1 Aufgabe 2

In diesem Modul werden wir Listen benötigen, also schreiben wir dies — und auch das Typsynonym — schon einmal hin:

```
> module Main(main) where
> import List
>
> type Rel a = [(a,a)]
```

1.1 *lub* und *glb*

Angenommen wir hätten die Relation schon vollständig spezifiziert, d.h. ihre transitive Hülle wäre uns bekannt, dann könnten wir die Funktionen `up` und `low` als einfache list comprehension hinschreiben:

```
> up x rel = nub [ a | (b, a) <- rel, b == x ]
> down x rel = nub [ a | (a, b) <- rel, b == x]
```

Das sind nämlich gerade all die Elemente `a`, die in Relation zu `x` stehen bzw. umgekehrt.

Es ist sinnvoll, nun zuerst Funktionen `ub` und `lb` für die oberen resp. unteren Schranken zu beschreiben. Dies können wir über ein `map` von `up` bzw. `down` auf der Liste der gegebenen Elemente erreichen.

```
> ub list rel =
>    map (\x -> up x rel) list

> lb list rel =
>    map (\x -> down x rel) list
```

Bilden wir dann von einer Liste von oberen (unteren) Schranken den Schnitt und wählen anschließend das kleinste (größte) Element bezüglich der Relation, so erhalten wir die Funktion `lub` (`glb`)

```
> lub list rel =
>    smallest (interSectionClosure (ub list rel)) rel

> smallest [x] rel = x
> smallest list rel =
```

```
>    case (nub l) of
>       [x] -> x
>       other -> error "not defined"
>    where
>       l = [ x | x <- list,
>                  all (\y -> if x /= y then elem (x,y) rel else True) list]

> glb list rel =
>    biggest (interSectionClosure (lb list rel)) rel

> biggest [x] rel = x
> biggest list rel =
>    case (nub l) of
>       [x] -> x
>       other -> error "not defined"
>    where
>       l = [ x | x <- list,
>                  all (\y -> if x /= y then elem (y,x) rel else True) list]

> interSectionClosure (list:lists) =
>    foldr intersect list lists
> interSectionClosure [] = []
```

1.2 Berechnung der transitiven Hülle

Die Berechnung der transitiven Hülle basiert darauf, sukzessive alle Paare `(a,b)` für die es ein `c` gibt, so daß `(a,c)` und `(c,b)` in Relation stehen. Dies wird gemacht, solange irgendein *neues* Element gefunden wird.

```
> transitiv (a,b) aus = [ (a,c) | (d,c) <- aus, b == d ]

> transitiveElemente zu aus1@(x:xs) aus2 =
>    transitiv zu aus1 ++
>    transitiv zu aus2 ++
>    transitiveElemente x xs (zu:aus2)
> transitiveElemente zu [] aus =
>    transitiv zu aus

> transitiveHuelle liste@((x1,x2):xs)
>    | any (\x -> notElem x liste) dazu = transitiveHuelle (dazu ++ liste)
>    | otherwise = (nub liste)
>    where
>       dazu = transitiveElemente (x1,x2) xs []
```

Für die Aufgabe schreiben wir die Relation als transitive Hülle der gegebenen geordneten Menge.

```
> rel = transitiveHuelle
>          [("bot", "a"), ("bot", "b"),
>           ("a", "c"), ("a", "d"),
>           ("b", "d"), ("b", "e"),
>           ("c", "k"), ("c", "f"),
>           ("d", "f"), ("d", "g"),
>           ("e", "h"),
>           ("k", "top"),
>           ("f", "j"),
>           ("g", "k"), ("g", "l"),
>           ("h", "i"), ("h", "j"),
>           ("j", "m"),
>           ("l", "top"),
>           ("i", "l"), ("i", "m"),
>           ("m", "top")]
```

Außerdem erleichtern wir uns den Testlauf und die Aufgabe 1, indem wir die Ausdrücke für die Teilaufgaben hinschreiben

```
> i    = glb ["l", "e"] rel
> ii   = glb [ glb ["l", "j"] rel, "k"] rel
> iii  = lub ["c", "e"] rel
> iv   = lub ["d", "e"] rel
> v    = lub ["c", lub ["d", "e"] rel ] rel
> vi   = lub ["c", "d", "e"] rel
> vii  = glb [ glb ["j", "l"] rel, "k"] rel
> viii = glb ["j", glb ["l", "k"] rel ] rel
> ix   = glb ["j", "k"] rel
```

2 Abstraktere Version

Die präsentierte Lösung ist im Hinblick auf die Länge des Programmtextes alles andere als optimal. Wir können es uns z.B. sparen, jeweils eine Funktion `smallest` und `biggest` zu schreiben, wenn wir einfach festlegen, was kleiner bzw. größer bezüglich der Relation heißt. Dazu schreiben wir zwei kleine Funktionen `smaller` und `bigger`. Dann können wir eine Funktion `most` angeben, die höherer Ordnung ist und für die `most smaller` bzw. `most bigger` das gleiche bedeutet wie `smallest` resp. `biggest`.

Für die transitive Hülle kann man eine ähnliche Abstraktion machen, wenn man sich nicht darauf festlegtt, die *transitive* Hülle haben zu wollen.

```
> module Main(main) where
> import List
>
> type Rel a = [(a,a)]
```

```
> smaller x y rel = elem (x,y) rel
> bigger x y rel = elem (y,x) rel

> up x rel = nub [ a | (_,a) <- rel, bigger a x rel ]
> down x rel = nub [ a | (a,_) <- rel, smaller a x rel ]
```

```
> ub list rel =
>    map (\x -> up x rel) list

> lb list rel =
>    map (\x -> down x rel) list
```

```
> lub list rel =
>    most smaller (interSectionClosure (ub list rel)) rel

> glb list rel =
>    most bigger (interSectionClosure (lb list rel)) rel

> most pred [x] rel = x
> most pred list rel =
>    case (nub m_list) of
>       [x] -> x
>       other -> error "element not defined!"
>    where
>       m_list = [ x | x <- list,
>                      all
>                         (\y -> if x /= y then pred x y rel else True)
>                         list ]

> interSectionClosure (list:lists) =
>    foldr intersect list lists
> interSectionClosure [] = []
```

```
> transitiv :: Eq a => (a,a) -> Rel a -> Rel a
> transitiv (a,b) aus = [ (a,c) | (d,c) <- aus, b == d ]

> reflexiv ::  (a,a) -> Rel a -> Rel a
> reflexiv (a,b) aus = [(a,a), (b,b)]

> neueElemente :: ((a,a) -> Rel a -> Rel a) -> (a,a) -> Rel a -> Rel a -> Rel a
> neueElemente was zu aus1@(x:xs) aus2 =
>    was zu aus1 ++
>    was zu aus2 ++
>    neueElemente was x xs (zu:aus2)
> neueElemente was zu [] aus =
>    was zu aus

> huelle :: Eq a => ((a,a) -> Rel a -> Rel a) -> [(a,a)] -> [(a,a)]
> huelle was liste@((x1,x2):xs)
>    | any (\x -> notElem x liste) dazu = huelle was (dazu ++ liste)
>    | otherwise = (nub liste)
>    where
>       dazu = neueElemente was (x1,x2) xs []

> rel = huelle transitiv
>          [("bot", "a"), ("bot", "b"),
>           ("a", "c"), ("a", "d"),
>           ("b", "d"), ("b", "e"),
>           ("c", "k"), ("c", "f"),
>           ("d", "f"), ("d", "g"),
>           ("e", "h"),
>           ("k", "top"),
>           ("f", "j"),
>           ("g", "k"), ("g", "l"),
>           ("h", "i"), ("h", "j"),
>           ("j", "m"),
>           ("l", "top"),
>           ("i", "l"), ("i", "m"),
>           ("m", "top")]



> i    = glb ["l", "e"] rel
> ii   = glb [ glb ["l", "j"] rel, "k"] rel
> iii  = lub ["c", "e"] rel
> iv   = lub ["d", "e"] rel
> v    = lub ["c", lub ["d", "e"] rel ] rel
> vi   = lub ["c", "d", "e"] rel
> vii  = glb [ glb ["j", "l"] rel, "k"] rel
```

```
> viii = glb ["j", glb ["l", "k"] rel ] rel
> ix   = glb ["j", "k"] rel
```

3 Lösung zu Aufgabe 1

```
? i
"b"
? ii
"
Program error: element not defined!


? iii
"j"
? iv
"
Program error: element not defined!

? v
"
Program error: element not defined!

? vi
"j"
? vii
"
Program error: element not defined!

? viii
"d"
? ix
"
Program error: element not defined!

?
```