Fachrichtung 6.2 — Informatik
Universität des Saarlandes
Tutorenteam der Vorlesung Programmierung 1

Programmierung 1 (Wintersemester 2012/13)

Erklärung 5

(Prä- und Postordnung)

Hinweis: Dieses Blatt enthält eine zusätzliche Erklärung erstellt von den Tutoren. Für die Richtigkeit besteht daher keine Gewähr.

Die Erklärung sowie ihr Thema sind für die Klausur weder relevant noch irrelevant.

In diesem Dokument wollen wir die Prä- und Postordnung neu erklären und die Zusammenhänge zu anderen Begriffen auf Bäumen darstellen. Prä- und Postordnung ist ein wichtiger Punkt bei der Verarbeitung von Bäumen.

Das Dokument enthält an wichtigen Stellen kleine Aufgaben, die das Verständnis weiter verbessern sollen. Es empfiehlt sich, diese Aufgaben an Ort und Stelle zu lösen!

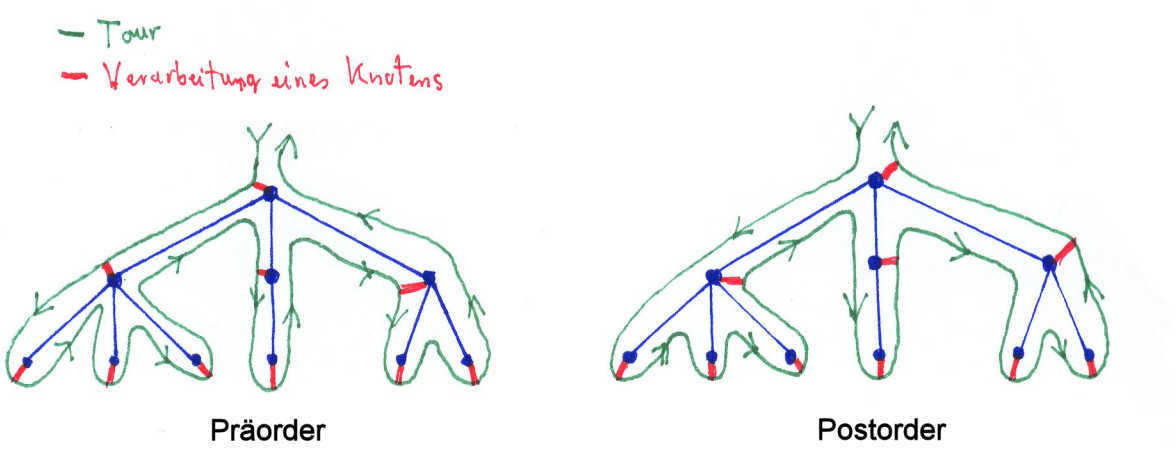
1 Ordnung

Die Prä- und Postordnung bezeichnen Methoden, einen Baum zu durchlaufen und dabei alle seine Knoten zu betrachten. Eine mögliche Sichtweise ist dabei: Bildlich gesprochen beginnt die **Präordnung** bei der Wurzel und steigt dann in die einzelnen Teilbäume ab. Die **Postordnung** beginnt beim linken unteren Blatt und steigt dann eine Ebene auf, wenn alle Unterbäume eines Knotens behandelt wurden. Diese Sicht ist schon sehr algorithmisch und kann, wie wir später sehen, relativ einfach in Rekursions gleichungen übersetzt werden.

Eine andere Vorstellung, die für Prä- und Postordnung hilfreich sein kann, ist die folgende: Man zeichnet eine Linie um den Baum herum, wie hier in grüner Farbe dargestellt. Dann nummeriert man die Knoten die man sieht bei 0 beginnend nach folgendem Schema:

- *Präordnung:* Die Knoten werden fortlaufend nummeriert, wenn man sie auf der linken Seite der (hier grünen) Linie sieht.
- *Postordnung:* Die Knoten werden fortlaufend nummeriert, wenn man sie auf der rechten Seite der (hier grünen) Linie sieht.

Die roten Linien im Bild verdeutlichen das sehr gut. Im linken Bild zeigen alle diese Linien nach links (vom Knoten aus gesehen), im rechten Bild nach rechts.



Nun können wir zwar die Prä- und Postordnung selbst angeben, unser Ziel ist es aber diese in Rekursionsgleichungen und dann in SML zu übersetzen.

Bei der Präordnung besucht man zuerst die Wurzel eines Baumes und dann rekursiv seine Teilbäume. Bei der Postordnung besucht man zuerst die Teilbäume und dann den Knoten. *Das ist sehr wichtig zum Schreiben von Prozeduren, die nach Prä- oder Postordnung vorgehen sollen.* Alle diese Prozeduren sind nach dem folgenden informellen Schema aufgebaut:

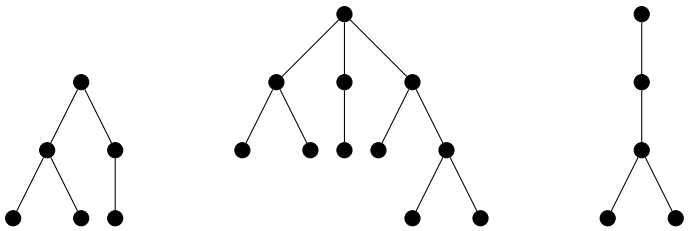
```
preorder  (T ts) = verarbeite_wurzel(T ts) @ verarbeite_alle_rekursiv(ts)
postorder (T ts) = verarbeite_alle_rekursiv(ts) @ verarbeite_wurzel(T ts)
```

Hier bezeichnet *ts* die Liste aller Untebäume. Eine der einfachsten Methoden ist es mit Hilfe der Prozedur *map* die Rekursion auf allen Unterbäumen durchzuführen. Dazu ein Beispiel:
Beispiel: Prälinearisierung.

```
fun prelin (T ts) = length ts :: foldl (fn (t,a) => a @ prelin t) nil ts     (ohne map)
(Schema:)           ( Wurzel )   (         Unterbäume rekursiv         )
fun prelin (T ts) = length ts ::      foldr op@ nil (map prelin ts)          (mit map)
```

Übungen

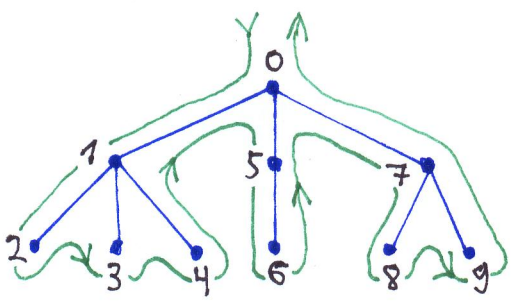
(a) Nummeriert die Knoten gemäß der Prä- und der Postorder.



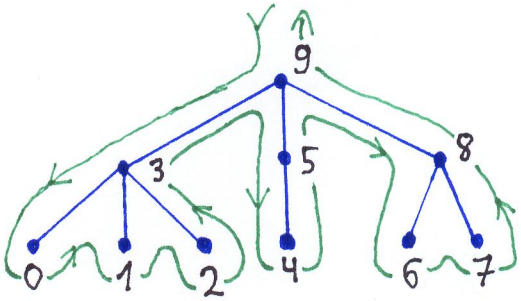
(b) Deklariert eine Prozedur *postlin* : *tree* $\rightarrow$ *int list* analog zu *prelin* und obigem Schema für die Postordnung.

2 Prä/Postnummerierung

Wie wir schon festegestellt haben geht man bei der Prä- bzw. Postnummerierung die Baumknoten in der entsprechenden Reihenfolge durch und **nummeriert** die Knoten in der Reihenfolge, in der man sie erreicht. Man beginnt bei 0 zu zählen. Falls noch nicht geschehen, ist es jetzt ein guter Zeitpunkt, sich die Nummerierung für die obigen Bäume zu überlegen.



Präorder



Postorder

3 Linearisierung

Auch bei der Linearisierung geht man die Knoten des Baumes in der gewählten Reihenfolge durch. Wir haben oben schon die Prozedur gesehen, hier jedoch nochmals das Vorgehen: Für jeden Knoten schreibt man hier die **Anzahl seiner Unterbäume** in eine Liste. Für einen Baum $T\ ts$ kann man die Anzahl mittels $length\ ts$ bestimmen. Um die Unterbäume durchzugehen, müssen wir gemäß dem Schema unsere Prozedur rekursiv auf alle Teilbäume anwenden: $map\ lin\ ts$. Dabei müssen die Ergebnisse alle in eine Liste, etwa so: $foldr\ op@\ nil\ (map\ lin\ ts)$. Nun können wir gemäß dem Schema die Prozeduren schreiben.

```
fun prelin  (T ts) = length ts :: foldlr op@ nil (map prelin ts)
fun postlin (T ts) =              foldlr op@ nil (map prelin ts) @ [length ts]
```

Beispiel: Die Prälinearisierung unseres Beispielbaumes ist $[3, \underbrace{3,0,0,0}, \underbrace{1,0}, \underbrace{2,0,0}]$, die Postlinearisierung ist $[\underbrace{0,0,0,3}, \underbrace{0,1}, \underbrace{0,0,2}, 3]$. Die Klammern markieren die Linearisierungen der Teilbäume.

Übungen

(a) Warum benötigen wir keinen Basisfall für die obigen Prozeduren? Was ist die Vorgehensweise?

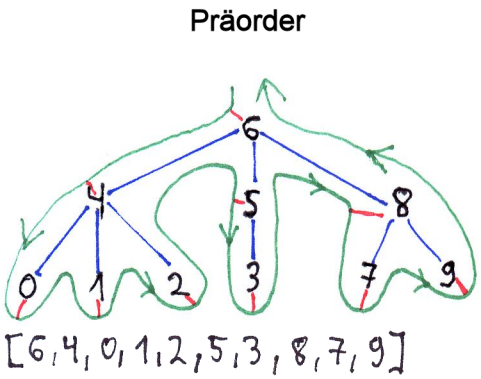
(b) Bestimmt die Linearisierungen der Bäume von oben.

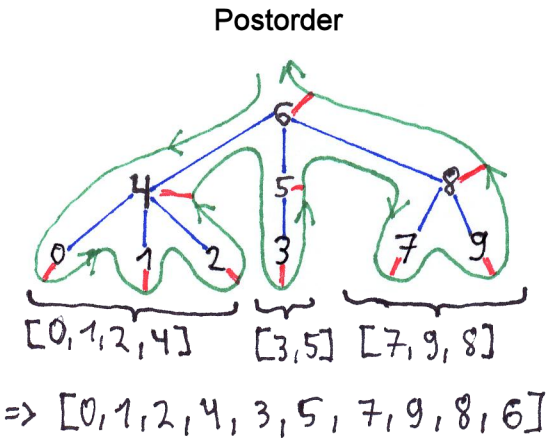
(c) Schreibt die Linearisierung für beliebige markierte Bäume ($\alpha\ ltr$) um.
Hinweis: Die Marken sollen in der Linearisierung nicht auftauchen.

(d) Warum verwende ich in den beiden Prozeduren (*prelin*/*postlin*) jeweils $foldr$ anstatt $foldl$? Was würde passieren, wenn man $foldr$ schreiben würde? Versucht diese Frage ohne Interpreter zu beantworten.

4 Projektion

Die Projektion ist der Linearisierung sehr ähnlich. Wir gehen entsprechend der Ordnung den Baum durch, anstatt für jeden Knoten die Anzahl der Unterbäume in die Liste zu schreiben, nehmen wir die **Marke des Knotens**. Die Projektion ist also eine **Liste aller Beschriftungen**, in Präorder- oder Postorder-Reihenfolge.

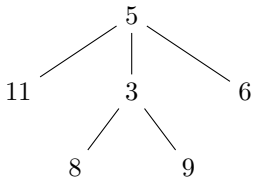


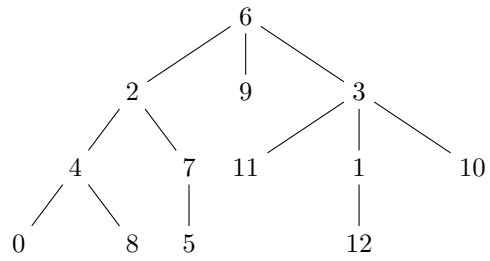



```
fun preproj (L (a,ts)) =    a   :: foldl (fn (t,a) => a @ preprop t) nil ts
                         (Wurzel)  (rekursiv die Unterbäume (ts) verarbeiten)
```

Übungen

(a) Bestimmt die Prä- und Postprojektion der folgenden Bäume:





(b) Schreibt analog $postproj : \alpha\, ltr- > \alpha\, list$. Schreibt *preproj* erneut in der Variante mit *map*.

(c) Vergleicht die Prozeduren zur Projektion mit denen zur Linearisierung und dem Schema.

5 Teilbaumzugriff

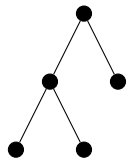
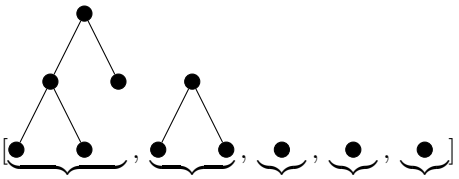
Auch der Teilbaumzugriff läuft sehr ähnlich ab. Der Zugriff auf den $n-$ten Teilbaum in Präordnung bedeutet, dass wir den $n-$ten Knoten samt Unterbaumliste wollen, den wir nach Präordnung erreichen. Wir beginnen hier bei 0 zu zählen. Im Buch sieht man die endrekursiven Varianten, da es aber einfacher ist, die Prozeduren nicht endrekursiv zu machen, werden wir die Vorgehensweise daran besprechen.
Wir suchen die Teilbäume eines Baumes nach einer Ordnung. Was liegt näher, als in der uns bekannten Ordnung alle Knoten abzugehen und ihren Teilbaum zu suchen? Wir erstellen daher wieder eine Liste (mit genau dem gleichen Ansatz wie eben!), aber im Gegensatz zu vorher schreiben für einen Knoten nicht seine Marke oder die Anzahl seiner Unterbäume in die Liste, sondern **den kompletten Teilbaum**. Wir erstellen also eine Liste aller Teilbäume eines Baumes:

```
fun preSubtrees (T ts) = (T ts) :: foldl (fn (t,a) => a @ preSubtrees t) nil ts
                         (Wurzel)  (   rekursive Anwendung auf die Unterbäume   )
```

Nun haben wir alle Teilbäume des Baumes in der richtigen Ordnung, wir müssen nur noch auf den richtigen Teilbaum zugreifen. Das macht die Prozedur *List.nth* für uns:

```
fun prest t n = List.nth(preSubtrees t, n)
```

Wir können nun also auf den $n-$ten Teilbaum eines Baumes zugreifen, nach bekanntem Schema.
Beispiel: $prest\ \underbrace{(T[\,T[T[],T[]]\,,\,T[]])}_{=t}\ 1$

$t =$  $;preSubtrees\ t = [$  $]$; also $prest\ t\ 1 =$ 

Übungen

(a) Schreibt die Prozedur $postSubtrees : tree \to treelist$, die alle Teilbäume eines Baumes in Postorder liefert. Schreibt dann *post*, die den Teilbaumzugriff über Postnummern realisiert.

(b) Schreibt die Prozeduren *preSubtrees* und *postSubtrees* erneut unter Verwendung von *map*. Ihr könnt euch an obigen Prozeduren orientieren.

(c) Macht euch die Ähnlichkeit der Nummerierung, Linearisierung, Projektion und Teilbaumzugriffe klar. Stellt euch dazu die Prozeduren dafür in eurer bevorzugten Schreibweise zusammen (mit $foldl/r$ und *map* oder nur mit $foldl$).

(d) Macht euch den Unterschied zwischen Prä- und Postordnung erneut an einem Beispiel, dem Schema von oben, und den einzelnen Prozeduren klar.