Errors in Haskell

- **let** `average` xs = `div` (`sum` xs) (`length` xs)
  in `average` []

Errors in Haskell

- **let** `average` xs = `div` (`sum` xs) (`length` xs)
  in `average` []

- **let** `tail` ($x : xs$) = xs
  in `tail` []

Errors in Haskell

- **let** `average` *xs* = `div` (`sum` *xs*) (`length` *xs*)
  in `average` []
- **let** `tail` (*x* : *xs*) = *xs*
  in `tail` []
- **if** ⋯ **then** `error` "some string" **else** ⋯

Errors in Haskell

- **let** `average` $xs =$ `div` (`sum` xs) (`length` xs)
  in `average` []

- **let** `tail` $(x : xs) = xs$
  in `tail` []

- **if** $\cdots$ **then** `error` "some string" **else** $\cdots$

- **let** `loop` $=$ `loop`
  in `loop`

Errors in Haskell

- **let** `average` xs = `div` (`sum` xs) (`length` xs)
  in `average` []

- **let** `tail` $(x : xs) = xs$
  in `tail` []

- **if** $\cdots$ **then** `error` "some string" **else** $\cdots$

- **let** `loop` = `loop`
  in `loop`

Traditionell, oft alle Fehlerursachen unter „$\bot$" subsumiert.

Errors in Haskell

- **let** `average` xs = `div` (`sum` xs) (`length` xs)
  in `average` []

- **let** `tail` $(x : xs) = xs$
  in `tail` []

- **if** $\cdots$ **then** `error` "some string" **else** $\cdots$

- **let** `loop` = `loop`
  in `loop`

Traditionell, oft alle Fehlerursachen unter „$\bot$“ subsumiert.

Besser, feinere Unterscheidung. Etwa wie folgt:

Ok v : nicht fehlerbehaftet

Bad "$\cdots$" : endlich fehlerbehaftet

$\bot$: nicht terminierend

Fortpflanzung von Fehlern

- `tail` $[1/0, 2.5] \leadsto \mathit{Ok}\ ((\mathit{Ok}\ 2.5) : (\mathit{Ok}\ []))$

Fortpflanzung von Fehlern

- `tail` $[1/0, 2.5] \rightsquigarrow Ok\ ((Ok\ 2.5) : (Ok\ [\,]))$

- $(\lambda x \to 3)$ (`error` "$\cdots$") $\rightsquigarrow Ok\ 3$

Fortpflanzung von Fehlern

- `tail` $[1/0, 2.5] \rightsquigarrow \mathit{Ok}\ ((\mathit{Ok}\ 2.5) : (\mathit{Ok}\ []))$

- $(\lambda x \rightarrow 3)$ (`error` "$\cdots$") $\rightsquigarrow \mathit{Ok}\ 3$

- (`error` s) $(\cdots) \rightsquigarrow \mathit{Bad}\ s$

Fortpflanzung von Fehlern

- $\texttt{tail}\ [1/0, 2.5] \leadsto \mathit{Ok}\ ((\mathit{Ok}\ 2.5) : (\mathit{Ok}\ []))$
- $(\lambda x \to 3)\ (\texttt{error}\ \text{``}\cdots\text{''}) \leadsto \mathit{Ok}\ 3$
- $(\texttt{error}\ s)\ (\cdots) \leadsto \mathit{Bad}\ s$
- $\textbf{case}\ (\texttt{error}\ s)\ \textbf{of}\ \{\cdots\} \leadsto \mathit{Bad}\ s$

Fortpflanzung von Fehlern

- $\texttt{tail}\ [1/0, 2.5] \ \rightsquigarrow\ \mathit{Ok}\ ((\mathit{Ok}\ 2.5) : (\mathit{Ok}\ []))$
- $(\lambda x \to 3)\ (\texttt{error}\ \text{“}\cdots\text{”}) \ \rightsquigarrow\ \mathit{Ok}\ 3$
- $(\texttt{error}\ s)\ (\cdots) \ \rightsquigarrow\ \mathit{Bad}\ s$
- $\textbf{case}\ (\texttt{error}\ s)\ \textbf{of}\ \{\cdots\} \ \rightsquigarrow\ \mathit{Bad}\ s$
- $(\texttt{error}\ s_1) + (\texttt{error}\ s_2) \ \rightsquigarrow$???

Fortpflanzung von Fehlern

- $\texttt{tail}\ [1/0, 2.5] \ \rightsquigarrow\ \mathit{Ok}\ ((\mathit{Ok}\ 2.5) : (\mathit{Ok}\ []))$
- $(\lambda x \to 3)\ (\texttt{error}\ \text{“}\cdots\text{”}) \ \rightsquigarrow\ \mathit{Ok}\ 3$
- $(\texttt{error}\ s)\ (\cdots) \ \rightsquigarrow\ \mathit{Bad}\ s$
- $\textbf{case}\ (\texttt{error}\ s)\ \textbf{of}\ \{\cdots\} \ \rightsquigarrow\ \mathit{Bad}\ s$
- $(\texttt{error}\ s_1) + (\texttt{error}\ s_2) \ \rightsquigarrow\ ???$

Abhängigkeit von Auswertungsreihenfolge führt zu erheblichen Einschränkungen der Implementationsfreiheit!

Imprecise Error Semantics [Peyton Jones et al. 1999]

Grundidee:

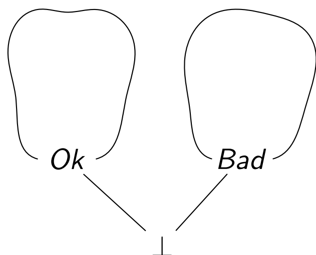
$$\begin{aligned}
\mathit{Ok}\ v &: \text{nicht fehlerbehaftet} \\
\mathit{Bad}\ \{\cdots\} &: \text{endlich fehlerbehaftet, nichtdeterministisch} \\
\bot &: \text{nicht terminierend}
\end{aligned}$$

Imprecise Error Semantics [Peyton Jones et al. 1999]

Grundidee:

$$\begin{aligned} \mathit{Ok}\ v &: \text{nicht fehlerbehaftet} \\ \mathit{Bad}\ \{\cdots\} &: \text{endlich fehlerbehaftet, nichtdeterministisch} \\ \bot &: \text{nicht terminierend} \end{aligned}$$

Definiertheits-Ordnung:

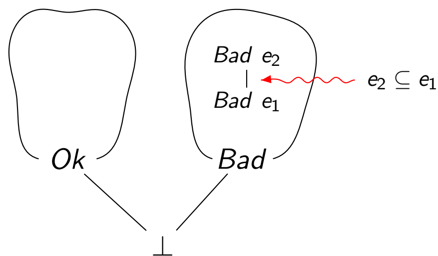


Imprecise Error Semantics [Peyton Jones et al. 1999]

Grundidee:

Ok v	: nicht fehlerbehaftet
Bad $\{\cdots\}$	: endlich fehlerbehaftet, nichtdeterministisch
$\perp$	: nicht terminierend

Definiertheits-Ordnung:



Imprecise Error Semantics [Peyton Jones et al. 1999]

Fortpflanzung von Fehlern:

- $(\texttt{error}\ s_1) + (\texttt{error}\ s_2) \ \rightsquigarrow\ \mathit{Bad}\ \{s_1, s_2\}$

Imprecise Error Semantics [Peyton Jones et al. 1999]

Fortpflanzung von Fehlern:

- $(\texttt{error}\ s_1) + (\texttt{error}\ s_2) \;\leadsto\; \mathit{Bad}\ \{s_1, s_2\}$

- $3 + (\texttt{error}\ s) \;\leadsto\; \mathit{Bad}\ \{s\}$

Imprecise Error Semantics [Peyton Jones et al. 1999]

Fortpflanzung von Fehlern:

- $(\texttt{error}\ s_1) + (\texttt{error}\ s_2) \ \rightsquigarrow\ \mathit{Bad}\ \{s_1, s_2\}$
- $3 + (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s\}$
- $\texttt{loop} + (\texttt{error}\ s) \ \rightsquigarrow\ \bot$

Imprecise Error Semantics [Peyton Jones et al. 1999]

Fortpflanzung von Fehlern:

- $(\texttt{error}\ s_1) + (\texttt{error}\ s_2) \ \rightsquigarrow\ \mathit{Bad}\ \{s_1, s_2\}$
- $3 + (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s\}$
- $\texttt{loop} + (\texttt{error}\ s) \ \rightsquigarrow\ \bot$
- $(\texttt{error}\ s_1)\ (\texttt{error}\ s_2) \ \rightsquigarrow\ \mathit{Bad}\ \{s_1, s_2\}$

Imprecise Error Semantics [Peyton Jones et al. 1999]

Fortpflanzung von Fehlern:

- $(\texttt{error}\ s_1) + (\texttt{error}\ s_2) \ \rightsquigarrow\ \mathit{Bad}\ \{s_1, s_2\}$
- $3 + (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s\}$
- $\texttt{loop} + (\texttt{error}\ s) \ \rightsquigarrow\ \bot$
- $(\texttt{error}\ s_1)\ (\texttt{error}\ s_2) \ \rightsquigarrow\ \mathit{Bad}\ \{s_1, s_2\}$
- $(\lambda x \to 3)\ (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Ok}\ 3$

Imprecise Error Semantics [Peyton Jones et al. 1999]

Fortpflanzung von Fehlern:

- $(\texttt{error}\ s_1) + (\texttt{error}\ s_2) \ \rightsquigarrow\ \mathit{Bad}\ \{s_1, s_2\}$
- $3 + (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s\}$
- $\texttt{loop} + (\texttt{error}\ s) \ \rightsquigarrow\ \bot$
- $(\texttt{error}\ s_1)\ (\texttt{error}\ s_2) \ \rightsquigarrow\ \mathit{Bad}\ \{s_1, s_2\}$
- $(\lambda x \rightarrow 3)\ (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Ok}\ 3$
- $\textbf{case}\ (\texttt{error}\ s_1)\ \textbf{of}\ \{(x, y) \rightarrow \texttt{error}\ s_2\} \ \rightsquigarrow\ \mathit{Bad}\ \{s_1, s_2\}$

Auswirkungen auf Programmäquivalenz

„Normalerweise“:

$$\texttt{takeWhile}\ p\ (\texttt{map}\ h\ l) \quad = \quad \texttt{map}\ h\ (\texttt{takeWhile}\ (p \circ h)\ l)$$

wobei:

$$\begin{array}{l}
\texttt{takeWhile} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha] \\
\texttt{takeWhile}\ p\ [\,] \qquad = [\,] \\
\texttt{takeWhile}\ p\ (a : as) \mid p\ a \qquad\quad = a : \texttt{takeWhile}\ p\ as \\
\qquad\qquad\qquad\qquad\quad\ \mid \texttt{otherwise} = [\,]
\end{array}$$

$$\begin{array}{l}
\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
\texttt{map}\ h\ [\,] \qquad = [\,] \\
\texttt{map}\ h\ (a : as) = h\ a : \texttt{map}\ h\ as
\end{array}$$

Auswirkungen auf Programmäquivalenz

„Normalerweise":

$$\texttt{takeWhile}\ p\ (\texttt{map}\ h\ l) \quad = \quad \texttt{map}\ h\ (\texttt{takeWhile}\ (p \circ h)\ l)$$

wobei:

$$\begin{array}{l}
\texttt{takeWhile} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha] \\
\texttt{takeWhile}\ p\ [\,] \qquad\quad = [\,] \\
\texttt{takeWhile}\ p\ (a : as) \mid p\ a \qquad\quad = a : \texttt{takeWhile}\ p\ as \\
\qquad\qquad\qquad\qquad\quad \mid \texttt{otherwise} = [\,]
\end{array}$$

$$\begin{array}{l}
\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
\texttt{map}\ h\ [\,] \qquad\ = [\,] \\
\texttt{map}\ h\ (a : as) = h\ a : \texttt{map}\ h\ as
\end{array}$$

Aber nun:

$$\begin{array}{c}
\texttt{takeWhile null (map tail (error}\ s\texttt{))} \\
\neq \\
\texttt{map tail (takeWhile (null} \circ \texttt{tail) (error}\ s\texttt{))}
\end{array}$$

Auswirkungen auf Programmäquivalenz

„Normalerweise“:

$$\texttt{takeWhile}\ p\ (\texttt{map}\ h\ l) \quad = \quad \texttt{map}\ h\ (\texttt{takeWhile}\ (p \circ h)\ l)$$

wobei:

$$\begin{array}{l}
\texttt{takeWhile} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha] \\
\texttt{takeWhile}\ p\ [\,] \qquad\qquad = [\,] \\
\texttt{takeWhile}\ p\ (a : as) \ \mid\ p\ a \qquad\qquad = a : \texttt{takeWhile}\ p\ as \\
\qquad\qquad\qquad\qquad\qquad\quad\ \mid\ \texttt{otherwise} = [\,]
\end{array}$$

$$\begin{array}{l}
\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
\texttt{map}\ h\ [\,] \qquad\quad = [\,] \\
\texttt{map}\ h\ (a : as) = h\ a : \texttt{map}\ h\ as
\end{array}$$

Aber nun:

$$\texttt{takeWhile}\ \texttt{null}\ (\texttt{map}\ \texttt{tail}\ (\texttt{error}\ s)) \qquad \text{↯}\, s$$

$$\neq$$

$$\texttt{map}\ \texttt{tail}\ (\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s)) \qquad \text{↯}\, s \text{ oder } \text{↯}\,\text{“empty list”}$$

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s, \text{“empty list”}\}$$

wobei:

$$\begin{aligned}
&\texttt{takeWhile}\ p\ [\,] && = [\,] \\
&\texttt{takeWhile}\ p\ (a : as)\ \mid\ p\ a && = a : (\texttt{takeWhile}\ p\ as) \\
&\phantom{\texttt{takeWhile}\ p\ (a : as)}\ \mid\ \texttt{otherwise} && = [\,]
\end{aligned}$$

$$\begin{aligned}
&\texttt{tail}\ [\,] && = \texttt{error}\ \text{“empty list”} \\
&\texttt{tail}\ (a : as) && = as
\end{aligned}$$

$$\begin{aligned}
&\texttt{null}\ [\,] && = \text{True} \\
&\texttt{null}\ (a : as) && = \text{False}
\end{aligned}$$

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s)\ \rightsquigarrow\ \mathit{Bad}\ \{s, \text{“empty list”}\}$$

wobei:

$$\begin{aligned}
&\texttt{takeWhile}\ p\ [\,] && = [\,] \\
&\texttt{takeWhile}\ p\ (a : as)\ \mid\ p\ a && = a : (\texttt{takeWhile}\ p\ as) \\
&\phantom{\texttt{takeWhile}\ p\ (a : as)}\ \mid\ \texttt{otherwise} && = [\,]
\end{aligned}$$

$$\begin{aligned}
&\texttt{tail}\ [\,] && = \texttt{error}\ \text{“empty list”} \\
&\texttt{tail}\ (a : as) && = as
\end{aligned}$$

$$\begin{aligned}
&\texttt{null}\ [\,] && = \text{True} \\
&\texttt{null}\ (a : as) && = \text{False}
\end{aligned}$$

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s, \text{“empty list”}\}$$

wobei:

$$\begin{array}{lll} \texttt{takeWhile}\ p\ [\,] & & = [\,] \\ \texttt{takeWhile}\ p\ (a : as) & \mid\ p\ a & = a : (\texttt{takeWhile}\ p\ as) \\ & \mid\ \texttt{otherwise} & = [\,] \end{array}$$

$$\begin{array}{ll} \texttt{tail}\ [\,] & = \texttt{error}\ \text{“empty list”} \\ \texttt{tail}\ (a : as) & = as \end{array}$$

$$\begin{array}{ll} \texttt{null}\ [\,] & = \text{True} \\ \texttt{null}\ (a : as) & = \text{False} \end{array}$$

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s, \text{"empty list"}\}$$

wobei:

```
takeWhile p []                   = []
takeWhile p (a : as) | p a       = a : (takeWhile p as)
                     | otherwise = []

tail []       = error "empty list"
tail (a : as) = as

null []       = True
null (a : as) = False
```

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s, \text{“empty list”}\}$$

wobei:

$$\begin{aligned}
&\texttt{takeWhile}\ p\ []\quad && = []\\
&\texttt{takeWhile}\ p\ (a : as)\ && \mid\ p\ a\ && = a : (\texttt{takeWhile}\ p\ as)\\
& && \mid\ \texttt{otherwise} && = []
\end{aligned}$$

$$\begin{aligned}
\texttt{tail}\ []\ &= \texttt{error}\ \text{“empty list”}\\
\texttt{tail}\ (a : as) &= as
\end{aligned}$$

$$\begin{aligned}
\texttt{null}\ []\ &= \text{True}\\
\texttt{null}\ (a : as) &= \text{False}
\end{aligned}$$

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s, \text{“empty list”}\}$$

wobei:

$$\begin{aligned}
&\texttt{takeWhile}\ p\ [\,] && = [\,] \\
&\texttt{takeWhile}\ p\ (a : as)\ \mid\ p\ a && = a : (\texttt{takeWhile}\ p\ as) \\
&\phantom{\texttt{takeWhile}\ p\ (a : as)}\ \mid\ \texttt{otherwise} && = [\,]
\end{aligned}$$

$$\begin{aligned}
&\texttt{tail}\ [\,] && = \texttt{error}\ \text{“empty list”} \\
&\texttt{tail}\ (a : as) && = as
\end{aligned}$$

$$\begin{aligned}
&\texttt{null}\ [\,] && = \text{True} \\
&\texttt{null}\ (a : as) && = \text{False}
\end{aligned}$$

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s, \text{“empty list”}\}$$

wobei:

$$\begin{array}{lll} \texttt{takeWhile}\ p\ [\,] & = [\,] & \\ \texttt{takeWhile}\ p\ (a : as) & \mid\ p\ a & = a : (\texttt{takeWhile}\ p\ as) \\ & \mid\ \texttt{otherwise} & = [\,] \end{array}$$

$$\begin{array}{ll} \texttt{tail}\ [\,] & = \texttt{error}\ \text{“empty list”} \\ \texttt{tail}\ (a : as) & = as \end{array}$$

$$\begin{array}{ll} \texttt{null}\ [\,] & = \text{True} \\ \texttt{null}\ (a : as) & = \text{False} \end{array}$$

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s, \text{“empty list”}\}$$

während:

$$\texttt{takeWhile null}\ (\texttt{map tail}\ (\texttt{error}\ s)) \ \rightsquigarrow\ \mathit{Bad}\ \{s\}$$

wobei:

$$\begin{aligned}
&\texttt{takeWhile}\ p\ []\ &&= []\\
&\texttt{takeWhile}\ p\ (a : as)\ &&\mid p\ a = a : (\texttt{takeWhile}\ p\ as)\\
& &&\mid \texttt{otherwise} = []
\end{aligned}$$

$$\begin{aligned}
&\texttt{map}\ h\ [] &&= []\\
&\texttt{map}\ h\ (a : as) &&= (h\ a) : (\texttt{map}\ h\ as)
\end{aligned}$$

$$\begin{aligned}
&\texttt{null}\ [] &&= \text{True}\\
&\texttt{null}\ (a : as) &&= \text{False}
\end{aligned}$$

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s, \text{“empty list”}\}$$

während:

$$\texttt{takeWhile}\ \texttt{null}\ (\texttt{map}\ \texttt{tail}\ (\texttt{error}\ s)) \ \rightsquigarrow\ \mathit{Bad}\ \{s\}$$

wobei:

$$\begin{aligned}
&\texttt{takeWhile}\ p\ [\,] && = [\,] \\
&\texttt{takeWhile}\ p\ (a : as)\ \mid\ p\ a && = a : (\texttt{takeWhile}\ p\ as) \\
&\phantom{\texttt{takeWhile}\ p\ (a : as)}\ \mid\ \texttt{otherwise} && = [\,]
\end{aligned}$$

$$\begin{aligned}
&\texttt{map}\ h\ [\,] && = [\,] \\
&\texttt{map}\ h\ (a : as) && = (h\ a) : (\texttt{map}\ h\ as)
\end{aligned}$$

$$\begin{aligned}
&\texttt{null}\ [\,] && = \text{True} \\
&\texttt{null}\ (a : as) && = \text{False}
\end{aligned}$$

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s, \text{“empty list”}\}$$

während:

$$\texttt{takeWhile}\ \texttt{null}\ (\texttt{map}\ \texttt{tail}\ (\texttt{error}\ s)) \ \rightsquigarrow\ \mathit{Bad}\ \{s\}$$

wobei:

```
takeWhile p []           = []
takeWhile p (a : as) | p a       = a : (takeWhile p as)
                     | otherwise = []

map h []       = []
map h (a : as) = (h a) : (map h as)

null []       = True
null (a : as) = False
```

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s)\ \rightsquigarrow\ \mathit{Bad}\ \{s, \text{“empty list”}\}$$

während:

$$\texttt{takeWhile}\ \texttt{null}\ (\texttt{map}\ \texttt{tail}\ (\texttt{error}\ s))\ \rightsquigarrow\ \mathit{Bad}\ \{s\}$$

wobei:

$$\begin{aligned}
&\texttt{takeWhile}\ p\ [\,] && = [\,] \\
&\texttt{takeWhile}\ p\ (a : as)\ \mid\ p\ a && = a : (\texttt{takeWhile}\ p\ as) \\
&\qquad\qquad\qquad\qquad\ \mid\ \texttt{otherwise} && = [\,]
\end{aligned}$$

$$\begin{aligned}
&\texttt{map}\ h\ [\,] && = [\,] \\
&\texttt{map}\ h\ (a : as) && = (h\ a) : (\texttt{map}\ h\ as)
\end{aligned}$$

$$\begin{aligned}
&\texttt{null}\ [\,] && = \text{True} \\
&\texttt{null}\ (a : as) && = \text{False}
\end{aligned}$$

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s) \leadsto \mathit{Bad}\ \{s, \text{“empty list”}\}$$

während:

$$\texttt{takeWhile}\ \texttt{null}\ (\texttt{map}\ \texttt{tail}\ (\texttt{error}\ s)) \leadsto \mathit{Bad}\ \{s\}$$

wobei:

$$\begin{array}{ll} \texttt{takeWhile}\ p\ [\,] & = [\,] \\ \texttt{takeWhile}\ p\ (a : as) & \mid p\ a \qquad\quad = a : (\texttt{takeWhile}\ p\ as) \\ & \mid \texttt{otherwise} = [\,] \end{array}$$

$$\begin{array}{ll} \texttt{map}\ h\ [\,] & = [\,] \\ \texttt{map}\ h\ (a : as) & = (h\ a) : (\texttt{map}\ h\ as) \end{array}$$

$$\begin{array}{ll} \texttt{null}\ [\,] & = \text{True} \\ \texttt{null}\ (a : as) & = \text{False} \end{array}$$

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s, \text{“empty list”}\}$$

während:

$$\texttt{takeWhile}\ \texttt{null}\ (\texttt{map}\ \texttt{tail}\ (\texttt{error}\ s)) \ \rightsquigarrow\ \mathit{Bad}\ \{s\}$$

wobei:

$$\begin{array}{lll} \texttt{takeWhile}\ p\ [\,] & & = [\,] \\ \texttt{takeWhile}\ p\ (a : as) & \mid\ p\ a & = a : (\texttt{takeWhile}\ p\ as) \\ & \mid\ \texttt{otherwise} & = [\,] \end{array}$$

$$\begin{array}{ll} \texttt{map}\ h\ [\,] & = [\,] \\ \texttt{map}\ h\ (a : as) & = (h\ a) : (\texttt{map}\ h\ as) \end{array}$$

$$\begin{array}{ll} \texttt{null}\ [\,] & = \text{True} \\ \texttt{null}\ (a : as) & = \text{False} \end{array}$$

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s) \leadsto \mathit{Bad}\ \{s, \text{“empty list”}\}$$

während:

$$\texttt{takeWhile}\ \texttt{null}\ (\texttt{map}\ \texttt{tail}\ (\texttt{error}\ s)) \leadsto \mathit{Bad}\ \{s\}$$

Also:

$$\texttt{takeWhile}\ \texttt{null}\ (\texttt{map}\ \texttt{tail}\ (\texttt{error}\ s)) \neq \texttt{map}\ \texttt{tail}\ (\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s))$$

Auswirkungen auf Programmäquivalenz

Denn:

$$\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s) \ \rightsquigarrow\ \mathit{Bad}\ \{s, \text{“empty list”}\}$$

während:

$$\texttt{takeWhile}\ \texttt{null}\ (\texttt{map}\ \texttt{tail}\ (\texttt{error}\ s)) \ \rightsquigarrow\ \mathit{Bad}\ \{s\}$$

Also:

$$\begin{array}{c} \texttt{takeWhile}\ \texttt{null}\ (\texttt{map}\ \texttt{tail}\ (\texttt{error}\ s)) \\ \neq \\ \texttt{map}\ \texttt{tail}\ (\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s)) \end{array}$$

Man stelle sich dies in folgendem Kontext vor:

$$\begin{array}{l} \texttt{catchJust}\ \texttt{errorCalls}\ (\texttt{evaluate}\ \cdots) \\ \qquad (\lambda s \to \textbf{if}\ s == \text{“empty list”} \\ \qquad\qquad \textbf{then}\ \texttt{return}\ [[42]] \\ \qquad\qquad \textbf{else}\ \ \texttt{return}\ [\,]) \end{array}$$

Freies Theorem

Bisheriger Kenntnisstand:

$$\texttt{g}\ p\ (\texttt{map}\ h\ l) \quad = \quad \texttt{map}\ h\ (\texttt{g}\ (p \circ h)\ l)$$

für jedes $\texttt{g} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$, wenn

- $p \neq \bot$,
- h strikt ($h\ \bot = \bot$) und
- h total ($\forall x \neq \bot.\ h\ x \neq \bot$).

Freies Theorem

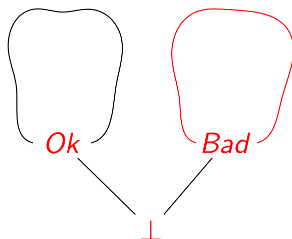
Bisheriger Kenntnisstand:

$$\texttt{g}\ p\ (\texttt{map}\ h\ l) \quad = \quad \texttt{map}\ h\ (\texttt{g}\ (p \circ h)\ l)$$

für jedes $\texttt{g} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$, wenn

- $p \neq \bot$,
- h strikt ($h\ \bot = \bot$) und
- h total ($\forall x \neq \bot.\ h\ x \neq \bot$).

Was sind entsprechende Bedingungen „in echt“?



Schweiß und Tränen . . .

. . . durchgängige Formalisierung der Semantik

Schweiß und Tränen . . .

. . . durchgängige Formalisierung der Semantik

. . . Einstieg in Beweis des Parametrizitäts-Theorems

Schweiß und Tränen . . .

. . . durchgängige Formalisierung der Semantik

. . . Einstieg in Beweis des Parametrizitäts-Theorems

. . . Betrachtung der interessanten Induktionsfälle

Schweiß und Tränen ...

...durchgängige Formalisierung der Semantik

...Einstieg in Beweis des Parametrizitäts-Theorems

...Betrachtung der interessanten Induktionsfälle

...Identifizierung geeigneter Bedingungen auf Relationsebene

Schweiß und Tränen . . .

. . . durchgängige Formalisierung der Semantik

. . . Einstieg in Beweis des Parametrizitäts-Theorems

. . . Betrachtung der interessanten Induktionsfälle

. . . Identifizierung geeigneter Bedingungen auf Relationsebene

. . . Anpassung relationaler Aktionen

Schweiß und Tränen ...

... durchgängige Formalisierung der Semantik

... Einstieg in Beweis des Parametrizitäts-Theorems

... Betrachtung der interessanten Induktionsfälle

... Identifizierung geeigneter Bedingungen auf Relationsebene

... Anpassung relationaler Aktionen

... Vervollständigung allgemeiner Beweis

Schweiß und Tränen . . .

. . . durchgängige Formalisierung der Semantik

. . . Einstieg in Beweis des Parametrizitäts-Theorems

. . . Betrachtung der interessanten Induktionsfälle

. . . Identifizierung geeigneter Bedingungen auf Relationsebene

. . . Anpassung relationaler Aktionen

. . . Vervollständigung allgemeiner Beweis

. . . Übertragung der Bedingungen auf Funktionsebene

Schweiß und Tränen ...

... durchgängige Formalisierung der Semantik

... Einstieg in Beweis des Parametrizitäts-Theorems

... Betrachtung der interessanten Induktionsfälle

... Identifizierung geeigneter Bedingungen auf Relationsebene

... Anpassung relationaler Aktionen

... Vervollständigung allgemeiner Beweis

... Übertragung der Bedingungen auf Funktionsebene

... Anwendung auf konkrete Funktionen

…Anwendung auf „takeWhile“

Für jedes $\texttt{g} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$,

$$\texttt{g}\ p\ (\texttt{map}\ h\ l) \quad = \quad \texttt{map}\ h\ (\texttt{g}\ (p \circ h)\ l)$$

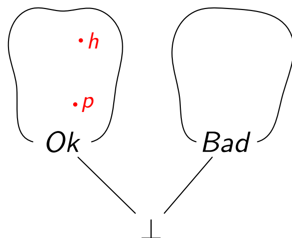
... Anwendung auf „takeWhile"

Für jedes $\mathtt{g} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$,

$$\mathtt{g}\ p\ (\mathtt{map}\ h\ l) \quad = \quad \mathtt{map}\ h\ (\mathtt{g}\ (p \circ h)\ l)$$

vorausgesetzt

- p und h nicht fehlerbehaftet,



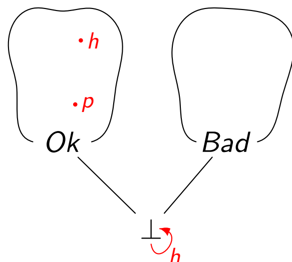
…Anwendung auf „takeWhile“

Für jedes $\texttt{g} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$,

$$\texttt{g}\ p\ (\texttt{map}\ h\ l) \quad = \quad \texttt{map}\ h\ (\texttt{g}\ (p \circ h)\ l)$$

vorausgesetzt

- p und h nicht fehlerbehaftet,
- $h \perp = \perp$,



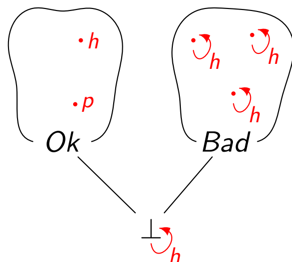
...Anwendung auf „takeWhile“

Für jedes g :: $(\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$,

$$\texttt{g}\ p\ (\texttt{map}\ h\ l) \quad = \quad \texttt{map}\ h\ (\texttt{g}\ (p \circ h)\ l)$$

vorausgesetzt

- p und h nicht fehlerbehaftet,
- $h \perp = \perp$,
- h verhält sich als Identität auf endlichen Fehlern, und



…Anwendung auf „takeWhile“

Für jedes $\texttt{g} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$,

$$\texttt{g}\ p\ (\texttt{map}\ h\ l) \quad = \quad \texttt{map}\ h\ (\texttt{g}\ (p \circ h)\ l)$$

vorausgesetzt

- p und h nicht fehlerbehaftet,
- $h \perp = \perp$,
- h verhält sich als Identität auf endlichen Fehlern, und
- h bildet Nichtfehler auf Nichtfehler ab.