This is is not a type:

T

Gradual typing for R

Jan Vitek, Northeastern University
Types enhance productivity
function(x) {
    var y = x ? 2 : "Y"
    if x    y += "ES"
    else    y += 40
    return y
}
Types prevent Johnny from going “wrong”

...well-typed programs cannot “go wrong”

A Theory of Type Polymorphism.

The compile-time type checker for this language has proven to be a valuable filter which traps a significant proportion of programming errors.

Once the type checker has accepted a program, code may be generated which assumes that no objects carry their types at run-time. This is widely accepted as yielding efficient object code.
m ( Object [] argh ) {
    argh [0] = new Object ()
}

m( new String[“hi”] )
The Tower of Programming Languages

JavaScript
PHP
Ruby
Smalltalk
Python
Racket
Clojure
A gradual type system can gradually enrich “scripts” with explicit and sound types without changing code

— Matthias Felleisen, TLDI 2010
From Static to Dynamic

Adding dynamic types to C#  
ECOOP 2010
Gavin Bierman¹, Erik Meijer², and Mads Torgersen²

dynamic doc = HtmlPage.Document;
dynamic win = HtmlPage.Window;
string latitude, longitude, name, address;
...

dynamic map = win.CreateInstance("VEMap", "myMap");
map.LoadMap();
map.DeleteAllShapes();

var x = win.CreateInstance("VELatLong", latitude, longitude);
var pin = map.AddPushpin(x);

doc.Title = "Information for: " + name;
pin.SetTitle(name);
pin.SetDescription(address);
map.SetCenterAndZoom(x, 9);
Runtime errors possible in dynamic operations.

Otherwise sound.
<?hh // strict

function annotating(?string $x): string {
    return $x === null ? "Hello" : "Bye";
}

function f(): void {
    // UNSAFE
    annotating(6);
}

function g(): void {
    // UNSAFE
    annotating(true);
}
Runtime errors may occur anywhere, the dynamic type system ensure memory safety but programs are unsound
A generic function's method table is equipped with the types of values. Defining methods on types directly is analogous to defining class methods in class-based object systems. With multi-methods, definitions can be associated with values that have the same type. Writing method definitions with the same name as the type inside the type definition block suggests that it should be just as easy to define or look up methods with types themselves as it is to define methods with values.

Consider the following Julia code snippet:

```julia
function Rational(num::T, den::T)
    if num == 0 && den == 0
        error("invalid rational: 0/0")
    end
    g = gcd(den, num)
    new(div(num, g), div(den, g))
end
```

This snippet defines a `Rational` type that represents a rational number. The type parameter `T` is a subtype of `Integer`, making `Rational` a composite type. Its constructors, defined using the `new` function, can be applied to constructs instances. The default constructor initializes a `Rational` value without any arguments.

The `Array Operators Using Multiple Dispatch` section discusses the use of multiple dispatch in dynamic languages like Julia, where methods are defined for types instead of values. This approach allows the language to be used for specifying domain-specific semantics, and it suggests that it should be possible to define or look up methods with types themselves as easily as with values.

The text emphasizes the importance of defining abstractions at the user level rather than hard-coding them into the language. It discusses the trade-offs between flexibility and compile-time efficiency, highlighting the importance of multiple dispatch and type inference in dynamic languages.
Runtime errors may occur at any function invocation as there are no checks, the dynamic type system ensure memory safety but programs are unsound
The Design and Implementation of Typed Scheme
POPL 2008
Sam Tobin-Hochstadt    Matthias Felleisen
PLT, Northeastern University

#lang racket

(provide (struct-out pt)
  distance)

(struct pt (x y))

; distance : pt pt -> real
(define (distance p1 p2)
  (sqrt (+ (sqr (- (pt-x p2) (pt-x p1)))
         (sqr (- (pt-y p2) (pt-y p1))))))

#lang typed/racket

(require/typed "distance.rkt"
  [#:struct pt ([x : Real] [y : Real]])
  [distance (-> pt pt Real)])

(distance (pt 3 5) (pt 7 0))
Typed Racket is sound but does not preserve all correct untyped programs.

Errors can occur anywhere but are caught and properly blamed.
fun move(p: like Point) {
    x := p.getX();
    y := p.getY();
    # p.hog(); raises compile-time err
}

fun move(p: Point) {
    x := p.getX();
    y := p.getY();
}
Runtime errors may occur in dynamic and like type code, they are dynamically caught.

Everywhere else we have soundness.
What Types for R?

- Core
- CRAN
- Users
Why Types for R?

```r
function (x, na.rm = FALSE, dims = 1L) {
    if (is.data.frame(x))
        x <- as.matrix(x)
    if (!is.array(x) || length(dn <- dim(x)) < 2L)
        stop("'x' must be an array of at least 2D")
    if (dims < 1L || dims > length(dn) - 1L)
        stop("invalid 'dims'")
}
```

Use types to systematize expectations made by a function on its arguments
function
  (x :~ Matrix(N,...),
   na.rm :: Logical = FALSE,
   dims :: Range(1,dim(x)) = 1L) {
Why Types for R?

```r
function (x, i) {
    while (x < i)
        x++
    ...

Use types to avoid unnecessary allocation and to generate efficient native code
```
function {T<:Numeric}(x :: T, i :: T) {
    while (x < i)
        x++
    ...
}
x :: Int
x :: Int[]
x :: Int[2]
$x :: \text{Int}[2,\ldots]$
x ~: Logical
x :: Int[2,...]
\{N\} (x::Int[N], y::Logical[N])
Open questions

• Types for data frames?
• Types for S3, S4, and … ?
• Types for functions…