

# Questions: Semantic and Computational Issues

Veneeta Dayal and Chung-chieh Shan

September 6, 2006

## 1. Getting started with Haskell

### Evaluating expressions

```
> 2 + 3
5
> (\x -> x + x) 3
6
> (\x -> \y -> x * x + y * y) 3 4
25
> (\x y -> x * x + y * y) 3 4
25
> 2 == 3
False
> 2 < 3 || 2 >= 3
True
> 2 < 3 && 2 >= 3
False
> False == True
False
> False /= not True
False
```

### Defining names

```
Test.hs
module Test where
-- Sum the square of two numbers
sumsq = \x y -> x * x + y * y
```

```
> sumsq 3 4
25
```

```
sumsq x y = x * x + y * y
```

```
sumsq x = \y -> x * x + y * y
```

## Type inference

```
> :type True
True :: Bool
> :type not
not :: Bool -> Bool
> :type not True
not True :: Bool
> not not
(Type error)
> False False
(Type error)
> :type \x y -> x && y
\x y -> x && y :: Bool -> Bool -> Bool
> :type (\x y -> x && y) True
(\x y -> x && y) True :: Bool -> Bool
> :type \f -> f False || f True
\f -> f False || f True :: (Bool -> Bool) -> Bool
> :type \x -> x
\x -> x :: a -> a
> :type \x -> x + x
\x -> x + x :: (Num a) => a -> a
> :type \x -> if x > 0 then x else 0
\x -> if x > 0 then x else 0 :: (Ord a, Num a) => a -> a
> :type \x -> x + not x
(Type error)
```

## Type signatures

```
positive :: Integer -> Bool
positive x = x > 0
```

## Type synonyms

```
type IPred = Integer -> Bool
positive :: IPred
positive x = x > 0
```

## Creating a new type

```
data Person = Alice | Bob | Carol deriving (Eq, Show)
female x = x == Alice || x == Carol
```

## 2. On puzzles and types

Playing with Haskell is like completing a puzzle, whose pieces' shapes are made up with rules that, at first, you seem not be able to grasp. You keep pushing the last piece, and it just doesn't fit in. Then you realize that shapes are actually types. —Andrea Rossato (Haskell Café list, Sep 1)

## 3. Resources

Haskell home page: <http://www.haskell.org/> Two implementations:

- Hugs (may be easier to install): <http://www.haskell.org/hugs/>
- GHC (may give clearer error messages): <http://www.haskell.org/ghc/>  
(The interpreter in GHC is called `ghci`.)

## 4. Exercises

Today's examples are online at <http://www.cs.rutgers.edu/~ccshan/questions/>

1. Give an example of something to which you can apply `good`, defined by

```
good f = f 0 && f 1
```

What is the type of `good`? Can you apply anything to `good`?

2. Define the proposition (of type `Prop`) that nobody saw anybody. How do you check that your definition is correct?
3. In Hamblin's semantics of questions, the extension of a question is not its set of true answers, but its set of answers. Change `q` in `Karttunen1.hs` and `Karttunen2.hs` to compute the Hamblin extension of *Who did Alice see?*. How do you check that your changes work?
4. The three lines in the definition of `q` in `Karttunen1.hs` is rather repetitive. Define an auxiliary function `f` to capture this repetition, so that the definition of `q` simplifies to

```
q w p = p w && (p == f Alice || p == f Bob || p == f Carol)
```

What is the type of your `f`? Can you simplify this definition of `q` further?

5. A semanticist can read *Two Dozen Short Lessons in Haskell* to learn Haskell. In particular, Chapter 4 explains *list comprehensions*, a convenient way to build lists. Use a list comprehension to make the definition of `q` in `Karttunen2.hs` less repetitive.