https://www.youtube.com/watch?v=wTP2RUD_cL0

Theorems for Free
Blame for All

Philip Wadler
University of Edinburgh

QCon São Paulo
25 April 2017

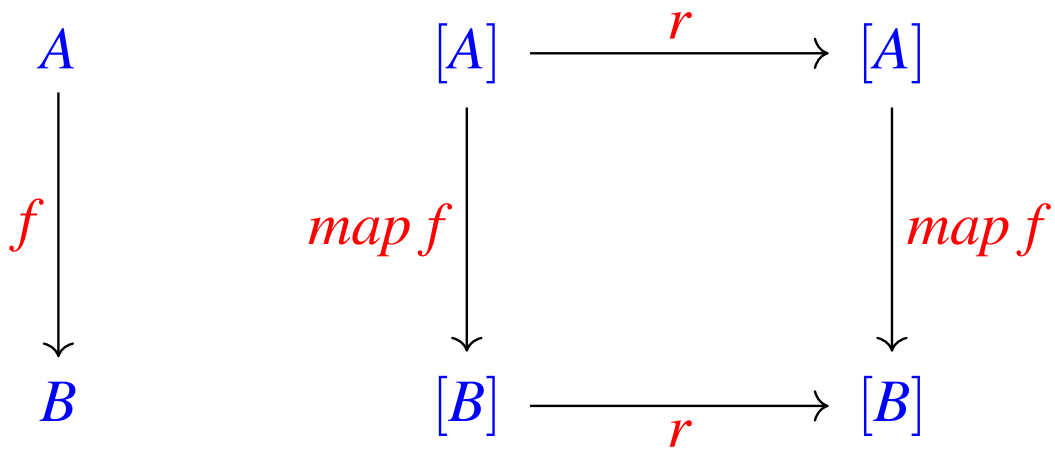
Part I

Theorems for Free

A magic trick

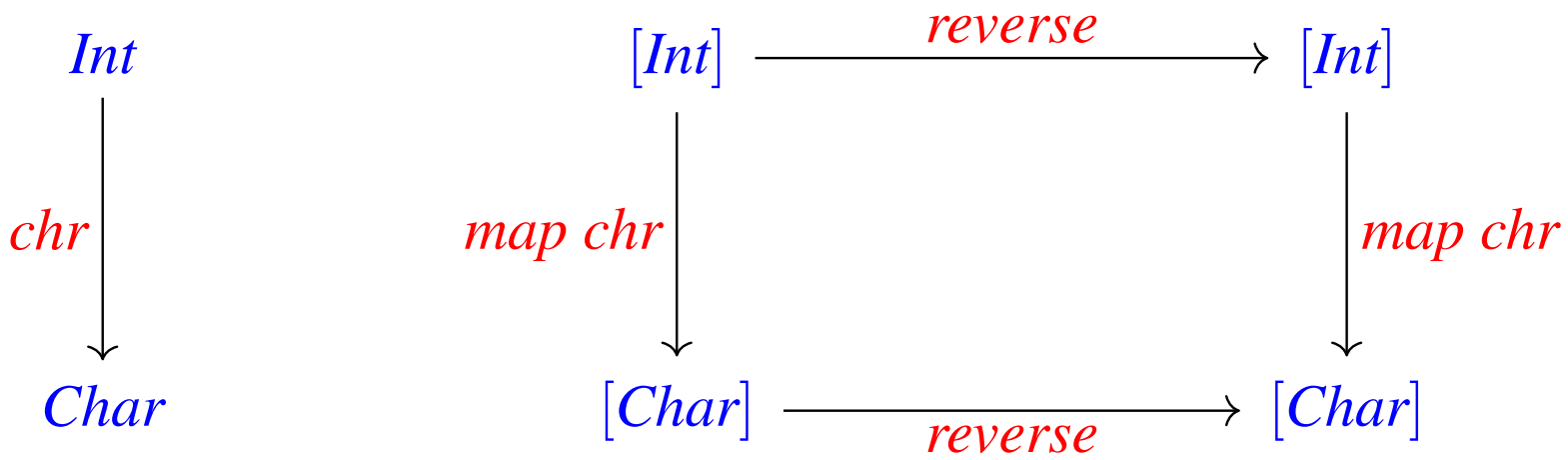
$$r \quad :: \quad \forall a.\, [a] \to [a]$$

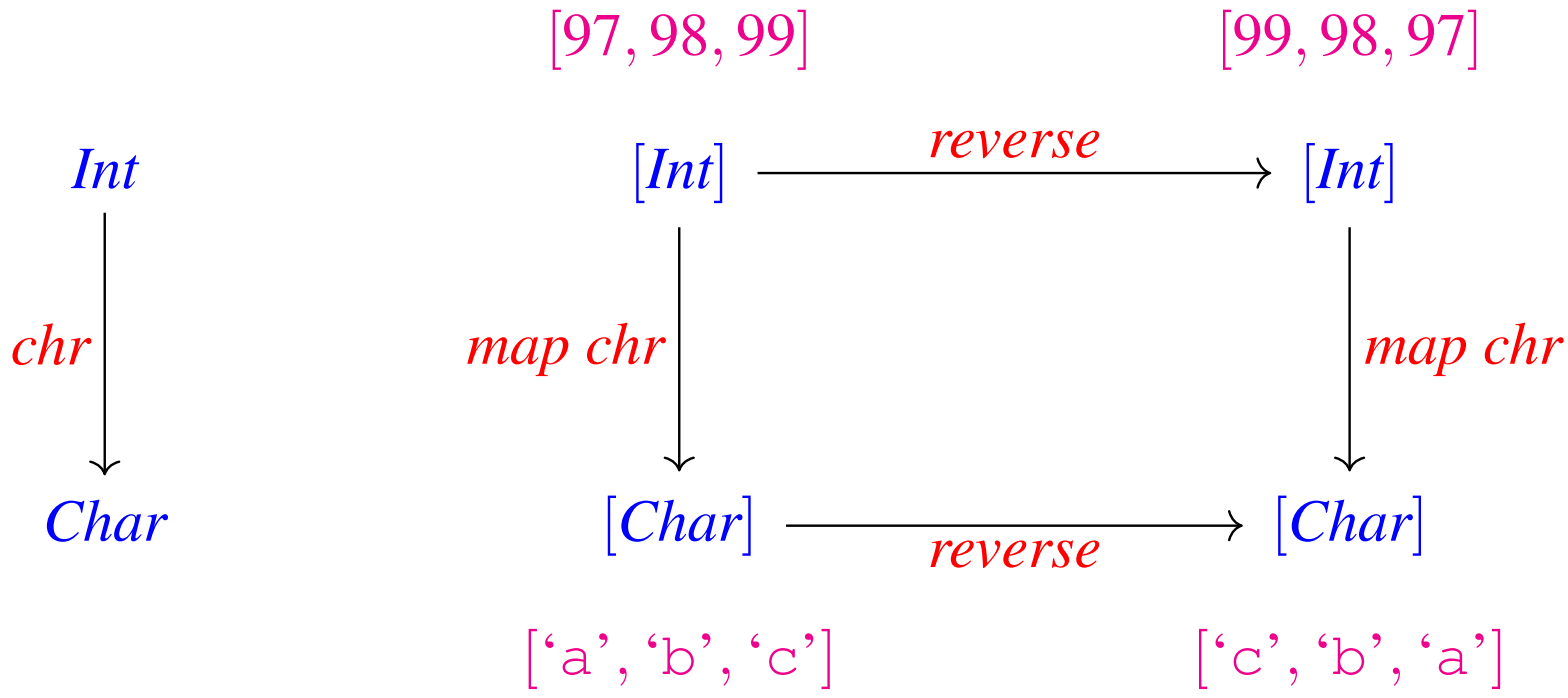
$$
\begin{aligned}
map &:: \forall a\, b.\, (a \to b) \to ([a] \to [b]) \\
f &:: A \to B \\
map\, f &:: [A] \to [B]
\end{aligned}
$$


A
f
B
[A]
r
[A]
map f
map f
[B]
r
[B]

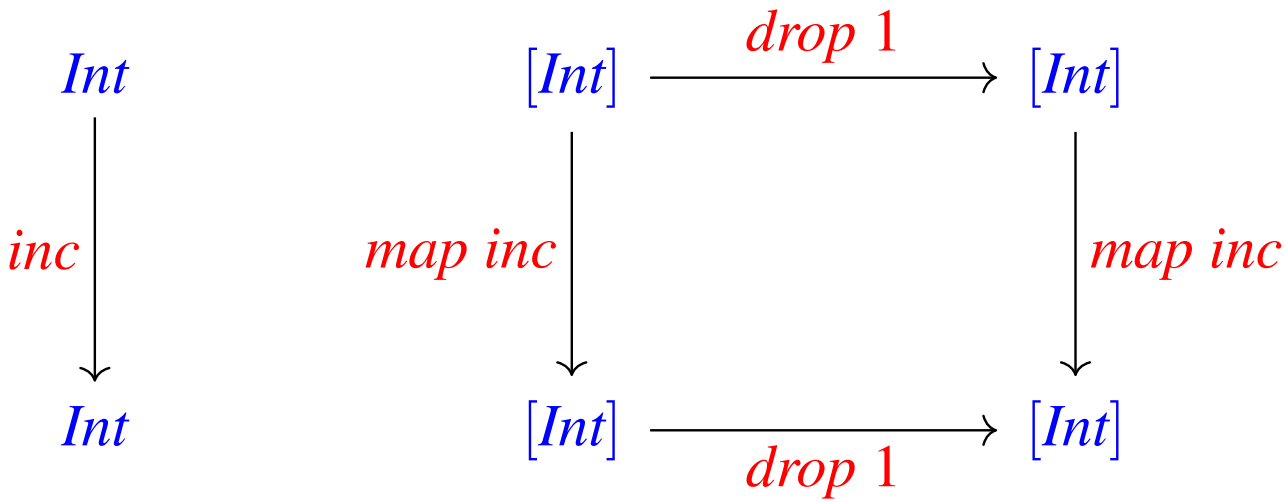
$$\mathit{reverse} \quad :: \quad \forall a.\, [a] \to [a]$$

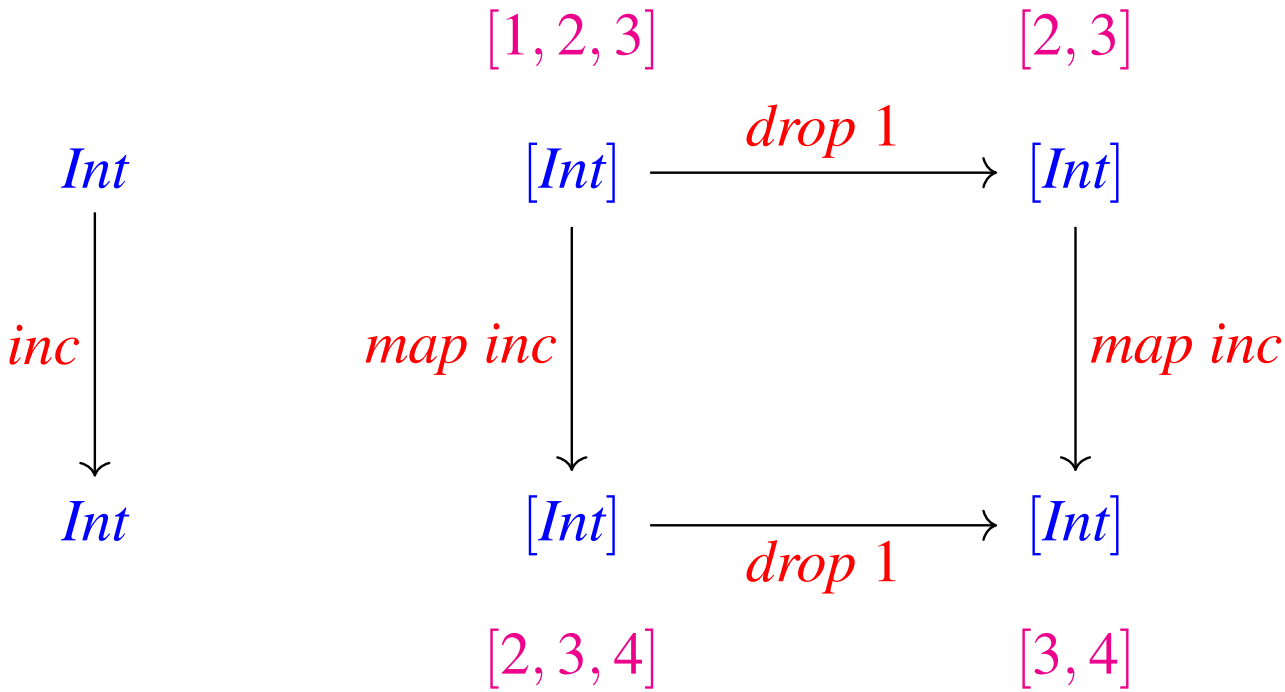
$$
\begin{array}{rcl}
map & :: & \forall a\, b.\, (a \to b) \to ([a] \to [b]) \\
chr & :: & Int \to Char \\
map\ f & :: & [Int] \to [Char]
\end{array}
$$


Int
chr
Char
[Int]
reverse
[Int]
map chr
map chr
[Char]
reverse
[Char]

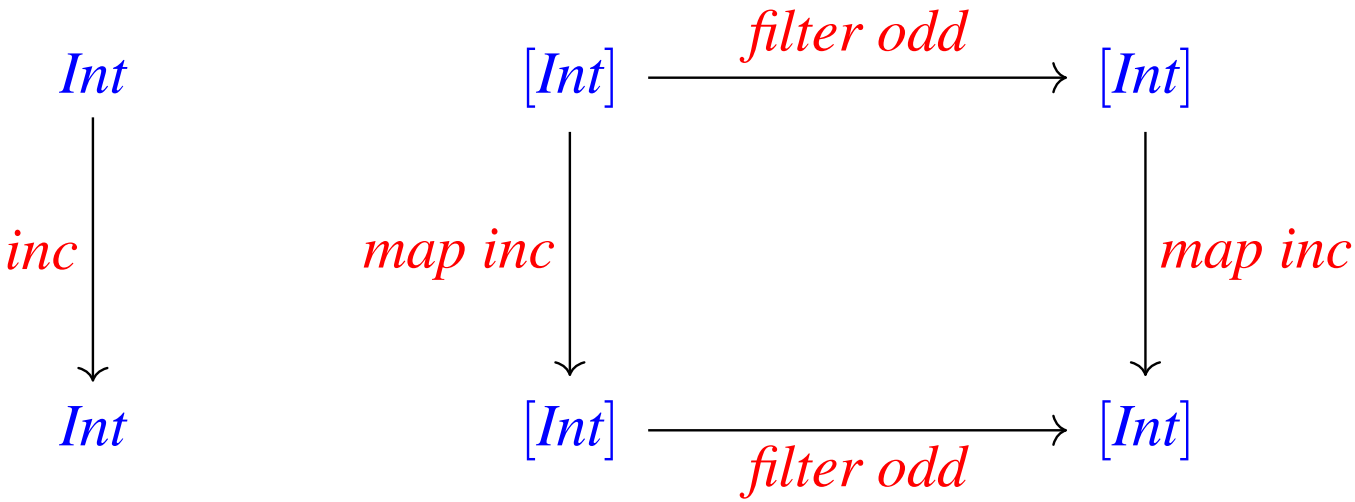

[97, 98, 99]
[99, 98, 97]
Int
[Int]
reverse
[Int]
chr
map chr
map chr
Char
[Char]
reverse
[Char]
['a', 'b', 'c']
['c', 'b', 'a']

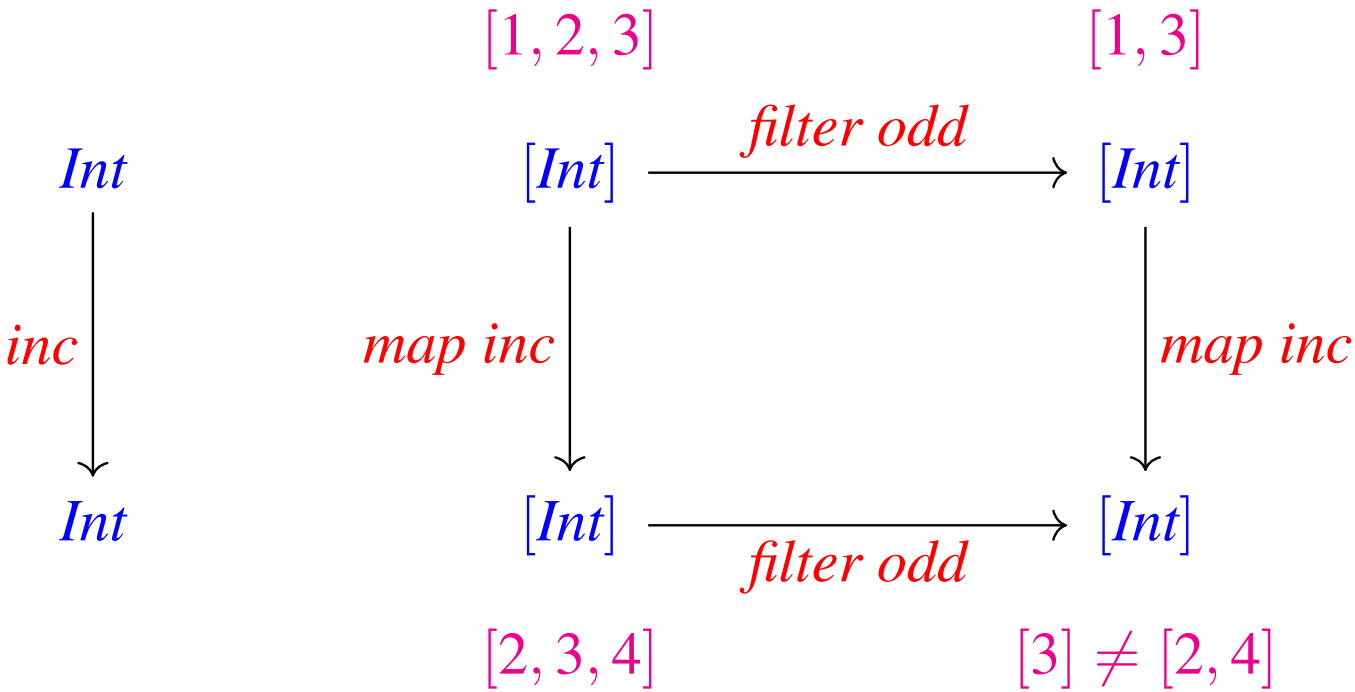
$$
\begin{array}{rcl}
\mathit{drop} & :: & \forall a.\, \mathit{Int} \to ([a] \to [a]) \\
\mathit{drop}\ 1 & :: & \forall a.\, [a] \to [a]
\end{array}
$$


Int
inc
Int
[Int]
drop 1
[Int]
map inc
map inc
[Int]
[Int]
drop 1

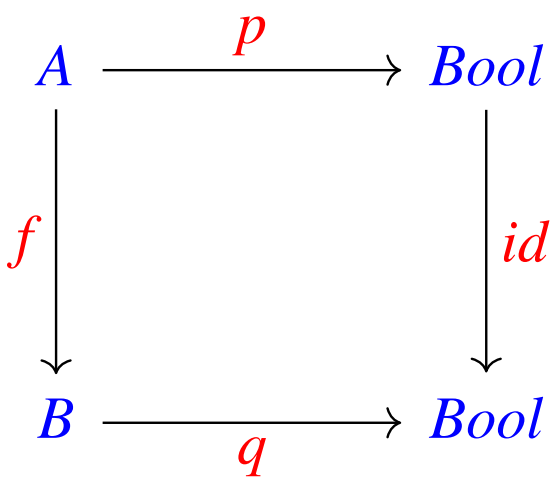

[1, 2, 3]
[2, 3]
drop 1
Int
[Int]
[Int]
inc
map inc
map inc
Int
[Int]
[Int]
drop 1
[2, 3, 4]
[3, 4]

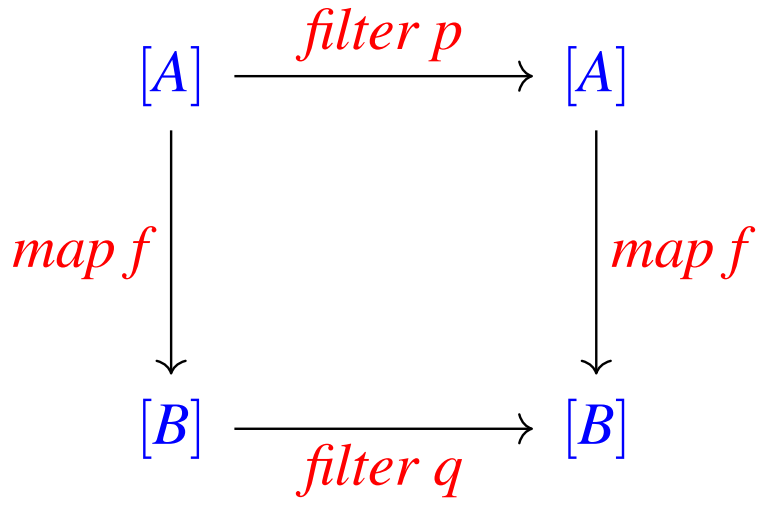
$$
\begin{aligned}
\mathit{filter} &:: \forall a.\,(a \to \mathit{Bool}) \to ([a] \to [a]) \\
\mathit{odd} &:: \mathit{Int} \to \mathit{Bool} \\
\mathit{filter\ odd} &:: [\mathit{Int}] \to [\mathit{Int}]
\end{aligned}
$$

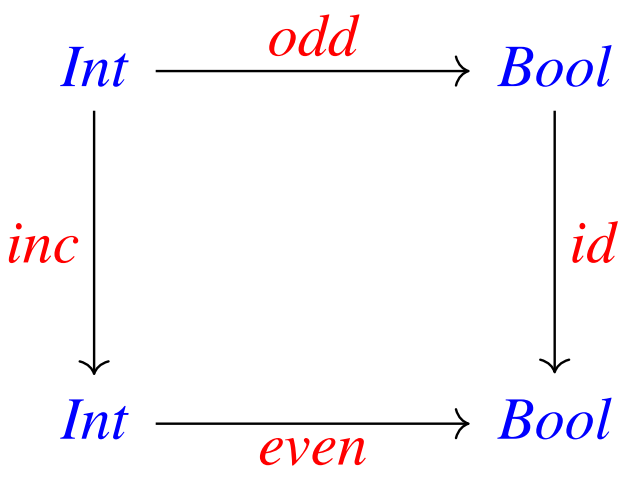

Int
inc
Int
[Int]
filter odd
[Int]
map inc
map inc
[Int]
[Int]
filter odd

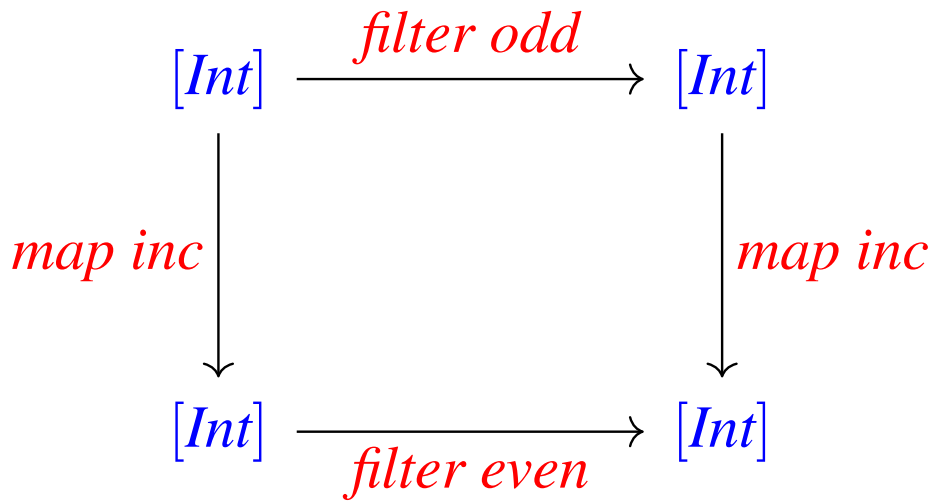

[1, 2, 3]
[1, 3]
filter odd
Int
[Int]
[Int]
inc
map inc
map inc
Int
[Int]
[Int]
filter odd
[2, 3, 4]
[3] ≠ [2, 4]

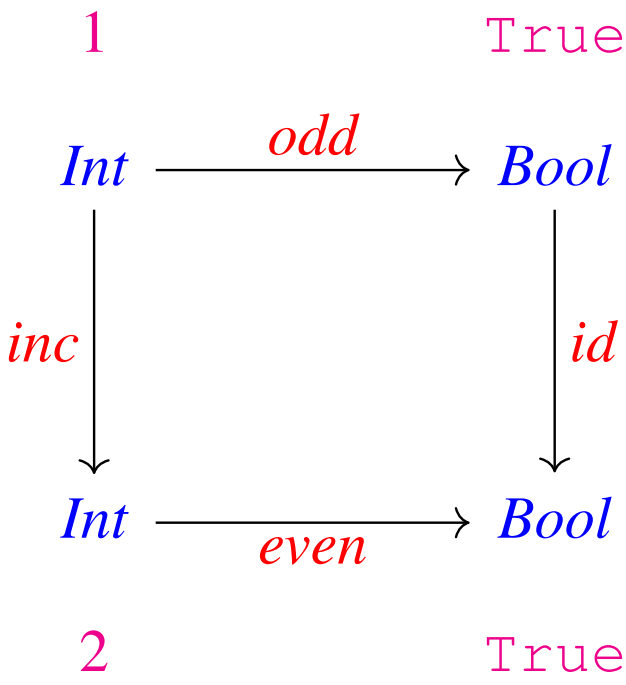
$$\mathit{filter} \quad :: \quad \forall a.\,(a \to \mathit{Bool}) \to ([a] \to [a])$$

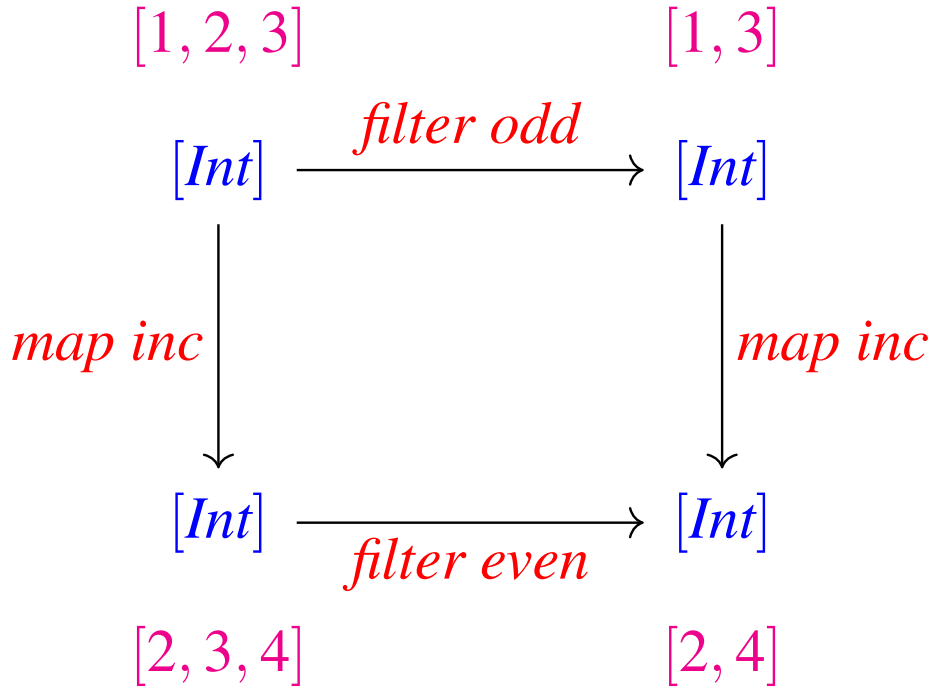

A
p
Bool
f
id
B
q
Bool


[A]
filter p
[A]
map f
map f
[B]
filter q
[B]


Int
odd
Bool
inc
id
Int
even
Bool


[Int]
filter odd
[Int]
map inc
map inc
[Int]
filter even
[Int]


1
True
Int
odd
Bool
inc
id
Int
even
Bool
2
True


[1, 2, 3]
[1, 3]
[Int]
filter odd
[Int]
map inc
map inc
[Int]
filter even
[Int]
[2, 3, 4]
[2, 4]

$$\mathcal{E}\,[\![A]\!]\,\rho = \{(e_1, e_2) \mid \exists v_1, v_2.\ e_1 \longrightarrow^* v_1 \text{ and } e_2 \longrightarrow^* v_2 \text{ and } (v_1, v_2) \in \mathcal{V}\,[\![A]\!]\,\rho\}$$

$$\mathcal{V}\,[\![\mathsf{int}]\!]\,\rho = \{(n, n) \mid n \in \mathbb{Z}\}$$

$$\mathcal{V}\,[\![\mathsf{bool}]\!]\,\rho = \{(b, b) \mid b \in \mathbb{B}\}$$

$$\mathcal{V}\,[\![A \times B]\!]\,\rho = \{(p_1, p_2) \mid (\mathsf{fst}(p_1), \mathsf{fst}(p_2)) \in \mathcal{E}\,[\![A]\!]\,\rho \text{ and } (\mathsf{snd}(p_1), \mathsf{snd}(p_2)) \in \mathcal{E}\,[\![A]\!]\,\rho\}$$

$$\mathcal{V}\,[\![A \to B]\!]\,\rho = \{(v_f, v_g) \mid \forall (v_1, v_2) \in \mathcal{V}\,[\![A]\!]\,\rho.\ (v_f\ v_1, v_g\ v_2) \in \mathcal{E}\,[\![B]\!]\,\rho\}$$

$$\mathcal{V}\,[\![\forall X.\, A]\!]\,\rho = \{(v_1, v_2) \mid \forall B_1, B_2, R \subseteq B_1 \times B_2.\ (v_1\ [B_1], v_2\ [B_2]) \in \mathcal{E}\,[\![A]\!]\,\rho(X{:=}R)\}$$

$$\mathcal{V}\,[\![\exists X.\, A]\!]\,\rho = \{(\mathsf{pack}(B_1, v_1), \mathsf{pack}(B_2, v_2)) \mid \exists R \subseteq B_1 \times B_2.\ (v_1, v_2) \in \mathcal{V}\,[\![A]\!]\,\rho(X{:=}R)\}$$

$$\mathcal{V}\,[\![X]\!]\,\rho = \rho(X)$$

Fig. 1. A Logical Relation for λF.

A Short Cut to Deforestation

Andrew Gill John Launchbury Simon L Peyton Jones
Department of Computing Science, University of Glasgow G12 8QQ
{andy,jl,simonpj}@dcs.glasgow.ac.uk

This paper is to appear in FPCA 1993.

Abstract

Lists are often used as "glue" to connect separate parts of a program together. We propose an automatic technique for improving the efficiency of such programs, by removing many of these intermediate lists, based on a single, simple, local transformation. We have implemented the method in the Glasgow Haskell compiler.

1 Introduction

which do not use intermediate lists. For example, it can be re-written like this:

```
all' p xs = h xs
  where h []     = True
        h (x:xs) = p x && h xs
```

Now no intermediate list is used, but this has been achieved at the cost of clarity and conciseness compared with the original definition.

We want to eat our cake and have it too. That is, we would like to write our programs in the style of `all`, but have the compiler automatically transform this into the more efficient version `all'`.

Part II

And now a word from our sponsor

Lambda Calculus

$$L, M, N \quad ::= \quad x \mid \lambda x.\, N \mid L\, M$$

Polymorphic Lambda Calculus

$$\begin{aligned} A, B, C \quad &::= \quad X \mid A \to B \mid \forall X.\, B \\ L, M, N \quad &::= \quad x \mid \lambda x{:}A.\, N \mid L\, M \mid \Lambda X.\, N \mid L\, A \end{aligned}$$

Part III

Blame for All

A simple typed program

$$
\begin{aligned}
&\mathit{inc} :: \mathit{Int} \to \mathit{Int} \\
&\mathit{inc}\ x = x + 1 \\
&\\
&\mathit{twice} :: \forall a.\, (a \to a) \to (a \to a) \\
&\mathit{twice}\ f\ x = f\ (f\ x) \\
&\\
&\mathit{main} :: \mathit{Int} \\
&\mathit{main} = \mathit{twice}\ \mathit{inc}\ 40 \\
\longmapsto & \\
&42 : \mathit{Int}
\end{aligned}
$$

A simple untyped program

$$
\begin{aligned}
& inc^{\star} :: \star \\
& inc^{\star}\ x = x + 1 \\
& \\
& twice^{\star} :: \star \\
& twice^{\star}\ f\ x = f\ (f\ x) \\
& \\
& main^{\star} :: \star \\
& main^{\star} = twice^{\star}\ inc^{\star}\ 40 \\
\longmapsto & \\
& 42 : \star
\end{aligned}
$$

A simple untyped program

$$
\begin{aligned}
& inc^\star :: \star \\
& inc^\star\ x = x + 1 \\
& \\
& twice^\star :: \star \\
& twice^\star\ f\ x = f\ (f\ x) \\
& \\
& main^\star :: \star \\
& main^\star = twice^\star\ inc^\star\ 40 \\
\longmapsto & \\
& 42 : \star
\end{aligned}
$$

Untyped is unityped

Widening: casting from typed to untyped

$$
\begin{array}{l}
inc^{\star} :: \star \\
inc^{\star}\ x = x + 1 \\
\\
twice :: \forall a.\, (a \to a) \to (a \to a) \\
twice\ f\ x = f\ (f\ x) \\
\\
twice^{\star} :: \star \\
twice^{\star} = \langle twice \rangle^{p} \\
\\
main :: \star \\
main = twice^{\star}\ inc^{\star}\ 40 \\
\\
\longmapsto \\
\\
42 : \star
\end{array}
$$

Widening: casting from typed to untyped

$$
\begin{array}{l}
inc^{\star} :: \star \\
inc^{\star}\ x = x + 1 \\
\\
twice :: \forall a.\,(a \to a) \to (a \to a) \\
twice\ f\ x = f\ (f\ x) \\
\\
twice^{\star} :: \star \\
twice = \langle twice \rangle^{p} \\
\\
main :: \star \\
main = \underline{twice^{\star}\ \texttt{40}\ inc^{\star}} \\
\longmapsto \\
\quad \texttt{blame}\ \overline{p}
\end{array}
$$

A cast from more-precise to less-precise type may blame the *context containing* the cast, but never the *term contained* in the cast.

Narrowing: casting from untyped to typed

$$
\begin{array}{l}
inc :: Int \to Int \\
inc\ x = x + 1 \\
\\
twice^{\star} :: \star \\
twice^{\star}\ f\ x = f\ (f\ x) \\
\\
twice :: \forall a.\, (a \to a) \to (a \to a) \\
twice = \langle twice^{\star} \rangle^{p} \\
\\
main :: Int \\
main = twice\ inc\ 40 \\
\\
\longmapsto \\
\\
42 : Int
\end{array}
$$

Narrowing: casting from untyped to typed

$$
\begin{array}{l}
inc :: Int \to Int \\
inc\ x = x + 1 \\
\\
twice^{\star} :: \star \\
twice^{\star}\ f\ x = \underline{42} \\
\\
twice :: \forall a.\, (a \to a) \to (a \to a) \\
twice = \langle twice^{\star} \rangle^{p} \\
\\
main :: Int \\
main = twice\ inc\ 40 \\
\\
\longmapsto \\
\\
\quad \texttt{blame}\ p
\end{array}
$$

A cast from less-precise to more-precise type may blame the *term contained* in the cast, but never the *context containing* the cast.

Polymorphic Blame Calculus

Reduction $\boxed{M \longmapsto N}$

$$
\begin{aligned}
op(\vec{V}) &\longmapsto [\![op]\!](\vec{V}) \\
(\lambda x{:}A.\, N[x])\, V &\longmapsto N[V] \\
(\Lambda X.\, V[X])\, X &\longmapsto V[X] \\
V : \iota \overset{Q}{\Longrightarrow} \iota &\longmapsto V \\
(V : A \to B \overset{Q}{\Longrightarrow} C \to D)\, W &\longmapsto V\, (W : C \overset{-Q}{\Longrightarrow} A) : B \overset{Q}{\Longrightarrow} D \\
(V : \forall X.A \overset{Q}{\Longrightarrow} \forall X.B)\, X &\longmapsto V\, X : A \overset{Q}{\Longrightarrow} B \qquad \text{if } X \notin Q \\
V : X \overset{Q}{\Longrightarrow} X &\longmapsto V \qquad \text{if } X \notin Q \\
V : A \overset{-X}{\Longrightarrow} X \overset{+X}{\Longrightarrow} A &\longmapsto V
\end{aligned}
$$

Reduction of configurations $\boxed{\Sigma \vdash M \longrightarrow \Sigma' \vdash N}$

$$\frac{M \longmapsto N}{\Sigma \vdash \mathcal{E}[M] \longrightarrow \Sigma \vdash \mathcal{E}[N]} \qquad \frac{X \notin \Sigma}{\Sigma \vdash \mathcal{E}[\nu X{:=}A.N] \longrightarrow \Sigma,\, X{:=}A \vdash \mathcal{E}[N]}$$

Reduction

$\boxed{M \longmapsto N}$ $\boxed{M \longmapsto \texttt{blame}\, p}$

$$V : \star \overset{Q}{\Longrightarrow} \star \longmapsto V$$

$$V : \iota \overset{p}{\Longrightarrow} \iota \longmapsto V$$

$$(V : A \to B \overset{p}{\Longrightarrow} C \to D)\, W \longmapsto V\, (W : C \overset{\bar{p}}{\Longrightarrow} A) : B \overset{p}{\Longrightarrow} D$$

$$(V : A \overset{p}{\Longrightarrow} \forall X.B)\, X \longmapsto V : A \overset{p}{\Longrightarrow} B$$

$$V : \forall X.A \overset{p}{\Longrightarrow} B \longmapsto (V\, \star) : A[X{:=}\star] \overset{p}{\Longrightarrow} B$$

$$V : X \overset{p}{\Longrightarrow} X \longmapsto V$$

$$V : \star \overset{p}{\Longrightarrow} \star \longmapsto V$$

$$V : A \overset{p}{\Longrightarrow} \star \longmapsto V : A \overset{p}{\Longrightarrow} G \overset{p}{\Longrightarrow} \star \qquad \text{if } \mathrm{ug}(A), A \prec G$$

$$V : \star \overset{p}{\Longrightarrow} A \longmapsto V : \star \overset{p}{\Longrightarrow} G \overset{p}{\Longrightarrow} A \qquad \text{if } \mathrm{ug}(A), G \prec A$$

$$V : G \overset{p}{\Longrightarrow} \star \overset{p'}{\Longrightarrow} G \longmapsto V$$

$$V : G \overset{p}{\Longrightarrow} \star \overset{p'}{\Longrightarrow} H \longmapsto \texttt{blame}\, p' \qquad \text{if } G \neq H$$

Theorems for Free for Free

Parametricity, With and Without Types

ANONYMOUS AUTHOR(S)

The polymorphic blame calculus integrates static typing, including universal types, with dynamic typing. The primary challenge with this integration is preserving parametricity: even dynamically-typed code should satisfy it once it has been cast to a universal type. Ahmed et al. (2011) employ runtime type generation in the polymorphic blame calculus to preserve parametricity, but a proof that it does so has been elusive. Matthews and Ahmed (2008) gave a proof of parametricity for a closely related system that combines ML and Scheme, but later found a flaw in their proof. In this paper we prove that the polymorphic blame calculus satisfies relational parametricity. The proof relies on a step-indexed Kripke logical relation. The step-indexing is required to make the logical relation well-defined in the case for the dynamic type. The possible worlds include the mapping of generated type names to their concrete types and the mapping of type names to relations. We prove the Fundamental Property of this logical relation and that it is sound with respect to contextual equivalence. To demonstrate the utility of parametricity in the polymorphic blame calculus, we derive two free theorems.

CCS Concepts: •**Software and its engineering** → **General programming languages**; •**Social and professional topics** → *History of programming languages;*

Additional Key Words and Phrases: keyword1, keyword2, keyword3

ACM Reference format:
Anonymous Author(s). 2017. Theorems for Free for Free. *PACM Progr. Lang.* 1, 1, Article 1 (January 2017), 23 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

Mixed Messages: Measuring Conformance and Non-Interference in TypeScript

Double-blind submission

Abstract

TypeScript participates in the recent trend among programming languages to support gradual typing. The DefinitelyTyped Repository for TypeScript supplies type definitions for over 2000 popular JavaScript libraries. However, there is no guarantee that implementations conform to their corresponding declarations.

We present a practical evaluation of gradual typing for TypeScript. We have developed a tool, based on the polymorphic blame calculus, for monitoring JavaScript libraries and TypeScript clients against the TypeScript definition. We apply our tool, TypeScript TNG, to those libraries in the DefinitelyTyped Repository which had adequate test code to use. Of the 122 libraries we checked, 59 had cases where either the library or its tests failed to conform to the declaration.

Gradual typing should satisfy *non-interference*. Monitoring a program should never change its behaviour except, to raise a type error should a value not conform to its declared type. However, our experience also suggests serious technical concerns with the use of the JavaScript proxy mechanism for enforcing contracts. Of the 122 libraries we checked, 22 had cases where the library or its tests violated non-interference.

1998 ACM Subject Classification D.2.5 [*Software Engineering*]: Testing and Debugging

Keywords and phrases Gradual Typing, TypeScript, JavaScript, Proxies

Part IV

Further Reading

Theorems for Free

Wadler; ICFP, 1989

Blame for All

Ahmed, Findler, Siek, Wadler; POPL, 2011

Theorems for Free for Free

Ahmed, Jamner, Siek, Wadler; ICFP, 2017

Mixed Messages

Williams, Morris, Wadler, Zalewski; ECOOP, 2017

Propositions as Types

Wadler; CACM, December 2015