

---

# FRP, Arrows e Robots

# Motivação

---

- Utilizar o poder expressivo das linguagens funcionais em sistemas híbridos.
- Sistemas híbridos: sistemas que funcionam em tempo real, com componentes contínuas e componentes discretas (e.g: Robot).

# FRP

---

- Functional Reactive Programming: paradigma para programar sistemas híbridos.
- YAMPA – DSL embebida em Haskell que resulta do refinamento da FRP
- Começou em YAle, terminou em Arrows e com Muita Programação pelo meio.

# FRP

---

- Sinal:

`Signal a = Time -> a`

- Exemplo :

Velocidade de um robot com duas rodas

`Signal (Speed, Speed)`

# FRP / Arrows

---

- Signal Functions

`SF a b = Signal a -> Signal b`

- Não é possível construir directamente Signal Functions.
- Não é possível aplicar Signal Functions a sinais.
- Para que servem e como se usam?

# FRP / Arrows

---

- Permitem, em conjunto com os sinais, tratar valores temporais contínuos.
- Estão munidas de um conjunto de “combinadores” que introduzem disciplina na programação.
- Signal Functions são uma instância da classe Arrow.

# Arrows

## ■ Combinadores

`arr :: (Arrow s) =>`

`(a->b) -> s a b`

`(>>>) :: (Arrow s) =>`

`s a b -> s b c -> s a c`

`first :: (Arrow s) =>`

`s a b -> s (a,c) (b,c)`

# Arrows

- Este é um conjunto mínimo de combinadores.
- `first` pode ser substituído por `(&&&)`  
$$(\&\&\&) :: (\text{Arrow } s) \Rightarrow s \ a \ b \rightarrow s \ a \ c \rightarrow s \ a \ (b, c)$$
- `first f = (arr fst >>> f) &&& (arr snd)`



# Arrows

---

Combinadores já definidos:

`identity :: SF a a`

`constant :: b -> SF a b`

`time :: SF a -> Time`

`arr2 :: (a -> b -> c)`

`-> SF (a,b) c`

`integral :: SF Double Double`

# Exemplo

---

- Equação física

$$x(t) = \frac{1}{2} \times \int_0^t (v_r(t) + v_l(t)) \times \cos \theta(t) dt$$

- Em FRP

- `x = (1/2) * integral ( (vr + vl) * cos theta )`

# Exemplo

- `vrSF, vlSF :: SF SimbotInput Speed`
- `theta :: SF SimbotInput Angle`
- Código yampa:

```
xSF :: SF SimbotInput Distance
```

```
xSF = let
```

```
    v = (vrSF &&& vlSF) >>> arr2 (+)
```

```
    t = thetaSF >>> arr cos
```

```
    in
```

```
(v &&& t) >>> arr (*) >>> integral >>> arr (/2)
```

# Arrows

---

- Açucar Sintático

- Exemplo Anterior

```
xSF :: SF SimbotInput Distance
```

```
xSF = proc inp -> do
```

```
    vr <- vrSF -< inp
```

```
    vl <- vlSF -< inp
```

```
    theta <- thetaSL -< inp
```

```
    i <- intergal -< (vr+vl)*cos theta
```

```
    retrunA -< (i/2)
```

# Escolha condicional

---

- `flag :: SF a Bool`

- Exemplo

```
sf :: SF a b
```

```
sf = proc inp -> do
```

```
    x <- sfx -< inp
```

```
    y <- sfy -< inp
```

```
    b <- flag -< inp
```

```
returnA -< if b then x else y
```

# Eventos

---

- A escolha condicional não é suficientemente expressiva

- Eventos

`Signal (Event b) ≅ Signal (Maybe b)`

- Gerador de Streams de eventos  
`SF a (Event b)`

# Switching

---

- Um gerador de Streams

```
rsStuck :: SF SimbotInput (Event ())
```

- Combinadores de “switching”

```
switch ::
```

```
  SF a (b, Event c) ->
```

```
    (c -> SF a b) -> SF a b
```

# Exemplo

---

- Objectivo : Parar o robot quando este está imobilizado por um obstáculo.

```
stop :: Speed -> SF SimbotInput Speed
stop v = (constant v &&& rsStuck)
        `switch`
        \() -> constant 0
```



# Mais combinadores de switching

---

- `dSwitch :: SF a (b, Event c) ->`  
`(c -> SF a b) -> SF a b`
- `rSwitch, drSwitch ::`  
`SF a b -> SF (a, Event (SF a b)) b`
- Os prefixos `d` e `r` significam  
“delayed” e “recurring”

# Funções sobre Eventos

---

- `tag :: Event a -> b -> Event b`
- `mergeBy :: (a->a->a->) ->`  
`Event a->Event a->Event a`
- `lmerge, rmerge, merge ::`  
`Event a-> Event a -> Event a`

# Funções geradoras de eventos

---

- `edge :: SF Bool (Event ())`
- `never :: b -> SF a (Event b)`
- `now :: b -> SF a -> (Event b)`
- `after :: Time -> b -> SF a (Event b)`
- `repeatedly :: Time -> b -> SF a (Event b)`

# Pequena Demo

---

- Exemplo 1
- ...

# Para saber mais

---

- Para saber muito mais:
  - [www.haskell.org/yampa](http://www.haskell.org/yampa)
  - [www.haskell.org/frp](http://www.haskell.org/frp)
  - [www.soi.city.ac.uk/~ross/arrows](http://www.soi.city.ac.uk/~ross/arrows)