Haskell at PIK

Haskell in Leipzig

5 December 2006

- System:

  f :: X -> F X

  We consider only the case where **F** is a *monad*

  data (Monad m) => Sys m a = Sys (a -> m a)

- A system can be *applied*:

  ```
  apply :: (Monad m) => Sys m a -> m a -> m a

  apply sys x = x >>= sys
  ```

  and *iterated*:

  ```
  iter :: (Monad m) => Sys m a -> m a -> Int -> m a

  iter sys x n = take n (iterate (>>= sys) x)
  ```

- Examples:

  m = Id Deterministic

  m = [] Non-deterministic

  m = SimpleProb Stochastic (M. Erwig)

  Most important: *Combinations* of these.

- *Deterioration*

  worse_than :: (Monad m) => m a -> m a -> Bool

  a partial strict order

  (apply sys x) ‘worse_than‘ x

  expresses a general idea of *deterioration* of the state.

- *Vulnerability*

  Deterioration in the context of a *socio-ecological* system:

  sys :: (X,Y) -> m (X,Y)

  or

  soc_sys :: (X,Y) -> m X

  eco_sys :: (X,Y) -> m Y

  sys = tau . pair(soc_sys, eco_sys)

- Problems:
  - Given a non-deterministic soc_sys and a stochastic eco_sys, how can they be combined?
  - Check that *worse_than* is a strict order
  - Extend strict orders from a -> a to m a -> m a
  - **Etc.**

- Formalisation of BSP

- Main elements:
  - Distributed data newtype D a = D (Proc -> a) e.g. pid = D id
  - Monad definition used to describe local computations

- Main elements (cont.):
  - Communication primitive: exch :: D [(a, Proc)] -> D [(a, Proc)]
  - *Reduction* of a constant value:

    val dx = dx 0, if dx is constant

    undefined otherwise.

- Usage:
  - Formulate problems
  - Implement and test proposed solution
  - Serve as documentation of C++ implementation