Well-typed programs can't be blamed

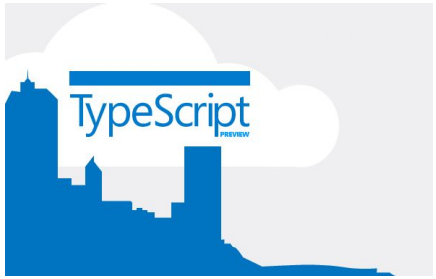
(ESOP 2009)

Robert Bruce Findler
Northwestern University

Philip Wadler
University of Edinburgh

Aussois, 14–18 October 2013

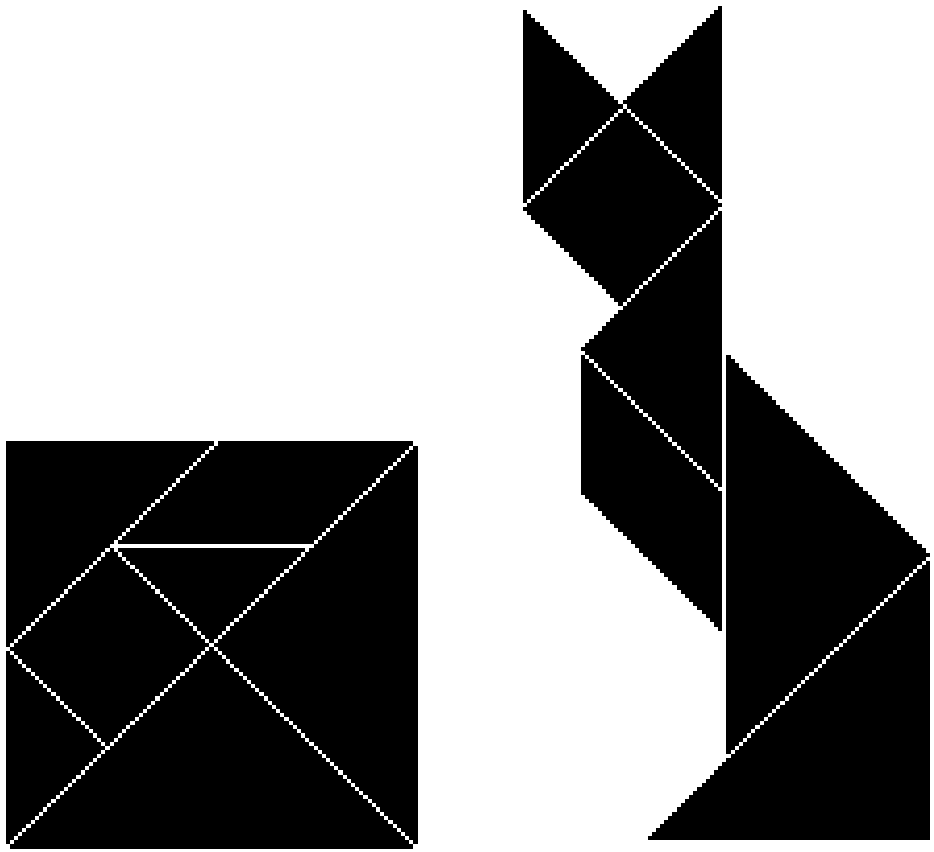

JavaScript


TypeScript


Microsoft
Visual F#


Lua


Scala



Part I

Evolving a program

An untyped program

$$
\begin{array}{l}
\lceil \texttt{let} \\
\quad x = 2 \\
\quad f = \lambda y.\, y + 1 \\
\quad h = \lambda g.\, g\,(g\,x) \\
\texttt{in} \\
\quad h\,f \rceil \\
\longrightarrow \\
\lceil 4 \rceil
\end{array}
$$

A typed program

$$
\begin{array}{l}
\texttt{let} \\
\quad x = 2 \\
\quad f = \lambda y : \texttt{Int}.\, y + 1 \\
\quad h = \lambda g : \texttt{Int} \to \texttt{Int}.\, g\,(g\,x) \\
\texttt{in} \\
\quad h\,f \\
\longrightarrow \\
\quad 4 : \texttt{Int}
\end{array}
$$

A partly typed program—narrowing

$$
\begin{array}{l}
\texttt{let} \\
\quad x = 2 \\
\quad f = \lceil \lambda y.\, y + 1 \rceil : \star \overset{p}{\Rightarrow} \texttt{Int} \to \texttt{Int} \\
\quad h = \lambda g : \texttt{Int} \to \texttt{Int}.\, g\,(g\,x) \\
\texttt{in} \\
\quad h\,f \\
\longrightarrow \\
\quad 4 : \texttt{Int}
\end{array}
$$

A partly typed program—narrowing

$$
\begin{array}{l}
\texttt{let} \\
\quad x = 2 \\
\quad f = \lceil \lambda y.\, \texttt{false} \rceil : \star \overset{p}{\Rightarrow} \texttt{Int} \to \texttt{Int} \\
\quad h = \lambda g : \texttt{Int} \to \texttt{Int}.\, g\,(g\,x) \\
\texttt{in} \\
\quad h\,f \\
\longrightarrow \\
\quad \texttt{blame}\, p
\end{array}
$$

Positive (covariant): blame the term contained in the cast

Narrowing can give rise to positive blame, but never negative blame

Another partly typed program—widening

$$
\begin{array}{l}
\texttt{let} \\
\quad x = \lceil 2 \rceil \\
\quad f = (\lambda y : \texttt{Int}.\, y + 1) : \texttt{Int} \to \texttt{Int} \stackrel{p}{\Rightarrow} \star \\
\quad h = \lceil \lambda g.\, g\,(g\,x) \rceil \\
\texttt{in} \\
\quad \lceil h\, f \rceil \\
\longrightarrow \\
\quad \lceil 4 \rceil
\end{array}
$$

Another partly typed program—widening

$$
\begin{array}{l}
\texttt{let} \\
\quad x = \lceil \texttt{true} \rceil \\
\quad f = (\lambda y : \texttt{Int}.\, y + 1) : \texttt{Int} \to \texttt{Int} \overset{p}{\Rightarrow} \star \\
\quad h = \lceil \lambda g.\, g\,(g\,x) \rceil \\
\texttt{in} \\
\quad \lceil h\, f \rceil \\
\longrightarrow \\
\quad \texttt{blame}\, \bar{p}
\end{array}
$$

Negative (contravariant): blame the context containg the cast

Widening can give rise to negative blame, but never positive blame

Part II

Blame calculus

Untyped = Uni-typed

$$
\begin{array}{lcll}
\lceil x \rceil & = & x & \\
\lceil c \rceil & = & c : A \overset{p}{\Rightarrow} \star & \text{if } \mathrm{ty}(c) = A \\
\lceil op(\vec{M}) \rceil & = & op(\lceil \vec{M} \rceil : \vec{\star} \overset{\vec{p}}{\Rightarrow} \vec{A}) : B \overset{p}{\Rightarrow} \star & \text{if } \mathrm{ty}(op) = \vec{A} \to B \\
\lceil \lambda x.\, N \rceil & = & (\lambda x : \star.\, \lceil N \rceil) : \star \to \star \Rightarrow \star & \\
\lceil L\ M \rceil & = & (\lceil L \rceil : \star \overset{p}{\Rightarrow} \star \to \star)\ \lceil M \rceil &
\end{array}
$$

(slogan due to Bob Harper)

Blame calculus: Compatibility

$$A \prec A \qquad A \prec \star \qquad \star \prec B$$

$$\frac{A' \prec A \qquad B \prec B'}{A \to B \prec A' \to B'}$$

Types

$$\frac{\mathrm{ty}(c) = \iota}{\Gamma \vdash c : \iota} \qquad \frac{\Gamma \vdash \vec{t} : \vec{A} \qquad \mathrm{ty}(op) = \vec{A} \to B}{\Gamma \vdash op(\vec{t}) : B} \qquad \frac{x : A \in \Gamma}{\Gamma \vdash x : A}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x{:}A.\, t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash s : A}{\Gamma \vdash t\ s : B}$$

$$\frac{\Gamma \vdash s : A \qquad A \prec B}{\Gamma \vdash (s : A \overset{p}{\Rightarrow} B) : B} \qquad \frac{\Gamma \vdash s : G}{\Gamma \vdash (s : G \Rightarrow \star) : \star}$$

Reductions

$$
\begin{array}{rcll}
(\lambda x{:}A.\, t)\, v & \longrightarrow & t[x := v] & \\
op(\vec{v}\,) & \longrightarrow & \delta(op, \vec{v}\,) & \\
v : A{\to}B \overset{p}{\Rightarrow} A'{\to}B' & \longrightarrow & \lambda x'{:}A'.\, (v\ (x' : A' \overset{\bar{p}}{\Rightarrow} A) : B \overset{p}{\Rightarrow} B') & \\
v : \iota \overset{p}{\Rightarrow} \iota & \longrightarrow & v & \\
v : A \overset{p}{\Rightarrow} \star & \longrightarrow & (v : A \overset{p}{\Rightarrow} G) : G \Rightarrow \star & \text{if } \star \neq A \prec G \\
v : (G \Rightarrow \star) : \star \overset{p}{\Rightarrow} A & \longrightarrow & v : G \overset{p}{\Rightarrow} A & \text{if } G \prec A \\
v : (G \Rightarrow \star) : G \overset{p}{\Rightarrow} A & \longrightarrow & \texttt{blame}\ p & \text{if } G \not\prec A
\end{array}
$$

$$
\frac{s \longrightarrow t}{E[s] \longrightarrow E[t]} \qquad \frac{s \longrightarrow \texttt{blame}\ p}{E[s] \longrightarrow \texttt{blame}\ p}
$$

Part III

Subtyping

$<:$ $<:^{+}$ $<:^{-}$ $<:_{n}$

Subtype

$$\frac{}{\star <: \star}$$

$$\frac{}{\iota <: \iota} \qquad \frac{A <: G}{A <: \star}$$

$$\frac{A' <: A \qquad B <: B'}{A \to B <: A' \to B'}$$

Example:

$$\frac{\dfrac{\dfrac{}{\texttt{Int} <: \texttt{Int}}}{\texttt{Int} <: \star} \qquad \dfrac{\dfrac{}{\texttt{Int} <: \texttt{Int}}}{\texttt{Int} <: \star}}{\star \to \texttt{Int} <: \texttt{Int} \to \star}$$

Positive subtype—widening

$$\frac{}{A <:^{+} \star}$$

$$\frac{}{\iota <: \iota}$$

$$\frac{A' <:^{-} A \qquad B <:^{+} B'}{A \to B <:^{+} A' \to B'}$$

Example:

$$\frac{\dfrac{}{\star <:^{-} \texttt{Int}} \qquad \dfrac{}{\texttt{Int} <:^{+} \star}}{\texttt{Int} \to \texttt{Int} <:^{+} \star \to \star}$$

Negative subtype—narrowing

$$\frac{}{\star <:^{-} A}$$

$$\frac{}{\iota <: \iota} \qquad \frac{A <:^{-} G}{A <:^{-} \star}$$

$$\frac{A' <:^{+} A \qquad B <:^{-} B'}{A \to B <:^{-} A' \to B'}$$

Example:

$$\frac{\dfrac{}{\texttt{Int} <:^{+} \star} \qquad \dfrac{}{\star <:^{-} \texttt{Int}}}{\star \to \star <:^{-} \texttt{Int} \to \texttt{Int}}$$

Naive subtype

$$\frac{}{A <:_n \star}$$

$$\frac{}{\iota <:_n \iota}$$

$$\frac{A <:_n A' \qquad B <:_n B'}{A \to B <:_n A' \to B'}$$

Example:

$$\frac{\dfrac{}{\texttt{Int} <:_n \star} \qquad \dfrac{}{\texttt{Int} <:_n \star}}{\texttt{Int} \to \texttt{Int} <: \star \to \star}$$

Part IV

The Blame Theorem

Safety

$$\frac{}{x \text{ sf } p} \qquad \frac{t \text{ sf } p}{\lambda x.\, t \text{ sf } p} \qquad \frac{s \text{ sf } p \qquad t \text{ sf } p}{s\ t \text{ sf } p}$$

$$\frac{s \text{ sf } p \qquad A <:^{+} B}{s : A \overset{p}{\Rightarrow} B \text{ sf } p}$$

$$\frac{s \text{ sf } p \qquad A <:^{-} B}{s : A \overset{\bar{p}}{\Rightarrow} B \text{ sf } p}$$

$$\frac{s \text{ sf } p \qquad p \neq q \qquad \bar{p} \neq q}{s : A \overset{q}{\Rightarrow} B \text{ sf } p}$$

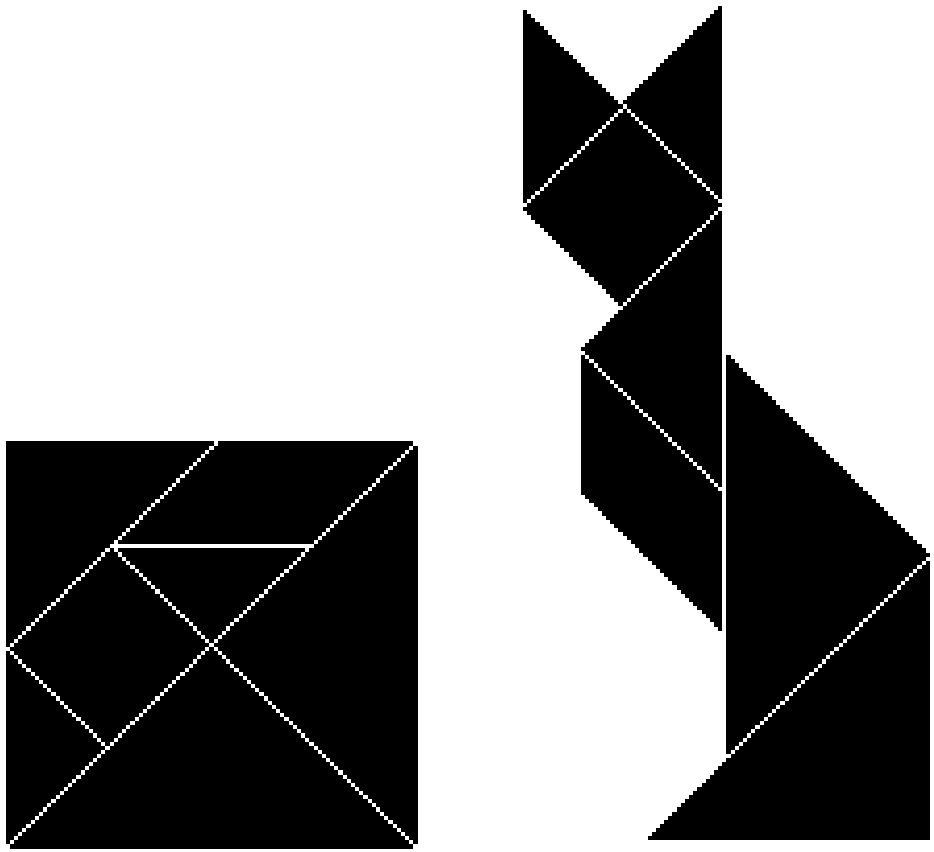
The Blame Theorem

Preservation

If s sf p and $s \longrightarrow t$ then t sf p.

Progress

If s sf p then $s \not\longrightarrow \texttt{blame}\ p$.



The First Tangram Theorem

$$A <: B \text{ if and only if } A <:^{+} B \text{ and } A <:^{-} B$$

The First Blame Corollary

Let t be a term where $s : A \overset{p}{\Rightarrow} B$ is the only subterm with label p. If $A <: B$ then $t \not\longrightarrow \texttt{blame}\ p$ and $t \not\longrightarrow \texttt{blame}\ \bar{p}$.

The Second Tangram Theorem

$$A <:_n B \text{ if and only if } A <:^+ B \text{ and } B <:^- A$$

The Second Blame Corollary

Let t be a term where $s : A \overset{p}{\Rightarrow} B$ is the only subterm with label p. If $A <:_n B$ then $t \not\longrightarrow \texttt{blame}\ p$.

Let t be a term where $s : A \overset{p}{\Rightarrow} B$ is the only subterm with label p. If $B <:_n A$ then $t \not\longrightarrow \texttt{blame}\ \overline{p}$.

A new slogan for type safety

Milner (1978):

Well-typed programs can't go wrong.

Felleisen and Wright (1994); Harper (2002):

Well-typed programs don't get stuck.

Wadler and Findler (2008):

Well-typed programs can't be blamed.

References

Well-typed programs can't be blamed

Robert Bruce Findler and Philip Wadler

ESOP 2009

Threesomes

Jeremy Siek and Philip Wadler

POPL 2010

Blame for all

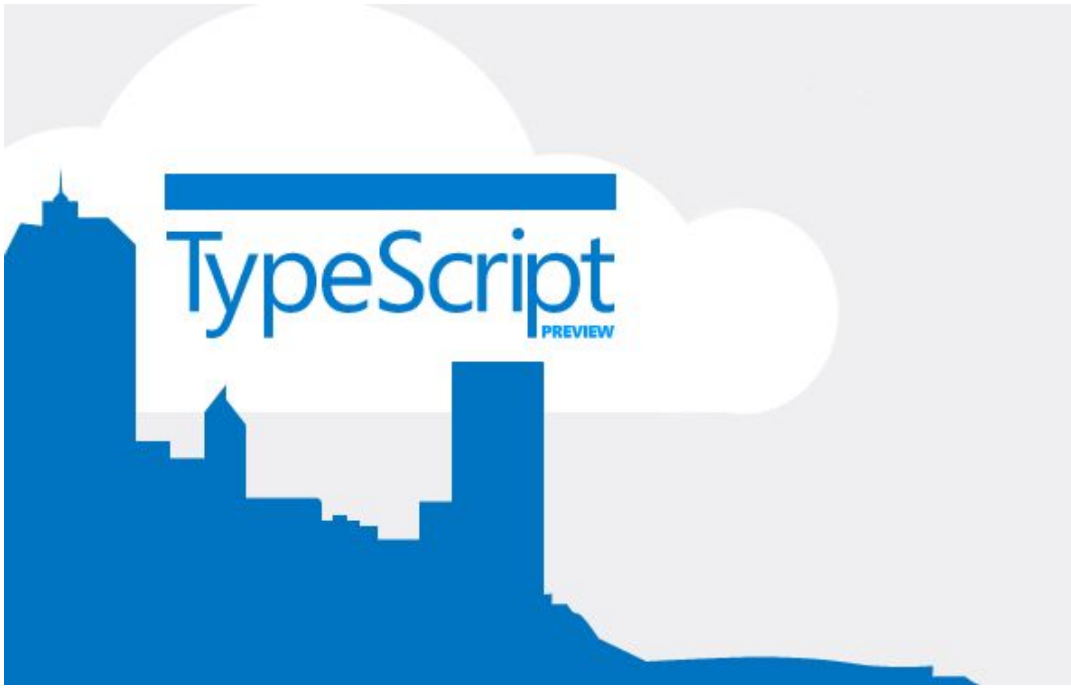
Amal Ahmed, Robert Bruce Findler, Jeremy Siek, Philip Wadler

POPL 2011

Part V

TypeScript: The Next Generation


JavaScript™


TypeScript
PREVIEW

F⋆, JavaScript, and TypeScript

A fully abstract compiler from F⋆ to JavaScript

Cedric Fournet, Nikhil Swamy, Juan Chen, Pierre-Evariste Dagand,
Pierre-Yves Strub, and Benjamin Livshits
POPL 2013

Gradual Typing Embedded Securely in JavaScript

K. Bhargavan, G. Bierman, J. Chen, C. Fournet, A. Rastogi,
P. Strub, N. Swamy
POPL 2014

TypeScript: The Next Generation

Joint proposal to MSR with Gavin Bierman.

TypeScript *interface* declares types for third-party module.
DefinitelyTyped repository declares types for 150 libraries.
But the declaration is taken of faith.

TypeScript TNG uses blame calculus to generate wrappers from *interface* declarations.

But there are problems!

- Wrappers on functions violate object identity.
- How to interpose type checks for update?
  Proxies may do the job.

Hypothesis: TypeScript TNG will aid debugging and increase reliablility of TypeScript and JavaScript code.

A wide-spectrum type system

Extend blame calculus to support a wide range of type systems:

- dynamic types (as in JavaScript or Racket)
- polymorphic types (as in F# or Haskell)
- dependent types (as in F$\star$ or Coq)

Hypothesis: a wide-spectrum type system will increase the utility of dependent types, by allowing dynamic checks to be used as a fallback when static validation is problematic.

Part VI

Other

Other

Propositions as Sessions

Philip Wadler

ICFP 2012

A Practical Theory of Language-Integrate Query

James Cheney, Sam Lindley, and Philip Wadler

ICFP 2013