Introduktion til programmering, ugeseddel 5

Version 1.1

27. september 2014

Den femte undervisningsuge handler om *rekursive datatyper*, *parameteriserede datatyper* og *gensidig rekursion*.

Omrokering:

Lige som i sidste uge bytter vi rundt på fredagsforelæsningen og mandagens repetitionstime. Mandag morgen vil dække nyt stof og fredagen vil blive brugt til repetition.

Ligeledes har vi igen fordelt tirsdagsøvelserne mellem tirsdag og fredag, så den første time begge dage bruges på at gennemgå øvelser fra ugesedlen og den anden time bruges på at arbejde på afleveringsopgaverne.

1 Plan for ugen

Mandag

Rekursive datatyper, gennemløb af træer.

Pensum: HR: 8.1-8.3.

Tirsdag

Gensidig rekursion og parametriserede datatyper.

Pensum: HR: 8.4-8.6

Fredag

Repetition af ugens pensum. Se ovenfor.

2 Mandagsopgaver

Emner: Rekursive datatyper, tilfældige tal.

Vi kan definere en datatype, der repræsenterer en udvælgelsesproces, der foretager tilfældige valg med en given sandsynlighed og til sidst ender med et specifikt tal:

```
datatype selection = Pick of int
                   | Choose of real * selection * selection
```

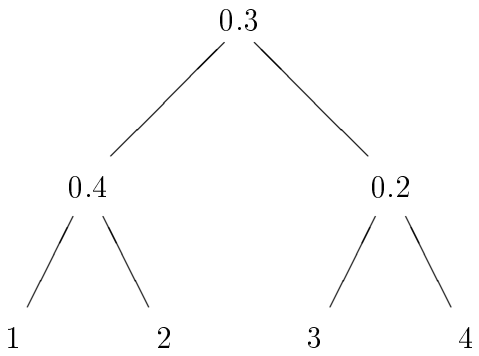
Udvælgelsesprocessen `Pick` n siger, at man altid (altså med sandsynlighed 1.0) vælger tallet n.

Udvælgelsesprocessen `Choose` (p, s, t) vælger mellem (med sandsynlighed p, hvor $0.0 < p < 1.0$), at fortsætte med udvælgelsesprocessen s eller (med sandsynlighed $1.0 - p$) at fortsætte med udvælgelsesprocessen t.

Man kan betragte en udvælgelsesproces som et binært træ med sandsynligheder i forgreningerne og heltal i bladene. For eksempel kan udvælgelsesprocessen

```
val s1234 = Choose (0.3,
                    Choose (0.4, Pick 1, Pick 2),
                    Choose (0.2, Pick 3, Pick 4))
```

tegnes som træet



En udvælgelse med udvælgelsesprocessen `s1234` sker på følgende måde:

1. Man starter i roden (toppen) af træet.
2. Med sandsynlighed 0.3 vælger man venstre undertræ og med sandsynlighed $1.0 - 0.3 = 0.7$ højre undertræ.
3. Hvis man derefter står i den knude, der er markeret 0.4, vælger man denne knudes venstre undertræ (tallet 1) med sandsynlighed 0.4 og knudens højre undertræ (tallet 2) med sandsynlighed $1.0 - 0.4 = 0.6$.
4. Hvis man efter trin 2 i stedet står i knuden markeret med 0.2, vælger man denne knudes venstre undertræ (tallet 3) med sandsynlighed 0.2 og knudens højre undertræ (tallet 4) med sandsynlighed $1.0 - 0.2 = 0.8$.

For at komme til tallet 1 skal man altså først træffe et valg med sandsynlighed 0.3 og derefter et valg med sandsynlighed 0.4, så sandsynligheden for at komme fra roden af

træet ned til tallet 1 er $0.3 \cdot 0.4 = 0.12$. Tilsvarende er sandsynligheden for at ende med tallet 2 lig med $0.3 \cdot 0.6 = 0.18$, sandsynligheden for at ende med tallet 3 er $0.7 \cdot 0.2 = 0.14$ og sandsynligheden for at ende med 4 er $0.7 \cdot 0.8 = 0.56$. Bemærk, at $0.12 + 0.18 + 0.14 + 0.56 = 1.0$, som forventet.

Det samme tal kan stå i flere `Pick`-knuder i samme udvælgelsesproces. Udvælgelsesprocessen `Choose (0.3, Pick 7, Pick 7)` er således en proces, der altid vælger tallet `7`.

En udvælgelsesproces siges at være *valid*, hvis alle de sandsynligheder p, der forekommer i knuder af formen `Choose` (p, s, t), ligger skarpt mellem 0.0 og 1.0.

5M1 En terning med n sider nummereret fra 1 til og med n siges at være *retfærdig*, hvis alle sider kommer op med lige stor sandsynlighed (som dermed er $\frac{1}{n}$).

Skriv en funktion `fairDie : int -> selection`, der givet et tal n, hvor $1 <= n$, returnerer en udvælgelsesproces, der svarer til en retfærdig terning med siderne 1 til og med n.

Kaldet `fairDie 2` kan for eksempel returnere udvælgelsesprocessen
`Choose (0.5, Pick 2, Pick 1)` og kaldet `fairDie 3` kan returnere
`Choose (0.333333333333, Pick 3, Choose (0.5, Pick 2, Pick 1))`

Vink: Kig på strukturen i eksemplerne herover og brug din forståelse af denne til at udfylde skitsen

```
fun fairDie 1 = ...
  | fairDie n = Choose (..., ..., fairDie (n-1))
```

5M2 Skriv en funktion `isValid : selection -> bool`, der afgør om en udvælgelsesproces er valid.

I alle de følgende opgaver kan det antages, at udvælgelsesprocesser er valide.

Vi bruger i den følgende opgave modulet `Random`, der omhandler generering af tilfældige tal[1]. Du får brug for funktionerne

```
Random.newgen : unit -> Random.generator
Random.random : Random.generator -> real
```

Kaldet `Random.newgen ()` returnerer en tilfældighedsgenerator g.
Kaldet `Random.random` g, hvor g er en generator, returnerer et tilfældigt tal mellem 0.0 og 1.0. Et eksempel på brug af funktionerne til at lave et par af to tilfældige tal mellem 0.0 og 1.1 er vist herunder.

```
- load "Random";
> val it = () : unit
- val g = Random.newgen ();
> val g = <generator> : generator
- (Random.random g, Random.random g);
> val it = (0.427284102155, 0.363904911729) : real * real
-
```

[1] Eller mere præcist *pseudotilfældige* tal, da de er konstrueret gennem en deterministisk proces.

5M3 Skriv en funktion `randomPick : selection -> Random.generator -> int`, der returnerer et tal, der er udvalgt fra udvælgelsesprocessen med de sandsynlighder, denne foreskriver.

Kaldet `randomPick g (Choose (0.3, Pick 7, Pick 9))`, hvor `g` er en tilfældighedsgenerator, skal altså returnere `7` i 30% af kaldene og `9` i 70% af kaldene. Tilsvarende skal kaldet `randomPick g (fairDie 5)` returnere værdier fra 1 til 5 med hver 20% sandsynlighed.

5M4 Skriv en funktion `probabilityOf : selection -> int -> real`, der beregner sandsynligheden for at en valid udvælgelsesproces vil vælge et givet tal.

Kaldet `probabilityOf s1234 1` skal altså returnere `0.12`, kaldet `probabilityOf s1234 2` skal returnere `0.18`, kaldet `probabilityOf s1234 3` skal returnere `0.14`, kaldet `probabilityOf s1234 4` skal returnere `0.56`, og kaldet `probabilityOf s1234` n skal returnere `0.0` for alle andre heltal n.

5M5 Skriv en funktion `average : selection -> real`, der beregner den gennemsnitlige værdi af de valg, en valid udvælgelsesproces kan træffe.

Kaldet `average s1234` skal altså returnere `3.14`, da $0.12 \cdot 1 + 0.18 \cdot 2 + 0.14 \cdot 3 + 0.56 \cdot 4 = 3.14$.

Hvis der er tid til overs, kan I begynde på tirsdagsopgaverne.

3 Tirsdagsopgaver

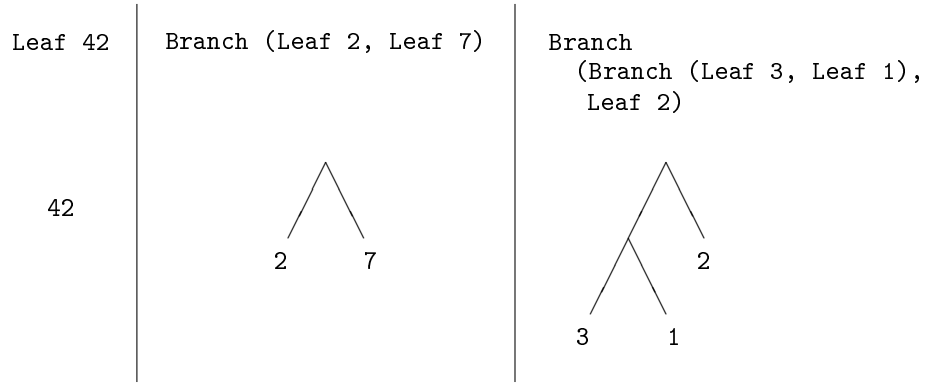
Emner: Rekursive datatyper, gennemløb at træer.

Det forventes, at du inden øvelserne tirsdag har forberedt dig på opgaverne ved at løse så mange som muligt på egen hånd.

Datatypen

```
datatype leafTree = Leaf of int
                  | Branch of leafTree * leafTree
```

definerer typen af træer med heltal i bladene og uden data i forgreningerne. Der er altid mindst et blad i træet. Herunder er vist et par eksempler på træer af typen `leafTree` både vist som SML-udtryk og i grafisk repræsentation.



Alle tirsdagsopgaverne omhandler denne slags træer.

5T1 Skriv en funktion `treeSum : leafTree -> int`, der givet et træ returnerer summen af alle træets blade. For eksempel skal kaldet

```
treeSum (Branch (Branch (Leaf 3, Leaf 1), Leaf 2))
```

returnere heltallet `6`.

5T2 Skriv en funktion `treeMax : leafTree -> int`, der givet et træ returnerer det største af tallene i alle træets blade. For eksempel skal kaldet

```
treeMax (Branch (Branch (Leaf 3, Leaf 1), Leaf 2))
```

returnere heltallet `3`.

5T3 Skriv en funktion `tree2List : leafTree -> int list`, der givet et træ returnerer en liste af tallene i alle træets blade i den rækkefølge, de forekommer i træet (dvs. at bladene i venstre gren af et træ kommer før bladene i højre gren af træet). For eksempel skal kaldet

```
tree2List (Branch (Branch (Leaf 3, Leaf 1), Leaf 2))
```

returnere listen `[3, 1, 2]`.

5T4 Tegn grafisk alle fem træer t : `leafTree`, hvor `tree2List` t returnerer listen `[1, 2, 3, 4]`.

5T5 Er det gennemløb, som `tree2List` definerer, et *præordens gennemløb*, et *inordens gennemløb* eller et *postordens gennemløb*? Giver distinktionen i det hele taget mening for typen `leafTree`? Se HR afsnit 8.3.2 – 8.3.3 for definitoner.

5T6 Skriv en funktion `mirrorTree : leafTree -> leafTree`, der givet et træ returnerer en spejling af træet: Alle grene byttes parvis om. For eksempel skal kaldet

```
tree2List (Branch (Branch (Leaf 3, Leaf 1), Leaf 2))
```

returnere træet `Branch (Leaf 2, Branch (Leaf 1, Leaf 3))`.

4 Fredagsopgaver

Emner: Parametriserede datatyper og gensidig rekursion.

Det forventes, at du inden øvelserne fredag har forberedt dig på opgaverne ved at løse så mange som muligt på egen hånd.

Fredagsøvelserne vil bruge følgende datatype, der beskriver træer med værdier i knuderne og et vilkårligt antal børn til hver knude. Et blad er dermed en knude uden børn:

```
datatype 'a generalTree = Node of 'a * 'a generalTree list
```

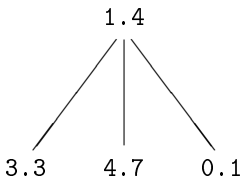
Herunder er vist et par eksempler på træer af typen `generalTree` både vist som SML-udtryk og i grafisk repræsentation.

```
Node (42, [])
```

42

```
Node(true,
     [Node(false, [])])
```


```
Node (1.4,
      [Node (3.3, []),
       Node (4.7, []),
       Node (0.1, [])])
```



5F1 Skriv en funktion `nodes : 'a generalTree -> int`, der returnerer antallet af knuder i træet. Anvendt på de tre herover viste træer skal `nodes` returnere henholdsvis 1, 2 og 4.

Brug to gensidigt rekursive funktioner `nodes : 'a generalTree -> int` og `nodesList : 'a generalTree list -> int`

5F2 Skriv en version af `nodes`, der ikke bruger gensidig rekursion, men bruger `List.foldr` eller `List.foldl` til at finde antallet af knuder i en liste af træer.

5F3 Skriv en funktion `preOrder : 'a generalTree -> 'a list`, der returnerer en liste af knuderne i træet i et *præordens gennemløb*. Se HR afsnit 8.3.2 for en definition.

Skriv `preOrder` både ved brug af to gensidigt rekursive funktioner og ved at bruge `List.map` og `List.concat` til at behandle listen af børn af en knude.

5 Opgavetema: Geometriske figurer

Vi bruger i ugeopgaven en datatype `figure` (der er en generalisering af datatypen, `figur`, der blev introduceret til mandagsforelæsningen) til at definere geometriske figurer i det reelle plan:

```
type point = real * real   (* et punkt i planen *)

datatype 'col figure
 = Circle of 'col * point * real  (* farve, center og radius *)
 | Rectangle of 'col * point * point
            (* farve, nederste venstre og oeverste hoejre hjoerne *)
 | Over of 'col figure * 'col figure
                                  (* foerste figur over anden figur *)
```

`Circle` (c, p, r) definerer en cirkel med farve c, centrum i p og radius r.

`Rectangle` (c, bl, tr) definerer en akseret rektangel (dvs. med lodrette og vandrette sider) med farve c, nederste venstre hjørne bl og øverste højre hjørne tr.

`Over` (f_1, f_2) definerer en figur af to underfigurer. Hvor de to underfigurer lapper over hinanden, ligger den første underfigur øverst.

Bemærk, at koordinatsystemet er som i matematik: Større værdier på første koordinat ligger længere mod højre, og større værdier på anden koordinat ligger længere oppe.

Bemærk endvidere, at typen `figure` er parametriseret over farven `'col`, så man kan f.eks. bruge typen `InstragraML.colour figure`, hvor farven altså er defineret som et RGB-tripel (`int * int * int`), eller bruge datatypen `primaryColours`:

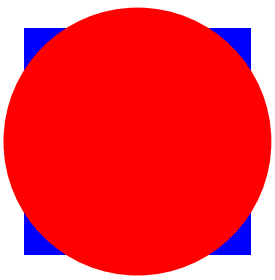
```
datatype primaryColours = Black | Red | Green | Blue
                        | White | Cyan | Magenta | Yellow
```

til at lave figurer af typen `primaryColours figure`.

Et eksempel på en figur af typen `primaryColours figure` er

```
val redCircleOverBlueSquare =
      Over (Circle (Red, (0.0, 0.0), 1.8),
            Rectangle (Blue, (~1.5, ~1.5), (1.5, 1.5)))
```

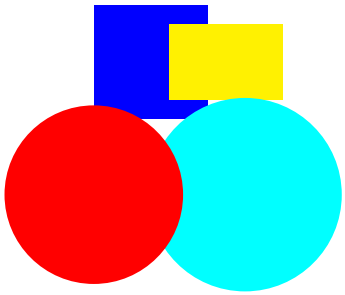
der definerer en rød cirkel med radius 1.8 og centrum i $(0, 0)$ ovenover et kvadrat med hjørner i $(-1.5, -1.5)$ og $(1.5, 1.5)$, som vist herunder:



En anden figur, der kan bruges til afprøvning i ugeopgaverne, er

```
val twoAndTwo =
      Over (Over (Circle (Red, (0.0, 0.0), 1.2),
                  Circle (Cyan, (2.0, 0.0), 1.3)),
            Over (Rectangle (Yellow, (1.0, 1.2), (2.5, 2.2)),
                  Rectangle (Blue, (0.0, 1.0), (1.5, 2.5))))
```

Denne figur viser to overlappende cirkler, rød over cyan, der delvist dækker to overlappende rektangler, gul over blå:



6 Gruppeaflevering

Gruppeafleveringen obligatorisk. Alle delspørgsmål skal besvares. Opgaven afleveres i Absalon. Der afleveres en fil pr. gruppe, men den skal angive alle deltageres fulde navne i kommentarlinjer øverst i filen. Filens navn skal være af formen `5G-`*`initialer`*`.sml`, hvor initialer er erstattet af gruppemedlemmernes initialer. Hvis f.eks. Bill Gates, Linus Torvalds, Steve Jobs og Gabe Logan Newell afleverer en opgave sammen, skal filen hedde `5G-BG-LT-SJ-GLN.sml`. Brug gruppeafleveringsfunktionen i Absalon.

Gruppeopgaven giver op til 2 point, som tæller til de 20 point, der kræves for eksamensdeltagelse. Genaflevering kan hæve pointtallet fra første aflevering med højest 1 point, så sørg for at gøre jeres bedste allerede i første aflevering.

5G1 Skriv en funktion `toRGB : primaryColours -> int * int * int`, der oversætter en farve af typen `primaryColours` til et RGB tripel (sådan som InstragraML repræsenterer farver). For eksempel skal `toRGB Red` returnere `(255, 0, 0)` og `toRGB Yellow` skal returnere `(255, 255, 0)`, da gul dannes ved blanding af rød og grøn.

Brug RGB-definitionerne fra `InstagraML.sml`, hvis du er i tvivl. For eksempel har `RGB.red` værdien `(255, 0, 0)`.

5G2 Skriv en funktion `reColour : 'a figure -> ('a -> 'b) -> 'b figure`, der omfarver et billede ved at anvende en farvetransformation på alle elementer i billedet. Bemærk, at den returnerede figur kan bruge en anden farvetype end den oprindelige. Kaldet `reColour redCircleOverBlueSquare toRGB` skal altså returnere værdien

```
Over (Circle ((255,0,0), (0.0, 0.0), 1.8),
      Rectangle ((0,0,255), (~1.5, ~1.5), (1.5, 1.5)))
```

5G3 Skriv en funktion `colourOf : 'a figure -> point -> 'a option`, der finder farven af et punkt i en figur.

Hvis figuren er en cirkel med farve c, gives `SOME` c, hvis punktet ligger i cirklen og `NONE`, hvis punktet ligger udenfor.

Hvis figuren er en rektangel med farve c, gives `SOME` c, hvis punktet ligger i rektanglen og `NONE`, hvis punktet ligger udenfor.

Hvis figuren er sammensat af to underfigurer (med konstruktoren `Over`) og punktet ligger i første underfigur (altså hvis farven af punktet i denne underfigur er forskellig fra `NONE`), returneres punktets farve i denne underfigur. Ellers returneres farven i den anden underfigur (hvilket kan være `NONE`).

For eksempel skal kaldet `colourOf redCircleOverBlueSquare (0.0, 0.0)` returnere `SOME Red` og kaldet `colourOf redCircleOverBlueSquare (3.0, 0.0)` skal returnere `NONE`.

Bemærk, at `colourOf` er en generalisering af funktionen `erI`, der blev introduceret ved mandagsforelæsningen.

5G4 Brug `colourOf` til at definere en funktion `hasAColour : 'a figure -> point -> bool`, der returnerer `true`, hvis punktet har en farve forskellig fra `NONE` i figuren.

For eksempel skal kaldet `hasAColour redCircleOverBlueSquare (0.0, 0.0)` returnere `true` og kaldet `hasAColour redCircleOverBlueSquare (3.0, 0.0)` skal returnere `false`.

5G5 Omskriv `colourOf`, så den ved behandling af `Over` kalder `hasAColour` for at se, om et punkt har en farve i første underfigur. Dermed bliver `colourOf` og `hasAColour` gensidigt rekursive, så brug nøgleordet `and` til at sikre dette.

5G6 Skriv en funktion

```
toInstagraML : InstagraML.colour figure * point * int * int * real
            -> InstagraML.image
```

der tager en figur f (parameteriseret med `InstagraML`s farvetype `(int * int * int)`), et punkt p , en bredde b, en højde h og en skaleringsfaktor s og returnerer et `InstagraML` billede. Baggrundsfarven (der, hvor figuren ikke dækker) skal være hvid.

Det genererede billede skal have bredde og højde (b, h) i pixels og punktet p skal svare til nederste venstre hjørne i billedet (med pixelkoordinaterne $(0, 0)$). Skaleringsfaktoren s er størrelsen af en pixel målt i den skala, som figuren bruger.

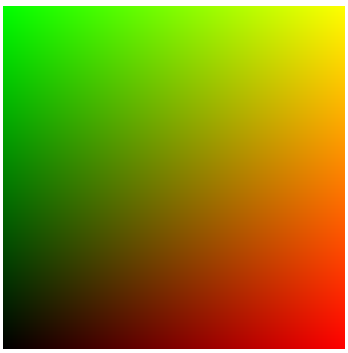
Brug de i afsnit 5 viste eksempler til at afprøve din funktion.

Vink: Til denne opgave kan du med fordel bruge funktionen

```
InstagraML.fromFunction
        : int * int * (int * int -> int * int * int) -> image
```

der konstruerer et billede ud fra en funktion.

I kaldet `InstagraML.fromFunction`$(w, h\, f)$ er w bredden af det ønskede billede, h er højden af billedet og f er en funktion fra koordinater i billedet til farver. For eksempel vil kaldet `InstagraML.fromFunction (256, 256, fn (x,y) => (x,y,0))` lave et billede, der er en glidende overgang mellem hjørner, der er sorte, røde, grønne og gule, som vist på billedet herunder.



7 Individuel aflevering

Den individuelle opgave er obligatorisk. Alle delspørgsmål skal besvares. Opgaven afleveres i Absalon som en fil med navnet `5I-`*`navn`*`.sml`, hvor *navn* er erstatttet med dit navn. Hvis du fx hedder Anders A. And, skal filnavnet være `5I-Anders-A-And.sml`. Skriv også dit fulde navn som en kommentar i starten af filen.

Den individielle opgave giver op til 3 point, som tæller til de 20 point, der kræves for eksamensdeltagelse. Genaflevering kan hæve pointtallet fra første aflevering med højest 1 point, så sørg for at gøre dit bedste allerede i første aflevering.

5I1 Skriv en funktion `reorder : 'a figure -> 'a figure`, der bytter om på rækkefølgen af elementerne i figuren, så de element, der i argumentet er øverst, i resultatet er nederst og så videre.

Kaldet `reorder redCircleOverBlueSquare` skal altså returnere figuren
`Over (Rectangle (Blue, (~1.5, ~1.5), (1.5, 1.5)), Circle (Red, (0.0, 0.0), 1.8))`.

5I2 Skriv en funktion `scale : 'a figure -> real -> 'a figure`, der skalerer en figur. Kaldet `scale` $f\ s$ skalerer alle koordinater og radier i figuren f ved at gange med s og returnerer den skalerede figur.

For eksempel skal kaldet `scale redCircleOverBlueSquare 2.0` returnere
`Over (Circle (Red, (0.0, 0.0), 3.6), Rectangle (Blue, (~3.0, ~3.0), (3.0, 3.0)))`.

5I3 Skriv en funktion `move : 'a figure -> real * real -> 'a figure`, der flytter en figur. Kaldet `move` $f\ (x, y)$ lægger (x, y) til alle koordinater i figuren f og returnerer den flyttede figur.

For eksempel skal kaldet `move redCircleOverBlueSquare (1.0, 2.0)` returnere
`Over (Circle (Red, (1.0, 2.0), 1.8), Rectangle (Blue, (~0.5, 0.5), (2.5, 3.5)))`.

5I4 Skriv en funktion `toList : 'a figure -> 'a figure list`, der givet en figur returnerer en liste af de cirkler og rektangler, der er i figuren, i rækkefølge fra øverst til nederst.

Kaldet `toList redCircleOverBlueSquare` skal altså returnere listen
`[Circle (Red, (0.0, 0.0), 1.8), Rectangle (Blue, (~1.5, ~1.5), (1.5, 1.5))]`.

5I5 Skriv en funktion `fromList : 'a figure list -> 'a figure`, der ud fra en liste af figurer konstruerer en figur, hvor figurerne fra listen er ordnet, så den første figur ligger over den anden osv. Hvis listen er tom, skal undtagelsen `Empty` rejses.

Kaldet
`fromList [Circle (Red, (0.0, 0.0), 1.8), Rectangle (Blue, (~1.5, ~1.5), (1.5, 1.5))]`
skal altså returnere en figur, der er identisk med `redCircleOverBlueSquare`.

5I6 Skriv funktionen `fromList` uden eksplicit rekursion ved at bruge `List.foldl` eller `List.foldr`.

8 Ugens nød

Denne uges nød bygger videre på mandagsopgaverne og handler derfor om udvælgelsesprocesser af typen `selection`. Vi starter med at generalisere typen ved at parameterisere med typen af de værdier, der kan udvælges:

```
datatype 'a selection = Pick of 'a
                      | Choose of real * selection * selection
```

5N1 Skriv en funktion `product : 'a selection * 'b selection -> ('a * 'b) selection`, der givet en udvælgelsesproces f for typen `'a` og en udvælgelsesproces g for typen `'b` returnerer en udvælgelsesproces $f \times g$ for typen `'a * 'b`, sådan at $f \times g$ udvælger et par (a, b) med sandsynlighed pq netop hvis f udvælger a med sandsynlighed p og g udvælger b med sandsynlighed q.

For eksempel skal kaldet `product (fairDie 2, fairDie 2)` returnere en udvælgelsesproces, der returnerer parrene (1,1), (1,2), (2,1) og (2,1) hver med sandsynlighed 0.25.

5N2 Skriv en funktion `probabilityThat : 'a selection -> ('a -> bool) -> real`, der givet en udvælgelsesproces f og et prædikat P returnerer sandsynligheden for, at P er opfyldt for et valg foretaget af f.

For eksempel skal kaldet `probabilityThat (fairDie 10) (fn n => n mod 3 <> 0)` returnere 0.7, da 7 ud af de 10 lige sandsynlige udfald ikke er delelige med 3.

5N3 Skriv en funktion `conditional : 'a selection -> ('a -> bool) -> 'a selection`, der givet en udvælgelsesproces f og et prædikat P returnerer en udvælgelsesproces, der kun kan returnere elementer a fra `'a`, hvor $P(a)$ er sand, og hvor sandsynligheden for at vælge a er lig med $\frac{p}{q}$, hvor p er sandsynligheden for at f udvælger a og q er sandsynligheden for at f udvælger et element b, hvor $P(b)$ er sand. Med andre ord er q lig med `probabilityThat` $f\ P$.

For eksempel skal kaldet `conditional (fairDie 10) (fn n => n mod 3 <> 0)` returnere en udvælgelsesproces f, som returnerer tallene 1, 2, 4, 5, 7, 8 og 10 med hver sandsynlighed $\frac{0.1}{0.7} = \frac{1}{7}$.