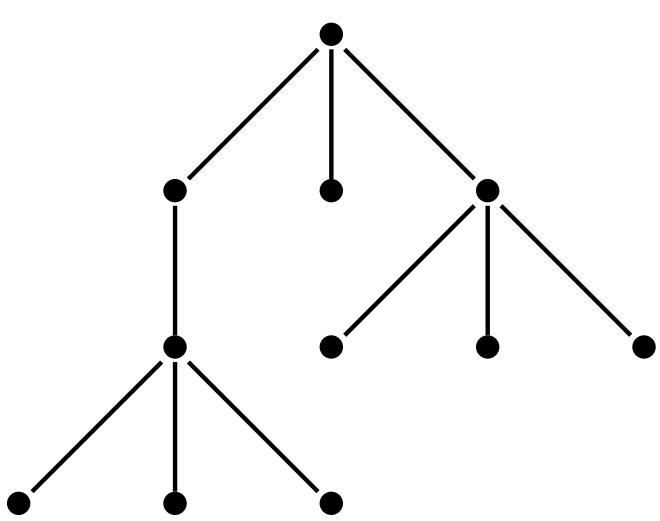
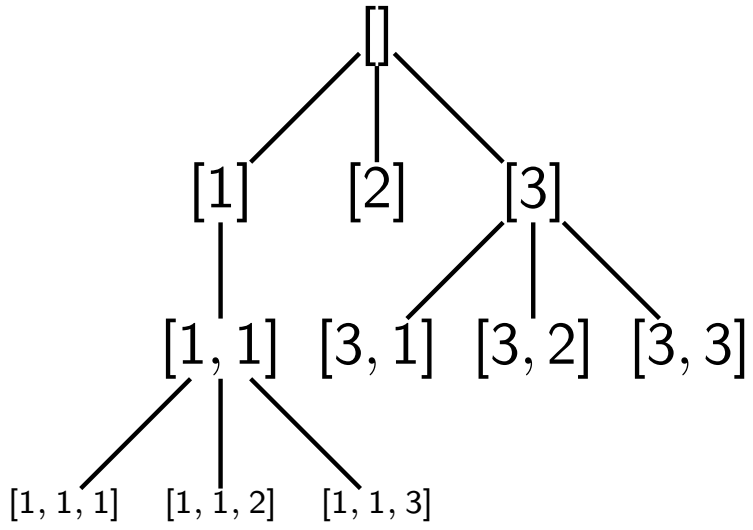
Adressen





- Finde Teilbaum zu gegebener Adresse:

```
fun ast t xs = foldl (fn (n,(T ts))
                         => List.nth(ts,n-1)) t xs
```

- Eine Adresse *a* heißt für einen Baum *t* *gültig*, wenn `ast t a` einen Teilbaum liefert. Ansonsten heißt *a* *ungültig*.

Begriffe

Sei ein Baum t gegeben, seien a und a' gültige Adressen für t. Wir sagen, dass

- der durch a' bezeichnete Knoten der **n-te Nachfolger** des durch a bezeichneten Knoten ist, wenn $a' = a@[n]$ gilt.
- der durch a' bezeichnete Knoten ein **Nachfolger** des durch a bezeichneten Knoten ist, wenn eine Zahl n mit $a' = a@[n]$ existiert.
- der durch a bezeichnete Knoten ein **Vorgänger** des durch a' bezeichneten Knoten ist, wenn eine Zahl n mit $a' = a@[n]$ existiert.
- der durch a' bezeichnete Knoten dem durch a bezeichneten Knoten **untergeordnet** ist, wenn eine Liste ns mit $a' = a@ns$ existiert.
- der durch a bezeichnete Knoten dem durch a' bezeichneten Knoten **übergeordnet** ist, wenn eine Liste ns mit $a' = a@ns$ existiert.

Propositionen

- *Proposition 7.1:* Zwei (reine) Bäume sind genau dann gleich, wenn sie die gleichen gültigen Adressen haben.

- *Proposition 7.2:* Für jeden Baum t gilt:
  1. [] ist eine gültige Adresse für t.
  2. Wenn a eine gültige Adresse für t ist, dann ist jedes Präfix von a eine gültige Adresse für t.
  3. Wenn $a@[n]$ eine gültige Adresse für t ist und $1 \leq k \leq n$, dann ist $a@[k]$ eine gültige Adresse für t.

- *Proposition 7.3:* Sei t ein Baum. Dann haben alle Knoten ausser der Wurzel einen Vorgänger, die Wurzel hat keinen Vorgänger.

Suche nach Teilbäumen

```
fun iT t (T ts) = (t = T ts) orelse List.exists (iT t) ts

fun exists f nil = false
  | exists f (x::xr) = f x orelse exists f xr
```

```
iT t2 t3 = iT t2 T[T[t2]],t1,t2]
= false orelse exists (iT t2) [T[T[t2]],t1,t2]
= exists (iT t2) [T[T[t2]],t1,t2]
= iT t2 T[T[t2]] orelse exists (iT t2) [t1,t2]
= (false orelse exists (iT t2) [T[t2]]) orelse ...
= exists t2 [T[t2]] orelse ...
= ((iT t2 T[t2]) orelse exists (iT t2) []) orelse ...
= ((false orelse exists (iT t2) [t2]) orelse ...) orelse ...
= (true orelse ...) orelse ...
= true
```

Zählen von Teilbäumen

```
fun zT t (T ts) = if (t = T ts) then 0
                  else foldl op+ 0 (map zT t) ts

fun map f nil = nil | map f (x::xr) = (f x)::map f xr
```

```
zT t1 t2 = zT t1 T[t1,t1,t1]
= if (t1=T[t1,1,t1]) then 1 else foldl op+ 0 (map zT t1) [t1,t1,t1]
= foldl op+ 0 (map zT t1) [t1,t1,t1]
= foldl op+ 0 (if (t1=t1) then 1 else ...)::(map zT t1 [t1 t1])
= foldl op+ 0 1::(map zT t1 [t1 t1])
= ...
= foldl op+ 0 [1,1,1]
= 3
```