O'Haskell types

Data types

```
data Datatype = constructor1 type11 ... type1n1 |
                constructor2 type21 ... type2n2 |
                ...

a = constructor1 term11 ... term1n1
b = constructor2 term21 ... term2n2
```

Records

```
struct Record = selector1 :: type1 -> type1'
                selector2 :: type2 -> type2'

record = struct selector1 t1 = term1 (non-recursive)
                selector2 t2 = term2 (non-recursive)

OR

record = struct selector1 = selector1
                selector2 = selector2
         where selector1 t1 = term1 (recursive)
               selector2 t2 = term2 (recursive)

a = record.selector1
b = record.selector2
```

Sub- and supertyping

```
struct RecordS < Record = selectorS1 :: typeS1
                          selectorS2 :: typeS2

data DatatypeS > Datatype = constructorS1 typeS11 ... typeS1nS1 |
                            constructorS2 typeS21 ... typeS2nS2 |

Action     < Cmd ()
Request a  < Cmd a
Template a < Cmd a

struct Methods = method1 :: type11 ... type1n1 -> Action
                 method2 :: type21 ... type2n2 -> Request type2
```

Object classes (templates)

```
class :: type1 -> type2 -> ... -> Template Methods

class x1 x2 ... = template stateVar1 := term1
                           stateVar2 := term2
                  in struct method1 = action  monad_term1 (non-recursive)
                            method2 = request monad_term2 (non-recursive)
                  where <local definitions>

OR

class x1 x2 ... = template stateVar1 := term1
                           stateVar2 := term2
                  in let <local definitions including
                          recursive actions or requests>
                         method1 = action  monad_term1 (recursive)
                         method2 = request monad_term2 (recursive)
                     in struct ..Methods
                  where <local definitions>

a <- class a1 a2 ...
```

Main program of Expander2

```
module Ecom where

import Tk

main tk = do
  win1 <- tk.window []
  win2 <- tk.window []
  fix solve1 <- solver tk "Solver1" win1 solve2 "Solver2" enum1 paint1
      solve2 <- solver tk "Solver2" win2 solve1 "Solver1" enum2 paint2
      paint1 <- painter tk "Solver1" solve1 "Solver2" solve2
      paint2 <- painter tk "Solver2" solve2 "Solver1" solve1
      enum1 <- enumerator tk solve1
      enum2 <- enumerator tk solve2
  solve1.buildSolve (0,20) solve1.skip
  solve2.buildSolve (20,20) solve1.skip
  win2.iconify
```

Template for processing widgets

```
struct Scanner = startScan0 :: Int -> Picture -> Action
                 startScan  :: Int -> Action
                 addScan    :: Picture -> Action
                 stopScan0  :: Action
                 stopScan   :: Action
                 isRunning  :: Request Bool

scanner :: TkEnv -> (Widget_ -> Action) -> Template Scanner
scanner tk act =
    template (as,run,running) := ([],undefined,False)
    in let startScan0 delay bs = action as := bs; startScan delay
           startScan delay = action if running then run.stop
                                    run0 <- tk.periodic delay loop
                                    run := run0; run.start; running := True
           loop = action case as of w:s -> if noRepeat w then as := s
                                           act w
                                           if isFast w then loop
                                    _ -> stopScan
           addScan bs = action as := bs++as
           stopScan0 = action stopScan; as := []
           stopScan = action if running then run.stop; running := False
           isRunning = request return running
       in struct ..Scanner
```

Painter template

```
struct Painter =
  callPaint      :: Bool -> [Picture] -> String -> [Int] -> Int -> Bool -> Bool
                        -> Action -> Action
  labSolver      :: String -> Action
  remote         :: Action -> Action
  setCurrInPaint :: Int -> Action
  setButtons     :: [ButtonOpt] -> [ButtonOpt] -> [ButtonOpt] -> Action
  setEvalInPaint :: String -> Pos -> Action

painter :: TkEnv -> String -> Solver -> String -> Solver -> Template Painter
painter tk solveName solve solveName2 solve2 =

  template ...
           scans := []
           ...

  in let

       ...
```

```
  drawPict pict = action
    if fast || all isFast pict then mapM_ drawWidget pict
    else let lgs = length scans
             (picts1,picts2) = splitAt lgs picts
             g sc pict = do run <- sc.isRunning
                            if run then sc.addScan pict else h sc pict
             h sc = sc.startScan0 delay
         zipWithM_ g scans picts1
         if lgp > lgs then scs <- accumulate $ replicate (lgp-lgs)
                                             $ scanner tk drawWidget
                           zipWithM_ h scs picts2
                           scans := scans++scs
    where picts = if New `elem` pict then f pict [] [] else [pict]
          f (New:pict) picts pict' = f pict (picts++[pict']) []
          f (w:pict) picts pict'   = f pict picts (pict'++[w])
          f _ picts pict'          = picts++[pict']
          lgp = length picts

  ...

in struct ..Painter
```

Tk environment

```
module Tk where

struct Tk =
    window    :: [WindowOpt]   -> Request Window
    bitmap    :: [BitmapOpt]   -> Request ConfBitmap
    photo     :: [PhotoOpt]    -> Request Photo
    delay     :: Int -> (String -> Cmd ()) -> Request String
    periodic  :: Int -> Cmd () -> Request Runnable
    bell      :: Action

primTk :: Template Tk
primTk =
  template in
   let window opts = request
         x <- primGetPath
         primExTcl_["toplevel",x]
         winsetcmd x opts
         win x
       bell        = primExTcl_ ["bell"]
       delay t a   = request
         n <- primNextCallBack
         tag <- primExTcl ["after",show t, "{doEvent ",show n,"}"]
         let tag' = drop 6 tag      -- all tags start with "after#"
         primAddCallBack (\_ -> a tag')
         return tag'
```

```
      periodic t a = request
        n <- primAddCallBack (\_ -> a)
        let ln = "loop"++show n
        primExTcl_["proc",ln,"{args} {haskellEvent ",show n,
                   "\nupdate\nafter",show t,ln,"}"]
        hnd ln
      bitmap opts = request
        os <- textOpts opts
        nm <- primExTcl["image create bitmap",os]
        btmp nm
      photo opts = request
        os <- textOpts opts
        nm <- primExTcl["image create photo",os]
        phto nm
  in struct ..Tk

primExTcl  = primExecuteTcl . unwords
primExTcl_ = primExecuteTcl_ . unwords

primitive primExecuteTcl "primExecuteTcl" :: String -> Request String
primitive primExecuteTcl_ "primExecuteTcl_" :: String -> Action
primitive primGetPath "primGetPath" :: Request String
primitive primAddCallBack "primAddCallBack" :: (String -> Cmd ()) -> Request Int
primitive primNextCallBack "primNextCallBack" :: Request Int
```

```
-- Windows

struct BasicWindow a < ConfWidget a =
    button      :: [ButtonOpt]      -> Request Button
    canvas      :: [CanvasOpt]      -> Request Canvas
    checkButton :: [CheckButtonOpt] -> Request CheckButton
    entry       :: [EntryOpt]       -> Request Entry
    frame       :: [FrameOpt]       -> Request Frame
    label       :: [LabelOpt]       -> Request Label
    listBox     :: [ListBoxOpt]     -> Request ListBox
    menuButton  :: [MenuButtonOpt]  -> Request MenuButton
    radioButton :: [RadioButtonOpt] -> Request RadioButton
    scrollBar   :: [ScrollBarOpt]   -> Request ScrollBar
    slider      :: [SliderOpt]      -> Request Slider
    textEditor  :: [TextEditorOpt]  -> Request TextEditor

type Pos = (Int,Int)

struct ManagedWindow =
    getGeometry :: Request (Pos,Pos)   -- size,position
    setSize     :: Pos -> Action
    setPosition :: Pos -> Action
    iconify     :: Action
    deiconify   :: Action
```

```
-- top level windows

struct Window < BasicWindow WindowOpt, ManagedWindow

-- Images

struct Image =
  imageName :: String

struct Bitmap < Image

struct ConfBitmap < Bitmap, Configurable BitmapOpt

struct PredefBitmap < Bitmap

struct Photo < Image, Configurable PhotoOpt =
  blank    :: Action
  putPixel :: Pos -> Color -> Action
  getPixel :: Pos -> Request Color
  copyFrom :: Photo -> Action  -- to be refined
  saveAs   :: FilePath -> Action

struct Runnable =
   start :: Action
   stop  :: Action

struct TkEnv < Tk, StdEnv
```

```
-- General widget structures

struct Widget =
    ident   :: String
    destroy :: Action
    exists  :: Request Bool
    focus, raise, lower :: Action
    bind    :: [Event] -> Action

struct Configurable a =
    set     :: [a] -> Action

struct ConfWidget a < Widget, Configurable a

-- Structures for subtyping by WWidgets

struct Cell a =
  setValue :: a -> Action
  getValue :: Request a

struct LineEditable =
  lines       :: Request Int
  getLine     :: Int -> Request String
  deleteLine  :: Int -> Action
  insertLines :: Int -> [String] -> Action
```

```
struct Invokable =
    invoke  :: Action

struct Packable =
    packIn :: String -> Dir -> Stretch -> Expansion -> Cmd ()
    wname  :: String

struct Scannable a =
    mark :: a -> Action
    drag :: a -> Action

struct WWidget a < ConfWidget a, Packable

struct ScrollWidget a < WWidget a =
    xview :: Request (Double,Double)
    yview :: Request (Double,Double)

-- Window widgets

struct Frame <  BasicWindow FrameOpt, WWidget FrameOpt

struct Slider < WWidget SliderOpt, Cell Int

struct Button < WWidget ButtonOpt, Invokable =
    flash   :: Action
```

```
struct CheckButton < Button =
   toggle   :: Action
   checked  :: Request Bool

struct RadioButton < Button =
   select   :: Action
   deselect :: Action

struct MenuButton < WWidget MenuButtonOpt =
   menu :: [MenuOpt] -> Request Menu

struct Label < WWidget LabelOpt

struct ListBox < ScrollWidget ListBoxOpt, LineEditable, Cell [Int],
                 Scannable Pos =
  view :: Int -> Action

struct TextEditor < ScrollWidget TextEditorOpt, LineEditable, Scannable Pos

struct Entry < ScrollWidget EntryOpt, Cell String, Scannable Int =
   cursorPos :: Request Int
```

```
struct Canvas < ScrollWidget CanvasOpt, Scannable Pos =
   oval      :: Pos -> Pos -> [OvalOpt]      -> Request Oval
   arc       :: Pos -> Pos -> [ArcOpt]       -> Request Arc
   rectangle :: Pos -> Pos -> [RectangleOpt] -> Request Rectangle
   line      :: [Pos]      -> [LineOpt]      -> Request Line
   polygon   :: [Pos]      -> [PolygonOpt]   -> Request Polygon
   text      :: Pos        -> [CTextOpt]     -> Request CText
   image     :: Pos        -> [CImageOpt]    -> Request CImage
   cwindow   :: Pos        -> [CWindowOpt]   -> Request CWindow
   clear     :: Action
   save      :: FilePath -> Action

struct ScrollBar < WWidget ScrollBarOpt =
   attach :: ScrollWidget BasicWOpt -> Dir -> Action
```

```
-- Canvas Widgets

struct CWidget a < ConfWidget a =
   getCoords :: Request [Pos]
   setCoords :: [Pos] -> Action
   move      :: Pos -> Action

struct Arc       < CWidget ArcOpt
struct Oval      < CWidget OvalOpt
struct Rectangle < CWidget RectangleOpt
struct Line      < CWidget LineOpt
struct Polygon   < CWidget PolygonOpt
struct CText     < CWidget CTextOpt
struct CImage    < CWidget CImageOpt
struct CWindow   < CWidget WindowOpt, BasicWindow WindowOpt

-- Menus

struct Menu < ConfWidget MenuOpt =
    mButton :: [MButtonOpt] -> Request MButton
    cascade :: [MButtonOpt] -> Request Menu

struct MButton < Configurable MButtonOpt, Invokable
```

```
-- Color

data Color = RGB Int Int Int deriving Eq

black  = RGB 0 0 0
white  = RGB 255 255 255
red    = RGB 255 0 0
green  = RGB 0 255 0
blue   = RGB 0 0 255
yellow = RGB 255 255 0

-- Auxiliary types for options

data None       = None
data AnchorType = NW | N | NE | W | C | E | SW | S | SE
data ReliefType = Raised | Sunken | Flat | Ridge | Solid | Groove
data VertSide   = Top | Bottom
data WrapType   = NoWrap | CharWrap | WordWrap
data SelectType = Single | Multiple
data Align      = LeftAlign | CenterAlign | RightAlign
data Round      = Round
data ArcStyleType = Pie | Chord | Perimeter
data CapStyleType  > Round = Butt | Proj
data JoinStyleType > Round = Bevel | Miter
data ArrowType     > None = First | Last | Both
```

```
-- Options

data Anchor      = Anchor AnchorType
...

-- widget option types

data BasicOpt       > Background, BorderWidth, Cursor, Relief
data BasicWOpt      > BasicOpt, Width
data DimOpt         > Height, Width
data StdOpt         > BasicWOpt, DimOpt
data FontOpt        > Font, Foreground, Anchor, Justify
data PadOpt         > Padx, Pady

data WindowOpt      > BasicOpt, Title
data PhotoOpt       > DimOpt, File
data BitmapOpt      > Background, Foreground, File, BitmapData

data ButtonOpt      > MenuButtonOpt, Command
data CanvasOpt      > StdOpt, ScrollRegion
data CheckButtonOpt > ButtonOpt, Indicatoron, SelectColor
data EntryOpt       > BasicWOpt, Justify, Font, Foreground, Enabled
type FrameOpt       = StdOpt
data LabelOpt       > StdOpt, FontOpt, PadOpt, Img, Btmp, Underline, Text
data ListBoxOpt     > StdOpt, Font, Foreground, SelectMode
data MenuButtonOpt  > LabelOpt, Enabled
type RadioButtonOpt = CheckButtonOpt
```

```
type ScrollBarOpt   = StdOpt
data SliderOpt      > BasicWOpt, From, To, Orientation, Length,
                      Font, Foreground, CmdInt, Enabled
data TextEditorOpt  > StdOpt, Font, Foreground, PadOpt, Wrap, Enabled

data CBasicOpt      > Fill, Width, Stipple
data CImageOpt      > Anchor, Img, Btmp
data CTextOpt       > Font, Justify, Text, Anchor, Fill
data CWindowOpt     > DimOpt, Anchor
data LineOpt        > CBasicOpt, Arrow, Smooth, CapStyle, JoinStyle
data PolygonOpt     > OvalOpt, Smooth
data ArcOpt         > OvalOpt, ArcStyle, Angles
data OvalOpt        > CBasicOpt, Outline
data RectangleOpt   > OvalOpt

data MenuOpt        > WindowOpt, Enabled
data MButtonOpt     > StdOpt, FontOpt, PadOpt, Img, Btmp, Underline,
                      CLabel, Enabled, Command

data AllOpt         > MenuOpt, CheckButtonOpt, TextEditorOpt, FrameOpt,
                      LineOpt, WindowOpt, ArcOpt, PolygonOpt,
                      OvalOpt, CTextOpt, RectangleOpt, SliderOpt, MButtonOpt,
                      CanvasOpt, ListBoxOpt, BitmapOpt, PhotoOpt, CImageOpt,
                      EntryOpt, CWindowOpt, ButtonOpt, MenuButtonOpt,
                      LabelOpt
```

```
--- Events

data ButtonPress = ButtonPress Int (Pos -> Cmd ())
                 | AnyButtonPress  (Int -> Pos -> Cmd ())

data MouseEvent > ButtonPress =
           ButtonRelease Int (Pos -> Cmd ())
         | AnyButtonRelease (Int -> Pos -> Cmd ())
         | Motion Int (Pos -> Cmd ())
         | AnyMotion  (Pos -> Cmd ())
         | Double ButtonPress
         | Triple ButtonPress

data WindowEvent = Enter (Cmd ())
                 | Leave (Cmd ())
                 | Configure (Pos -> Cmd ())

data SimpleKeyEvent = KeyPress String (Cmd ())
                    | KeyRelease String (Cmd ())
                    | AnyKeyPress (String -> Cmd ())

data KeyEvent > SimpleKeyEvent = Mod [Modifier] SimpleKeyEvent

data DestroyEvent = Destroy (Cmd ())

data Event > MouseEvent, KeyEvent, WindowEvent, DestroyEvent
```

Internal representation of terms and formulas

```
data Term a = V a | F a [Term a] | Hidden Special deriving (Show,Eq)

type TermS = Term String

data Special = Dissect [(Int,Int,Int,Int)] | BoolMat [String] [String] Pairs |
               ListMat [String] [String] Triples | ListMat2 [String] Triples2 |
               ListMat2L [String] Triples2L | LRarr (Array (Int,Int) ActLR) |
               ERR deriving (Show,Eq)
```

Positions and pointer modifications

```
-- label t p returns the root of the subterm at position p of t.

label t []       = root t
label (F _ ts) p = label' ts p
label t _        = undef

label' (t:_) (0:p)  = label t p
label' (_:ts) (n:p) = label' ts $ (n-1):p
label' _ _          = undef
```

```
-- getSubterm t p returns the subterm at position p of t.

 getSubterm t []       = t
 getSubterm (F _ ts) p = getSubterm' ts p
 getSubterm t _        = Hidden ERR

 getSubterm' (t:_) (0:p)  = getSubterm t p
 getSubterm' (_:ts) (n:p) = getSubterm' ts $ (n-1):p
 getSubterm' _ _          = Hidden ERR

-- dropFromPoss p t removes the prefix p from each pointer of t below p.

 dropFromPoss p = if null p then id else mapT f
                  where f x = if isPos x && p <<= q
                              then mkPos0 $ drop (length p) q else x
                              where q = getPos x

-- getSubterm1 t p returns the subterm u at position p of t and replaces each
-- pointer p++q in u by q.

 getSubterm1 t p = dropFromPoss p $ getSubterm t p
```

```
-- addToPoss p t adds the prefix p to all pointers of t that point to subterms
-- of t.

 addToPoss p t = if null p then t else mapT f t
                 where f x = if isPos x && q `elem` positions t
                             then mkPos0 $ p++q else x where q = getPos x

-- changePoss p q t replaces the prefix p of all pointers of t with prefix p by
-- q.

 changePoss p q = mapT f where f x = if isPos x && p <<= r
                                     then mkPos0 $ q++drop (length p) r else x
                                     where r = getPos x
```

```
-- replace t p u expands t at all pointers into the subterm v of t at position
-- p. Pointers to the same subterm are expanded only once, the others are
-- redirected. Afterwards v is replaced by u.

 replace t p0 u = h [] t where posFun = g (const []) $ f [] t
                               f p _ | p == p0 = []
                               f p (F x ts)    = concat $ zipWithSucs f p ts
                               f p (V x) | isPos x && p0 <<= q
                                               = [(q,p)] where q = getPos x
                               f _ t           = []
                               g f ((p,q):ps) = g (upd f p [q]) ps; g f _ = f
                               h p _ | p == p0 = u
                               h p (F x ts)    = F x $ zipWithSucs h p ts
                               h p (V x) | isPos x && p0 <<= q
                                               = if p == r then movePoss t q p
                                                           else mkPos r
                                                 where q = getPos x
                                                       r:_ = posFun q
                               h _ t           = t

-- replace2 t p u q copies the subterm at position p of t to position q of u and
-- replaces each pointer p++r in the modified term by q++r.

 replace2 t p0 u q0 = replace u q0 $ changePoss p0 q0 $ f [] $ getSubterm t p0
                      where f p (F x ts) = F x $ zipWithSucs f p ts
                            f p (V x) | isPos x && q0 << q && not (p0 <<= q)
                                        = movePoss t q p where q = getPos x
```