Shapes

— ett hyfsat funktionellt ritspråk

Henrik Tidefelt

LiTH

11 september 2007

Mål

Med den här UppLYSningen hoppas jag

- Att ni ska få ett hum om vad Shapes är.
- Få höra era invändningar mot designen som den ser ut idag.
- Lyckas hitta någon testpilot.
- Väcka intresse för utvecklingssamarbete.

Plan

De stora inslagen idag är:

- Beskriva hur språkets struktur ser ut idag.
- Visa lite av de funktioner som kärnan erbjuder.
- Diskutera intressanta utmaningar för framtiden.

Introduktion

Historia

- Hösten 2003: Första kontakt med MetaPost.
- Sommaren 2004: Toolbox för plottning i Matlab tar form.
- Hösten 2004: Börjar undersöka möjligheten att ersätta MetaPost.
- September 2005: Shapes, då kallat *MetaPDF*, versionshanteras.
- Januari 2007: Shapes, då kallat *Drool*, har använts till stort antal figurer i en bok.
- April 2007: Kontrollerade tillstånd.
- September 2007: Språket heter *Shapes*, och presenteras för första gången.

Rötter

Shapes har sina rötter i många av de språk jag varit i kontakt med:

- MetaPost (en omarbetning av Knuths MetaFont) — Shapes kom till när jag ledsnade på MetaPost.
- Scheme — syntax och funktions-begreppet.
- Haskell — för sina rena ideal.
- C++ — utmatningssyntaxen.

Alternativ

Några andra ritspråk som finns och/eller används idag:

- MetaPost
- Asymptote
- PGF och TikZ
- Haskell PDF

Varför Shapes?

Givet utbudet av alternativa rit-språk, varför utveckla ett till? Här är några skäl:

- Inte funktionellt orienterade (alla utom Haskell PDF).
- Dålig beräkningskapacitet (MetaPost och PDF/TikZ).
- Saknar domän-specifik syntax (Haskell PDF).
- Inte publicerade när Shapes påbörjades (Asymptote och Haskell PDF).

Hello, shaper!

```
•page << [stroke (0cm,0cm)--(1cm,1cm)]

•page << stroke [] ((0cm,0cm)--(1cm,1cm))

[(\ •dst pth .> { •dst << stroke [] pth } )
  •page  (0cm,0cm)--(1cm,1cm)]
```

Språkets struktur

Exempel på enkla typer

- Flyttal: `14`, `14.5`, `1`
- Heltal: `'5`, `'~12`, `'0xFF`
- Längd: `7cm`, `~3mm`, `72bp`
- Sträng (mer detaljer senare): `` `Hej!´ ``
- Symbol: `'left`

Lexikala bindningar

Lexikala bindningar fungerar som i Scheme, men kan inte bindas om.

```
a: 42
```

Räckvidden (eng: *scope*) är begränsad till en *code bracket*:

```
{
  a: 42
  •stdout << a
}
```

Lexikala bindningar — detaljer

- Högerledet evalueras i samma scope som bindningen tillhör. (Jämför `letrec` i Scheme.)

```
odd: \ n .> [if n = '0 false [even n - '1]]
even: \ n .> [if n = '0 true [odd n - '1]]
```

- Skuggade bindningar kan nås:

```
a: ../a + 7
```

Dynamiska bindningar

Dynamisk bindning infördes som ett sätt att undvika den imperativa spagetti-struktur som ett skrivbart *graphics state* lätt kan leda till.

```
@width:4bp | [stroke mypath]
```

- Dynamiska variabler inleds med `@`.
- Den dynamiska variabeln tillsammans med ett värde blir ett nytt värde som representerar en potentiell dynamisk bindning.
- Bindningsvärden kan kombineras:
  `@width:4bp & @dash:[dashpattern 1cm 4mm]`
- Dynamiska bindningar sätts i scope med en “pipe”.
- En dynamisk variabel har ett filter och ett skönsvärde (eng: *default value*).

Dynamiska värden

En dynamisk variabel kan bindas till ett *dynamiskt värde*.

- Ser ut så här:
  ```
  @bigmargin: dynamic 1.3 * @smallmargin
  ```
- Undviker behovet av att binda alla dynamiska variabler till argumentlösa funktioner.

Funktionsdefinitioner

Exempel:

```
\ x y .> x * x + y * y
```

- Argumentens namn är en del av funktionens signatur.
- En slask (eng: *sink*) kan ta hand om ytterligare argument.
  ```
  \ x y <> rest .> x + y + (foo [] <>rest)
  ```
- Vilka argument som helst kan få skönsvärden:
  ```
  \ x:3 y z:2 .> x + y + z
  ```

Enkla funktionsanrop

Ett enkelt funktionsanrop kan ange argument både genom ordning och genom namn.

```
hypot: \ x y .> [sqrt x*x + y*y]
```

- Ordnade argument: se anrop till `sqrt`.
- Namngivna argument: `[hypot y:3 x:4]`.
- Blandat: Ordnade argument måste komma först.
- Endast ett argument: `square [] 3` eller `square [] x:3`

Märk att namngivna argument kan inte ändra betydelsen av ordnade argument!

Snitt

Scheme: *evaluated cuts*

```
[hypot 3 ...]
[hypot y:4 ...]
```

- Ordnade argument blir helt osynliga i den nya funktionen.
- Namngivna argument (er)sätter skönsvärden.
- Endast ett argument: `hypot [...] 3` eller `hypot [...] y:4`

Kontrollerade tillstånd

Structures

Lat evaluering

Continuation passing style

Funktioner i kärnan

Kurv-konstruktion

Grundläggande ritning

Travare

2D

3D

PDF

LaTeX och strängar

Utmaningar för framtiden

Trixelering

Kompilera funktioner till PDF

Mer grund-struktur

- Namespaces/packages
- Användar-typer

Sammanfattning

Sammanfattning

- Shapes är...

Slut.