YFrob (1)

YFrob, Yampa version of Frob: Functional Robotics.

• Framework for robot programming on top of Yampa.
• Intended to be generic:
  - Programs written in terms of specific features: specific kinds of sensors and actuators.
  - A program will (in principle) run on any specific platform that provides the assumed features.

YFrob (2)

• Platforms:
  - Pioneer (historical)
  - RobotSim: a simulated environment providing the Simbot platform.
Robot Controller

```haskell
type SimbotController = SimbotProperties -> SF SimbotInput SimbotOutput
```

Input Features (1)

```haskell
class HasRobotStatus i where
  rsBattStat :: i -> BatteryStatus
  rsIsStuck :: i -> Bool

data BatteryStatus = BSHigh | BSLow | BSCritical
  deriving (Eq, Show)
```

Input Features (2)

```haskell
-- derived event sources:
rsBattStatChanged :: HasRobotStatus i => SF i (Event BatteryStatus)
rsBattStatLow :: HasRobotStatus i => SF i (Event ()
rsBattStatCritical :: HasRobotStatus i => SF i (Event ()
rsStuck :: HasRobotStatus i => SF i (Event ()
```
class HasOdometry i where
  odometryPosition :: i -> Position2
  odometryHeading  :: i -> Heading

class HasRangeFinder i where
  rfRange        :: i -> Angle -> Distance
  rfMaxRange     :: i -> Distance

  -- derived range finders:
  rfFront  :: HasRangeFinder i => i -> Distance
  rfBack   :: HasRangeFinder i => i -> Distance
  rfLeft   :: HasRangeFinder i => i -> Distance
  rfRight  :: HasRangeFinder i => i -> Distance

class HasAnimateObjectTracker i where
  aotOtherRobots :: i -> [(RobotType, RobotId, Angle, Distance)]
  aotBalls       :: i -> [(Angle, Distance)]

class HasTextualConsoleInput i where
  tciKey        :: i -> Maybe Char
  tciNewKeyDown :: HasTextualConsoleInput i => Maybe Char -> SF i (Event Char)
  tciKeyDown    :: HasTextualConsoleInput i => SF i (Event Char)
**Output Features**

class MergeableRecord o => HasDiffDrive o where
ddBrake :: MR o
ddVelDiff :: Velocity -> Velocity -> MR o
ddVelTR :: Velocity -> RotVel -> MR o

class MergeableRecord o => HasTextConsoleOutput o
tcoPrintMessage :: Event String -> MR o

**Mergable Records**

mrMerge :: MergeableRecord a => MR a -> MR a -> MR a
mrFinalize :: MergeableRecord a => MR a -> MR a

For example, the expression:
sbo :: SimbotOutput
sbo = mrFinalize
    (ddVelDiff vel1 vel2 `mrMerge` tcoPrintMessage)
merges the velocity output with a console message.

**The Task Monad**

You might have noticed that the type of switch looks a lot like monadic bind:

switch :: SF a (b, Event c)
    -> (c -> SF a b)
    -> SF a b

A task is a signal function along with a terminating event. Instance of monad. Useful for sequencing.

**YFrob Installation (1)**

- Download YFrob-0.4.tar.gz from the course web page.
- Unpack it.
- Go to the top directory: cd YFrob.
- Compile and install (Linux/Unix):
  - cabal configure
  - cabal build
  - sudo cabal install --global
YFrob Installation (2)

- Try one of the applications, e.g. afp-soccer (Linux/Unix):
  - cd afp-soccer
  - make
  - ./afp-soccer

YRSC 2010 “Protocol” (1)

To make it easy to set up games for the Yampa Robot Soccer Cup (YRSC) 2010, follow this “protocol”:

- Each player writes a single module with a distinct module name (e.g. using his or her own name).
- This module exports all the robot controllers the player wants to use for controlling the robots of his or her team.

YRSC 2010 “Protocol” (2)

- If a controller needs to know what team the robot it controls belong to (likely), it should have an extra parameter to allow this information to be passed in from the code that sets up an initial game configuration. For example:

  attacker :: Int -> SimbotController

  For simplicity, let us say the convention is that 1 stands for the left team, and 2 for the right team.

YRSC 2010 “Protocol” (3)

- For identifying team mates, use the animate object tracker. The left team have IDs 1, 2, 3, the right team 11, 12, 13.