2. Semantics of Functional Programs

up to now: defined syntax of Haskell
now: Semantics of Haskell

needed to implement compilers/interpreters and to know when a program is correct.

2 ways to define the semantics of programming languages:

a2 • denotational semantics: define semantics by a mapping from constructs of the prog. language to mathematical objects

a3 • operational semantics: define semantics by providing an interpreter for the language. Every other interpreter should behave in the same way.

2.1: Mathematical Foundations
2.2: Def. of Semantics of Haskell

2.1. Complete Partial Orders and Fixpoints

2.1.1: math. objects needed for Haskell expressions
2.1.2: — u —— Haskell functions
2.1.3: How can one infer the semantics of a recursively defined Haskell function?
2.1.1. Partially Defined Values

Goal: Define a mapping \( Val \) that maps every Haskell-expression to a mathematical object.
For an expression \( \text{exp} \), \( Val(\text{exp}) \) should be the meaning/semantics of \( \text{exp} \).

Ex: Haskell-expression 5 is mapped to the number 5,
i.e.: \( Val(5) = 5 \).

What about Haskell-functions?

\[
\text{square} :: \text{Int} \rightarrow \text{Int} \\
\text{square} \ x = x \times x
\]

\( Val(\text{square}) \) is the function that takes a number \( n \)
from \( \mathbb{Z} \) and returns \( n \cdot n \).

What about the following non-terminating function?

\[
\text{non-term} :: \text{Int} \rightarrow \text{Int} \\
\text{non-term} \ x = \text{non-term} \ (x + 1)
\]

What is \( Val(\text{non-term} \ 0) \)?
To express undefinedness, we introduce a special undefined value \( \bot \) ("bottom").
Then \( Val(\text{non-term} \ 0) = Val(\text{non-term} \ 1) = \bot \).
So for the semantics of Haskell, we need mathematical objects like \( \mathbb{Z}, \mathbb{L}, \mathbb{B} \), tuples, functions, etc.

Booleans

We will only define semantics for well-typed expressions. At the moment, we disregard polymorphism and assume that every expression has a unique type. For each type we now define a domain of mathematical objects, such that \( \text{Val} \) maps expressions of this type to objects of the corresponding domain.

Domain for type \( \text{Int} \): \( \mathbb{Z}_+ = \{ \mathbb{L}_2, 0, 1, -1, 2, -2, \ldots \} \)

Domain for type \( \text{Bool} \): \( \mathbb{B}_+ = \{ \mathbb{L}_\mathbb{B}, \text{True}, \text{False} \} \)

Similarly for \( \text{Char}, \text{Float}, \ldots \)

For every domain \( D \) we use a partial ordering \( \leq_D \) which compares elements of \( D \) by their definedness. So \( x \leq_D y \) means that \( x, y \in D \) and \( x \) is less defined or equal to \( y \).

So for \( \mathbb{Z}_+ \) we have: \( \mathbb{L}_{\mathbb{Z}_+} \leq_{\mathbb{Z}_+} 0 \) (Slide 29)

\[
\begin{array}{cccc}
-2 & -1 & 0 & 1 & 2 & 3 & \ldots \\
- & - & 0 & \neq & 1 & \neq & \end{array}
\]
Sud domains are called flat, because elements are either completely defined or completely undefined. "≤" is partial because there are elements that can’t be compared with "≤" (e.g., 0 ≤ 1, 1 ≤ 0).

An order (ordering) is a relation with certain properties.

1. \( x \leq x \) (reflexivity)
2. \( x \leq y \) and \( y \leq z \) implies \( x \leq z \) (transitivity)
3. \( x \leq y \) and \( y \leq x \) implies \( x = y \) (antisymmetry)

A transitive antisymmetric relation is called an ordering.

Thus: \( \leq \) is a reflexive ordering.

\( \mathbb{Z}_\perp, \mathbb{B}_\perp, \mathbb{C}_\perp, \perp \) is the set of floating point numbers

\( \mathbb{L}_\perp, \mathbb{P}_\perp, \mathbb{I}_\perp, \perp \) are flat domains.

\( \mathbb{C} = \{ \text{a, b, c, ...} \} \) set of characters

**Def. 2.1.1 \( \leq \) on base domains**

Let \( D \) be a base domain (i.e., \( \mathbb{Z}_\perp, \mathbb{B}_\perp, \mathbb{C}_\perp, \perp \)). Then for all \( d, d' \in D \) we have: \( d \leq_D d' \) iff \( d = d' \) or \( d = \perp \) or \( d' = \perp \).

Sud domains are called flat. "if and only if"
Now we want to look at a domain for tuple expressions. If \( \exp_1, \exp_2, \ldots, \exp_n \) are expressions where 
\( \text{Val}(\exp_1) \in D_1, \ldots, \text{Val}(\exp_n) \in D_n \), then 
\( \text{Val}(\langle \exp_1, \ldots, \exp_n \rangle) \in D_1 \times \cdots \times D_n \).
So the Cartesian product of domains is again a domain.

**Def 2.1.2 (Product Domains)**
Let \( D_1, \ldots, D_n \) be domains, where \( n \geq 2 \). Then \( D_1 \times \cdots \times D_n \) is also a domain. We define 
\( (d_1, \ldots, d_n) \sim_{D_1 \times \cdots \times D_n} (d'_1, \ldots, d'_n) \) iff \( d_i \sim_{D_i} d'_i \) for all \( 1 \leq i \leq n \).
Thus, the smallest element of \( D_1 \times \cdots \times D_n \) is 
\( \bot_{D_1 \times \cdots \times D_n} = (\bot_{D_1}, \ldots, \bot_{D_n}) \).

**Ex:** \( \mathbb{Z}_1 \times \mathbb{Z}_1 \)

\[
\begin{array}{ccccccc}
(0,0) & & (0,1) & & (1,0) & & (1,1) & \\
\vdots & \times & \times & \times & \times & \vdots & \\
(0,1) & & (1,0) & & (1,1) & & (0,1) & \\
\vdots & & & & & & \vdots & \\
(1,1) & & (1,0) & & (0,1) & & (1,1) & \\
\end{array}
\]

Now we have 3 degrees of definedness.
$\subseteq$ is still a reflexive ordering (on product domains)

Lemma 2.1.3 (\(\subseteq\) on product domains)

If all \(\subseteq_{D_i}\) are reflexive orderings, then 
\(\subseteq_{D_n \times \cdots \times D_n}\) is also reflexive ordering.

Proof:

**Reflexivity:** \((d_n, \ldots, d_n) \subseteq_{D_n \times \cdots \times D_n} (d_n, \ldots, d_n)\), since 
\(d_n \subseteq_{D_n} d_n, \ldots, d_n \subseteq_{D_n} d_n\) (which holds by reflexivity of 
\(\subseteq_{D_n}\)), hence \(\subseteq_{D_n}\).

**Transitivity:** Let \((d_n, \ldots, d_n) \subseteq_{D_n \times \cdots \times D_n} (d'_n, \ldots, d'_n)\) and 
\((d'_n, \ldots, d'_n) \subseteq_{D_n \times \cdots \times D_n} (d''_n, \ldots, d''_n)\).

By definition, we have \(d_i \subseteq_{D_i} d'_i\) and \(d'_i \subseteq_{D_i} d''_i\) for all \(1 \leq i \leq n\). Since \(\subseteq_{D_i}\) is transitive, we have \(d_i \subseteq_{D_i} d''_i\). This implies 
\((d_n, \ldots, d_n) \subseteq_{D_n \times \cdots \times D_n} (d''_n, \ldots, d''_n)\).

**Antisymmetry:** analogous.

For expressions of function type, we also regard functions 
from \(D_1\) to \(D_2\) (for domains \(D_1, D_2\)).
(The corresponding function domain will only consist of a subset of the functions from $D_1 \rightarrow D_2$.)
But $E$ can easily be defined for functions as well.

Def 2.1.4 ( $E$ on functions)

Let $f, g$ be functions from a domain $D_1$ to a domain $D_2$.
Then we define $f \equiv_D g$ iff $f(d) \equiv_{D_2} g(d)$ for all $d \in D_1$.
Thus, the smallest function from $D_1$ to $D_2$ (denoted $\bot_{D_1 \rightarrow D_2}$) is the function that returns $\bot_{D_2}$ for all arguments (of $D_1$).

For functions, there can even be infinite chains such that

$$f_0 \equiv_{D_1 \rightarrow D_2} f_1 \equiv_{D_1 \rightarrow D_2} f_2 \equiv_{D_1 \rightarrow D_2} \ldots$$

Lemma 2.1.5 ($E$ on functions)

If $E_{D_2}$ is a reflexive ordering, then $E_{D_1 \rightarrow D_2}$ is also a reflexive ordering.

Proof: Similar to Lemma 2.1.3