binäre Bäume auffassen. Dabei liefern die Konstruktoren C und V atomare und die Konstruktoren A und M zusammengesetzte Bäume.

a) Schreiben Sie eine Prozedur *prelin* : *exp* → *int list list*, die die Adressen eines Ausdrucks in Präordnung liefert.

b) Schreiben Sie eine Prozedur *postlin* : *exp* → *int list list*, die die Adressen eines Ausdrucks in Postordnung liefert.

Aufgabe 7.11

Schreiben Sie zwei Prozeduren *prelin* und *postlin* des Typs *tree* → *int list list*, die die Adressen eines Baumes in Prä- beziehungsweise Postordnung liefern.

Aufgabe 7.12

Schreiben Sie eine Prozedur $prest' : tree \to int \to tree$, die zu einem Baum und einer Pränummer den entsprechenden Teilbaum liefert. Wenn die angegebene Zahl keine Pränummer des Baums ist, soll die Ausnahme *Subscript* geworfen werden.

Aufgabe 7.13

a) Geben Sie die Prä- und die Postlinearisierung des Baums $T[T[], T[T[]], T[T[], T[]]]$ an.

b) Gibt es Listen über $\mathbb{N}$, die gemäß der Prä- oder Postlinearisierung keine Bäume darstellen?

Aufgabe 7.14

Schreiben Sie eine Prozedur *direct* : *tree* → *tree*, die zu einem Baum einen gerichteten Baum liefert, der die gleiche Menge darstellt. Orientieren Sie sich an Sortieren durch Mischen.

Aufgabe 7.15

Seien finitäre Mengen durch gerichtete Bäume dargestellt.

a) Schreiben Sie Prozeduren *union*, *intersect* und *diff* des Typs *tree* → *tree* → *tree*, die zu zwei Mengen X und Y die Vereinigung $X \cup Y$, den Schnitt $X \cap Y$ und die Differenz $X - Y$ liefern.

b) Deklarieren Sie Prozeduren des Typs *tree* → *tree* → *bool*, die zu zwei Mengen X, Y testen, ob $X \in Y$ beziehungsweise $X \subseteq Y$ gilt.

Aufgabe 7.16

Schreiben Sie eine Prozedur *find* : $(\alpha \to bool) \to \alpha\ ltr \to \alpha\ option$, die zu einer Prozedur und einem Baum die gemäß der Präordnung erste Marke des Baums liefert, für die die Prozedur *true* liefert. Orientieren Sie sich an der Prozedur *prest* in Abschnitt 7.6.1.

Aufgabe 7.17

Schreiben Sie eine Prozedur *sum* : *int tree* $\to$ *int*, die die Summe aller Marken eines Baums liefert. Wenn eine Marke mehrfach auftritt, soll sie auch mehrfach in die Summe eingehen.

Aufgabe 7.18

Schreiben Sie eine Prozedur *lmap* : $(\alpha \to \beta) \to \alpha\ tree \to \beta\ tree$, die eine Prozedur auf alle Marken eines Baums anwendet. Verwenden Sie die Prozedur *map* für Listen.

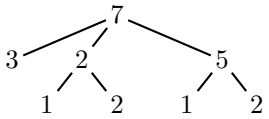
Aufgabe 7.19 (Projektionen)

a) Schreiben Sie eine Prozedur *prep* : $\alpha\ ltr \to \alpha\ list$, die die Präprojektion eines markierten Baumes liefert.

b) Schreiben Sie eine Prozedur *pop* : $\alpha\ ltr \to \alpha\ list$, die die Postprojektion eines markierten Baums liefert.

Aufgabe 7.20

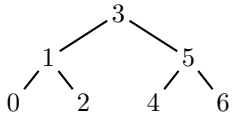
Die *n-te Ebene* eines Baums ist die Liste der Marken der Knoten der Tiefe n, angeordnet entsprechend der Anordnung der Knoten. Als Beispiel betrachten wir den Baum



Die nullte Ebene ist [7], die erste Ebene ist $[3, 2, 5]$, die zweite Ebene ist $[1, 2, 1, 2]$, und die dritte und alle nachfolgenden Ebenen sind []. Schreiben Sie eine Prozedur *level* : $\alpha\ ltr \to int \to \alpha\ list$, die die n-te Ebene eines Baums liefert.

Aufgabe 7.21

Bei der Inprojektion eines binären Baums erscheinen die Marken der inneren Knoten jeweils nach den Marken des linken und vor den Marken des rechten Unterbaums: $inpro(L(x, [t_1, t_2])) = inpro\ t_1\ @[x]\ @\ inpro\ t_2$. Beispielsweise ist $[0, 1, 2, 3, 4, 5, 6]$ die Inprojektion des Baums



Schreiben Sie eine polymorphe Prozedur *inpro* : $\alpha\ ltr \to \alpha\ list$, die die Inprojektion eines binären Baums liefert. Wenn die Prozedur auf einen Baum angewendet wird, der nicht binär ist, soll die Ausnahme *Subscript* geworfen werden.