Types and Type Checking (What is it good for?)

Stephen Chang
Racket Summer School 2018

(Thursday morning)

A quick survey

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{f} \in \mathtt{F} \Rightarrow \mathtt{f}' \qquad \Gamma \vdash \mathtt{F} \uparrow \mathtt{All}\,(\overline{\mathtt{X}}\mathtt{<:}\overline{\mathtt{S}})\overline{\mathtt{T}}\mathtt{\to R} \qquad \Gamma \vdash \overline{\mathtt{e}} \in \overline{\mathtt{U}} \Rightarrow \overline{\mathtt{e}}' \\ |\overline{\mathtt{X}}| > 0 \qquad \overline{\mathtt{X}} \cap FV(\overline{\mathtt{S}}) = \emptyset \qquad \Gamma \vdash \overline{\mathtt{A}} \mathrel{\mathtt{<:}} \overline{\mathtt{S}} \qquad \Gamma \vdash \overline{\mathtt{U}} \mathrel{\mathtt{<:}} [\overline{\mathtt{A}}/\overline{\mathtt{X}}]\overline{\mathtt{T}} \\ \forall\overline{\mathtt{B}}.\quad (\Gamma \vdash \overline{\mathtt{B}} \mathrel{\mathtt{<:}} \overline{\mathtt{S}} \quad \text{and} \quad \Gamma \vdash \overline{\mathtt{U}} \mathrel{\mathtt{<:}} [\overline{\mathtt{B}}/\overline{\mathtt{X}}]\overline{\mathtt{T}} \quad \text{imply} \quad \Gamma \vdash [\overline{\mathtt{A}}/\overline{\mathtt{X}}]\mathtt{R} \mathrel{\mathtt{<:}} [\overline{\mathtt{B}}/\overline{\mathtt{X}}]\mathtt{R})\end{array}}{\Gamma \vdash \mathtt{f}\,(\overline{\mathtt{e}}) \in [\overline{\mathtt{A}}/\overline{\mathtt{X}}]\mathtt{R} \Rightarrow \mathtt{f}'\,[\overline{\mathtt{A}}]\,(\overline{\mathtt{e}}')}$$

(App-InfSpec)

?

The Holy Grail of PL Research …(one of)

… is to predict behavior of programs
(without running them)

Prevent bugs

Avoid malware

Prove equivalence

Abandon all hope?

"All non-trivial, semantic properties
of programs are undecidable"

--- Rice's theorem

Enter … Type Systems

“A lightweight, syntactic analysis
that approximates program behavior”

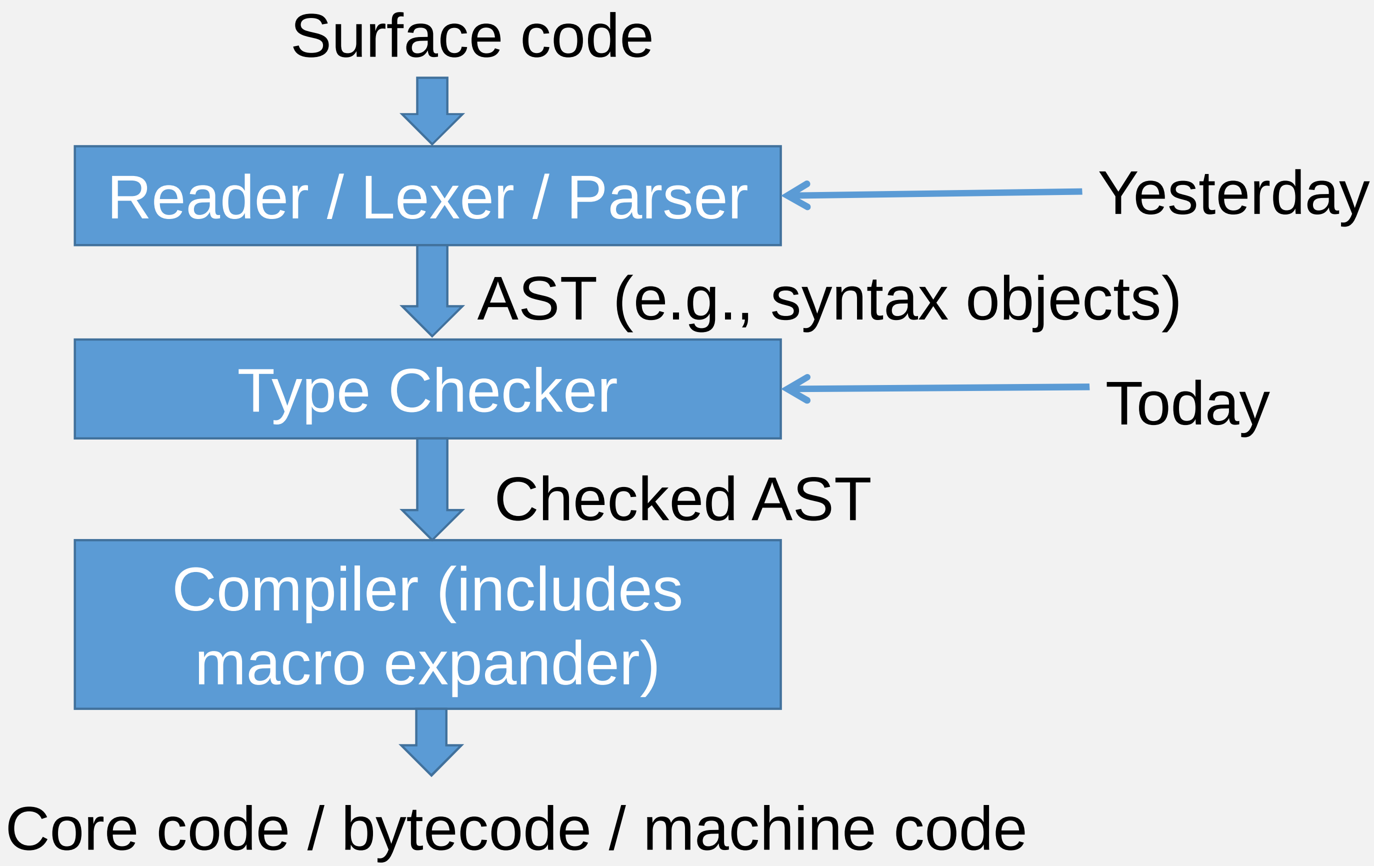
Typed Racket

Validate function arguments
(Prevent bugs)

Check memory safety
(Avoid malware)

Verify program properties
(Prove equivalence)

Typed Languages



How to create typed languages?

1. Incorporate types into the grammar.
2. Come up with a language of types.
3. Develop type rules for each language construct.
4. Implement a type checker.

How to create typed languages?

1. Incorporate types into the grammar

```
Definition = (define-
              function (Variable [Variable : Type] ...) : Type Expression)

Expression = (function-application Expression Expression ...)
           | (λ ([x : Type] ...) Expression)
           | (if Expression Expression Expression)
           | (+ Expression Expression)
           | Variable
           | Number
           | Boolean
           | String
```

How to create typed languages?

2. Come up with language of types

```
Type = (-> Type ...)
     | Number
     | Boolean
     | String
```

Specifying Type Systems: Inference Rules

3. Develop type rules for each language construct.

Premise 1

IF: Premise 2

Premise 3

THEN: Conclusion

If type systems were a video game …

IF:

THEN:

Type Judgements

$$\vdash e : \tau$$

“It is true, that expression e has type τ”

How to create typed languages?

3. Develop rules for each language construct

```
Definition = (define-
             function (Variable [Variable : Type] ...)  : Type Expression)

Expression = (function-application Expression Expression ...)
           | (λ ([x : Type] ...) Expression)
           | (if Expression Expression Expression)
           | (+ Expression Expression)
           | Variable
           | Number
           | Boolean
           | String
```

A rule for string literals

$$\frac{}{\vdash \; < string > : \text{String}}$$

"It is true, that string literals have type String"

Function Application Type Rule

“If: it is true, that expression e1 has type $\tau_{in} \rightarrow \tau_{out}$”

“and e2 has type τ_{in}”

$$\frac{\vdash e_1 : \tau_{in} \rightarrow \tau_{out} \qquad \vdash e_2 : \tau_{in}}{\vdash e_1\ e_2 : \tau_{out}}$$

“Then: it is true that e1 e2 has type τ_{out}”

If Type Rule

$$\frac{\vdash e_1 : Bool \qquad \vdash e_2 : \tau \qquad \vdash e_3 : \tau}{\vdash if\ e_1\ e_2\ e_3 : \tau}$$

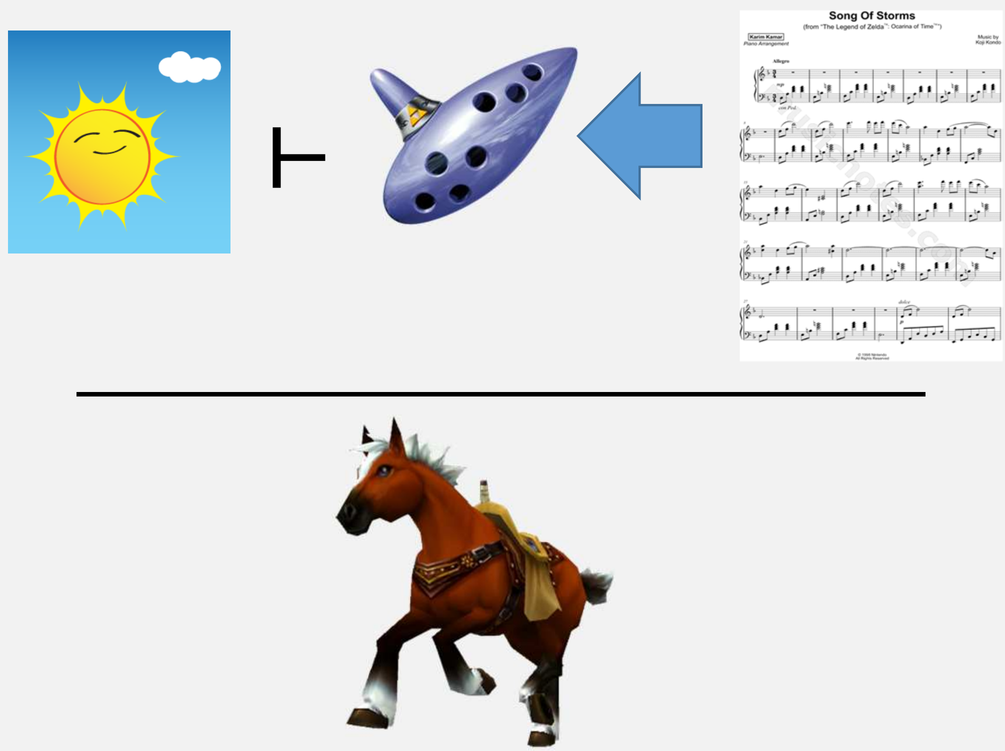
Plus type rule

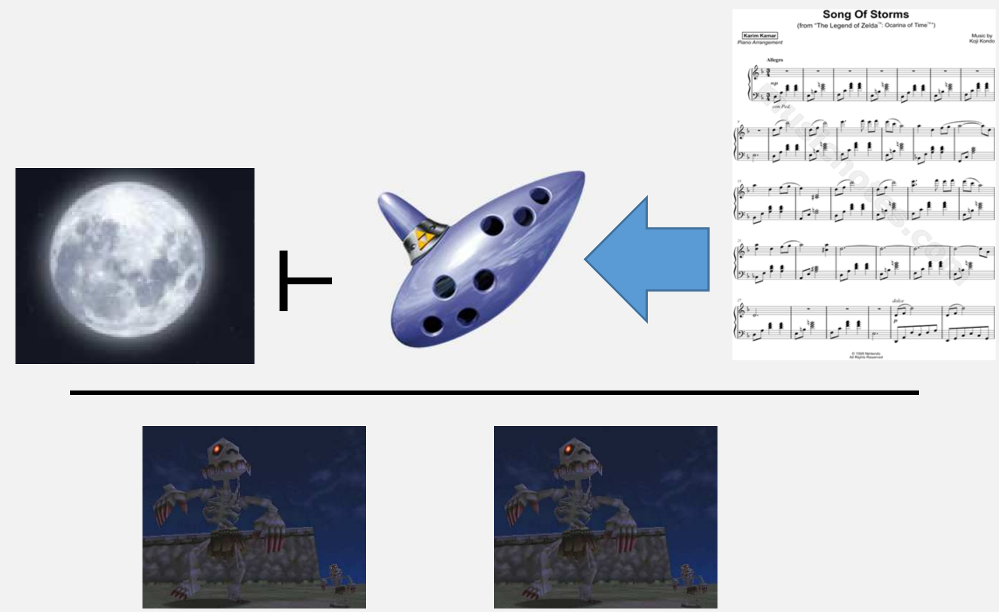
$$\frac{\begin{array}{c}\vdash e_1 : Int \\ \vdash e_2 : Int\end{array}}{\vdash e_1 + e_2 : Int}$$

Variables?

$$\frac{}{\vdash x : ???}$$

A variable's meaning depends on its context


Song Of Storms
(from "The Legend of Zelda™: Ocarina of Time™")


Song Of Storms
(from "The Legend of Zelda™: Ocarina of Time™")

Variables : Type depends on context

“In context Γ, x has type τ”

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

Context (or Type Environment)

$$\Gamma = x : \tau, \ldots$$

Type rule for lambda functions

"In context Γ, *extended with* x, which has type τin, e has type τout"

$$\frac{\Gamma, x{:}\tau_{in} \vdash e : \tau_{out}}{\Gamma \vdash \lambda x{:}\tau_{in}.e \; : \; \tau_{in} \to \tau_{out}}$$

From Type *System* … to Type *Checking*

These type rules say nothing about how to check types,
i.e., they are not an algorithm

If rule: a <u>specification</u>

$$\frac{\vdash e_1 : Bool \qquad \vdash e_2 : \tau \qquad \vdash e_3 : \tau}{\vdash if\ e_1\ e_2\ e_3 : \tau}$$

<u>IF</u>: e1 has type Bool, and e2 has type τ, and e3 has type τ
<u>THEN</u>: if e1 e2 e3 has type τ

If rule: a checking algorithm

$$\frac{\vdash e_1 : \tau_1 \quad \tau_1 = Bool \quad \vdash e_2 : \tau_2 \quad \vdash e_3 : \tau_3 \quad \tau_2 = \tau_3}{\vdash if\ e_1\ e_2\ e_3 : \tau_2}$$

1. Compute e1’s type: τ1
2. Check that τ1 is Bool
3. Compute e2’s type: τ2
4. Compute e3’s type: τ3
5. Check that e2’s type equals e3’s type
6. Assign (if e1 e2 e3) e2’s type

“Bidirectional” judgements: better for specifying algorithms

$\Leftarrow$ = “check” the type

$\Rightarrow$ = “compute” the type

A bidirectional If rule

$$\frac{\vdash e_1 \Leftarrow Bool \qquad \vdash e_2 \Rightarrow \tau \qquad \vdash e_3 \Leftarrow \tau}{\vdash if\ e_1\ e_2\ e_3 \Rightarrow \tau}$$

1. Check that e1 has type Bool
2. Compute e2’s type as τ
3. Check that e3 has type τ
4. Assign (if e1 e2 e3) type τ

“Bidirectional” judgements

$\Leftarrow$ = “check” the type

$\Rightarrow$ = “compute” the type

But now we have two judgements!

Another bidirectional If rule

$$\frac{\vdash e_1 \Leftarrow Bool \quad \vdash e_2 \Leftarrow \tau \quad \vdash e_3 \Leftarrow \tau}{\vdash if\ e_1\ e_2\ e_3 \Leftarrow \tau}$$

1. Check that e1 has type Bool
2. Compute e2’s type as τ
3. Check that e3 has type τ
4. Assign (if e1 e2 e3) type τ

LAB, part 1

- Develop bidirectional rules for our language. Focus on the expressions first.
- It might help to first review the “conventional” type rules.
- Make sure to come up with both “check” and “compute” versions of the bidirectional rules.

LAB, part 2

- Use your rules to implement a “compute” type checking function.