Dienstag, 10. Mai 2016 14:00

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:

Ch2 • denotational semantics: define semantics by a mapping from constructs of the prog. language to mathematical objects

Ch3 • 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: — " ——— 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 $\mathcal{Val}$ that maps every Haskell-expression to a mathematical object.
For an expression $\underline{exp}$, $\mathcal{Val}(\underline{exp})$ should be the meaning/semantics of $\underline{exp}$.

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

What about Haskell-functions?

```
square :: Int -> Int
square x = x * x
```

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

What about the following non-terminating fct?

```
non_term :: Int -> Int
non_term x = non_term (x + 1)
```

What is $\mathcal{Val}(\text{non\_term } 0)$?

To express undefinedness, we introduce a special undefined value $\perp$ ("bottom").

Then $\mathcal{Val}(\text{non\_term } 0) = \mathcal{Val}(\text{non\_term } 1) = \perp$.

So for the semantics of Haskell, we need mathematical objects like $\mathbb{Z}$, $\bot$, $\underset{\substack{\uparrow \\ \text{Booleans}}}{\mathbb{B}}$, tuples, functions, etc.

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 <u>domain</u> of mathematical objects, such that Val maps expressions of this type to objects of the corresponding domain.

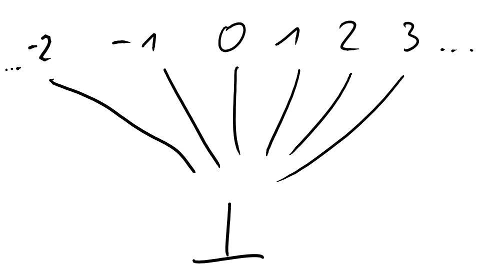
Domain for type Int: $\mathbb{Z}_\bot = \{ \bot_{\mathbb{Z}_\bot}, 0, 1, -1, 2, -2, \ldots \}$
Domain for type Bool: $\mathbb{B}_\bot = \{ \bot_{\mathbb{B}_\bot}, \text{True}, \text{False} \}$
Similarly for Char, Float, ...

For every domain D we use a partial ordering $\sqsubseteq_D$ which compares elements of D by their definedness.
So $x \sqsubseteq_D y$ means that $x, y \in D$ and
x is <u>less defined or equal</u> to y.

So for $\mathbb{Z}_\bot$ we have: $\bot_{\mathbb{Z}_\bot} \sqsubseteq_{\mathbb{Z}_\bot} 0$ (Slide 29)

$\bot \sqsubseteq 1$

... -2 -1 0 1 2 3 ... $\qquad 0 \not\sqsubseteq 1$



$0 \not\sqsubseteq 1$

$1 \not\sqsubseteq 0$

$0 \sqsubseteq 0$

Such domains are called <u>flat</u>, because elements are either completely defined or completely undefined.

"$\sqsubseteq$" is <u>partial</u> because there are elements that can't be compared with "$\sqsubseteq$" (e.g., $0 \not\sqsubseteq 1$, $1 \not\sqsubseteq 0$).

An <u>order</u> (<u>ordering</u>) is a relation with certain properties.

(1) $x \sqsubseteq x$ <u>reflexivity</u>

(2) $x \sqsubseteq y$ and $y \sqsubseteq z$ implies $x \sqsubseteq z$ <u>transitivity</u>

(3) $x \sqsubseteq y$ and $y \sqsubseteq x$ implies $x = y$ <u>antisymmetry</u>

A transitive antisymmetric relation is called an <u>ordering</u>.

Thus: $\sqsubseteq$ is a reflexive ordering

$\mathbb{Z}_\perp$, $\mathbb{B}$, $C_\perp$, $F_\perp$ are <u>flat domains</u>. (F is the set of floating point numbers)

$C = \{'a', 'b', \dots\}$ set of characters

<u>Def. 2.1.1</u> ($\sqsubseteq$ on base domains)

Let D be a base domain (i.e., $\mathbb{Z}_\perp$, $\mathbb{B}_\perp$, $C_\perp$, $F_\perp$).

Then for all $d, d' \in D$ we have: $d \sqsubseteq_D d'$ iff $d = \perp_D$ or $d = d'$.

"if and only if"

Such domains are called <u>flat</u>.

Now we want to look at a domain for tuple expressions.

If $\underline{exp}_1, \underline{exp}_2, \dots, \underline{exp}_n$ are expressions where $Val(\underline{exp}_1) \in D_1, \dots, Val(\underline{exp}_n) \in D_n$ then

$$Val\left((\underline{exp}_1, \dots, \underline{exp}_n) \right) \in D_1 \times \dots \times D_n.$$

So the Cartesian product of domains is again a domain.

<u>Def 2.1.2</u> (Product Domains)

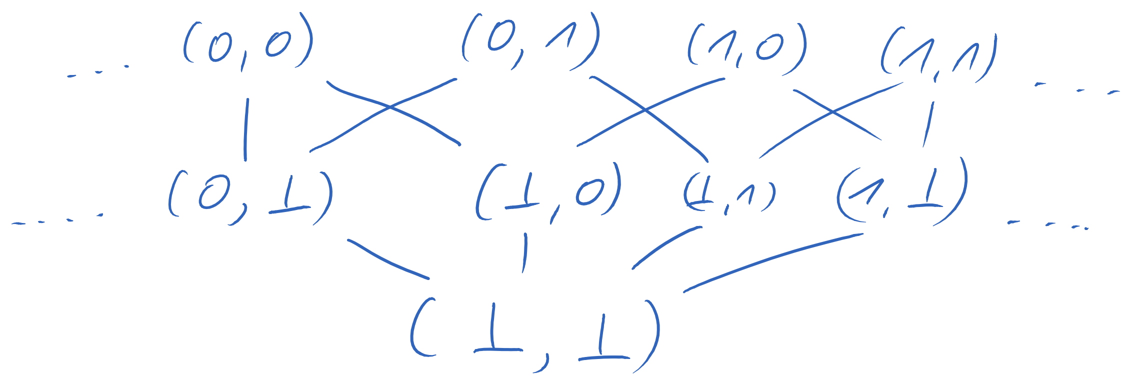
Let $D_1, \dots, D_n$ be domains, where $n \geq 2$. Then $D_1 \times \dots \times D_n$ is also a domain. We define

$$(d_1, \dots, d_n) \sqsubseteq_{D_1 \times \dots \times D_n} (d_1', \dots, d_n') \text{ iff } d_i \sqsubseteq_{D_i} d_i' \text{ for all } 1 \leq i \leq n.$$

Thus, the smallest element of $D_1 \times \dots \times D_n$ is

$$\bot_{D_1 \times \dots \times D_n} = (\bot_{D_1}, \dots, \bot_{D_n}).$$

Ex: $\mathbb{Z}_\bot \times \mathbb{Z}_\bot$



Now we have 3 degrees of definedness.

$\sqsubseteq$ is still a reflexive ordering (on product domains)

<u>Lemma 2.1.3</u> ($\sqsubseteq$ on product domains)
If all $\sqsubseteq_{D_i}$ are reflexive orderings, then $\sqsubseteq_{D_1 \times \ldots \times D_n}$ is also reflexive ordering.

<u>Proof</u>:

<u>Reflexivity</u>: $(d_1, \ldots, d_n) \sqsubseteq_{D_1 \times \ldots \times D_n} (d_1, \ldots, d_n)$, since $d_1 \sqsubseteq_{D_1} d_1, \ldots, d_n \sqsubseteq_{D_n} d_n$, which holds by reflexivity of $\sqsubseteq_{D_1}, \ldots, \sqsubseteq_{D_n}$.

<u>Transitivity</u>: Let $(d_1, \ldots, d_n) \sqsubseteq_{D_1 \times \ldots \times D_n} (d_1', \ldots, d_n')$ and

$$(d_1', \ldots, d_n') \sqsubseteq_{D_1 \times \ldots \times D_n} (d_1'', \ldots, d_n'').$$

By definition, we have $d_i \sqsubseteq_{D_i} d_i'$ and $d_i' \sqsubseteq_{D_i} d_i''$ for all $1 \leq i \leq n$. Since $\sqsubseteq_{D_i}$ is transitive, we have $d_i \sqsubseteq_{D_i} d_i''$. This implies
$(d_1, \ldots, d_n) \sqsubseteq_{D_1 \times \ldots \times D_n} (d_1'', \ldots, d_n'')$.

<u>Antisymmetry</u>: analogous. $\blacksquare$

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 $D_1 \to D_2$.)
But $\sqsubseteq$ can easily be defined for functions as well.

<u>Def 2.1.4</u> ($\sqsubseteq$ on functions)
Let f, g be functions from a domain D_1 to a domain D_2.
Then we define $f \sqsubseteq_{D_1 \to D_2} g$ iff $f(d) \sqsubseteq_{D_2} g(d)$ for all $d \in D_1$.
Thus, the smallest function from D_1 to D_2 (denoted $\bot_{D_1 \to D_2}$) is the function that returns $\bot_{D_2}$ for all arguments (of D_1).

For functions, there can even be <u>infinite</u> chains such that

$$f_0 \sqsubseteq_{D_1 \to D_2} f_1 \sqsubseteq_{D_1 \to D_2} f_2 \sqsubseteq_{D_1 \to D_2} \dots$$

<u>Lemma 2.1.5</u> ($\sqsubseteq$ on functions)
If $\sqsubseteq_{D_2}$ is a reflexive ordering, then $\sqsubseteq_{D_1 \to D_2}$ is also a reflexive ordering.
<u>Proof</u>: Similar to Lemma 2.1.3 $\square$