Opdracht 1: Functionele figuren

INTRODUCTIE

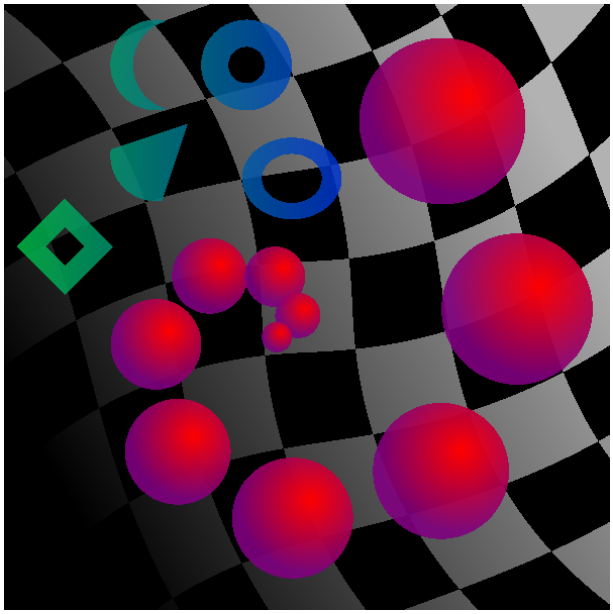
Dit is de eerste practicumopdracht van de cursus Functioneel programmeren. In dit practicum worden plaatjes gegenereerd van figuren die zijn beschreven als een (wiskundige) functie. In de figuren wordt onder andere gebruik gemaakt van rotaties, transparantie, clipping en gradiënten.

LEERDOELEN
Na het maken van deze opdracht, wordt verwacht dat u
- hogere-orde functies kunt definiëren en deze functies kunt gebruiken
- figuren kunt beschrijven als functies en hiervan plaatjes kunt genereren
- kunt werken met tupels en lijsten
- functies met een polymorf type kunt definiëren
- kunt werken met typesynoniemen.

Studeeraanwijzing
Deze opdracht is te maken na het bestuderen van de leereenheden 1 tot en met 7. In de opdracht worden typesynoniemen gebruikt om de types leesbaar te houden. Typesynoniemen worden eigenlijk pas in latere leereenheden behandeld: er wordt zoveel toelichting gegeven als nodig is voor het maken van de opgaven. Alle functies die input/output (*IO*) verzorgen zijn gegeven.
Een startversie van de Haskell module `Figuur.hs` is gegeven waarin een aantal functies nog gedefinieerd moet worden (herkenbaar aan de waarde *undefined*). Vrijwel alle gevraagde functies kunnen als een 'one-liner' worden opgeschreven. Ga dus niet op zoek naar ingewikkelde oplossingen. Mocht je vast komen te zitten, stel dan je vraag via yOUlearn of in een e-mail aan de begeleider. Deze opdracht heeft een studielast van ongeveer 12 uur.

Software
De opdracht is te maken met met GHC. Voor de opdracht zijn geen extra bibliotheken nodig. Vanaf onderdeel 7 ('PPM rendering') worden plaatjes gegenereerd in het PPM bitmap formaat. Dit formaat is gekozen vanwege zijn eenvoud. Er zijn weinig applicaties die dit formaat bitmap direct kunnen tonen, maar er bestaan vele gratis tools die dit formaat kunnen converteren naar bekende formaten zoals PNG, GIF en JPEG. Een goed conversietool is ImageMagick, te downloaden via `http://www.imagemagick.org` en beschikbaar voor allerlei besturingssystemen. ImageMagick kan vanuit de command-line worden aangeroepen met het commando `convert test.ppm test.png`.

Inleveren
Lever uw uitwerking in via yOUlearn. Vermeld in uw uitwerking duidelijk uw naam, uw studentnummer en het versienummer van de opdracht. Uitwerkingen worden beoordeeld als voldoende of onvoldoende. Bij een onvoldoende wordt u door de examinator geïnformeerd over de aanvullende inspanning die nodig is om het resultaat te compenseren.



FIGUUR 1.1 Voorbeeld van een functioneel figuur

Opdracht 1: Functionele figuren

In deze practicumopdracht gaan we figuren beschrijven met functies. Een voorbeeld van een figuur dat aan het einde van deze opdracht kan worden gemaakt is te zien in Figuur 1.1. Stapsgewijs worden de verschillende technieken uitgewerkt die zijn gebruikt in dit figuur. Er wordt geen gebruik gemaakt van een externe bibliotheek voor het tekenen van dergelijke figuren.
Er zijn verschillende mogelijkheden om een figuur te representeren zoals bijvoorbeeld een bitmap. Niet alle operaties zijn even gemakkelijk uit te voeren op een bitmap (denk bijvoorbeeld aan rotaties en het tekenen van cirkels). De representatie die voor deze opdracht wordt gebruikt is een functie die voor iedere positie een waarde oplevert. Deze waarde kan bijvoorbeeld een kleur zijn.

$$\textbf{type}\ \mathit{Figuur}\ a = \mathit{Pos} \to a$$

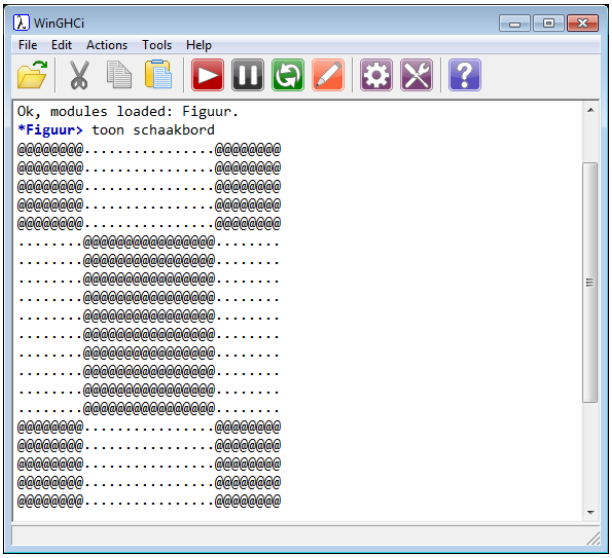
Typedeclaraties (of typesynoniemen) worden pas in Leereenheid 8 (en 10) van het werkboek uitgelegd. Ze helpen om ingewikkelde types leesbaar te houden. We zullen extra toelichting geven bij deze declaraties waar dat nodig is voor deze opdracht. Een positie bestaat uit een tupel met een x-waarde en een y-waarde:

$$\textbf{type}\ \mathit{Pos} = (\mathit{Double}, \mathit{Double}) \quad \text{-- (x, y)}$$

Voor posities gebruiken we *double-precision floating-point* getallen. Een eigenaardigheid van de gekozen representatie voor figuren is dat figuren geen rand hebben: ze lopen als het ware door tot in de oneindigheid. Als conventie hanteren we echter dat we het gebied tussen de posities $(-1,-1)$ (linksonder) en $(1,1)$ (rechtsboven) gaan visualiseren.
Als voorbeeld geven we de definitie van een schaakbord als figuur. De hokjes van dit bord krijgen afwisselend de booleanwaarden *True* en *False*. Het type van *schaakbord* is dus *Figuur Bool*[1]. Een waarde van het type *Figuur Bool* kan worden beschouwd als een zwart-wit plaatje.

$$\mathit{schaakbord} :: \mathit{Figuur\ Bool}$$
$$\mathit{schaakbord}\ (x,y) = \mathit{even}\ (\mathit{round}\ x) == \mathit{even}\ (\mathit{round}\ y)$$

[1] Door de typesynoniemen uit te schrijven is te zien dat het type *Figuur Bool* gelijk is aan het functietype $(\mathit{Double}, \mathit{Double}) \to \mathit{Bool}$.



FIGUUR 1.2 Het tonen van *schaakbord* in WinGHCi

Op dit moment kan een figuur nog niet worden bekeken. De eerstvolgende stap is het schrijven van een simpele rendering van figuren. Het resultaat daarvan is te zien in Figuur 1.2. Omdat een figuur eigenlijk een functie is kunnen we al wel handmatig de waarden van enkele posities inspecteren:

```
Figuur> schaakbord (0,0)
True

Figuur> schaakbord (1,0)
False

Figuur> schaakbord (1,-1)
True
```

1 ASCII rendering

Om een figuur te tonen moet deze eerst worden afgebeeld op een tweedimensionaal grid, gerepresenteerd als een lijst van lijsten. De dimensie van het grid kan vrij worden gekozen:

type $Dimensie = (Int, Int)$ -- (breedte, hoogte)

Voor de ASCII rendering gebruiken we als breedte 32 en als hoogte 20.

a Schrijf een functie die alle coördinaten van een dimensie per rij oplevert:

$coordinaten :: Dimensie \rightarrow [[(Int, Int)]]$

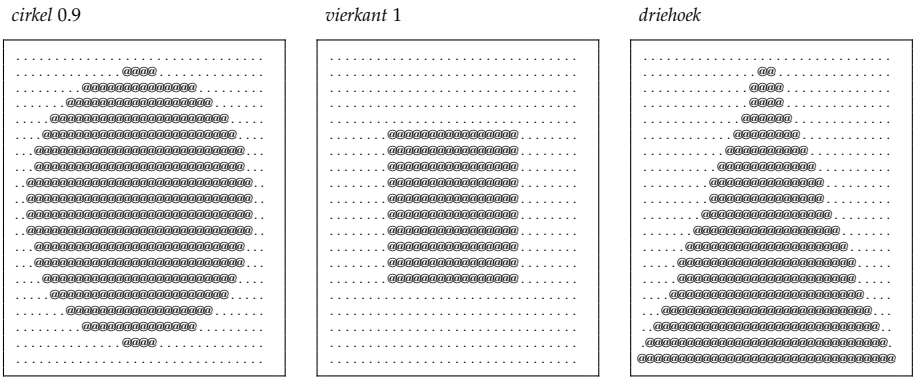
Controleer je definitie met de volgende aanroep:

```
Figuur> coordinaten (3,2)
[[(0,0),(1,0),(2,0)],[(0,1),(1,1),(2,1)]]
```

b Definieer de functie *render* om een figuur af te beelden op een grid met een bepaalde afmeting.

$render :: Dimensie \rightarrow Figuur\ a \rightarrow [[a]]$

Als conventie wordt het gebied tussen de posities $(-1,-1)$ (linksonder) en $(1,1)$ (rechtsboven) gebruikt: dit is dus onafhankelijk van de gekozen dimensie. Positie $(0,0)$ is dus het midden, positie $(1,0)$ midden rechts en positie $(1,-1)$ rechtson-



FIGUUR 1.3 Enkele Basisvormen

der. Maak gebruik van de volgende hulpfunctie om coördinaten te vertalen naar posities binnen het te tonen gebied.

$$
\begin{aligned}
&\mathit{naarPositie} :: \mathit{Dimensie} \to (\mathit{Int}, \mathit{Int}) \to \mathit{Pos} \\
&\mathit{naarPositie}\ d\ (x, y) = (\mathit{fromIntegral}\ x * 2\ /\ b - 1, 1 - \mathit{fromIntegral}\ y * 2\ /\ h) \\
&\quad \textbf{where} \\
&\qquad b = \mathit{fromIntegral}\ (\mathit{fst}\ d - 1) \\
&\qquad h = \mathit{fromIntegral}\ (\mathit{snd}\ d - 1)
\end{aligned}
$$

Controleer je definitie met de volgende aanroep:

```
Figuur> render (3,3) schaakbord
[[True,False,True],[False,True,False],[True,False,True]]
```

c Schrijf een functie die voor een booleanwaarde een bepaald karakter oplevert. In de voorbeelden worden de karakters `'@'` en `'.'` gebruikt.

$$\mathit{boolChar} :: \mathit{Bool} \to \mathit{Char}$$

d Schrijf een functie om de waarden van een figuur te veranderen:

$$\mathit{verander} :: (a \to b) \to \mathit{Figuur}\ a \to \mathit{Figuur}\ b$$

Aanwijzing: Uitschrijven geeft $\mathit{verander} :: (a \to b) \to (\mathit{Pos} \to a) \to \mathit{Pos} \to b$.

De functie om een figuur te tonen is gegeven. Controleer of het resultaat van de expressie *toon schaakbord* overeenkomt met het screenshot in Figuur 1.2.

$$
\begin{aligned}
&\mathit{toon} :: \mathit{Figuur\ Bool} \to \mathit{IO}\ () \\
&\mathit{toon} = \mathit{putStrLn} \circ \mathit{unlines} \circ \mathit{render}\ (32, 20) \circ \mathit{verander\ boolChar}
\end{aligned}
$$

2 Basisvormen

Schrijf functies die enkele basisvormen opleveren: een cirkel met een gegeven straal, een vierkant met een gegeven lengte, en een gelijkbenige driehoek. Alle figuren zijn gedefinieerd ten opzichte van de oorsprong.

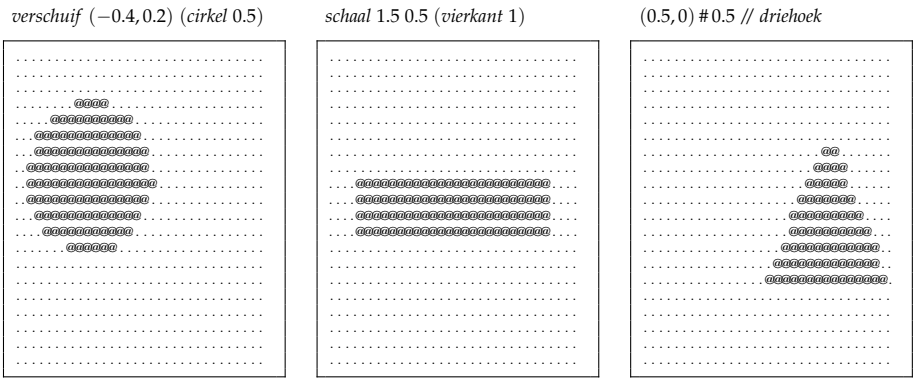
$$\mathit{cirkel} :: \mathit{Double} \to \mathit{Figuur\ Bool}$$

$$\mathit{vierkant} :: \mathit{Double} \to \mathit{Figuur\ Bool}$$

$$\mathit{driehoek} :: \mathit{Figuur\ Bool}$$

Het resultaat van deze vormen is te zien in Figuur 1.3.

Aanwijzing: Uitschrijven geeft $\mathit{cirkel} :: \mathit{Double} \to \mathit{Pos} \to \mathit{Bool}$. Voor een positie (x, y) in de tweede parameter moet worden bepaald of deze wel of niet in de cirkel valt.



FIGUUR 1.4 Transformaties op figuren

3 Transformaties

Omdat figuren functies zijn is het vrij eenvoudig om een aantal transformaties te definiëren zoals het verschuiven (transleren) en schalen van een figuur.

a Schrijf eerst een hogere-orde functie om een transformatie (een functie van *Pos* naar *Pos*) toe te passen op een figuur.

$$transform :: (Pos \to Pos) \to Figuur\ a \to Figuur\ a$$

Aanwijzing: Door de typesynoniem *Figuur* uit te schrijven is te zien dat *transform* het type $(Pos \to Pos) \to (Pos \to a) \to Pos \to a$ heeft.

b Definieer nu functies voor het verschuiven – gegeven een $(\Delta x, \Delta y)$ – en het schalen – gegeven een horizontale en vertikale factor – van een figuur. Maak gebruik van *transform*.

$$verschuif :: (Double, Double) \to Figuur\ a \to Figuur\ a$$
$$schaal :: Double \to Double \to Figuur\ a \to Figuur\ a$$

c Aangezien deze transformaties veel zullen worden gebruikt introduceren we ook twee operatoren om de transformaties bondig op te kunnen schrijven. Geef de definitie van deze operatoren door *verschuif* en *schaal* te hergebruiken. Merk op dat de operator // in beide richtingen met dezelfde factor schaalt.

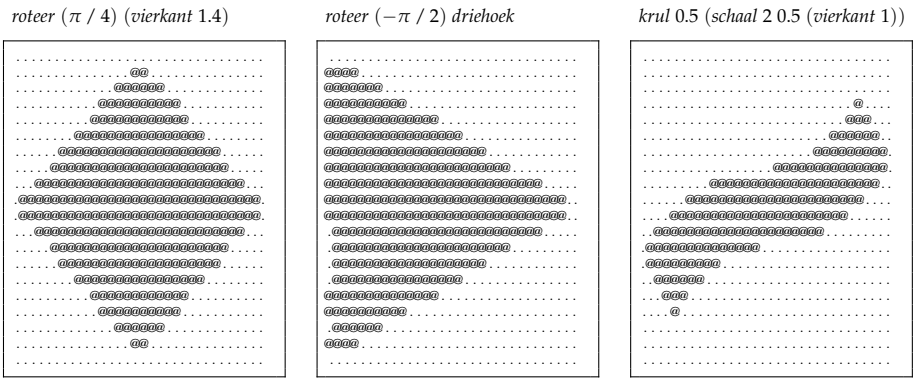
```
infixr 7 #
infixr 7 //
(#) :: (Double, Double) → Figuur a → Figuur a    -- verschuiven
(//) :: Double → Figuur a → Figuur a    -- schalen
```

De resultaten van de transformaties zijn te zien in Figuur 1.4.

4 Transformaties met poolcoördinaten

Andere transformaties laten zich makkelijker beschrijven als we gebruik maken van poolcoördinaten in plaats van (x, y)-coördinaten. Een poolcoördinaat bestaat uit de afstand van een punt tot de oorsprong en de hoek ten opzichte van de oorsprong. Functies om 'gewone' coördinaten te converteren van en naar poolcoördinaten zijn gegeven.

```
type Polar = (Double, Double)    -- (afstand, hoek)
toPolar :: Pos → Polar
toPolar (x, y) = (sqrt (x ^ 2 + y ^ 2), hoek)
  where hoek | x == 0    = π / 2 * signum y
             | x < 0     = atan (y / x) + π
             | otherwise = atan (y / x)
```



FIGUUR 1.5 Transformaties met poolcoördinaten

$$fromPolar :: Polar \to Pos$$
$$fromPolar\ (r,h) = (cos\ h * r, sin\ h * r)$$

a Defineer de functie *transPolar* die een transformatie op poolcoördinaten toepast op een figuur.

$$transPolar :: (Polar \to Polar) \to Figuur\ a \to Figuur\ a$$

Aanwijzing: Hergebruik de functie *transform*.

b Definieer rotatie met een gegeven hoek en de *krul* transformatie die het poolcoördinaat (r,a) afbeeldt op $(r, a - d * r)$, gegeven de parameter d.

$$roteer :: Double \to Figuur\ a \to Figuur\ a$$
$$krul :: Double \to Figuur\ a \to Figuur\ a$$

De resultaten van de rotaties zijn te zien in Figuur 1.5.

5 Composities

Figuren worden pas echt interessant als ze kunnen worden samengesteld tot complexere figuren. Om twee figuren te kunnen samenstellen moeten we weten hoe de waarden van beide figuren moeten worden gecombineerd.

a Schrijf de functie *compositie* om twee figuren samen te stellen. De types van deze figuren en van het resultaat mogen verschillend zijn.

$$compositie :: (a \to b \to c) \to Figuur\ a \to Figuur\ b \to Figuur\ c$$

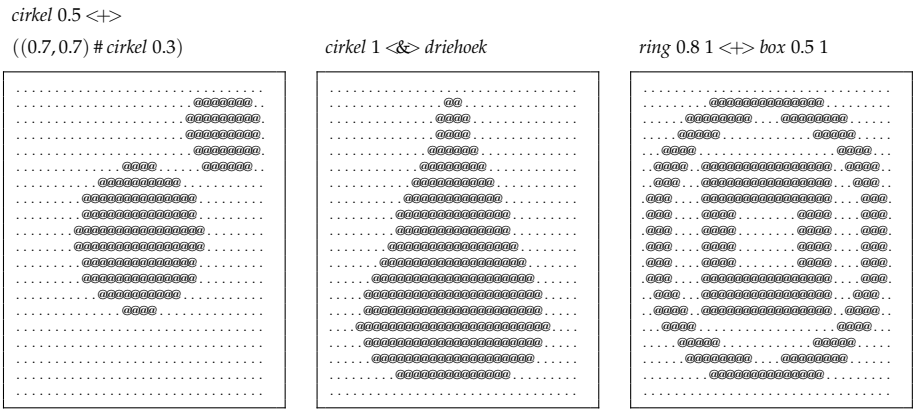
Aanwijzing: De functie van het type $a \to b \to c$ wordt gebruikt om elk punt van *Figuur a* en *Figuur b* te combineren tot een punt van *Figuur c*. Door het typesynoniem *Figuur* uit te schrijven zien we dat *compositie* het type $(a \to b \to c) \to (Pos \to a) \to (Pos \to b) \to Pos \to c$ heeft. De definitie bij dit polymorfe type is maar op één manier te schrijven.

b Definieer nu drie compositie operatoren door verschillende functies van het type $Bool \to Bool \to Bool$ mee te geven aan *compositie*.

$$(<+>) :: Figuur\ Bool \to Figuur\ Bool \to Figuur\ Bool$$
$$(<\&>) :: Figuur\ Bool \to Figuur\ Bool \to Figuur\ Bool$$
$$(<->) :: Figuur\ Bool \to Figuur\ Bool \to Figuur\ Bool$$

De betekenis van deze operatoren is:
- figuur p <+> q bekijkt of p of q waar is (voor een gegeven positie)
- figuur p <&> q bekijkt of zowel p als q waar is
- figuur p <-> q bekijkt of p waar is en q onwaar (clipping).

Zie ook de voorbeelden in Figuur 1.6.



FIGUUR 1.6 Verschillende composities

c Voor twee bijzondere clippings introduceren we een eigen definitie. Een ring bestaat uit een cirkel met daarin een cirkel weggelaten. Eerst wordt de straal van de binnenste cirkel meegegeven, daarna de straal van de buitenste. De functie *box* werkt met vierkanten in plaats van cirkels. Geef de definities van *ring* en *box*.

$$ring :: Double \to Double \to Figuur\ Bool$$
$$box :: Double \to Double \to Figuur\ Bool$$

6 Kleuren

De volgende stap is om een type te introduceren voor kleuren. Omdat we transparantie willen nabootsen in figuren gebruiken we een codering met vier waarden: een roodwaarde, een groenwaarde, een blauwwaarde en een alpha-waarde. Deze waarden liggen altijd tussen 0 en 1. De alpha-waarde 0 correspondeert met een volledig doorzichtige kleur. De alpha-waarde 1 hoort bij een ondoorzichtige kleur. De volgende definities zijn gegeven:

$$\textbf{type}\ Kleur = (Double, Double, Double, Double) \quad \text{-- (rood, groen, blauw, alpha)}$$
$$alpha :: Kleur \to Double$$
$$alpha\ (\_, \_, \_, a) = a$$
$$rood, groen, blauw, zwart, wit, leeg :: Kleur$$
$$\begin{aligned} rood &= (1,0,0,1)\\ groen &= (0,1,0,1)\\ blauw &= (0,0,1,1)\\ zwart &= (0,0,0,1)\\ wit &= (1,1,1,1)\\ leeg &= (0,0,0,0) \quad \text{-- volledig doorzichtig} \end{aligned}$$

a Schrijf een functie om een kleur te veranderen door een gegeven functie toe te passen op alle vier de waarden van een kleur:

$$veranderKleur :: (Double \to Double) \to Kleur \to Kleur$$

b Definieer een functie om de transparantie aan te passen. Doe dit door *iedere* kleurwaarde te vermenigvuldigen met een meegegeven factor [2]. Gebruik *veranderKleur*.

$$transparant :: Double \to Kleur \to Kleur$$

Controleer je definitie met de volgende aanroep:

```
Figuur> transparant 0.3 rood
(0.3,0.0,0.0,0.3)
```

[2] Om technische redenen ('premultiplied alpha') moeten ook de RGB waarden worden aangepast.

c Schrijf een functie die twee kleuren waardegewijs combineert:

$$zipKleur :: (Double \to Double \to Double) \to Kleur \to Kleur \to Kleur$$

Bijvoorbeeld:

```
Figuur> zipKleur max rood blauw
(1.0,0.0,1.0,1.0)
```

d Definieer een functie die twee kleuren mixt volgens een gegeven verhouding. Bijvoorbeeld, *mixKleur* 0.3 *rood blauw* betekent 30% rood en 70% blauw. Ook de alpha-waarde wordt gemixt. Gebruikt *zipKleur*.

$$mixKleur :: Double \to Kleur \to Kleur \to Kleur$$

Een voorbeeldaanroep is:

```
Figuur> mixKleur 0.3 rood blauw
(0.3,0.0,0.7,1.0)
```

7 PPM rendering

Voor figuren met kleuren of met een hogere resolutie volstaat het niet langer om een rendering naar ASCII karakters te gebruiken. We gaan daarom bitmaps genereren in het simpele maar inefficiënte PPM formaat. Er bestaan meerdere gratis tools die dit formaat kunnen converteren naar meer gangbare formaten zoals PNG, GIF en JPEG. Een tool dat hiervoor kan worden gebruikt is ImageMagick[3].

a De eerste regel van een PPM bestand beschrijf achtereenvolgens het type (altijd P6), de breedte, de hoogte en de maximale kleurwaarde (gebruik 255). De regel wordt afgesloten met een newline karakter. Definieer de functie:

$$headerPPM :: Dimensie \to String$$

Controleer je definitie met de volgende aanroep:

```
Figuur> headerPPM (32,20)
"P6 32 20 255\n"
```

b Schrijf vervolgens de functie die een kleur afbeeldt op een string met drie karakters. De alpha-waarde doet niet mee. De rood-, groen- en blauw-waarden liggen tussen 0 en 1 en moeten eerst worden geschaald naar maximaal 255. Gebruik de standaardfunctie *chr* om een *Int* te converteren naar een *Char*.

$$kleurPPM :: Kleur \to String$$

Controleer je definitie met de volgende aanroep:

```
Figuur> kleurPPM (0.4,0.3,0.2,0)
"fL3"
```

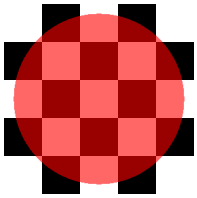
c Definieer tenslotte de functie die de inhoud van een PPM bestand oplevert. Na de header volgen de gecodeerde kleuren. Er is geen scheidingsteken tussen de kleuren of tussen de rijen.

$$maakPPM :: Dimensie \to Figuur\ Kleur \to String$$

De volgende functies zijn nodig om een string weg te schrijven naar een bestand:

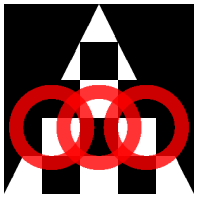
```
schrijf :: FilePath → Figuur Kleur → IO ()
schrijf file = writeBinaryFile file ∘ maakPPM (300,300)

writeBinaryFile :: FilePath → String → IO ()
writeBinaryFile file s = do
  h ← openBinaryFile file WriteMode
  hPutStr h s
  hClose h
```

[3] http://www.imagemagick.org/



$$kleur\ (transparant\ 0.6\ rood)\ (cirkel\ 0.9)$$
$$<>\ kleur\ wit\ (0.4\ //\ schaakbord)$$

$$kleur\ (transparant\ 0.6\ (mixKleur\ 0.5\ blauw\ wit))$$
$$((0.4, 0.4)\ \#\ vierkant\ 1)$$
$$<>\ kleur\ rood\ (cirkel\ 0.5)$$



$$stapel\ \ [\ kleur\ (transparant\ 0.8\ rood)$$
$$((x\ /\ 2, -0.3)\ \#\ ring\ 0.3\ 0.45)$$
$$|\ x \leftarrow [-1\,..\,1]]$$
$$<>\ kleur\ wit\ ((0.4\ //\ schaakbord)\ <\&>\ driehoek)$$

FIGUUR 1.7 Composities van gekleurde figuren met transparante delen

In de functie *schrijf* wordt de dimensie $(300, 300)$ meegegeven. Deze dimensie kan worden aangepast afhankelijk van de tijd die het kost om een PPM bestand te genereren. Het enige dat nog nodig is om het wegschrijven uit te proberen is een functie om een *Figuur Bool* in te kleuren:

$$kleur :: Kleur \to Figuur\ Bool \to Figuur\ Kleur$$
$$kleur = kleurMet \circ const$$

$$kleurMet :: Figuur\ Kleur \to Figuur\ Bool \to Figuur\ Kleur$$
$$kleurMet = compositie\ (\lambda k\ b \to \textbf{if}\ b\ \textbf{then}\ k\ \textbf{else}\ leeg)$$

Probeer de functies uit met de volgende actie:

```
Figuur> schrijf "d:\\test.ppm" (kleur rood driehoek)
```

Wie ImageMagick gebruikt moet in een terminal het volgende commando nog uitvoeren om het resultaat te kunnen bekijken als een PNG bestand:

```
convert d:\test.ppm d:\test.png
```

Aanwijzing: De bestanden test.ppm en test.png zijn als voorbeeld meegeleverd bij deze opdracht. Gebruik deze bestanden eventueel om te achterhalen wat er fout gaat.

8 Kleuren composities

Nu we figuren met kleuren kunnen maken hebben we ook een operator nodig om deze samen te stellen. De compositie operatoren die tot nog toe zijn geschreven (te weten <+>, <&> en <–>) werken namelijk alleen op figuren van het type *Figuur Bool*. We gaan één operator schrijven om een gekleurd figuur over een ander gekleurd figuur te leggen, rekening houdend met de doorzichtigheid in het bovenste figuur. De formule om twee kleuren waardegewijs te combineren is als volgt:

$$k = k_0 + k_1 \cdot (1 - \alpha_0)$$

waarbij α_0 de alpha-waarde is in het bovenste figuur, k_0 de kleurwaarde in het bovenste figuur, k_1 de kleurwaarde in het onderste figuur en k het resultaat.

FIGUUR 1.8 Horizontale en cirkelvormige gradiënten

a Definieer de functie die twee kleuren combineert volgens de bovenstaande formule:

$$over :: Kleur \to Kleur \to Kleur$$

Aanwijzing: De formule moet dus vier keer worden gebruikt, oftewel voor elke kleurwaarde. De implementatie kan worden gecontroleerd met:

```
Figuur> over (transparant 0.4 rood) blauw
(0.4,0.0,0.6,1.0)
```

b Schrijf nu de operator die figuren samenstelt met de functie *over*:

$$(<>) :: Figuur\ Kleur \to Figuur\ Kleur \to Figuur\ Kleur$$

c Definieer tenslotte een functie die een lijst van figuren samenstelt met de <>-operator:

$$stapel :: [Figuur\ Kleur] \to Figuur\ Kleur$$

Aanwijzing: Deze functie is te schrijven met *foldr*: denk goed na over het basisgeval voor de lege lijst.

Enkele composities met gekleurde figuren zijn te zien in Figuur 1.7.

9 Gradiënten

Figuren kunnen aantrekkelijker worden gemaakt door gradiënten te gebruiken. Er is sprake van een gradiënt als in een figuur een kleur langzaam overgaat in een andere kleur. In deze opdracht bekijken we horizontale gradiënten waarbij naar de x-positie wordt gekeken en cirkelvormige gradiënten waarbij naar de afstand tot de oorsprong wordt gekeken.

a Definieer eerst een horizontale gradiënt:

$$gradient :: Kleur \to Kleur \to Figuur\ Kleur$$

Links van de x-waarde -1 is enkel de eerste kleur te zien: rechts van de x-waarde 1 is enkel de tweede kleur te zien. Daartussenin is een lineaire overgang tussen de twee kleuren.

b Definieer nu een cirkelvormige gradiënt:

$$gradientCirkel :: Kleur \to Kleur \to Figuur\ Kleur$$

Punten verder dan afstand 1 van de oorsprong krijgen de eerste kleur: de oorsprong krijgt de tweede kleur (met weer een lineaire overgang tussen de kleuren). *Aanwijzing:* Gebruik poolcoördinaten.

10 Voorbeelden

Aan het einde van het Haskell startbestand zijn een aantal voorbeeldfiguren voorgedefinieerd, waaronder het figuur dat te zien is in Figuur 1.1 met als naam *eindvb*. Probeer deze figuren zelf eens te genereren en rapporteer eventuele verschillen.