Parametricity for Haskell with Imprecise Error Semantics

Florian Stenger[1] and Janis Voigtländer

Technische Universität Dresden

TLCA'09

[1]Supported by the DFG under grant VO 1512/1-1.

Parametricity for Haskell with Imprecise Error Semantics

Florian Stenger[1] and Janis Voigtländer

Technische Universität Dresden

TLCA'09
=
Applications

[1]Supported by the DFG under grant VO 1512/1-1.

Reasoning in Haskell: An Example

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

Reading Haskell: An Example

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

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

Reasoning in Haskell: An Example

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

$$
\begin{aligned}
&\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
&\texttt{map}\ f\ [\,] \qquad\ \ = [\,] \\
&\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as)
\end{aligned}
$$

For every choice of p, f, and l:

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

Provable by induction.

Reasoning in Haskell: An Example

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

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

For every choice of p, f, and l:

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

Provable by induction.

Or as a "free theorem" [Wadler, FPCA'89].

Reasoning in Haskell: An Example

$\texttt{takeWhile} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$

$\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta]$
$\texttt{map}\ f\ [\,] \quad = [\,]$
$\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as)$

For every choice of p, f, and l:

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

Provable by induction.

Or as a "free theorem" [Wadler, FPCA'89].

Reasoning in Haskell: An Example

$$\texttt{takeWhile} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

$$\texttt{filter} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

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

For every choice of p, f, and l:

$$\begin{array}{rcl}
\texttt{takeWhile}\ p\ (\texttt{map}\ f\ l) & = & \texttt{map}\ f\ (\texttt{takeWhile}\ (p \circ f)\ l) \\
\texttt{filter}\ p\ (\texttt{map}\ f\ l) & = & \texttt{map}\ f\ (\texttt{filter}\ (p \circ f)\ l)
\end{array}$$

Reasoning in Haskell: An Example

$$\texttt{takeWhile} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

$$\texttt{filter} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

$$\texttt{g} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

$$\begin{aligned}
&\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
&\texttt{map}\ f\ [\,] \qquad\quad = [\,] \\
&\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as)
\end{aligned}$$

For every choice of p, f, and l:

$$\begin{aligned}
\texttt{takeWhile}\ p\ (\texttt{map}\ f\ l) &= \texttt{map}\ f\ (\texttt{takeWhile}\ (p \circ f)\ l) \\
\texttt{filter}\ p\ (\texttt{map}\ f\ l) &= \texttt{map}\ f\ (\texttt{filter}\ (p \circ f)\ l) \\
\texttt{g}\ p\ (\texttt{map}\ f\ l) &= \texttt{map}\ f\ (\texttt{g}\ (p \circ f)\ l)
\end{aligned}$$

Errors in Haskell

Errors in Haskell

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

Errors in Haskell

- $\textbf{let}$ `average` $l =$ `div` $($`sum` $l)$ $($`length` $l)$
  $\textbf{in}$ `average` $[\,]$
- $\textbf{let}$ `tail` $(a : as) = as$
  $\textbf{in}$ `tail` $[\,]$

Errors in Haskell

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

- **let** `tail` $(a : as) = as$
  in `tail` []

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

Errors in Haskell

- **let** `average` $l =$ `div` $($`sum` $l)$ $($`length` $l)$
  in `average` $[\,]$
- **let** `tail` $(a : as) = as$
  in `tail` $[\,]$
- **if** $\cdots$ **then** `error` "some string" **else** $\cdots$
- **let** `loop` $=$ `loop`
  in `loop`

Errors in Haskell

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

- **let** `tail` $(a : as) = as$
  in `tail` []

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

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

Traditionally, all error causes subsumed under "$\perp$".

Errors in Haskell

- **let** `average` $l =$ `div` $($`sum` $l)$ $($`length` $l)$
  in `average` $[\,]$

- **let** `tail` $(a : as) = as$
  in `tail` $[\,]$

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

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

Traditionally, all error causes subsumed under "$\perp$".

Better, explicit distinction. Like:

$$\begin{array}{rcl} \mathit{Ok}\ v & : & \text{nonerroneous} \\ \mathit{Bad}\ \text{“}\cdots\text{”} & : & \text{finitely failing} \\ \perp & : & \text{nonterminating} \end{array}$$

Naive Propagation of Errors

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

Naive Propagation of Errors

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

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

Naive Propagation of Errors

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

Naive Propagation of Errors

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

Naive Propagation of Errors

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

Naive Propagation of Errors

- $\texttt{tail}\ [1/0, 2.5] \ \leadsto \ Ok\ ((Ok\ 2.5) : (Ok\ []))$
- $(\lambda x \rightarrow 3)\ (\texttt{error}\ \text{``}\cdots\text{''}) \ \leadsto \ Ok\ 3$
- $(\texttt{error}\ s)\ (\cdots) \ \leadsto \ Bad\ s$
- $\textbf{case}\ (\texttt{error}\ s)\ \textbf{of}\ \{\cdots\} \ \leadsto \ Bad\ s$
- $(\texttt{error}\ s_1) + (\texttt{error}\ s_2) \ \leadsto \ ???$

Dependence on evaluation order leads to considerably less freedom for implementors to rearrange computations, to optimise!

Imprecise Error Semantics [Peyton Jones et al., PLDI'99]

Basic idea:

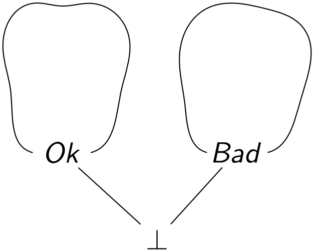
$$
\begin{aligned}
\mathit{Ok}\ v &: \text{nonerroneous} \\
\mathit{Bad}\ \{\cdots\} &: \text{finitely failing, nondeterministic} \\
\bot &: \text{nonterminating}
\end{aligned}
$$

Imprecise Error Semantics [Peyton Jones et al., PLDI'99]

Basic idea:

$$\begin{aligned} \mathit{Ok}\ v &: \text{nonerroneous} \\ \mathit{Bad}\ \{\cdots\} &: \text{finitely failing, nondeterministic} \\ \bot &: \text{nonterminating} \end{aligned}$$

Definedness order:

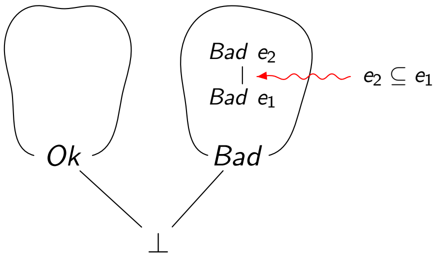


Imprecise Error Semantics [Peyton Jones et al., PLDI'99]

Basic idea:

$$\begin{array}{rcl} \mathit{Ok}\ v & : & \text{nonerroneous} \\ \mathit{Bad}\ \{\cdots\} & : & \text{finitely failing, nondeterministic} \\ \bot & : & \text{nonterminating} \end{array}$$

Definedness order:



Imprecise Error Semantics [Peyton Jones et al., PLDI'99]

Actual Propagation of Errors:

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

Imprecise Error Semantics [Peyton Jones et al., PLDI'99]

Actual Propagation of Errors:

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

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

Imprecise Error Semantics [Peyton Jones et al., PLDI'99]

Actual Propagation of Errors:

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

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

- $\texttt{loop} + (\texttt{error}\ s) \ \rightsquigarrow\ \bot$

Imprecise Error Semantics [Peyton Jones et al., PLDI'99]

Actual Propagation of Errors:

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

Imprecise Error Semantics [Peyton Jones et al., PLDI'99]

Actual Propagation of Errors:

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

Imprecise Error Semantics [Peyton Jones et al., PLDI'99]

Actual Propagation of Errors:

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

Impact on Program Equivalence

"Normally":

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

where:

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

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

Impact on Program Equivalence

"Normally":

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

where:

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

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

But now:

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

$$\neq$$

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

Impact on Program Equivalence

"Normally":

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

where:

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

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

But now:

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

$$\neq$$

$$\texttt{map}\ \texttt{tail}\ (\texttt{takeWhile}\ (\texttt{null} \circ \texttt{tail})\ (\texttt{error}\ s)) \quad \lightning\, s \text{ or } \lightning\, \text{"empty list"}$$

Impact on Program Equivalence

Because:

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

where:

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

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

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

Impact on Program Equivalence

Because:

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

where:

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

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

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

Impact on Program Equivalence

Because:

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

where:

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

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

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

Impact on Program Equivalence

Because:

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

where:

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

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

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

Impact on Program Equivalence

Because:

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

where:

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

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

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

Impact on Program Equivalence

Because:

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

where:

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

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

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

Impact on Program Equivalence

Because:

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

where:

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

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

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

Impact on Program Equivalence

Because:

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

while:

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

where:

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

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

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

Impact on Program Equivalence

Because:

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

while:

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

where:

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

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

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

Impact on Program Equivalence

Because:

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

while:

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

where:

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

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

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

Impact on Program Equivalence

Because:

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

while:

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

where:

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

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

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

Impact on Program Equivalence

Because:

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

while:

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

where:

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

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

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

Impact on Program Equivalence

Because:

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

while:

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

where:

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

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

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

Impact on Program Equivalence

Because:

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

while:

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

Thus:

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

Impact on Program Equivalence

Because:

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

while:

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

Thus:

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

Now, imagine this in the following program context:

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

How to Revise Free Theorems?

[Wadler, FPCA'89] : for every $\texttt{g} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$,

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

How to Revise Free Theorems?

[Wadler, FPCA'89] : for every $\texttt{g} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$,

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

- if f strict.

How to Revise Free Theorems?

[Wadler, FPCA'89] : for every $\texttt{g} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$,

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

- if f strict.

[Johann & V., POPL'04] : in presence of **seq**, if additionally:

- $p \neq \bot$ and
- f total.

How to Revise Free Theorems?

[Wadler, FPCA'89] : for every $\mathtt{g} :: (\alpha \to \mathrm{Bool}) \to [\alpha] \to [\alpha]$,

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

- if f strict ($f\ \perp = \perp$).

[Johann & V., POPL'04] : in presence of **seq**, if additionally:

- $p \neq \perp$ and
- f total ($\forall x \neq \perp.\ f\ x \neq \perp$).

How to Revise Free Theorems?

[Wadler, FPCA'89] : for every g :: $(\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$,

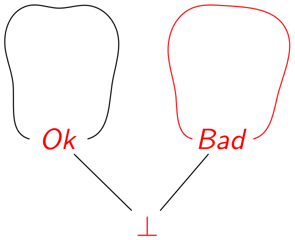
$$\texttt{g}\ p\ (\texttt{map}\ f\ l) \;=\; \texttt{map}\ f\ (\texttt{g}\ (p \circ f)\ l)$$

- if f strict ($f\ \bot = \bot$).

[Johann & V., POPL'04] : in presence of **seq**, if additionally:

- $p \neq \bot$ and
- f total ($\forall x \neq \bot.\ f\ x \neq \bot$).

What are corresponding conditions "in real"?



Sweat and Tears . . .

. . . provide full formalisation of the semantic setup

Sweat and Tears . . .

. . . provide full formalisation of the semantic setup

. . . enter general proof of parametricity theorem

Sweat and Tears . . .

. . . provide full formalisation of the semantic setup

. . . enter general proof of parametricity theorem

. . . consider the interesting induction cases

Sweat and Tears ...

...provide full formalisation of the semantic setup

...enter general proof of parametricity theorem

...consider the interesting induction cases

...identify appropriate restrictions on the level of relations

Sweat and Tears ...

... provide full formalisation of the semantic setup

... enter general proof of parametricity theorem

... consider the interesting induction cases

... identify appropriate restrictions on the level of relations

... adapt relational actions

Sweat and Tears . . .

. . . provide full formalisation of the semantic setup

. . . enter general proof of parametricity theorem

. . . consider the interesting induction cases

. . . identify appropriate restrictions on the level of relations

. . . adapt relational actions

. . . complete general proof

Sweat and Tears . . .

. . . provide full formalisation of the semantic setup

. . . enter general proof of parametricity theorem

. . . consider the interesting induction cases

. . . identify appropriate restrictions on the level of relations

. . . adapt relational actions

. . . complete general proof

. . . transfer restrictions to the level of functions

Sweat and Tears ...

... provide full formalisation of the semantic setup

... enter general proof of parametricity theorem

... consider the interesting induction cases

... identify appropriate restrictions on the level of relations

... adapt relational actions

... complete general proof

... transfer restrictions to the level of functions

... apply to concrete functions

. . . Application to `takeWhile`

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

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

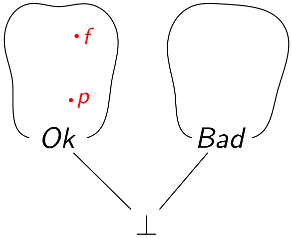
. . . Application to `takeWhile`

For every `g` :: $(\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$,

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

provided

- p and f are nonerroneous,



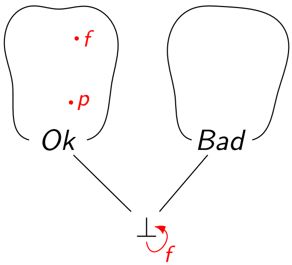
... Application to `takeWhile`

For every $\texttt{g} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$,

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

provided

- p and f are nonerroneous,
- $f\ \perp = \perp$,



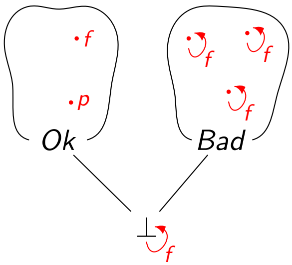
. . . Application to takeWhile

For every $\texttt{g} :: (\alpha \to \mathrm{Bool}) \to [\alpha] \to [\alpha]$,

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

provided

- p and f are nonerroneous,
- $f\ \bot = \bot$,
- f acts as identity on erroneous values, and



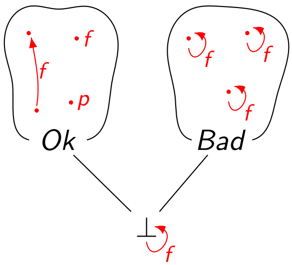
...Application to `takeWhile`

For every `g` :: $(\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$,

$$\mathtt{g}\ p\ (\mathtt{map}\ f\ l) \;=\; \mathtt{map}\ f\ (\mathtt{g}\ (p \circ f)\ l)$$

provided

- p and f are nonerroneous,
- $f\ \bot = \bot$,
- f acts as identity on erroneous values, and
- f maps nonerroneous values to nonerroneous values.



Conclusion

Types:

- constrain the behaviour of programs
- thus lead to interesting theorems about programs

Conclusion

Types:

- constrain the behaviour of programs
- thus lead to interesting theorems about programs

In a "real" language:

- interaction with language features can be quite nontrivial
- here: finite failures, imprecise semantics as in implementations

Conclusion

Types:

- constrain the behaviour of programs
- thus lead to interesting theorems about programs

In a "real" language:

- interaction with language features can be quite nontrivial
- here: finite failures, imprecise semantics as in implementations

Not in the paper (but [Stenger & V., Technical Report]):

- inequational free theorems under weaker restrictions

Conclusion

Types:

- constrain the behaviour of programs
- thus lead to interesting theorems about programs

In a “real” language:

- interaction with language features can be quite nontrivial
- here: finite failures, imprecise semantics as in implementations

Not in the paper (but [Stenger & V., Technical Report]):

- inequational free theorems under weaker restrictions
- dealing with exceptions in a limited way

Conclusion

Types:

- constrain the behaviour of programs
- thus lead to interesting theorems about programs

In a “real” language:

- interaction with language features can be quite nontrivial
- here: finite failures, imprecise semantics as in implementations

Not in the paper (but [Stenger & V., Technical Report]):

- inequational free theorems under weaker restrictions
- dealing with exceptions in a limited way

Push further:

- proper exception handling in the IO monad

Conclusion

Types:

- constrain the behaviour of programs
- thus lead to interesting theorems about programs

In a “real” language:

- interaction with language features can be quite nontrivial
- here: finite failures, imprecise semantics as in implementations

Not in the paper (but [Stenger & V., Technical Report]):

- inequational free theorems under weaker restrictions
- dealing with exceptions in a limited way

Push further:

- proper exception handling in the IO monad
- nondeterminism generally, not just on errors

Conclusion

Types:

- constrain the behaviour of programs
- thus lead to interesting theorems about programs

In a “real” language:

- interaction with language features can be quite nontrivial
- here: finite failures, imprecise semantics as in implementations

Not in the paper (but [Stenger & V., Technical Report]):

- inequational free theorems under weaker restrictions
- dealing with exceptions in a limited way

Push further:

- proper exception handling in the IO monad
- nondeterminism generally, not just on errors
- systematic study of impact on other reasoning techniques

References I

P. Johann and J. Voigtländer.
Free theorems in the presence of seq.
In *Principles of Programming Languages, Proceedings*, pages 99–110. ACM Press, 2004.

P. Johann and J. Voigtländer.
A family of syntactic logical relations for the semantics of Haskell-like languages.
Information and Computation, 207(2):341–368, 2009.

A. Moran, S.B. Lassen, and S.L. Peyton Jones.
Imprecise exceptions, Co-inductively.
In *Higher Order Operational Techniques in Semantics, Proceedings*, volume 26 of *ENTCS*, pages 122–141. Elsevier, 1999.

References II

- S.L. Peyton Jones, A. Reid, C.A.R. Hoare, S. Marlow, and F. Henderson.
  A semantics for imprecise exceptions.
  In *Programming Language Design and Implementation, Proceedings*, pages 25–36. ACM Press, 1999.

- F. Stenger and J. Voigtländer.
  Parametricity for Haskell with imprecise error semantics.
  Technical Report TUD-FI08-08, Technische Universität Dresden, 2008.

- P. Wadler.
  Theorems for free!
  In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 347–359. ACM Press, 1989.