TYPES ARE LIKE THE WEATHER, TYPE SYSTEMS ARE LIKE WEATHERMEN

MATTHIAS FELLEISEN, RACKETEER
Mr. Misunderstood

sing:

Two four six eight.
Who do we appreciate?

Types We think it's Types
### My Own Path to Appreciation

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1978</td>
<td>Algol 60, Simula 67, Pascal, C</td>
</tr>
<tr>
<td>1981</td>
<td>Prolog</td>
</tr>
<tr>
<td>1984</td>
<td>Scheme 84</td>
</tr>
<tr>
<td>1985</td>
<td>Russel</td>
</tr>
<tr>
<td>1987</td>
<td>Types for Scheme</td>
</tr>
<tr>
<td></td>
<td>Robert &quot;Corky&quot; Cartwright</td>
</tr>
<tr>
<td>1991</td>
<td>The Meaning of Types</td>
</tr>
<tr>
<td></td>
<td>Harper, Lee, Reynolds &amp; Scott</td>
</tr>
<tr>
<td>1993</td>
<td>CMU: ML</td>
</tr>
<tr>
<td>1994</td>
<td>Soft Scheme, HM-based inference</td>
</tr>
<tr>
<td></td>
<td>Andrew Wright</td>
</tr>
<tr>
<td>1995</td>
<td>Racket, née PLT Scheme</td>
</tr>
<tr>
<td></td>
<td>Matthew Flatt</td>
</tr>
<tr>
<td>1997</td>
<td>MrSpidey, SBA-based inference</td>
</tr>
<tr>
<td></td>
<td>Cormac Flanagan</td>
</tr>
<tr>
<td>2005</td>
<td>Typed Racket, big-bang &amp; universe</td>
</tr>
<tr>
<td></td>
<td>Sam Tobin-Hochstadt</td>
</tr>
<tr>
<td>Year</td>
<td>Event</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>1978</td>
<td>Algol 60, Simula 67, Pascal, C</td>
</tr>
<tr>
<td>1981</td>
<td>Prolog</td>
</tr>
<tr>
<td>1984</td>
<td>Scheme 84</td>
</tr>
<tr>
<td>1985</td>
<td>Russel</td>
</tr>
<tr>
<td>1987</td>
<td>Types for Scheme</td>
</tr>
<tr>
<td>1991</td>
<td>The Meaning of Types</td>
</tr>
<tr>
<td>1993</td>
<td>CMU: ML</td>
</tr>
<tr>
<td>1994</td>
<td>Soft Scheme, HM-based inference</td>
</tr>
<tr>
<td>1995</td>
<td>Racket, née PLT Scheme</td>
</tr>
<tr>
<td>1997</td>
<td>MrSpidey, SBA-based inference</td>
</tr>
<tr>
<td>2005</td>
<td>Typed Racket, big-bang &amp; universe</td>
</tr>
</tbody>
</table>
TWO TYPES OF TYPES

In some languages (C), types are merely instructions to the compiler.

In others (ML), types assist developers with maintaining software.
MAINTENANCE OF LARGE CODE BASES

Maintain >>500,000 of Racket

```java
int x = 10;
```

In others (ML), types assist developers with maintaining software
TWO MEANINGS OF "DEVELOP"

int x = 10;

In others (ML), types assist developers with maintaining software.

Maintain 100Kloc – 500Kloc
TYPES ARE LIKE THE WEATHER . . .
THERE IS NOTHING YOU CAN DO ABOUT IT. WEATHER HAPPENS.
THERE IS NOTHING YOU CAN DO ABOUT IT. COMPUTATION HAPPENS.
OF TEN, EVERYTHI NG’S JUST FINE. AN D THE ANSWER IS ALWAYS 4 2
BUT IF YOU’RE OLD ENOUGH, YOU REMEMBER THE BLUE SCREEN OF DEATH.

An error has occurred. To continue:

Press Enter to return to Windows, or

Press CTRL+ALT+DEL to restart your computer. If you do this, you will lose any unsaved information in all open applications.

Error: 0E : 016F : BFF9B3D4

Press any key to continue _
AND YES, YOU CAN GET THOSE ON UNIX AND LINUX SYSTEMS, TOO.

HP-UX 11i v3 coreadm *

```
# coreadm

global core file pattern:
init(1M) core file pattern:
global core dumps: disabled
per-process core dumps: enabled
global setid core dumps: disabled
per-process setid core dumps: disabled
```
PROGRESS! ALL YOU GOT WAS A NULL POINTER EXCEPTION.
WELL, YES. EXCEPTIONS EXIST IN YOUR FAVORITE LANGUAGE, TOO.

```
user=> (pst)
    clojure.core/eval    core.clj: 2852
...                   
user/eval2007         REPL Input
user/make-exception   user.clj:  31
user/update-row       user.clj:  23
user/make-jdbc-update-worker/reify/do-work user.clj:  18
user/jdbc-update      user.clj:   7

java.sql.SQLException: Database failure
SELECT FOO, BAR, BAZ
FROM GNIP
failed with ABC123

SQLState: "ABC"
errorCode: 123
java.lang.RuntimeException: Failure updating row
java.lang.RuntimeException: Request handling exception
nil
user=>
```
TYPE SYSTEMS ARE LIKE THE WEATHERMEN
WEATHERMEN USE MATHEMATICAL MODELS TO PREDICT THE WEATHER

This prediction is **partial** but **useful**.

It is mostly accurate.
WEATHERMEN USE MATHEMATICAL MODELS TO PREDICT THE WEATHER

\[
\frac{\partial u}{\partial t} = k \frac{\partial^2 u}{\partial x^2} + \frac{Q(x,t)}{c \rho} \\
\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \\
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = \nabla^2 u = 0 \\
\frac{\partial u}{\partial t} - 4 \frac{\partial^2 u}{\partial t^2} = \frac{\partial^3 u}{\partial x^3} + 8u - g(x,t)
\]

some partial predictions

and the emphasis is on mostly in accurate
LANGUAGES USE MATHEMATICAL MODELS TO PREDICT COMPUTATIONS

‣ This prediction is *partial* but *useful*.

‣ It is *mostly accurate*.
LANGUAGES USE MATHEMATICAL MODELS TO PREDICT COMPUTATIONS

\[
\begin{array}{c}
\text{\(f(\)}\ x\text{\()=x\)}
\end{array}
\]

\[
\begin{array}{c}
\text{\(f(\)}\ x\text{\()=x^2\)}
\end{array}
\]

And what about accuracy?
Types are the language of prediction.
Type systems use them to make more predictions.
The questions are:
- *Is that useful?*
- *Is it meaningful?*
THE MEANING OF TYPES ~ SOUNDNESS
A COMPUTATION IS A RANDOM WALK IN THE UNIVERSE OF BITS.

(def main []
  ...
  (+ x 23) ...
)

This + means machine addition, and it doesn’t care where the bits come from.

Can that happen?

What if these bits don’t represent numbers?
(def main []
... (+ x 23) ...)

Yes, in an Unsafe Language. And Life Goes on. Bits are bits.
What happens next?
IN AN **UNSOUND** LANGUAGE SUCH AS **C++**:  

```lisp
(def main []
  (... (+ x 23) ...)
)
```

If you’re lucky:  

```
0010 1000
+ 0100 0110
```

```
0110 1110
```

The computation ends in a segfault.
IN AN UNSOUND LANGUAGE SUCH AS C++:

(def main []
  ...
  (+ x 23) ...
)

And if not:

```
0010 1000
+ 0100 0110
```

```
0110 1110
```

42

The computation ends in ‘42’ and you never, ever find out that something went wrong.
Problematic bit manipulations may escape discovery during testing, even if your testing covers the particular path on which things go wrong.
Now imagine a program that controls your grandmother’s heart pacemaker.
IN AN **SOUND LANGUAGE SUCH AS ML**:

```
(def main []
  ...
  (+ x 23) ...
)
```

And in a sound language?
In an *sound* language such as ML:

```
(def main []
  
  (+ x 23)
  
)
```

It *immediately* raises an EXN.
IN AN **SOUND** LANGUAGE SUCH AS **ML**:

(def main []
  ...
  (+ x 23) ...
)

Are developers better off?

0010 1000
+ 0100 0110
0110 1110

**THIS IS THE SOURCE (THOUGH NOT NECESSARILY THE LOGICAL BUG).**
IN AN **SOUND LANGUAGE SUCH AS ML**: 

```
(def main []
  ...
  (+ x 23) ...)
```

Are users better off?

```
0010 1000
+ 0100 0110
-----
0110 1110
```

**SOMETHING BAD HAPPENED. SOMETHING WORSE MAY HAVE BEEN PREVENTED.**
As a user, don’t trust anything a program outputs.

As a developer, beware of programs that seem to work.

Even segfaults can happen far, far away in different galaxy.

As a user, consider yourself lucky when you encounter an exception.

As a developer, an EXN puts you much closer to the bug than a segfault.

The benefits of soundness make up a wide spectrum, but they shouldn’t be ignored.
Clojure comes with a single type: “the program will run”.

Bob Harper citing Dana Scott

A language with a single type can be sound.

Matthias with Andrew Wright
THE USEFULNESS OF TYPES
The expressive power of types

A single type isn’t very useful, except that it frees the developer from writing it down everywhere.

(let [m {:adam 1 :eve "paradise"}]
  .. do stuff ..)

(fn f [] "hello world")

(def f [x] ... x ...)
In an imperative world, *Void* is almost like the one type that some languages provide.

```java
for x in Map do {
    .. do stuff ..
}

void f()
   .. x ..

void f()
   .. x ..

f = “hello world”;
```
Clojure developers have many types in their mind. They just don’t have a language to write them down.
Developers have these thoughts because this is how they ‘predict’ that their programs work correctly. But some languages do not provide the means to write down these thoughts other than in comments.

And that is a problem, because code is written for others to understand the developers thoughts, and it accidentally runs on computers.
(define (compile-block decls statements next-label context add-to-top-level?)

(let* ([labels-with-numbers (map car statements)]

[labels (map (lambda (l)

  (if (stx-number? l)

    (datum->syntax l (string->symbol (format "~a" (syntax-e l))) l l)

    l))

  labels-with-numbers)]

  .. 138 more lines like this .. ))

SO HERE IS A RACKET PROGRAM FROM 15 YEARS AGO

A MISTAKE!
OH NO!!

20 MINUTES LATER; THE L SHOULD HAVE BEEN A 1, EASY!
WE LEARNED OUR LESSON. WE WROTE DOWN COMMENTS!

4 INPUT TYPES FOR 5 PARAMETERS!

WHAT'S THE PROBLEM NOW?
WRITE DOWN TYPES WHEN YOU STRUGGLE TO RECONSTRUCT THEM, AND GET THEM CHECKED.

;; start reading here:

(: compile-block [Listof Declarations] [Listof Statement] [Listof Symbol] [Listof Symbol] Boolean
   -> Code)

(define (compile-block decls statements next-label context add-to-top-level?)
  (let* ([labels-with-numbers (map car statements)]
         [labels (map (lambda (l)
                        (if (stx-number? l)
                            (datum->syntax l (string->symbol (format "a" (syntax-e l))) l l)
                            l))
                    labels-with-numbers])
   ... 138 more lines like this ...))

TYPES ARE CHECKED

A MAINTAINER CAN RELY ON THEM
TYPES ALSO HELP DEVELOP MAINTAINABLE CODE IN THE FIRST PLACE

... even in an Untyped language such as Clojure ...

MIT Press

ccs.neu.edu/home/matthias/HtDP2e/
All developers “think” types while they create code.

In some languages they can’t write down those thoughts and get them cross-checked with the program.

If they can’t write down types, they must reconstruct them.

That costs time (with spouses, kids, vacation) and money.

What can we do about this?
CAN'T WE JUST INFER THE TYPES?
HOW ABOUT TYPE INFERENCE? HASKELL IS SO COOL.

No.
How about type inference? ML has it, too.

No, it’s really not a good idea.
Why are you asking again? I said ‘no’ twice.
CAN'T WE JUST RECONSTRUCT THEM FROM THE SOURCE TEXT?

- Hindley-Milner type inference (ML, Haskell)
- Hindley-Milner with revised type algebra
- Type inference with set-based analysis
- ... with support from contracts after 15 years of research

Fundamentally, we need a language of types first, and untyped languages don't have one by definition.
ADDING TYPES TO AN UNTYPED LANGUAGE
HOW ADDING **EXPLICIT STATIC TYPES** OUGHT TO WORK

Incremental

When you have a code base of 500,000 lines, you *cannot* add types to all of this at once.

Idiomatic

Just add types. Otherwise code must not change, because it works.

Sound

The addition of types ought to narrow down the source of exceptions to cut down on future development time.
(define (f x) ;; [NEListof Number] -> Number
  .. (g x) ..)

(define (g y) ;; [NEListof Number] -> Number
  .. (h y) ..)

(define (h z) ;; [NEListof Number] -> Number
  .. (first z) ..)

(f '())

WHAT'S THE PROBLEM?
SOUNDNESS IN AN TYPED WORLD

(define (f x) :([NEListof Number] -> Number .. (g x) ..)

(define (g y) :([NEListof Number] -> Number .. (h y) ..)

(define (h z) :([NEListof Number] -> Number .. (first z) ..)

(f '())

THIS IS NOT NON-EMPTY.
One and the same variable has different types – depending on where it occurs.
A type system for an untyped language must understand this too.

```
;; shape is one of:
;;   — [square size]
;;   — [circle radius], or
;;   — a cons-pair of two shapes

(define s [square 1])
(define c [circle 2])
(define p (cons s c))

(define (area~ s)
  (cond
    [(circle? s) (area~ci s)]
    [(square? s) (area~sq s)]
    [(cons? s) (+ (area~ (car s)) (area~ (cdr s)))]))

((define (area~ s)
  (cond
    [(circle? s) (area~ci s)]
    [(square? s) (area~sq s)]
    [(cons? s) (+ (area~ (car s)) (area~ (cdr s)))]))

; Occurrence typing combines simple set-based reasoning with basic logic.
```

SHAPE

SQUARE

CIRCLE

SHAPE

SHAPE

SHAPE
HOW ARE TYPES ADDED INCREMENTALLY?

In **Typed Racket**, developers must equip entire *modules* with type annotations.

In **Reticulated Python**, developers may add types to any name, whenever, wherever.
#lang racket

(provide redo)

;; String Natural -> String
(define (redo s n) ...)
#lang racket

(provide redo)

;; String Natural -> String
(define (delete s n)
  .. (string-ref s n) ..)

#lang racket

(require "redo.rkt")

.. (delete s0 n0) ..
.. (delete s1 n1) ..
#lang typed/racket

(provide redo)

(: delete (String Natural -> String))

(define (delete s n)
  .. (string-ref s n) ..)

#lang racket

(require "redo.rkt")

.. (delete s0 n0) ..

.. (delete s1 n1) ..
Function abuse in an unchecked module

What should happen?

WHAT’S THE PROBLEM?
What should happen when the mistake happens far away?

Function abuse in an unchecked module far, far away ...

TYPED RACKET
Typed Racket generates contracts between TYPED and UNTYPED modules, & contract violations pinpoint the source, even far, far away.
Once again, the developer saves time.

Function abuse in an unchecked module
What happens if we don’t generate contracts?
#lang untyped

(require "voting-machine.rkt")

.. (setup '("Donald Duck" ...)) ..

.. (update "Donald Duck" -234) ..

#lang typed

(provide setup update ..)

(: setup (-> [Listof Name] a))

(define (setup lon) ..)

(: update (-> Name N a))

(define (update name precinct) ..)

No Contracts.

Nothing. The computation proceeds and Donald Duck loses 234 votes. Nobody will ever notice.

WHAT'S THE PROBLEM HERE?
And that’s precisely what *Typed Clojure* does ~ it masks the bugs.

Without contracts, you get all the unsoundness of C++ back.
Types for Untyped languages

▸ .. must speak the *grown idioms*.

▸ .. must allow *gradual additions*.

▸ .. ought to come with *soundness* because
  ▸ it reduces developer time
  ▸ it won’t mask errors

**THE COST IS AN OPEN PROBLEM.**
THE BIG TAKE-AWAY
Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.  

John F. Woods
THE BIG TAKE-AWAY: VALUE YOUR DEVELOPERS AND USERS

UNTYPED PROGRAMMING MAKES FOR A GOOD START

ADD TYPES IF YOU VALUE YOUR DEVELOPER’S TIME.

ADD TYPES IF YOU VALUE YOUR GRANDMOTHER’S LIFE.

WE ARE BUILDING HYBRID LANGUAGES BUT TO SOME EXTENT, IT’S ALL STILL RESEARCH.
Matthew Flatt, the Racket Man
Robby Findler, Dr. Racket, a Man with Contracts
Cormac Flanagan, Mr. Spidey
Stevie Strickland, with Class
Sam Tobin-Hochstadt, Typed
Asumu Takikawa, TOOR
Ben Greenman and Max New, Performance Matters
Alex Knauth, Alexis King, 2 wonderful freshmen

... and many many others for contributions to the code base
and even more for theoretical underpinnings, ideas, etc.
QUESTIONS?