# Functional Programming: Exercise Session 3

- Comments on sheet 2:
  - Mind Thy Syntax Trees
  - Kickstart your modeling career
  - Haskell
  - Headache and its Aspirin
- Practice for sheet 3
  - Induction over naturals
  - Some functions on lists

# Thou Shalt Mind Thy Syntax Trees

- The most common mistakes had to deal with using inapplicable rules
- You have to "parse" the formula correctly. Refresher:

*Form*, the formulae in first-order logic, is the smallest set where

1. $\bot \in Form$,
2. $p^n(t_1, \ldots, t_n) \in Form$ if $p^n \in \mathcal{P}$ and $t_j \in Term$, for all $1 \leq j \leq n$,
3. $A \circ B \in Form$ if $A \in Form$, $B \in Form$, and $\circ \in \{\wedge, \vee, \rightarrow\}$, and
4. $Qx.\, A \in Form$ if $A \in Form$, $x \in \mathcal{V}$, and $Q \in \{\forall, \exists\}$

# Thou Shalt Mind Thy Syntax Trees (cont)

- Example: if you have a sequent of the form $\Gamma \vdash \forall x.P(x) \to Q$, you *cannot* apply imp-I to it
- Example 2: can't use $\wedge ER$ to conclude $\Gamma \vdash P(x)$ from $\Gamma \vdash \exists x.P(x) \wedge Q(x)$
- Why?
- Can you think of a wrong proof exploting such "rules"?
- Side note: if a rule has side conditions, **check them**! And let us know that you did (a remark at the end is fine)
- Example: Assignment 3. (b) and (c)

# Existential (Introduction/Elimination) Issues

- Correct proof, with $\Gamma := \exists x.\, P(x) \wedge Q(x)$

$$\dfrac{\dfrac{}{\Gamma \vdash \exists x.\, P(x) \wedge Q(x)}\,\text{AX} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma,\, P(x) \wedge Q(x) \vdash P(x) \wedge Q(x)}\,\text{AX}}{\Gamma,\, P(x) \wedge Q(x) \vdash Q(x)}\,\wedge\text{ER}}{\Gamma,\, P(x) \wedge Q(x) \vdash \exists y.\, Q(y)}\,\exists\text{I}}{\exists x.\, P(x) \wedge Q(x) \vdash \exists y.\, Q(y)}\,\exists\text{E}^{**}$$

- Failed proof attempt, with same $\Gamma := \exists x.\, P(x) \wedge Q(x)$

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \exists {\color{red}z}.\, P({\color{red}z}) \wedge Q({\color{red}z})}\,\text{AX} \qquad \dfrac{\dfrac{???}{\Gamma,\, P({\color{red}z}) \wedge Q({\color{red}z}) \vdash ?? \wedge Q(x)}}{\Gamma,\, P({\color{red}z}) \wedge Q({\color{red}z}) \vdash Q(x)}\,\wedge\text{ER}}{\Gamma \vdash Q(x)}\,\exists\text{E}^{**}}{\exists x.\, P(x) \wedge Q(x) \vdash \exists y.\, Q(y)}\,\exists\text{I}$$

Note use of ${\color{red}z}$. Using $x$ instead is not allowed by $\exists$E's side condition!

# Models

- Parts where we asked for models of formulas were mostly correct
- Mistakes when asked for non-models
- $\forall x. (\exists y. r(x, y) \wedge q(y)) \rightarrow (\forall y. r(x, y) \rightarrow q(y))$
  - $I(r) = (a, b), (b, c), (c, a)$, $I(q) = a$
- $\forall x. \forall y. r(x, y) \rightarrow r(y, x) \rightarrow x = y$
  - Hint: can you reformulate this to use conjunction? What property of the relation $r$ does formula specify?
  - $I(r) = \{(a, b), (b, a)\}$
  - $I(r) = \{(a, a), (a, b)\}$

# Haskell

- Exercise 4 not too difficult and pretty much everybody nailed it
  - Except the I/O: repeat on good input, stop on bad!
- Style remark

```
if x==0 then True else False => x==0
foo x | <expr> = True        => foo x = <expr>
      | otherwise = False
```

- Exercise 5 becomes much easier once we know how to work with lists

# Headache of the week

**Claim:** If $f : \mathbb{R} \to \mathbb{R}$ is $f(x) = x$, for all $x \in \mathbb{R}$, then $f$ is continuous.
**Proof:** Let $c$ be an arbitrary real number. Let $\epsilon$ be an arbitrary positive real number. Choose $\delta = \epsilon$. Note that $\delta > 0$ since $\epsilon > 0$. Let $x$ be an arbitrary real number. Assume that $|x - c| < \delta$. As $f(x) = x$ and $f(c) = c$, it follows that $|f(x) - f(c)| < \epsilon$, because we chose $\delta = \epsilon$.
**Q.E.D.**

Turn the above argumentation into a formal proof, that is, use the proof rules from the lecture to derive

$$\vdash \big(\forall x.\, f(x) = x\big) \to \forall c.\, \forall \epsilon.\, \epsilon > 0 \to \exists \delta.\, \delta > 0 \wedge \forall x.\, |x - c| < \delta \to |f(x) - f(c)| <$$

# Induction over Nat

- Assume we're given a function *sum* with the following definition:

```
sum 0 = 0                        -- sum.1
sum n = sum (n-1) + n            -- sum.2
```

- Let's prove: $\forall n \in \mathbb{N}.\ sum\ n = \frac{n(n+1)}{2}$
- *Note:* when do we use *sum*.1, and when *sum*.2?

# Functions on Lists

- Lists are one of the main tools in Haskell
- *Lots* of predefined functions (e.g. in *Prelude*, *Data.List*)

```
-- test if list is equal to the empty list ([])
null :: [a] -> Bool
null [] = True
null _  = False

-- return head of the list
head :: [a] -> a
head []     = error "head: not defined on empty list"
head (x:_) = x

tail :: [a] -> [a]
tail []     = error "tail: not defined on empty list"
tail (_:xs) = xs

(:) :: a -> [a] -> [a]
```

# Functions on Lists (cont.)

```
-- return the last element of the list
last :: [a] -> a

-- return all elements of the list except the last one
init :: [a] -> [a]


(++) :: [a] -> [a] -> [a]
length :: [a] -> Int
reverse :: [a] -> [a]
(!!) :: Int -> [a] -> a -- zero-based
```

Exercise: Implement 3 of the functions listed above. E.g., *last*, *init*, ...

# More Functions

```
| reverse :: [a] -> [a]                        | reverse [1,2,3]
| map :: (a -> b) -> [a] -> [b]                | map (+1) [1,2,3]
| intersperse :: a -> [a] -> [a]               | intersperse ','
| intercalate :: [a] -> [ [a] ]-> [a]          | intercalate ", "
|                                              | "Haskell, C#, Ja
| take :: Int -> [a] -> [a]                    | take 3 [1..] = [
| drop :: Int -> [a] -> [a]                    | drop 7 [1..10] =
| splitAt :: Int -> [a] -> ([a], [a])          | splitAt 3 "FOO,B
| break ::                                     | break (==',') "F
| (a -> Bool) -> ([a] -> ([a], [a]))           |
| elem :: Eq a => a -> [a] -> Bool             | 1 `elem` [2,3,1]
| concat :: [ [a] ]-> [a]                      | concat [[1],[2,3
```

Exercise: Implement 2 of the functions listed above.