Denotational semantics: map every Haskell expression to some mathematical object.

Now: How do we map Haskell functions to continuous functions on corresponding domains?

Example without recursion:

\[ \text{conv} :: \text{Bool} \to \text{Int} \]

\[ \text{conv} = \lambda b \to \text{if } b = \text{True} \text{ then } 1 \text{ else } 0 \]

If function declarations have no recursion, we can assume that we already know the semantics of the right-hand sides of let declarations. Then \(\text{conv}\) should get the same semantics as the rhs of its declaration:

\[ \begin{array}{c}
\mathbf{f} : \mathbb{B}_I \to \mathbb{Z}_I \\
\mathbf{f}(b) = \begin{cases} \\
1, & \text{if } b = \text{True} \\
0, & \text{if } b = \text{False} \\
\bot_{\mathbb{Z}_I}, & \text{if } b = \bot_{\mathbb{B}_I} \end{cases}
\end{array} \]

Example with recursion:

\[ \text{fact} :: \text{Int} \to \text{Int} \]

\[ \text{fact} = \lambda x \to \text{if } x \leq 0 \text{ then } 1 \text{ else } \text{fact}(x-1) \times x \]

Semantics of \(\text{fact}\) should be a \(\text{let}\) from \(\mathbb{Z}_I \to \mathbb{Z}_I\).

We would like to compute the semantics of the rhs and then assign this semantics to \(\text{fact}\).

Problem: rhs contains \(\text{fact}\)!
We will now present 2 solutions to this problem, i.e., 2 ways how one could define the semantics of such recursive functions. It will turn out that these 2 alternatives lead to the same result.

Solution 1: Replace the recursive function definition by a sequence of non-recursive function definitions:

We use an auxiliary symbol \( \bot \) that is always undefined (i.e., \( \bot \) has the semantics \( \perp \perp \)).

\[
\begin{align*}
\text{fact}_0 &= \lambda x \rightarrow \bot \\
\text{fact}_n &= \lambda x \rightarrow \text{if } x \leq 0 \text{ then } 1 \text{ else } \text{fact}_0 (x-1) \times x \\
\text{fact}_2 &= \lambda x \rightarrow \text{if } x \leq 0 \text{ then } 1 \text{ else } \text{fact}_1 (x-1) \times x \\
\vdots \\
\end{align*}
\]

Now we can compute the semantics \( \text{fact}_0 \) of \( \text{fact}_0 \), the semantics \( \text{fact}_1 \) of \( \text{fact}_1 \), etc.

\[
\begin{align*}
\text{fact}_0 (x) &= 1 \text{ for all } x \in \mathbb{Z} \\
\text{fact}_n (x) &= \begin{cases} 
 x! & \text{for } 0 \leq x < 1 \\
 1 & \text{for } x < 0 \\
 1 & \text{for } x = 1 \text{ or } 1 \leq x \\
\end{cases} \\
\text{fact}_2 (x) &= \begin{cases} 
 x! & \text{for } 0 \leq x < 2 \\
 1 & \text{for } x < 0 \\
 1 & \text{for } x = 1 \text{ or } 2 \leq x \\
\end{cases} \\
\vdots \\
\text{fact}_n \text{ is like fact, but the } n\text{-th recursive call is replaced by } 1. 
\end{align*}
\]
To obtain the semantics fact for fact, we compute the semantics for its non-recursive approximations fact₀, fact₁, ... and then take their limit, i.e.: 

\[
\text{fact} = \lim \{ \text{fact}_n, \text{fact}_{n+1}, \ldots \}
\]

The step from one approximation \(\text{fact}_n\) to \(\text{fact}_{n+1}\) is done by the following function \(\text{ff} : \langle \mathbb{Z}_+ \to \mathbb{Z}_+ \rangle \to \langle \mathbb{Z}_+ \to \mathbb{Z}_+ \rangle\):

\[
(\text{ff}(g))(x) = \begin{cases} 1 & \text{if } x \leq 0 \\ g(x-1), & \text{otherwise} \end{cases}
\]

\(\text{ff}\) can also be implemented as a Haskell function:

\[
\text{fact}_0 = \text{ff}^0 (1)
\]

\[
\text{fact}_1 = \text{ff}^1 (1)
\]

\[
\text{fact}_2 = \text{ff}^2 (1)
\]

\[
\vdots
\]

\[
\text{fact} = \lim \{ \text{ff}^n (1) \mid n \in \mathbb{N}^2 \}
\]

(Slide 34)

Solution 2: alternative defin. of the semantics of recursively defined functions

Idea: regard the defining equations as constraints. The semantics should be a function that satisfies these constraints.
these constraints.

fact should be a function that satisfies
\[ \text{fact}(x) = \begin{cases} 1 & \text{if } x \leq 0 \\ \text{ff} \left( \text{fact} \right) & \text{else} \end{cases} \]

In other words: fact should be a fixpoint of \( \text{ff} \)
\[ \text{ff} \left( \text{fact} \right) = \text{fact} \]

In general: \( x \) is a fixpoint of a function \( f \) if
\[ f(x) = x. \]

Here: we search for a fixpoint of the function
\[ \text{ff} : \langle \mathbb{Z}_+ \rightarrow \mathbb{Z}_+ \rangle \rightarrow \langle \mathbb{Z}_+ \rightarrow \mathbb{Z}_+ \rangle \]
\[ (\text{ff} \left( g \right)) (x) = \begin{cases} 1, & \text{if } x \leq 0 \\ g(x-1) \cdot x, & \text{otherwise} \end{cases} \]

The only fixpoint of the function \( \text{ff} \) is:
\[ f(x) = \begin{cases} 1, & \text{if } x \leq 0 \\ x !, & \text{if } x > 0 \\ 1, & \text{if } x = 1 \end{cases} \]
\[ (\text{ff} \left( f \right)) (x) = \begin{cases} 1, & \text{if } x \leq 0 \\ f(x-1) \cdot x, & \text{if } x > 0 \\ 1, & \text{if } x = 1 \end{cases} \]

In general, a function can have several fixpoints:
\non-term :: \text{Int} \rightarrow \text{Int}
non-term = \ x \to\ non-term\ (x+1)

The semantics of non-term should now be a fixpoint of
the following higher-order function \(\text{un}\):
\[
\text{un} : \ (\text{Int} \to \text{Int}) \to \ (\text{Int} \to \text{Int})
\]
\[
\text{un} \ g = \ x \to\ g\ (x+1)
\]
\[
(\text{un} \ g)\ (x) = g\ (x+1)
\]

Which functions are fixpoints of \(\text{un}\)?
For which functions \(g\) do we have \(\text{un} \ g = g\) ?
All constant functions!

For the semantics, we should take the smallest
fixpoint (w.r.t. \(\leq\)), i.e., the fixpoint that is
"as undefined as possible". So the semantics of
non-term is \(g : \mathbb{Z}_1 \to \mathbb{Z}_1\) with \(g\ (x) = 1\)
for all \(x \in \mathbb{Z}_1\).

We usually call this the least fixpoint (lfp).

We now have 2 alternative definitions for the
semantics of recursively defined functions:
(Slide 35)
\[
\text{lfp} \ s^f = \bigcup \{ s^i \mid i \in \mathbb{N} \}
\]
These 2 definitions are equivalent!

Thm 2.1.17 (Fixpoint Theorem, Tarski+Kleene)
Let $\mathcal{E}$ be a cpo on $D$ and let $f : D \to D$ be continuous. Then $f$ has a least fixpoint and we have $\operatorname{lfp} f = \bigsqcup \{ f^i(\bot) \mid i \in \mathbb{N} \}$.

Proof: (1) Show that $\{ f^i(\bot) \mid i \in \mathbb{N} \}$ is a chain (i.e., $\bigsqcup \{ f^i(\bot) \mid i \in \mathbb{N} \}$ exists).

To this end, we prove $f^i(\bot) \leq f^{i+1}(\bot)$ by induction on $i$.

Ind. Base ($i = 0$): $f^0(\bot) = f(\bot) \checkmark$

Ind. Step ($i > 0$): Ind. Hyp $f^{i-1}(\bot) \leq f^i(\bot)$.

Since $f$ is continuous, it is also monotonic (Thm. 2.1.15 (a)).

Hence: $f(f^{i-1}(\bot)) \leq f(f^i(\bot))$.

$\therefore f^i(\bot) \leq f^{i+1}(\bot)$

(2) Show that $\bigsqcup \{ f^i(\bot) \mid i \in \mathbb{N} \}$ is a fixpoint of $f$.

$f(\bigsqcup \{ f^i(\bot) \mid i \in \mathbb{N} \}) = \bigsqcup \{ f(\bot) \mid i \in \mathbb{N} \}$ since $f$ is continuous

$\therefore \bigsqcup \{ f^{i+1}(\bot) \mid i \in \mathbb{N} \} = \bigsqcup \{ f^i(\bot) \mid i \in \mathbb{N} \}$ since $\bot$ is the smallest element

$\bigsqcup \{ f^i(\bot) \mid i \in \mathbb{N} \}$

(3) Show that $\bigsqcup \{ f^i(\bot) \mid i \in \mathbb{N} \}$ is smaller or equal to any fixpoint of $f$.

Let $d$ be a fixpoint of $f$.

To show: $\bigsqcup \{ f^i(\bot) \mid i \in \mathbb{N} \} \leq d$
It suffices to show that \( d \) is an upper bound of 
\[ \{ f^i(L) \mid i \in \mathbb{N} \} \].

We show \( f^i(L) \subseteq d \) by induction on \( i \).

**Ind. Base (\( i=0 \)):** \( f^0(L) \subseteq d \) \( \checkmark \)

**Ind. Step (\( i>0 \)):** Ind. Hyp \( f^{i-1}(L) \subseteq d \)

Since \( f \) is continuous and hence, monotonic, this implies
\[ f(f^{i-1}(L)) \subseteq f(d) \]
\[ f^i(L) \subseteq d \] because \( d \) is a fixpoint of \( f \).

Functions like \( f \) that are obtained from programs are computable and therefore always continuous.