# Computational Semantics with Haskell

Yulia Zinova

Winter 2016/2017

# Sets and Set Notation

- A set is a collection of definite, distinct objects
- Examples of sets?

# Sets and Set Notation

- A set is a collection of definite, distinct objects
- Examples of sets?
- Set of words in a particular book
- Set of colors of the German flag
- Set of letters in the Greek alphabet
- Set of even natural numbers greater than 5

We follow Van Eijck and Unger 2010, electronic access from the library

# Sets and Set Notation

- Elements of a set a called its *members*
- $a$ is an element of set $A$: $a \in A$
- $a$ is not an element of set $A$: $a \notin A$
- Elements of the set can be very different:

# Sets and Set Notation

- Elements of a set a called its *members*
- $a$ is an element of set $A$: $a \in A$
- $a$ is not an element of set $A$: $a \notin A$
- Elements of the set can be very different:
- words, colors, letters, numbers, other sets
- Example: set $A$ containing two sets – set $B$ of even numbers and set $C$ of odd numbers

## Sets and Set Notation

- Elements of a set a called its *members*
- $a$ is an element of set $A$: $a \in A$
- $a$ is not an element of set $A$: $a \notin A$
- Elements of the set can be very different:
- words, colors, letters, numbers, other sets
- Example: set $A$ containing two sets – set $B$ of even numbers and set $C$ of odd numbers
- Set $A$ has 2 members, sets $B$ and $C$ have an infinite number of members

# Sets and Set Notation

- Two sets are the same if they have the same members
- All sets are fully determined by their members – principle of extensionality
- Several ways to specify a set:
  1. Give a list of its members:

# Sets and Set Notation

- Two sets are the same if they have the same members
- All sets are fully determined by their members – principle of extensionality
- Several ways to specify a set:
  1. Give a list of its members: set having as its members numbers *1*, *2* and *3*.
  2. Provide a semantic description:

## Sets and Set Notation

- Two sets are the same if they have the same members
- All sets are fully determined by their members – principle of extensionality
- Several ways to specify a set:
  1. Give a list of its members: set having as its members numbers *1*, *2* and *3*.
  2. Provide a semantic description: set of colors of the German flag
  3. Separate a set out of a larger set *(set comprehension)*:

# Sets and Set Notation

- Two sets are the same if they have the same members
- All sets are fully determined by their members – principle of extensionality
- Several ways to specify a set:
  1. Give a list of its members: set having as its members numbers *1*, *2* and *3*.
  2. Provide a semantic description: set of colors of the German flag
  3. Separate a set out of a larger set *(set comprehension)*: even natural numbers are natural numbers such that the division by 2 leaves no reminder $E = \{2n \mid n \in \mathbb{N}\}$ `http://directpoll.com/r?XDbzPBd3ixYqg8p5Yh47q1CL4dJyUfDjWycpEuEv`

# Sets and Set Notation

- Some important sets have special names: $\mathbb{N}$ is a set of natural numbers, $\mathbb{Z}$ is the set of integer numbers, $\emptyset$ is the empty set.
- Exercise 1.1 Explain why $\emptyset \subseteq A$ holds for every $A$.
- Exercise 1.2 Explain the difference between $\{\emptyset\}$ and $\emptyset$
- The complement of a set $A$ with respect to some fixed universe $U$ (called domain) with $A \subseteq U$, is a set consisting of all objects in $U$ that are not elements of $A$.
  $\bar{A} = \{x \mid x \in U, x \notin A\}$
- Exercise 1.3 Check that $\bar{\bar{A}} = A$

# Relations

- Sets are collections of objects, but we also need relations.
- *Relation* between two sets $A$ and $B$ is a collection of ordered pairs $(a, b)$ such that $a \in A$ and $b \in B$
- Set of all ordered pairs such that the first element is taken from the set A and the second is taken from the set B is called *Cartesian product* and written as $A \times B$
- If $A = \{a, b, \dots, h\}$, $B = \{1, 2, \dots, 8\}$, $C = \{King, Queen, Knight, Bishop, Pawn, Rook\}$, $D = \{White, Black\}$, how can we obtain
  1. set of all possible positions,
  2. set of all figures,
  3. set of piece positions on the board,
  4. set of all the moves (not necessarily legal) on a board?

# Relations

- Sets of ordered pairs are called binary relations.
- Sets of triples are ternary relations.
- Example of a ternary relation?

# Relations

- Sets of ordered pairs are called binary relations.
- Sets of triples are ternary relations.
- Example of a ternary relation? Borrowing something from someone (who borrowed, owner, thing)
- *n*-ary relation is a set of *n*-tuples (ordered sequences of *n* objects)
- Unary relations are called *properties*.

# Composition

- If $R$ and $S$ are binary relations on a set $U$, i.e. $RsubseteqU^2$ and $SsubseteqU^2$, then the composition of $R$ and $S$ ($R \circ S$) is a set of pairs $(x, y)$ such that there is some $z$ with $(x, z) \in R$ and $(z, y) \in S$
- `http://directpoll.com/r?XDbzPBd3ixYqg8xopZWtE9oUDSLndsGvLua0BpIrP`

# Converse

- $R^{\vee} = \{(y, x) | (x, y) \in R\}$
- if a binary relation has the property that $R^{\vee} \subseteq R$, $R$ is called *symmetric*
- Exercise 1.4 Show that it follows from $R^{\vee} \subseteq R$ that $R^{\vee} = R$

# Identity relation, reflexive relations, transitive relations

- If $U$ is a set, the relation $I = \{(x, x) | x \in U\}$ is called the identity relation.
- If a relation $R$ on $U$ has the property that $I \subseteq R$, $R$ is called *reflexive*
- A relation $R$ is called *transitive* if it holds for all $x$, $y$, $z$ that if $(x, y) \in R$ and $(y, z) \in R$, then also $(x, z) \in R$
- If one says that the relation of friendship is transitive, what does it mean?

# Identity relation, reflexive relations, transitive relations

- If $U$ is a set, the relation $I = \{(x,x)|x \in U\}$ is called the identity relation.
- If a relation $R$ on $U$ has the property that $I \subseteq R$, $R$ is called *reflexive*
- A relation $R$ is called *transitive* if it holds for all $x$, $y$, $z$ that if $(x,y) \in R$ and $(y,z) \in R$, then also $(x,z) \in R$
- If one says that the relation of friendship is transitive, what does it mean?
- `http://directpoll.com/r?XDbzPBd3ixYqg8sGl2JvOqFN6XQsixLOQzf5GuNwU`

# Functions

- Functions are relations such that for any $(a, b)$ and $(a, c)$ in the relation it has to hold that $b$ and $c$ are equal.
- A *function* from a set $A$ (domain) to a set $B$ (range) is a relation between $A$ and $B$ such that for each $a \in A$ there is one and only one associated $b \in B$.
- Functions allow us to express *dependence*.
- Examples?

# Functions

- Functions can be given by tables (possibly infinite) – *extensional view.*
- Functions can be given as instructions for computation – *intensional view*
- Functions can be composed. If $g$ is a function that converts from Kelvin to Celcius and $f$ is a function that converts from Celcius to Fahrenheit, them $f\dot{g}$ is the function that converts from Kelvin to Fahrenheit.
- Exercise 1.5 The successor function $s : \mathbb{N} \rightarrow \mathbb{N}$ on natural numbers is given by $n \mapsto n + 1$. What is the composition of $s$ with itself?

# Characteristic function

- The *characteristic function* of a subset $A$ of some domain $U$ is a function that maps all members of $A$ to the truth-value **True** and all elements of $U$ that are not members of $A$ to **False**.
- As we described relations as sets, we can represent every relation as a characteristic function.
- Exercise 1.6 $\leq$ is a binary relation on the natural numbers. What is the corresponding characteristic function?

# Lambda calculus

- $\lambda$-calculus was developed by the mathematician Alonzo Church in the 1930s as a proposal for a precise definition of the notion of mechanical computation.
- Around the same time Alan Turing developed a different notion of computable functions in terms of a Turing machine.
- Those notions turned out to be equivalent (defining the same class of functions, also called the *recursive functions*)
- In 1960s the seminal work of Richard Montague showed the way towards beautiful applications of lambda calculus in linguistics.

# Lambda calculus

- Consider the notation $x \mapsto x^2 + y$
- This can be thought of as the function mapping $x$ to $x^2 + y$ for some fixed $y$.
- To make a clead distinction between the *bound* variable $x$ and the *unbound* variable $y$ we write
  $\lambda x \mapsto x^2 + y$ or $\lambda x.x^2 + y$
- The lambda operator indicates that this is the function that depends on one parameter $x$.
- If we want to bound $y$ too, we need to write $\lambda x \lambda y \mapsto x^2 + y$

# Function application

- Let us apply the function $\lambda x \lambda y \mapsto x^2 + y$ to an argument 3. It is written as
  $(\lambda x \lambda y \mapsto x^2 + y)3$
  and provides the following result
  $\lambda y \mapsto 3^2 + y$

# Language of lambda calculus

- Let us define all the possible expressions E:
  $E ::= v|(EE)|(\lambda v \mapsto E)$
- This is a context-free grammar in Backus-Naur Form (BNF)
- Construct at least 3 different expressions using various combinations of rules. Save them for later.
- Keeping in mind this definition, answer the following question:
  http://directpoll.com/r?XDbzPBd3ixYqg81BahqggV1lojc4u9XCcBrHS6Z8g

# Functions as arguments

- Functions can take functions as arguments
- Imagine we have a function $\lambda f \mapsto (f\, dragon)$
- Now we need to feed it the pluralization function as an argument.

# Functions as arguments

- Functions can take functions as arguments
- Imagine we have a function $\lambda f \mapsto (f\, dragon)$
- Now we need to feed it the pluralization function as an argument.
- $(\lambda f \mapsto (f\, dragon))(\lambda x \mapsto x +\!\!+\, s)$
- After function application

# Functions as arguments

- Functions can take functions as arguments
- Imagine we have a function $\lambda f \mapsto (f\, dragon)$
- Now we need to feed it the pluralization function as an argument.
- $(\lambda f \mapsto (f\, dragon))(\lambda x \mapsto x +\!\!+\ s)$
- After function application $(\lambda x \mapsto x +\!\!+\ s) dragon$
- After another application

# Functions as arguments

- Functions can take functions as arguments
- Imagine we have a function $\lambda f \mapsto (f\, dragon)$
- Now we need to feed it the pluralization function as an argument.
- $(\lambda f \mapsto (f\, dragon))(\lambda x \mapsto x +\!\!+\, s)$
- After function application $(\lambda x \mapsto x +\!\!+\, s) dragon$
- After another application *dragons*
- Exercise 1.7 Reduce the following expression $(\lambda f \lambda x \mapsto f(fx))(\lambda y \mapsto 1 + y)$
- Exercise 1.8 Reduce the following expression $(\lambda x \mapsto xx)(\lambda x \mapsto xx)$

# Types in Grammar and Computation

- To allow only sensible expressions and applications, we need to introduce types.
  $\tau ::= b | (\tau \rightarrow \tau)$
- Now, each lambda expression is assigned a type. This is written as $E : \tau$ ($E$ is an expression of type $\tau$)

# How to obtain the type of

- **Variables**: for each type $\tau$ we have variable for that type, e.g. $x : \tau$, $x' : \tau$, etc.
- **Abstraction**: if $x : \delta$ and $E : \tau$, then $(\lambda x \mapsto E) : \delta \rightarrow \tau$.
- **Application**: if $E_1 : \delta \rightarrow \tau$ and $E_2 : \delta$, then $(E_1 E_2) : \tau$.
- Exercise 1.9 Find types for the expressions you created
- Exercise 1.10 Find a type for $(\lambda x \mapsto xx)(\lambda x \mapsto xx)$

# Types in Haskell

- `Int` – the type of integers
- `Bool` – the type of truth-values
- `Char` – the type of characters
- type variables `a, b, ...` which stand for arbitrary types
- list types `[a]`
- function types `a` $\rightarrow$ `b`
- user-defined types

# Types

- Types in the lambda calculus have a lot in common with syntactic categories in grammars.
- Basic types can be thought of as corresponding to terminal categories in grammars (complete expressions)
- Function types characterize incomplete expressions like verb phrases.
- Such approach is called categorical grammar.
- Exercise 1.11 Assume that adjectives are of type $N \rightarrow N$, as they take a noun and return an expression of the same category. What is a type of an adverbial *very*?

# Functional Programming

- Programms are similar to functions. Side effects – change of state of the system.
- We can consider two results of performing something: the evaluation of an expression and a change of state this evaluation brings about.
- Imperative programming focuses on the state and how to modify it.
- Declarative programming focuses on the evaluation itself.
- For functional programming, computation corresponds to the evaluation of functions.
- Basic Haskell syntax: $E ::= x | E\ E | \lambda x \rightarrow E | let\ x = E\ in\ E$

- $((\lambda x.(xy))(\lambda z.z))$
- $((\lambda x.((\lambda y.(xy))x))(\lambda z.w))$
- $((((\lambda f.(\lambda g.(\lambda x.((fx)(gx)))))(\lambda m.(\lambda n.(nm))))(\lambda n.z))p)$

## References:

Van Eijck, J. and Unger, C. (2010). *Computational semantics with functional programming*. Cambridge University Press.