

Pre-Defined Functions

- $\text{bot} :: a$
- $\text{isa}_{\text{constr}} :: \underline{\text{type}} \rightarrow \text{Bool}$
for all $\text{constr} \in \text{Con}$, where $\text{constr} :: \underline{\text{type}}_1 \rightarrow \dots \rightarrow \underline{\text{type}}_n \rightarrow \underline{\text{type}}$
- $\text{argof}_{\text{constr}} :: \underline{\text{type}} \rightarrow (\underline{\text{type}}_1, \dots, \underline{\text{type}}_n)$
for all $\text{constr} \in \text{Con}$, wobei $\text{constr} :: \underline{\text{type}}_1 \rightarrow \dots \rightarrow \underline{\text{type}}_n \rightarrow \underline{\text{type}}$
- $\text{isa}_{n\text{-tuple}} :: (a_1, \dots, a_n) \rightarrow \text{Bool}$
for all $n \in \{0, 2, 3, \dots, k\}$
- $\text{sel}_{n,i} :: (a_1, \dots, a_n) \rightarrow a_i$
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)

⇓ (1)

append = \x1 x2 -> case (x1, x2) of (Nil, z) -> z

(Cons x y, z) -> Cons x (append y z)

⇓ (2)

append = \x1 ->

(\x2 -> case (x1, x2) of (Nil, z) -> z

(Cons x y, z) -> Cons x (append y z)).

⇓ (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
      (sel2,1 (x1, x2))
      (match z
        (sel2,2 (x1, x2))
        z
        (match (Cons x y)
          (sel2,1 (x1, x2))
          (match z
            (sel2,2 (x1, x2))
            (Cons x (append y z))
            bot)
          bot))
      (match (Cons x y)
        (sel2,1 (x1, x2))
        (match z
          (sel2,2 (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 (isaNil x1)
    then (match () (argofNil x1) x2
          (if (isaCons x1)
              then (match (x, y)
                        (argofCons x1)
                        (Cons x (append y x2))
                        bot)
              else bot))
    else (if (isaCons x1)
            then (match (x, y)
                        (argofCons x1)
                        (Cons x (append y x2))
                        bot)
            else bot))

```

⇓ (8)

```

append = \x1 -> (\x2 ->
  if (isaNil x1)
  then (if (isa() (argofNil x1))
    then x2
    else (if (isaCons x1)
      then (match (x, y)
        (argofCons x1)
        (Cons x (append y x2))
        bot)
      else bot))
  else (if (isaCons x1)
    then (match (x, y) (argofCons x1) (Cons x (append y x2)) bot)
    else bot))

```

⇓ (9), (5)

```

append = \x1 -> (\x2 ->
  if (isaNil x1)
  then x2
  else (if (isaCons x1)
    then (Cons (sel2,1 (argofCons x1))
      (append (sel2,2 (argofCons x1)) x2))
    else bot))

```

Transformation Into Simple HASKELL-Programs

(1) Transformation of function declarations into pattern declarations

$$\frac{\underline{\text{var}} \underline{\text{pat}}_1^1 \dots \underline{\text{pat}}_n^1 = \underline{\text{exp}}^1; \dots; \underline{\text{var}} \underline{\text{pat}}_1^k \dots \underline{\text{pat}}_n^k = \underline{\text{exp}}^k}{\underline{\text{var}} = \lambda x_1 \dots x_n \rightarrow \text{case } (x_1, \dots, x_n) \text{ of } \left\{ \begin{array}{l} (\underline{\text{pat}}_1^1, \dots, \underline{\text{pat}}_n^1) \rightarrow \underline{\text{exp}}^1; \\ \vdots \\ (\underline{\text{pat}}_1^k, \dots, \underline{\text{pat}}_n^k) \rightarrow \underline{\text{exp}}^k \end{array} \right\}}$$

where x_1, \dots, x_n must be new variables, $n > 0$,
and these are all declarations for $\underline{\text{var}}$

(2) Transforming lambda-expressions with several patterns

$$\frac{\lambda \underline{\text{pat}}_1 \dots \underline{\text{pat}}_n \rightarrow \underline{\text{exp}}}{\lambda \underline{\text{pat}}_1 \rightarrow (\lambda \underline{\text{pat}}_2 \rightarrow \dots (\lambda \underline{\text{pat}}_n \rightarrow \underline{\text{exp}}) \dots)} \quad \text{where } n \geq 2$$

(3) **Transforming lambda-patterns into case**

$$\frac{\backslash \underline{\text{pat}} \rightarrow \underline{\text{exp}}}{\backslash \underline{\text{var}} \rightarrow \text{case } \underline{\text{var}} \text{ of } \underline{\text{pat}} \rightarrow \underline{\text{exp}}}$$

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

(4) **Transforming case into match**

$$\frac{\text{case } \underline{\text{exp}} \text{ of } \{ \underline{\text{pat}}_1 \rightarrow \underline{\text{exp}}_1; \dots; \underline{\text{pat}}_n \rightarrow \underline{\text{exp}}_n \}}{\text{match } \underline{\text{pat}}_1 \underline{\text{exp}} \underline{\text{exp}}_1 \\ (\text{match } \underline{\text{pat}}_2 \underline{\text{exp}} \underline{\text{exp}}_2 \\ \dots \\ (\text{match } \underline{\text{pat}}_n \underline{\text{exp}} \underline{\text{exp}}_n \text{ bot}) \dots)}$$

(5) **match of variables**

$$\frac{\text{match } \underline{\text{var}} \underline{\text{exp}} \underline{\text{exp}}_1 \underline{\text{exp}}_2}{(\backslash \underline{\text{var}} \rightarrow \underline{\text{exp}}_1) \underline{\text{exp}}}$$

(6) match of joker pattern

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

(7) match of constructors

$$\frac{\text{match } (\underline{\text{constr}} \underline{\text{pat}}_1 \dots \underline{\text{pat}}_n) \underline{\text{exp}} \underline{\text{exp}}_1 \underline{\text{exp}}_2}{\text{if } (\text{isa}_{\underline{\text{constr}}} \underline{\text{exp}}) \text{ then } (\text{match } (\underline{\text{pat}}_1, \dots, \underline{\text{pat}}_n) (\text{argof}_{\underline{\text{constr}}} \underline{\text{exp}}) \underline{\text{exp}}_1 \underline{\text{exp}}_2) \text{ else } \underline{\text{exp}}_2}$$

(8) match of empty tuple

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

(9) **match of non-empty tuple**

$$\frac{\text{match } (\underline{\text{pat}}_1, \dots, \underline{\text{pat}}_n) \underline{\text{exp}} \underline{\text{exp}}_1 \underline{\text{exp}}_2}{\text{if } (\text{isa}_{n\text{-tuple}} \underline{\text{exp}}) \text{ then match } \underline{\text{pat}}_1 (\text{sel}_{n,1} \underline{\text{exp}}) (\text{match } \underline{\text{pat}}_2 (\text{sel}_{n,2} \underline{\text{exp}}) \dots (\text{match } \underline{\text{pat}}_n (\text{sel}_{n,n} \underline{\text{exp}}) \underline{\text{exp}}_1 \underline{\text{exp}}_2) \dots \underline{\text{exp}}_2) \underline{\text{exp}}_2 \text{ else } \underline{\text{exp}}_2} \text{ where } n \geq 2$$

(10) **Separation of Declarations**

$$\frac{\text{let } P \text{ in } \underline{\text{exp}}}{\text{let } P_1 \text{ in} \quad \text{let } P_2 \text{ in} \quad \dots \quad \text{let } P_k \text{ in } \underline{\text{exp}}}$$

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

(11) Transforming sequences of declarations into a single declaration

$$\frac{\{\underline{\text{var}}_1 = \underline{\text{exp}}_1; \dots; \underline{\text{var}}_n = \underline{\text{exp}}_n\}}{(\underline{\text{var}}_1, \dots, \underline{\text{var}}_n) = (\underline{\text{exp}}_1, \dots, \underline{\text{exp}}_n)}$$

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

(12) Declaration of several variables

$$\frac{\text{let } (\underline{\text{var}}_1, \dots, \underline{\text{var}}_n) = \underline{\text{exp}} \text{ in } \underline{\text{exp}}'}{\text{let } \underline{\text{var}} = \text{match } (\underline{\text{var}}_1, \dots, \underline{\text{var}}_n) \text{ (sel}_{n,1} \underline{\text{var}}, \dots, \text{sel}_{n,n} \underline{\text{var}}) \underline{\text{exp}} \text{ bot} \\ \text{in match } (\underline{\text{var}}_1, \dots, \underline{\text{var}}_n) \underline{\text{var}} \underline{\text{exp}}' \text{ bot}}$$

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