



---

# Programming Languages

**CS 251**  
*Fall 2021*

---

*Carolyn Anderson*

# Type-Checking

# Types

---

A type is...

- ◆ a set of values that **share some property**
- ◆ a **promise to produce** a member of a certain set of values
- ◆ a **prediction about the value** an expression will yield

# Our type judgments

## Numbers

$$\Gamma \vdash e : \text{number}$$

## Booleans

$$\Gamma \vdash e : \text{bool}$$

## Variables

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

## Addition

$$\frac{\Gamma \vdash e_1 : \text{number} \quad \Gamma \vdash e_2 : \text{number}}{\Gamma \vdash (+ e_1 e_2) : \text{number}}$$

## Functions

$$\frac{\Gamma(x \vdash \text{tau}_1) \quad \Gamma \vdash e : \text{tau}_2}{\Gamma \vdash (\text{lambda } (x : \text{tau}_1) e) : \text{tau}_1 \rightarrow \text{tau}_2}$$

## Function Applications

$$\frac{\Gamma \vdash e_1 : \text{tau}_1 \rightarrow \text{tau}_2 \quad \Gamma \vdash e_2 : \text{tau}_3}{\Gamma \vdash (e_1 e_2) : \text{tau}_2 \text{ if } \text{tau}_1 = \text{tau}_3}$$

# Practice:

---

Check the following expression using our type judgments:

```
(( (lambda (y: number)
  (lambda (x: number)
    (+ x y) ) ) 10 ) 9)
```

# Conditionals

Let's enrich our language with **if**-expressions.

$$\frac{\Gamma \vdash p:??? \quad \Gamma \vdash t:??? \quad \Gamma \vdash e:???}{\Gamma \vdash (\text{if } p \text{ } t \text{ } e):???}$$

# Conditionals

The predicate is the easy part: it must be boolean.

$$\frac{\Gamma \vdash p:\mathbf{bool} \quad \Gamma \vdash t:??? \quad \Gamma \vdash e:???}{\Gamma \vdash (\text{if } p \text{ } t \text{ } e):???$$

# Conditionals

---

What about the **if** and **else** branches?

$$\frac{\Gamma \vdash p : \mathbf{bool} \quad \Gamma \vdash t : ??? \quad \Gamma \vdash e : ???}{\Gamma \vdash (\mathbf{if} \ p \ t \ e) : ???}$$

# Conditionals

---

What about the **if** and **else** branches?

**Consider:**

`(if (= 0 x) #f (/ 10 x))`

$\Gamma \vdash p : \mathbf{bool}$      $\Gamma \vdash t : ???$      $\Gamma \vdash e : ???$

---

$\Gamma \vdash (\mathbf{if} \ p \ t \ e) : ???$

# Conditionals

---

What about the **if** and **else** branches?

Maybe they can be any type!

$$\frac{\Gamma \vdash p : \mathbf{bool} \quad \Gamma \vdash t : \mathbf{tau1} \quad \Gamma \vdash e : \mathbf{tau2}}{\Gamma \vdash (\mathbf{if} \ p \ t \ e) : ???}$$

# Conditionals

---

What about the **if** and **else** branches?

Maybe they can be any type!

**Problem:** now we can't type the whole expression!

$$\frac{\Gamma \vdash p:\mathbf{bool} \quad \Gamma \vdash t:\mathbf{tau1} \quad \Gamma \vdash e:\mathbf{tau2}}{\Gamma \vdash (\mathbf{if} \ p \ t \ e):???$$

# Conditionals

---

What about the **if** and **else** branches?

**Problem:** now we can't type the whole expression!

**Solution:** force both branches to be the same type.

$$\frac{\Gamma \vdash p:\mathbf{bool} \quad \Gamma \vdash t:\mathbf{tau} \quad \Gamma \vdash e:\mathbf{tau}}{\Gamma \vdash (\mathbf{if} \ p \ t \ e):\mathbf{tau}}$$

# Conditionals

---

**Solution:** force both branches to be the same type.

$$\frac{\Gamma \vdash p : \mathbf{bool} \quad \Gamma \vdash t : \mathbf{tau} \quad \Gamma \vdash e : \mathbf{tau}}{\Gamma \vdash (\text{if } p \text{ } t \text{ } e) : \mathbf{tau}}$$

# Evaluating our type judgment rules

---

How do we evaluate our type-checking system?

**Idea # 1:** Try out some programs

# Evaluating our type judgment rules

---

How do we evaluate our type-checking system?

**Idea # 1:** Try out some programs

**Idea # 2:** Implement and try out more programs

# Implementing a Racket Type-Checker

# Evaluating our type judgment rules

---

How do we evaluate our type-checking system?

**Idea # 1:** Try out some programs

**Idea # 2:** Implement and try out more programs

# Evaluating our type judgment rules

---

How do we evaluate our type-checking system?

**Idea # 1:** Try out some programs

**Idea # 2:** Implement and try out more programs

**Idea # 3:** Prove it correct

# Type soundness

---

The goal of static typing is to learn some things about our program's behavior without running it. This lets us catch errors early and makes our programs safer.

*Well-typed programs do not go wrong.*

**Robin Milner**

# Type soundness

---

One useful property is **type soundness**. A sound type system is one that correctly predicts the type of the values that the program produces.

More formally:

**Type soundness:** for all programs  $p$ , if (1) the type checker assigns  $p$  the type  $\tau$ , and if (2) the semantics says that  $p$  evaluates to a value  $v$ , then the type checker will also assign  $v$  the type  $\tau$ .

# Type soundness

---

**Type soundness:** for all programs  $p$ , if (1) the type checker assigns  $p$  the type  $\tau$ , and if (2) the semantics says that  $p$  evaluates to a value  $v$ , then the type checker will also assign  $v$  the type  $\tau$ .

What does this say about cases where the program doesn't evaluate to a value, such as:

- ◆ side effects
- ◆ failure to terminate
- ◆ errors

# Type soundness

---

Also, note the two components of this definition!

Type soundness requires **participation from the run-time system** (which implements the semantics).

**Type soundness:** for all programs  $p$ , if (1) the type checker assigns  $p$  the type  $\tau$ , and if (2) the semantics says that  $p$  evaluates to a value  $v$ , then the type checker will also assign  $v$  the type  $\tau$ .

# Type safety

---

**Type safety:** no primitive operation ever applies to values of the wrong type.

This can't be guaranteed by the static type-checker alone. As we have seen, many aspects of program behavior can't be statically checked.

Instead, this is a property of the type system + the run-time system: a safe language is one where the run-time system respects type abstractions.

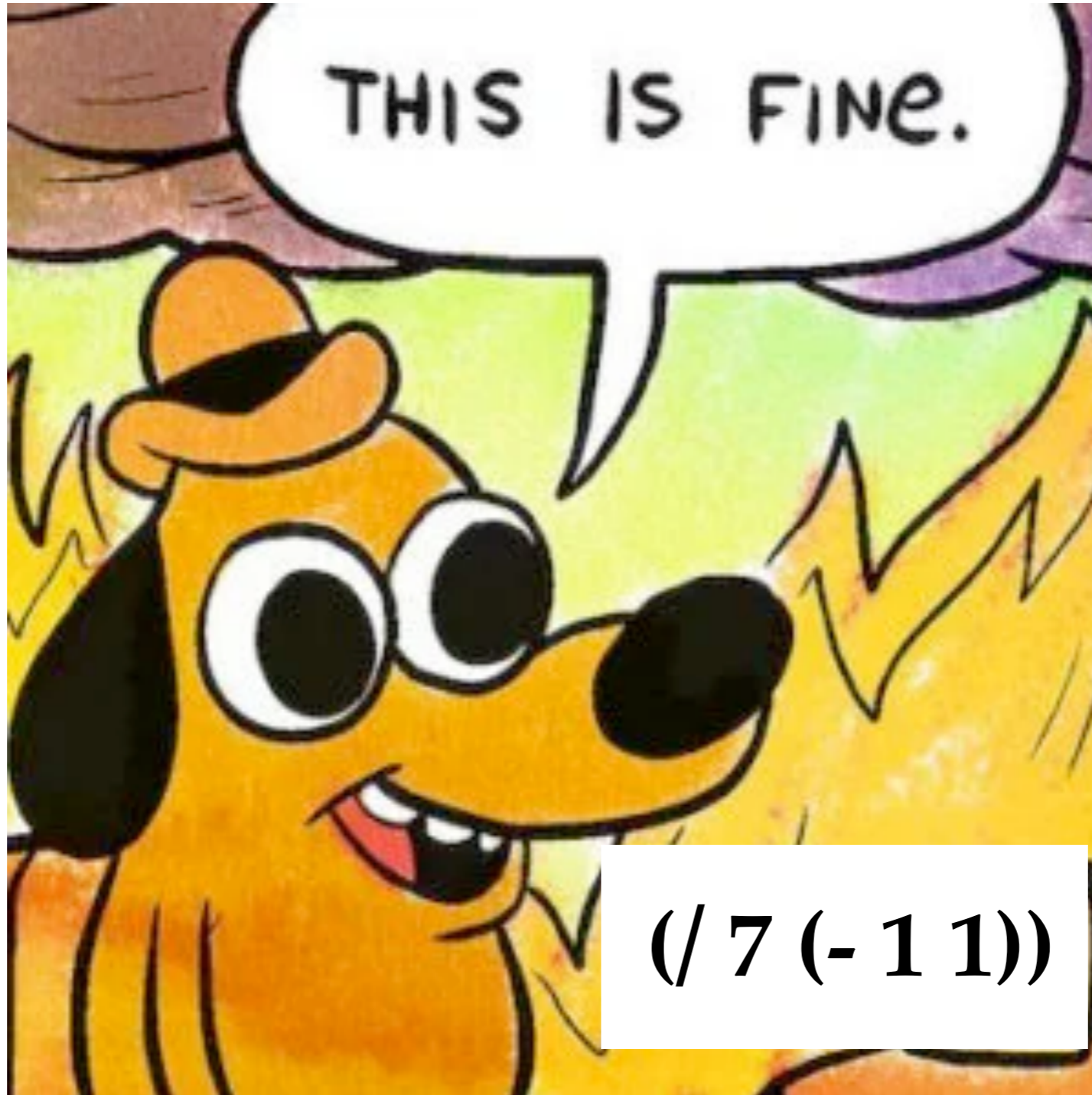
# Can all errors be statically caught?

---

We've seen some success in catching errors with our simple static type-checker. But are there errors that can't be caught?

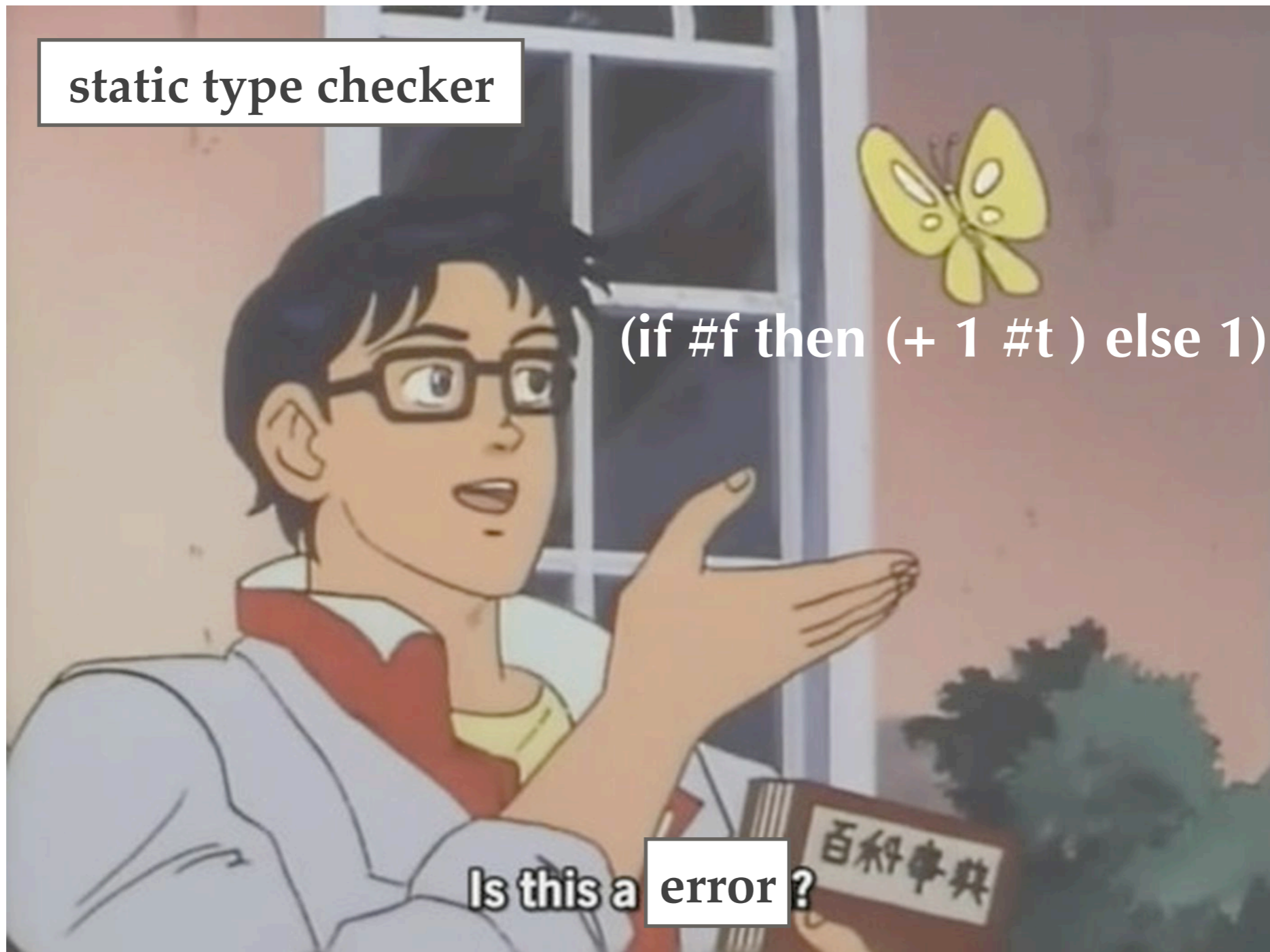
# The limits of static type-checking

Some type errors can't be caught statically. 😞



# The limits of static type-checking

Static type-checking restricts some dynamic behavior:



# Static versus Dynamic Type-Checking

# Static Type Checking

---

- ◆ May reject a program after parsing, before running.
- ◆ Catches some bugs early (before running)
- ◆ May reject programs that would not have produced an error (over-cautious)

# Dynamic Type Checking

---

- ◆ Only rejects syntax errors before running
- ◆ Doesn't reject any programs unnecessarily

# Which is better?

---

## **Convenience:**

It's annoying to write types... but it's also annoying to have bugs.

# Which is better?

---

## **Convenience:**

It's annoying to write types... but it's also annoying to have bugs.

## **Expressiveness:**

Static typing rejects some programs that run and can make some abstractions difficult to implement.

# Which is better?

---

## **Convenience:**

It's annoying to write types... but it's also annoying to have bugs.

## **Expressiveness:**

Static typing rejects some programs that run and can make some abstractions difficult to implement.

## **Security:**

Static typing catches more bugs... but not all bugs.

# Which is better?

---

## **Convenience:**

It's annoying to write types... but it's also annoying to have bugs.

## **Expressiveness:**

Static typing rejects some programs that run and can make some abstractions difficult to implement.

## **Security:**

Static typing catches more bugs... but not all bugs.

## **Efficiency:**

Compilers can use static types to optimize code... compilers can also remove unnecessary dynamic checks.

# Which is better?

---

## **Convenience:**

It's annoying to write types... but it's also annoying to have bugs.

## **Expressiveness:**

Static typing rejects some programs that run and can make some abstractions difficult to implement.

## **Security:**

Static typing catches more bugs... but not all bugs.

## **Efficiency:**

Compilers can use static types to optimize code... compilers can also remove unnecessary dynamic checks.

## **Code reuse:**

Untyped code can be applied more flexibly... but advanced type systems also allow flexible use. Types are useful for documentation.

# Summary

---

- ◆ It's a complex issue!
- ◆ Static versus dynamic may be too broad of a debate
- ◆ Better questions: - **what** checks should be enforced?  
- **when** should checks be enforced?