Amb

From HaskellWiki

This is an implementation of the `amb` operator (http://www.randomhacks.net/articles/2005/10/11/amb-operator) in Haskell. Interestingly, it is identical to the list monad: remove 'amb' and the examples below work fine (apart, of course, from the IO one).

Notably, AmbT could be considered ListT done right.

```
module Amb (AmbT, Amb, amb, cut, runAmbT, runAmb) where

import Control.Monad.Cont
import Control.Monad.State
import Control.Monad.Identity

newtype AmbT r m a = AmbT { unAmbT :: StateT [AmbT r m r] (ContT r m) a }
type Amb r = AmbT r Identity

instance MonadTrans (AmbT r) where
    lift = AmbT . lift . lift

instance (Monad m) => Monad (AmbT r m) where
    AmbT a >>= b = AmbT $ a >>= unAmbT . b
    return = AmbT . return

backtrack :: (Monad m) => AmbT r m a
backtrack = do xss <- AmbT get
               case xss of
                 [] -> fail "amb tree exhausted"
                 (f:xs) -> do AmbT $ put xs; f; return undefined

addPoint :: (Monad m) => (() -> AmbT r m r) -> AmbT r m ()
addPoint x = AmbT $ modify (x () :)

amb :: (Monad m) => [a] -> AmbT r m a
amb []     = backtrack
amb (x:xs) = ambCC $ \exit -> do
               ambCC $ \k -> addPoint k >> exit x
               amb xs
    where ambCC f = AmbT $ callCC $ \k -> unAmbT $ f $ AmbT . k

cut :: (Monad m) => AmbT r m ()
cut = AmbT $ put []

runAmbT :: (Monad m) => AmbT r m r -> m r
runAmbT (AmbT a) = runContT (evalStateT a []) return

runAmb :: Amb r r -> r
runAmb = runIdentity . runAmbT
```

And some examples:

```haskell
example :: Amb r (Integer,Integer)
example = do x <- amb [1,2,3]
             y <- amb [4,5,6]
             if x*y == 8
               then return (x,y)
               else amb []

factor :: Integer -> Amb r (Integer,Integer)
factor a = do x <- amb [2..]
              y <- amb [2..x]
              if x*y == a
                then return (x,y)
                else amb []

factorIO :: Integer -> AmbT r IO (Integer,Integer)
factorIO a = do lift $ putStrLn $ "Factoring " ++ show a
                x <- amb [2..]
                y <- amb [2..x]
                lift $ putStrLn $ "Trying " ++ show x ++ " and " ++ show y
                if x*y == a
                  then do lift $ putStrLn "Found it!"
                          return (x,y)
                  else do lift $ putStrLn $ "Nope (" ++ show (x*y) ++ ")"
                          amb []
```

The extra 'r' can be avoided if you're not using strict Haskell-98:

```haskell
type AmbT' m a = forall r. AmbT r m a
type Amb' a = AmbT' Identity a
```


Categories:

- Monad
- Code
- Idioms
