Pre-Defined Functions

- `bot` :: `a`
- $\texttt{isa}_{\underline{\texttt{constr}}}$ :: $\underline{\texttt{type}}$ -> `Bool`
  for all $\underline{\texttt{constr}} \in$ Con, where $\underline{\texttt{constr}}$ :: $\underline{\texttt{type}}_1$ -> ... -> $\underline{\texttt{type}}_n$ -> $\underline{\texttt{type}}$
- $\texttt{argof}_{\underline{\texttt{constr}}}$ :: $\underline{\texttt{type}}$ -> $(\underline{\texttt{type}}_1, \ldots, \underline{\texttt{type}}_n)$
  for all $\underline{\texttt{constr}} \in$ Con, wobei $\underline{\texttt{constr}}$ :: $\underline{\texttt{type}}_1$ -> ... -> $\underline{\texttt{type}}_n$ -> $\underline{\texttt{type}}$
- $\texttt{isa}_{n-\texttt{tuple}}$ :: $(\texttt{a}_1, \ldots, \texttt{a}_n)$ -> `Bool`
  for all $n \in \{0, 2, 3, \ldots, k\}$
- $\texttt{sel}_{n,i}$ :: $(\texttt{a}_1, \ldots, \texttt{a}_n)$ -> $\texttt{a}_i$
  for all $2 \leq n \leq k, 1 \leq i \leq n$.

```
append Nil z = z
append (Cons x y) z = Cons x (append y z)
```

$\Downarrow$ (1)

```
append = \x1 x2 -> case (x1, x2) of (Nil, z) -> z
                                    (Cons x y, z) -> Cons x (append y z)
```

$\Downarrow$ (2)

```
append = \x1 ->
         (\x2 -> case (x1, x2) of (Nil, z) -> z
                                  (Cons x y, z) -> Cons x (append y z)).
```

$\Downarrow$ (4)

```
append = \x1 ->
         (\x2 -> match (Nil, z) (x1, x2) z
                 (match (Cons x y, z) (x1, x2) (Cons x (append y z)) bot))
```

```
append = \x1 ->
         (\x2 -> match (Nil, z) (x1, x2) z
                 (match (Cons x y, z) (x1, x2) (Cons x (append y z)) bot))
```

⇓ (9)

```
append = \x1 ->
         (\x2 ->
            match Nil
                  (sel_{2,1} (x1, x2))
                  (match z
                         (sel_{2,2} (x1, x2))
                         z
                         (match (Cons x y)
                                (sel_{2,1} (x1, x2))
                                (match z
                                       (sel_{2,2} (x1, x2))
                                       (Cons x (append y z))
                                       bot)
                                bot))
                  (match (Cons x y)
                         (sel_{2,1} (x1, x2))
                         (match z
                                (sel_{2,2} (x1, x2))
                                (Cons x (append y z))
                                bot)
                         bot))
```

```
append = \x1 -> (\x2 -> match Nil
                        x1
                        (match z
                               x2
                               z
                               (match (Cons x y)
                                      ⋮
                                      )
                        (match (Cons x y)
                               x1
                               (match z
                                      x2
                                      (Cons x (append y z))
                                      bot)
                               bot))
```

⇓ (5)

```
append = \x1 -> (\x2 -> match Nil
                        x1
                        (\z -> z) x2
                        (match (Cons x y)
                               x1
                               (\z -> (Cons x (append y z))) x2
                               bot))
```

```
append = \x1 ->
         (\x2 ->
            match Nil x1 x2
                  (match (Cons x y) x1 (Cons x (append y x2)) bot))
```

⇓ (7)

```
append = \x1 ->
         (\x2 ->
            if   (isa_Nil x1)
            then (match () (argof_Nil x1) x2
                        (if   (isa_Cons x1)
                         then (match (x, y)
                                     (argof_Cons x1)
                                     (Cons x (append y x2))
                                     bot)
                         else bot))
            else (if  (isa_Cons x1)
                  then (match (x, y)
                              (argof_Cons x1)
                              (Cons x (append y x2))
                              bot)
                  else bot))
```

⇓ (8)

```
append = \x1 -> (\x2 ->
           if   (isa_Nil x1)
           then (if   (isa_() (argof_Nil x1))
                 then x2
                 else (if   (isa_Cons x1)
                       then (match (x, y)
                                   (argof_Cons x1)
                                   (Cons x (append y x2))
                                   bot)
                       else bot))
           else (if  (isa_Cons x1)
                 then (match (x, y) (argof_Cons x1) (Cons x (append y x2)) bot)
                 else bot))
```

$\Downarrow$ (9), (5)

```
append = \x1 -> (\x2 ->
           if   (isa_Nil x1)
           then x2
           else (if  (isa_Cons x1)
                 then (Cons (sel_2,1 (argof_Cons x1))
                            (append (sel_2,2 (argof_Cons x1)) x2))
                 else bot))
```

Transformation Into Simple HASKELL-Programs

(1) Transformation of function declarations into pattern declarations

$$\frac{\underline{\text{var}}\ \underline{\text{pat}}_1^1\ \ldots\ \underline{\text{pat}}_n^1\ =\ \underline{\text{exp}}^1;\ \ldots;\ \underline{\text{var}}\ \underline{\text{pat}}_1^k\ \ldots\ \underline{\text{pat}}_n^k\ =\ \underline{\text{exp}}^k}{\begin{array}{l}\underline{\text{var}}\ =\ \backslash\ \mathtt{x}_1\ \ldots\ \mathtt{x}_n\ \texttt{->}\ \texttt{case}\ (\mathtt{x}_1,\ldots,\mathtt{x}_n)\ \texttt{of}\ \{(\underline{\text{pat}}_1^1,\ldots,\underline{\text{pat}}_n^1)\ \texttt{->}\ \underline{\text{exp}}^1;\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\vdots\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad(\underline{\text{pat}}_1^k,\ldots,\underline{\text{pat}}_n^k)\ \texttt{->}\ \underline{\text{exp}}^k\}\end{array}}$$

where $\mathtt{x}_1,\ldots,\mathtt{x}_n$ must be new variables, $n > 0$,
and these are all declarations for $\underline{\text{var}}$

(2) Transforming lambda-expressions with several patterns

$$\frac{\backslash\ \underline{\text{pat}}_1\ \ldots\ \underline{\text{pat}}_n\ \texttt{->}\ \underline{\text{exp}}}{\backslash\ \underline{\text{pat}}_1\ \texttt{->}\ (\backslash\ \underline{\text{pat}}_2\ \texttt{->}\ \ldots\ (\backslash\ \underline{\text{pat}}_n\ \texttt{->}\ \underline{\text{exp}})\ldots)}\quad \text{where } n \geq 2$$

(3) Transforming lambda-patterns into `case`

$$\frac{\texttt{\textbackslash}\ \underline{\text{pat}}\ \texttt{->}\ \underline{\text{exp}}}{\texttt{\textbackslash}\ \underline{\text{var}}\ \texttt{->}\ \texttt{case}\ \underline{\text{var}}\ \texttt{of}\ \underline{\text{pat}}\ \texttt{->}\ \underline{\text{exp}}}$$

where $\underline{\text{pat}}$ is no variable, $\underline{\text{var}}$ is new

(4) Transforming `case` into `match`

$$\frac{\texttt{case}\ \underline{\text{exp}}\ \texttt{of}\ \{\underline{\text{pat}}_1\ \texttt{->}\ \underline{\text{exp}}_1 ; \ldots ; \underline{\text{pat}}_n\ \texttt{->}\ \underline{\text{exp}}_n\}}{\begin{array}{l}\texttt{match}\ \underline{\text{pat}}_1\ \underline{\text{exp}}\ \underline{\text{exp}}_1 \\ \quad (\texttt{match}\ \underline{\text{pat}}_2\ \underline{\text{exp}}\ \underline{\text{exp}}_2 \\ \quad\quad \ldots \\ \quad\quad (\texttt{match}\ \underline{\text{pat}}_n\ \underline{\text{exp}}\ \underline{\text{exp}}_n\ \ \texttt{bot}) \ldots)\end{array}}$$

(5) `match` of variables

$$\frac{\texttt{match}\ \underline{\text{var}}\ \underline{\text{exp}}\ \underline{\text{exp}}_1\ \underline{\text{exp}}_2}{(\texttt{\textbackslash}\ \underline{\text{var}}\ \texttt{->}\ \underline{\text{exp}}_1)\ \underline{\text{exp}}}$$

(6) `match` **of joker pattern**

$$\frac{\texttt{match}\ _\ \underline{\text{exp}}\ \underline{\text{exp}}_1\ \underline{\text{exp}}_2}{\underline{\text{exp}}_1}$$

(7) `match` **of constructors**

$$\frac{\texttt{match}\ (\underline{\text{constr}}\ \underline{\text{pat}}_1\ \ldots\ \underline{\text{pat}}_n)\ \underline{\text{exp}}\ \underline{\text{exp}}_1\ \underline{\text{exp}}_2}{\begin{array}{l}\texttt{if}\ (\texttt{isa}_{\underline{\text{constr}}}\ \underline{\text{exp}})\ \texttt{then}\ (\texttt{match}\ \ (\underline{\text{pat}}_1, \ldots, \underline{\text{pat}}_n) \\ \qquad\qquad\qquad\qquad (\texttt{argof}_{\underline{\text{constr}}}\ \underline{\text{exp}})\ \underline{\text{exp}}_1\ \underline{\text{exp}}_2)\ \texttt{else}\ \underline{\text{exp}}_2\end{array}}$$

(8) `match` **of empty tuple**

$$\frac{\texttt{match}\ (\,)\ \underline{\text{exp}}\ \underline{\text{exp}}_1\ \underline{\text{exp}}_2}{\texttt{if}\ (\texttt{isa}_{\texttt{0-tuple}}\ \underline{\text{exp}})\ \texttt{then}\ \underline{\text{exp}}_1\ \texttt{else}\ \underline{\text{exp}}_2}$$

(9) `match` of non-empty tuple

$$\frac{\texttt{match}\ (\underline{\text{pat}}_1, \ldots, \underline{\text{pat}}_n)\ \underline{\text{exp}}\ \underline{\text{exp}}_1\ \underline{\text{exp}}_2}{\text{where } n \geq 2}$$

$$\begin{array}{l}
\texttt{if}\quad (\texttt{isa}_{n-\texttt{tuple}}\ \underline{\text{exp}}) \\
\texttt{then match}\ \underline{\text{pat}}_1\ (\texttt{sel}_{n,1}\ \underline{\text{exp}}) \\
\qquad (\texttt{match}\ \underline{\text{pat}}_2\ (\texttt{sel}_{n,2}\ \underline{\text{exp}}) \\
\qquad\qquad \ldots\ (\texttt{match}\ \underline{\text{pat}}_n\ (\texttt{sel}_{n,n}\ \underline{\text{exp}})\ \underline{\text{exp}}_1\ \underline{\text{exp}}_2)\ \ldots \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \underline{\text{exp}}_2)\ \underline{\text{exp}}_2 \\
\texttt{else}\ \underline{\text{exp}}_2
\end{array}$$

(10) Separation of Declarations

$$\frac{\texttt{let}\ P\ \texttt{in}\ \underline{\text{exp}}}{\begin{array}{l}\texttt{let}\ P_1\ \texttt{in} \\ \qquad \texttt{let}\ P_2\ \texttt{in} \\ \qquad\qquad \ldots \\ \qquad\qquad\quad \texttt{let}\ P_k\ \texttt{in}\ \underline{\text{exp}}\end{array}}$$

where $P_1, \ldots, P_k$ is a separation of the declaration P

(11) Transforming sequences of declarations into a single declaration

$$\frac{\{\underline{\mathsf{var}}_1 \;=\; \underline{\mathsf{exp}}_1;\; \ldots;\; \underline{\mathsf{var}}_n \;=\; \underline{\mathsf{exp}}_n\}}{(\underline{\mathsf{var}}_1, \ldots, \underline{\mathsf{var}}_n) \;=\; (\underline{\mathsf{exp}}_1, \ldots, \underline{\mathsf{exp}}_n)}$$

where $n \geq 2$, $\underline{\mathsf{var}}_i \neq \underline{\mathsf{var}}_j$ for $i \neq j$ and $\underline{\mathsf{var}}_1 \sim_P \ldots \sim_P \underline{\mathsf{var}}_n$.
Here, $P = \{\underline{\mathsf{var}}_1 \;=\; \underline{\mathsf{exp}}_1; \ldots; \underline{\mathsf{var}}_n \;=\; \underline{\mathsf{exp}}_n\}$.

(12) Declaration of several variables

$$\frac{\texttt{let}\ (\underline{\mathsf{var}}_1, \ldots, \underline{\mathsf{var}}_n) = \underline{\mathsf{exp}}\ \texttt{in}\ \underline{\mathsf{exp}}'}{\begin{array}{l}\texttt{let}\ \underline{\mathsf{var}} = \texttt{match}\ (\underline{\mathsf{var}}_1, \ldots, \underline{\mathsf{var}}_n)\ (\texttt{sel}_{n,1}\ \underline{\mathsf{var}}, \ldots, \texttt{sel}_{n,n}\ \underline{\mathsf{var}})\ \underline{\mathsf{exp}}\ \texttt{bot} \\ \qquad \texttt{in}\ \texttt{match}\ (\underline{\mathsf{var}}_1, \ldots, \underline{\mathsf{var}}_n)\ \underline{\mathsf{var}}\ \ \underline{\mathsf{exp}}'\ \texttt{bot}\end{array}}$$

where $n \geq 2$ and $\underline{\mathsf{var}}$ is a new variable