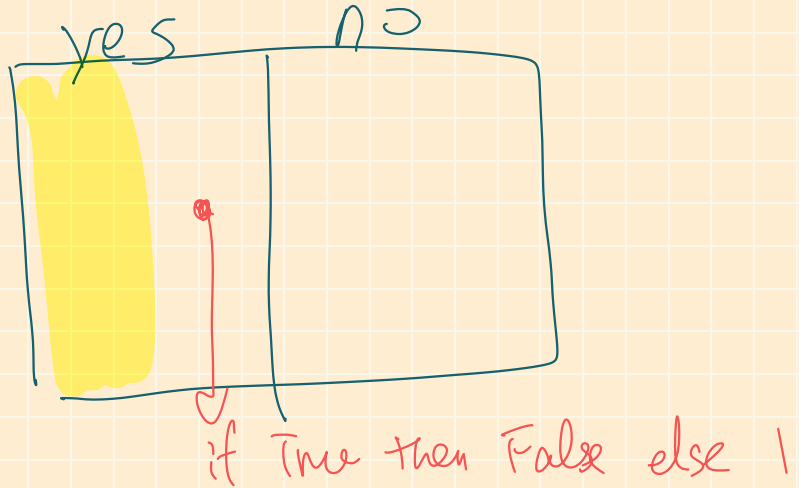
Review

Exception ⇒ undefined behaviors (stuck)

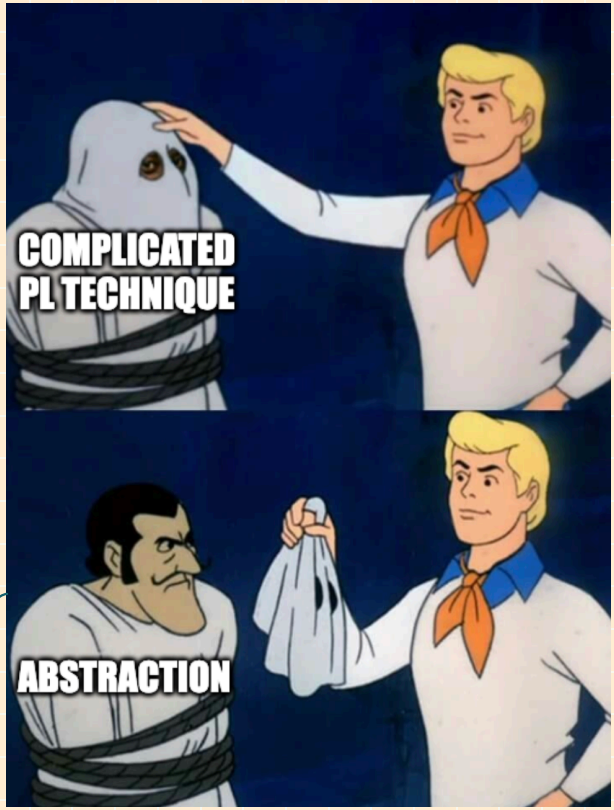
1 + True

1. What question are type systems designed to predict?
2. Draw a picture of a sound but incomplete static analysis.
3. Type systems are meant to be sound / complete.
4. Can a static analysis be both sound and complete?



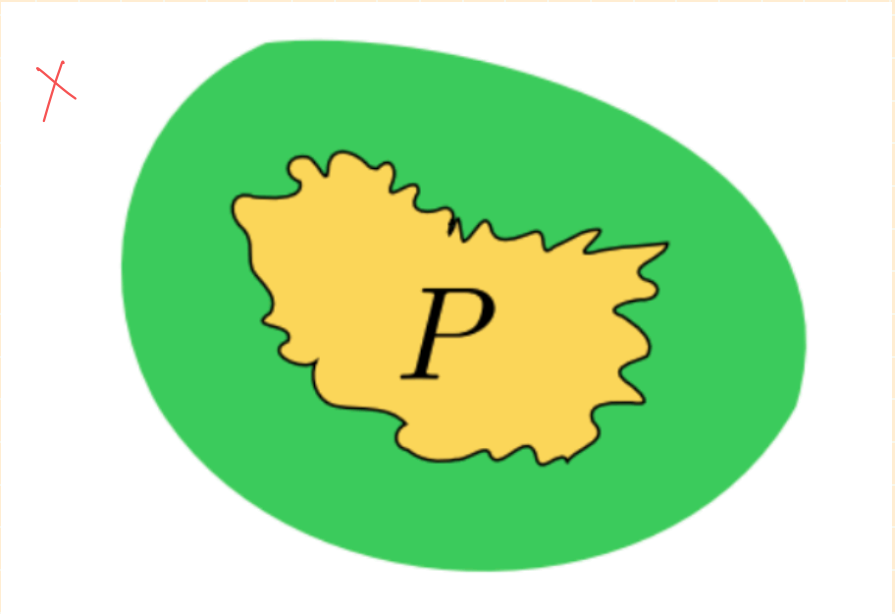
yes

What is the only general method for designing sound analysis?

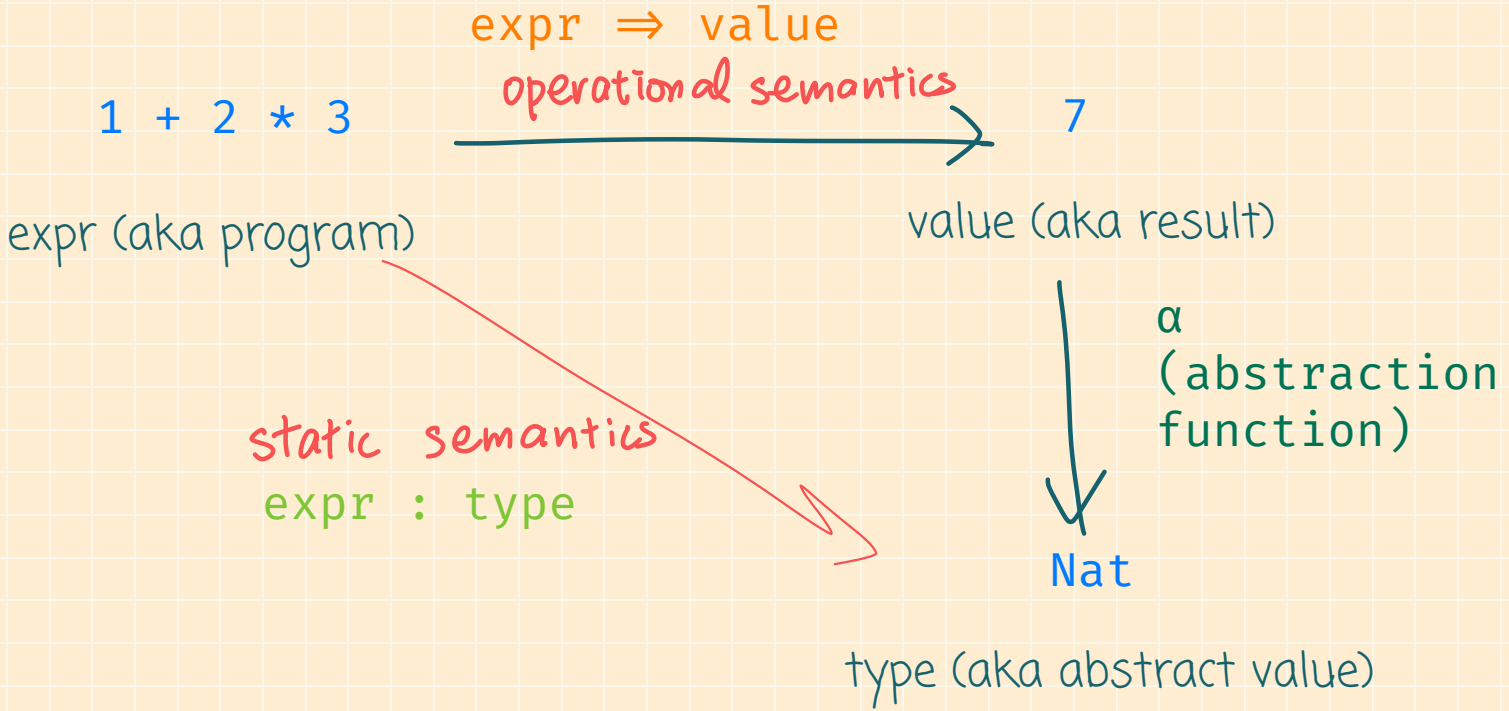


Over-Approximation

Draw a picture to illustrate how abstraction/over-approximation works



Draw the 2 steps of designing a type system



Operational semantics (dynamic/runtime) $e \Rightarrow v$

Abstract semantics (static/compile-time) $e : t$

$$\frac{}{n \Rightarrow n}\ \textsc{Nat}$$

$$n : \text{Nat}$$

$$\frac{}{b \Rightarrow b}\ \textsc{Bool}$$

$$b : \text{Bool}$$

$$\frac{e_1 \Rightarrow n_1 \quad e_2 \Rightarrow n_2 \quad (\oplus \in \{+, -, *\}) \quad (n_1 \oplus n_2 = n_3)}{e_1 \oplus e_2 \Rightarrow n_3}\ \textsc{Arith}$$

$$\frac{e_1 : \text{Nat} \quad e_2 : \text{Nat}}{e_1 + e_2 : \text{Nat}}$$

$$\frac{e_1 \Rightarrow n_1 \quad e_2 \Rightarrow n_2 \quad (n_1 = n_2)}{e_1 == e_2 \Rightarrow \text{True}}\ \textsc{EqTrue}$$

$$\frac{e_1 \Rightarrow n_1 \quad e_2 \Rightarrow n_2 \quad (n_1 \neq n_2)}{e_1 == e_2 \Rightarrow \text{False}}\ \textsc{EqFalse}$$

if True then 1 else False

$$\frac{e_1 \Rightarrow \text{True} \quad e_2 \Rightarrow v}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \Rightarrow v}\ \textsc{IfTrue}$$

$$\frac{e_1 \Rightarrow \text{False} \quad e_3 \Rightarrow v}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \Rightarrow v}\ \textsc{IfFalse}$$

$$\frac{e_1 : \text{Bool} \quad e_2 : t_2 \quad e_3 : t_3 \quad (t_2 = t_3)}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 : t_2}$$

Theorem (Type Soundness)

If e : t for some t, then e => v for some v

∅ = base

>∅ = inductive

"Well-typed programs don't go wrong."

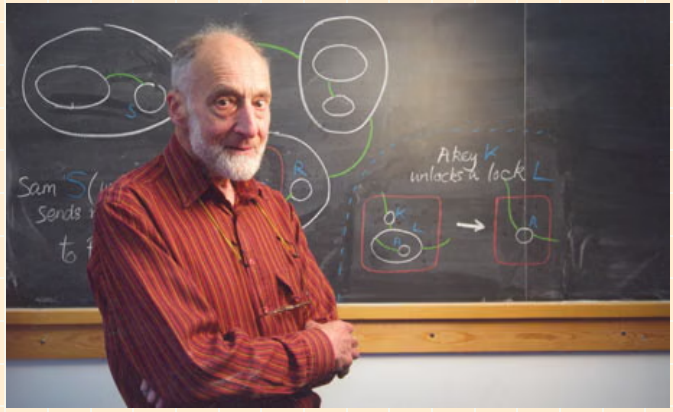
Proof (attempt 1)

Induction on AST size

Base cases (size(e) = 0) ← leaf nodes

$$\frac{}{e = b : \text{Bool}}\ \text{T-Bool} \qquad \frac{}{b \Rightarrow b}\ \text{Bool}$$

$$\frac{}{e = n : \text{Nat}}\ \text{T-Nat} \qquad \frac{}{n \Rightarrow n}\ \text{Nat}$$



Robin Milner (1934-2010)

Theorem (Type Soundness)

If e : t for some t, then e => v for some v

$v ::= n \mid b$

size(e) > 0 $e_1 + e_2$

Inductive cases (size(e) > 0)

Can we apply IH? If so, to what?

$$\frac{e1 : Nat \qquad e2 : Nat}{e1 + e2 : Nat}\ \text{T-Nat}$$

By the induction hypothesis, we have

- e1 => v1 for some v1 $= n_1 \mid b_1$
- e2 => v2 for some v2 $= n_2 \mid b_2$

Then?

$v_1 = n_1$, $v_2 = b_2$

Our induction hypothesis is too weak

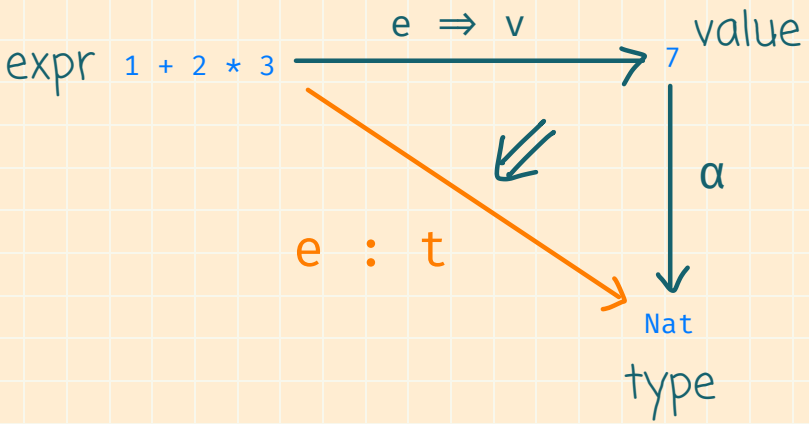
Theorem (Type Soundness)

If e : t for some t, then e => v for some v

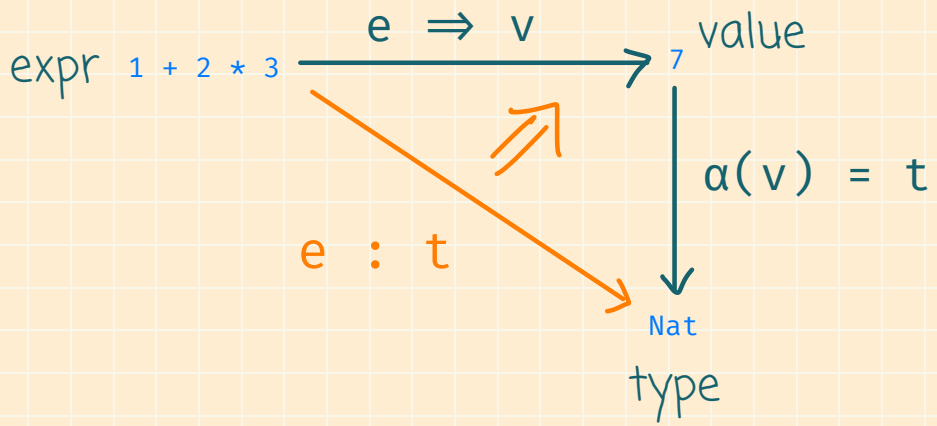
Lemma (Type Abstraction)

If e : t for some t, then e => v for some v, AND α(v) = t

Where the heck does this lemma come from?



Design a type system by composing arrows



Prove soundness by inverting the arrows

Lemma (Type Abstraction)

If e : t for some t, then e => v for some v, AND α(v) = t

Proof by induction in AST size

Lemma (Type Abstraction)

If e : t for some t, then e => v for some v, AND α(v) = t

Base cases (size(e) = 0)

$$\frac{}{b : \text{Bool}}\ \text{T-Bool}$$

$$\frac{}{n : \text{Nat}}\ \text{T-Nat}$$

Inductive cases (size(e) > 0)

$$\frac{e1 : \text{Nat} \qquad e2 : \text{Nat}}{e1 + e2 : \text{Nat}}\ \text{T-Arith}$$

$$\frac{e1 : \text{Nat} \qquad e2 : \text{Nat}}{e1 == e2 : \text{Bool}}\ \text{T-Eq}$$

$b \Rightarrow b$ $\quad \alpha(b) = 1$ $\quad$ Int

$e_1 \Rightarrow v_1$, $e_2 \Rightarrow v_2$ $\quad \alpha(v_1) = \text{Nat}$, $\alpha(v_2) = \text{Nat}$

$v_1 = n_1$

$v_2 = n_2$

HW + Quiz 2

$$\frac{e1 : \text{Bool} \quad e2 : t2 \quad e3 : t3 \quad (t2 = t3)}{\text{if } e1 \text{ then } e2 \text{ else } e3 : t2} \text{ T-If}$$

By IH, $e_1 \Rightarrow v_1$, $\alpha(v_1) = \text{Bool}$

$e_2 \Rightarrow v_2$, $\alpha(v_2) = t_2$ $\Rightarrow$ $\alpha(v_2) = t_2$

$e_3 \Rightarrow v_3$, $\alpha(v_3) = t_3$

$$\frac{e_1 \Rightarrow \text{True} \qquad e_2 \Rightarrow v_2}{\text{if } e_1 \cdots \Rightarrow v_2} \text{ If True}$$

$$\frac{e_1 \Rightarrow \text{False} \qquad e_3 \Rightarrow v_3}{\text{if } e_1 \cdots \Rightarrow v_3} \text{ If False}$$

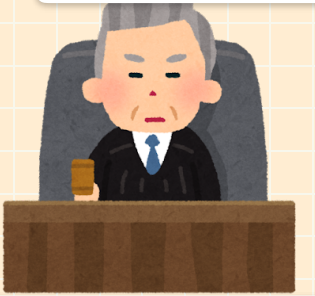
$\alpha(v_3) = t_2 = t_3$

That wasn't too hard... right?

Code needs to be updated

You're hired to fix their shit & prove it's unbreakable

AI writes shit code



Company gets sued

Bugs lead to disasters

Both the operational semantics and the type system are carefully designed to ensure soundness.

What could have gone wrong?

$$\frac{e_1 : \text{Bool} \quad e_2 : t_2}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 : t_2} \; ?$$

Supporting variables

$e : t$

$\Gamma \vdash e : t$

$$\Gamma ::= \cdot \quad \text{empty} \qquad \Gamma, x : t \quad \text{binding}$$

$$\frac{e_1 \Rightarrow v_1 \qquad ([v_1/x]e_2 = e_2') \qquad e_2' \Rightarrow v_2}{\mathsf{let}\, x = e_1 \,\mathsf{in}\, e_2 \Rightarrow v_2}\ \textsc{Let}$$

$$\frac{\Gamma \vdash e_1 : t_1 \qquad \Gamma, x : t_1 \vdash e_2 : t_2}{\Gamma \vdash \mathsf{let}\, x = e_1 \,\mathsf{in}\, e_2 : t_2}\ \textsc{T-Let}$$

(No rule for free variables)

$$\frac{}{\Gamma, x : t \vdash x : t}\ \textsc{T-Id} \qquad \frac{\Gamma \vdash x : t \qquad (x \neq y)}{\Gamma, y : t \vdash x : t}\ \textsc{T-Weaken}$$

$$\frac{e_1 \Rightarrow v_1 \qquad e_2 \Rightarrow v_2}{(e_1, e_2) \Rightarrow (v_1, v_2)}\ \textsc{Pack}$$

$$\frac{\Gamma \vdash e_1 : t_1 \qquad \Gamma \vdash e_2 : t_2}{\Gamma \vdash (e_1, e_2) : (t_1, t_2)}\ \textsc{T-Pack}$$

$$\frac{e_1 \Rightarrow (v_{11}, v_{12}) \qquad ([v_{11}/x_1, v_{12}/x_2]e_2 = e_2') \qquad e_2' \Rightarrow v_2}{\mathsf{let}\ (x_1, x_2) = e_1 \,\mathsf{in}\, e_2 \Rightarrow v_2}\ \textsc{Unpack}$$

$$\frac{\Gamma \vdash e_1 : (t_{11}, t_{12}) \qquad \Gamma, x_1 : t_{11}, x_2 : t_{12} \vdash e_2 : t}{\Gamma \vdash \mathsf{let}\ (x_1, x_2) = e_1 \,\mathsf{in}\, e_2 : t}\ \textsc{T-Unpack}$$

$$\frac{}{\lambda x.\ e \Rightarrow \lambda x.\ e}\ \text{Lambda}$$

$$\frac{\Gamma, x : t_i \vdash e : t_o}{\Gamma \vdash \lambda x.\ e : t_i \to t_o}\ \textsc{T-Lambda}$$

$$\frac{e_1 \Rightarrow \lambda x.\ e \qquad e_2 \Rightarrow v \qquad ([v/x]e = e') \qquad e' \Rightarrow v'}{e_1\ e_2 \Rightarrow v'}\ \text{App}$$

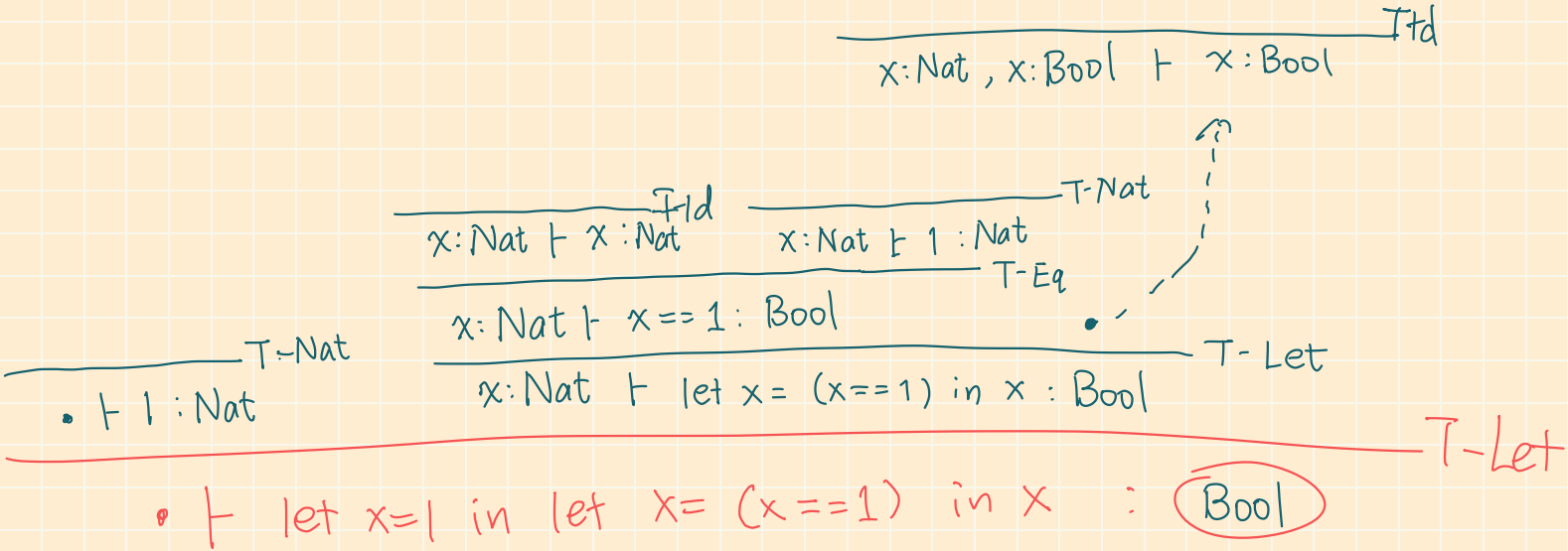
$$\frac{\Gamma \vdash e_1 : t_i \to t_o \qquad \Gamma \vdash e_2 : t_i}{\Gamma \vdash e_1\ e_2 : t_o}\ \textsc{T-App}$$

Exercise:

Weaken rule should be

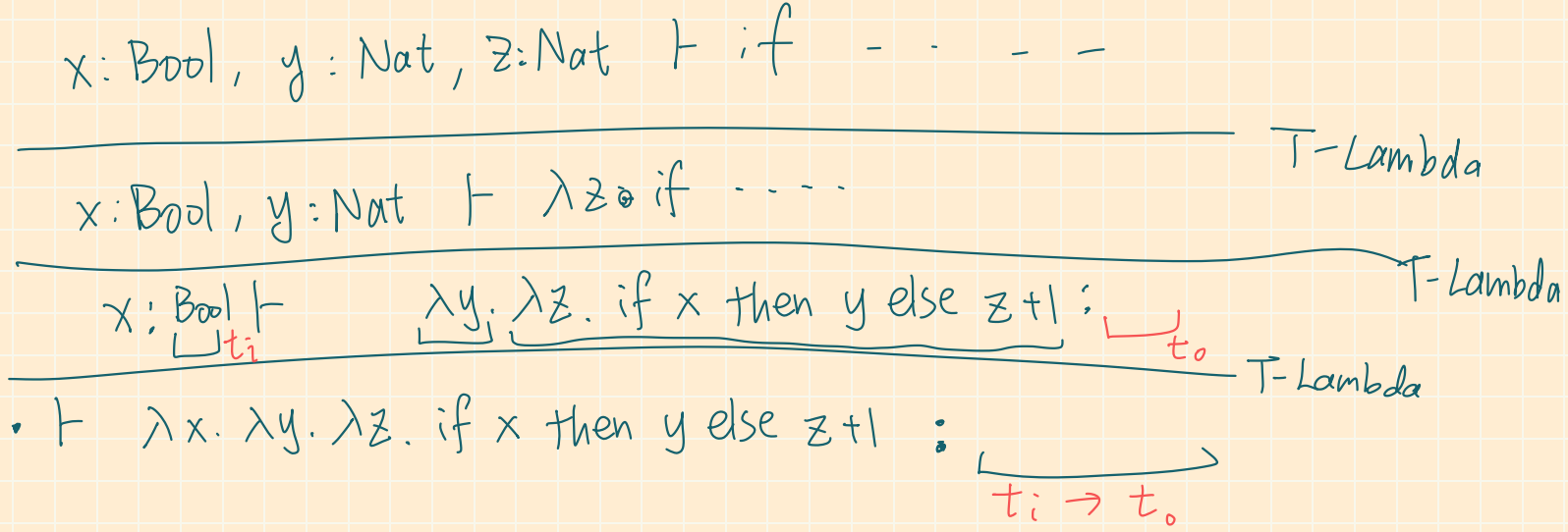
$$\frac{(x \neq y) \qquad \Gamma \vdash x : t_1}{\Gamma, y : t_2 \vdash x : t_1}\ \text{Weaken}$$

Draw the typing derivation tree for let x = 1 in let x = (x == 1) in x



Exercise:

Draw the typing derivation tree for \x. \y. \z. if x then y else z + 1



Problem: Our typing judgment isn't "algorithmic"

$$\Gamma \vdash e : t$$

1. Sometimes, we can compute t from e
2. Other times, an oracle needs to tell us what t is

Exercise: for each of the following rule, determine whether you can compute the t below the line as an output.

$$\frac{}{\Gamma \vdash n : \mathsf{Nat}}\ \textsc{T-Nat}$$

$$\frac{}{\Gamma, x : t \vdash x : t}\ \textsc{T-Id}$$

$$\frac{\Gamma, x : t_i \vdash e : t_o}{\Gamma \vdash \lambda x.\ e : t_i \to t_o}\ \textsc{T-Lambda}$$

$$\frac{\Gamma \vdash e_1 : t_i \to t_o \qquad \Gamma \vdash e_2 : t_i}{\Gamma \vdash e_1\ e_2 : t_o}\ \textsc{T-App}$$

Solution: decompose the typing judgment into 2 sub-judgments

```
def synth(gamma, e: Expr) → Type
```

Type synthesis: $\Gamma \vdash e \downarrow t$

output

$\Gamma \vdash e : t$

1. Sometimes, we can compute t from e
2. Other times, an oracle needs to tell us what t is

input

Type checking: $\Gamma \vdash e \uparrow t$

```
def check(gamma, e: Expr, t: Type) → bool
```

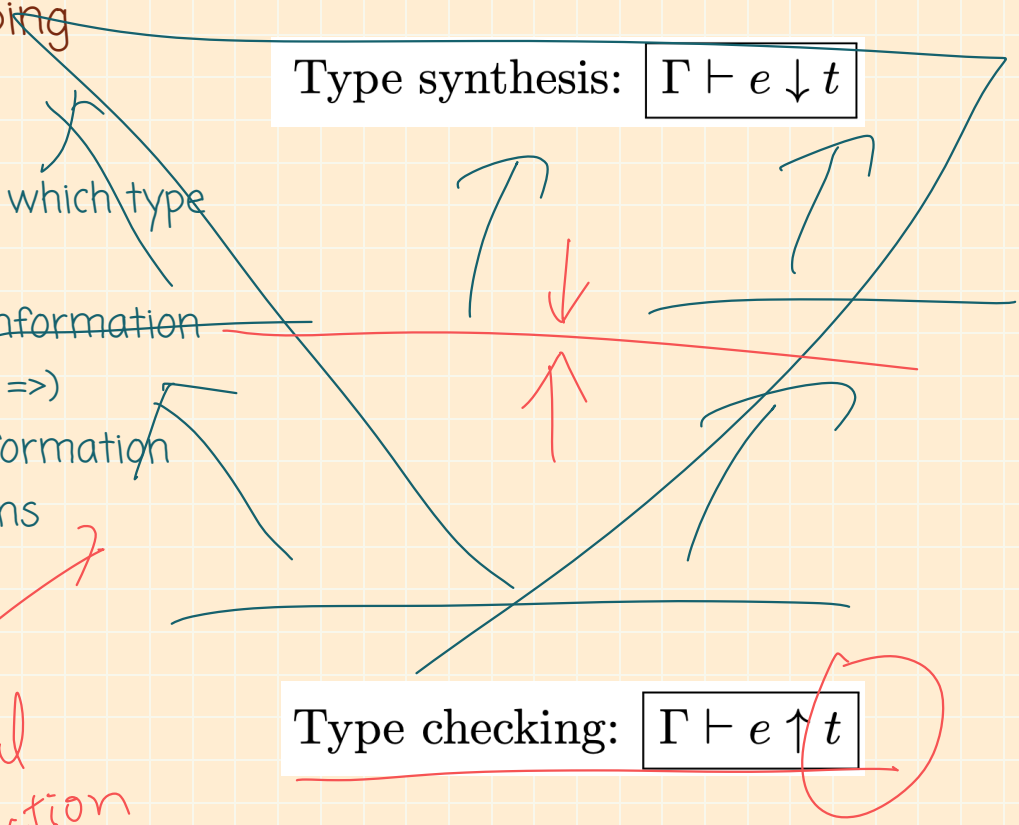
"Bidirectional Typing"

- check & synth mutually recursive
- VERY powerful technique in type system implementation
- Minimimal type annotations required (programmers only need to annotate top-level abbreviations)
- Scales to advanced type system features

Intuition of bidirectional typing

Arrows indicate the directions in which type information flows

- Down arrow = compute type information from sub-expression (just like =>)
- Up arrow = propagate type information from context to sub-expressions



Challenge: we need to decide for every $\Gamma \vdash e : t$

to implement it as $\Gamma \vdash e \downarrow t$ or $\Gamma \vdash e \uparrow t$

$$\frac{}{\Gamma, x : t \vdash x : t}\ \text{T-Id} \qquad \frac{\Gamma \vdash x : t \quad (x \neq y)}{\Gamma, y : t \vdash x : t}\ \text{T-Weaken}$$

$$\frac{}{\Gamma \vdash n : \mathsf{Nat}}\ \text{T-Nat} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Nat} \quad \Gamma \vdash e_2 : \mathsf{Nat} \quad (\oplus \in \{+, -, *\})}{\Gamma \vdash e_1 \oplus e_2 : \mathsf{Nat}}\ \text{T-Arith} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Nat} \quad \Gamma \vdash e_2 : \mathsf{Nat}}{\Gamma \vdash e_1 == e_2 : \mathsf{Bool}}\ \text{T-Eq}$$

$$\frac{}{\Gamma \vdash b : \mathsf{Bool}}\ \text{T-Bool} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Bool} \quad \Gamma \vdash e_2 : t_2 \quad \Gamma \vdash e_3 : t_3 \quad (t_2 = t_3)}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : t_2}\ \text{T-If}$$

(No rule for variables) $\qquad \frac{(\Delta(X) = t) \quad \Gamma \vdash e : t}{\Gamma \vdash X : t}\ \text{T-Abbrev} \qquad \frac{\Gamma \vdash e_1 : t_1 \quad \Gamma, x : t_1 \vdash e_2 : t_2}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : t_2}\ \text{T-Let}$

$$\frac{\Gamma, x : t_i \vdash e : t_o}{\Gamma \vdash \lambda x.\ e : t_i \to t_o}\ \text{T-Lambda} \qquad \frac{\Gamma \vdash e_1 : t_i \to t_o \quad \Gamma \vdash e_2 : t_i}{\Gamma \vdash e_1\ e_2 : t_o}\ \text{T-App}$$

$$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma \vdash e_2 : t_2}{\Gamma \vdash (e_1, e_2) : (t_1, t_2)}\ \text{T-Pack} \qquad \frac{\Gamma \vdash e_1 : (t_{11}, t_{12}) \quad \Gamma, x_1 : t_{11}, x_2 : t_{12} \vdash e_2 : t}{\Gamma \vdash \mathsf{let}\ (x_1, x_2) = e_1\ \mathsf{in}\ e_2 : t}\ \text{T-Unpack}$$

$$\frac{\Gamma \vdash e : t}{\Gamma \vdash (e : t) : t}\ \text{T-Ann}$$

BUT HOW ?????????????

Solution 1 https://www.youtube.com/watch?v=ZCPN9SfdH7c

$$\frac{}{\Gamma, x : t \vdash x : t}\ \text{T-Id} \qquad \frac{\Gamma \vdash x : t \quad (x \neq y)}{\Gamma, y : t \vdash x : t}\ \text{T-Weaken}$$

$$\frac{}{\Gamma \vdash n : \mathsf{Nat}}\ \text{T-Nat} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Nat} \quad \Gamma \vdash e_2 : \mathsf{Nat} \quad (\oplus \in \{+, -, *\})}{\Gamma \vdash e_1 \oplus e_2 : \mathsf{Nat}}\ \text{T-Arith} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Nat} \quad \Gamma \vdash e_2 : \mathsf{Nat}}{\Gamma \vdash e_1 == e_2 : \mathsf{Bool}}\ \text{T-Eq}$$

$$\frac{}{\Gamma \vdash b : \mathsf{Bool}}\ \text{T-Bool} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Bool} \quad \Gamma \vdash e_2 : t_2 \quad \Gamma \vdash e_3 : t_3 \quad (t_2 = t_3)}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : t_2}\ \text{T-If}$$

(No rule for variables) $\qquad \frac{(\Delta(X) = t) \quad \Gamma \vdash e : t}{\Gamma \vdash X : t}\ \text{T-Abbrev} \qquad \frac{\Gamma \vdash e_1 : t_1 \quad \Gamma, x : t_1 \vdash e_2 : t_2}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : t_2}\ \text{T-Let}$

$$\frac{\Gamma, x : t_i \vdash e : t_o}{\Gamma \vdash \lambda x.\ e : t_i \to t_o}\ \text{T-Lambda} \qquad \frac{\Gamma \vdash e_1 : t_i \to t_o \quad \Gamma \vdash e_2 : t_i}{\Gamma \vdash e_1\ e_2 : t_o}\ \text{T-App}$$

$$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma \vdash e_2 : t_2}{\Gamma \vdash (e_1, e_2) : (t_1, t_2)}\ \text{T-Pack} \qquad \frac{\Gamma \vdash e_1 : (t_{11}, t_{12}) \quad \Gamma, x_1 : t_{11}, x_2 : t_{12} \vdash e_2 : t}{\Gamma \vdash \mathsf{let}\ (x_1, x_2) = e_1\ \mathsf{in}\ e_2 : t}\ \text{T-Unpack}$$

$$\frac{\Gamma \vdash e : t}{\Gamma \vdash (e : t) : t}\ \text{T-Ann}$$

Non-solution: try all possibilities.

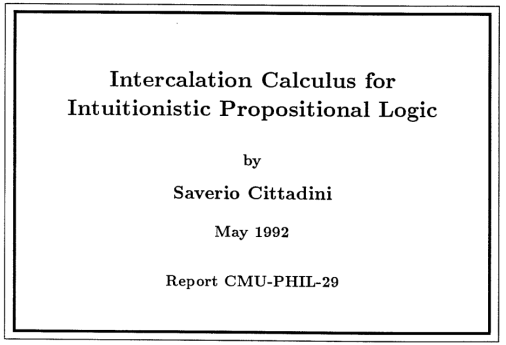
- There're 33 $\Gamma \vdash e : t$ in our type system
- Every e: t has two choices
- 2^33 = 8.5 billion alternative implementations

Solution 2: talk to a logician and steal their work

PL designer

Work of mathematicians & logicians

The Pfenning Recipe for designing bidirectional type systems



Intercalation Calculus for
Intuitionistic Propositional Logic

by

Saverio Cittadini

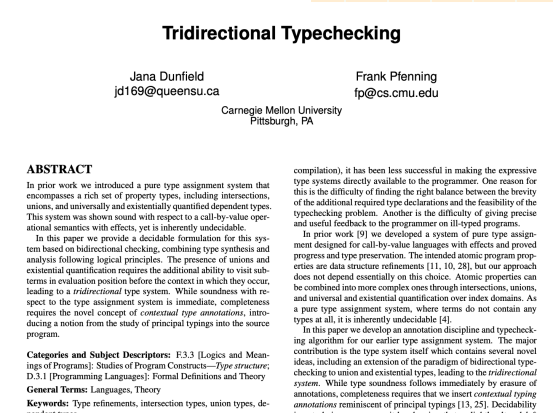
May 1992

Report CMU-PHIL-29

Carnegie Mellon

Philosophy
Methodology
Logic

Pittsburgh, Penn



Tridirectional Typechecking

Jana Dunfield
jd169@queensu.ca

Frank Pfenning
fp@cs.cmu.edu

Carnegie Mellon University
Pittsburgh, PA

ABSTRACT

In prior work we introduced a pure type assignment system that encompasses a rich set of property types, including intersections, unions, and universally and existentially quantified dependent types. This system was shown sound with respect to a call-by-value operational semantics with effects, yet is inherently undecidable.

In this paper we provide a decidable formulation for this system based on bidirectional checking, combining type synthesis and analysis following logical principles. The presence of unions and existential quantification requires the additional ability to visit subterms in evaluation position before the context in which they occur, leading to a *tridirectional* type system. While soundness with respect to the type assignment system is immediate, completeness requires the novel concept of *contextual type annotations*, introducing a notion from the study of principal typings into the source program.

Categories and Subject Descriptors: F.3.3 [Logics and Meanings of Programs]: Studies of Program Constructs—*Type structure*; D.3.1 [Programming Languages]: Formal Definitions and Theory

General Terms: Languages, Theory

Keywords: Type refinements, intersection types, union types, de-

compilation), it has been less successful in making the expressive type systems directly available to the programmer. One reason for this is the difficulty of finding the right balance between the brevity of the additional required type declarations and the feasibility of the typechecking problem. Another is the difficulty of giving precise and useful feedback to the programmer on ill-typed programs.

In prior work [9] we developed a system of pure type assignment designed for call-by-value languages with effects and proved progress and type preservation. The intended atomic program properties are data structure refinements [11, 10, 28], but our approach does not depend essentially on this choice. Atomic properties can be combined into more complex ones through intersections, unions, and universal and existential quantification over index domains. As a pure type assignment system, where terms do not contain any types at all, it is inherently undecidable [4].

In this paper we develop an annotation discipline and typechecking algorithm for our earlier type assignment system. The major contribution is the type system itself which contains several novel ideas, including an extension of the paradigm of bidirectional typechecking to union and existential types, leading to the *tridirectional system*. While type soundness follows immediately by erasure of annotations, completeness requires that we insert *contextual typing annotations* reminiscent of principal typings [13, 25]. Decidability

Make vs Use

Step 1: For every type,

- identify its introduction form (the language construct that "makes" / "constructs" something of that type).
- identify its elimination form (the language construct that "uses" / "consumes" something of that type).

How to make a Bool?

How to use a Bool?

How to make a Nat?

How to use a Nat?

How to make a T1 -> T2?

How to use a T1 -> T2?

How to make a (T1, T2)?

How to use a (T1, T2)?

Step 2: For every typing rule, identify it as

- an introduction rule for a type (where the intro form of the type appears, usually appear below the line)
- an elimination rule for a type (the elim form of the type appears, usually below the line)
- others

Step 3: For every intro / elim rule, identify its principal judgment

- = the judgment where the type being introduced / eliminated appears

Step 4: Assign arrows using the rules:

- principal judgment of intro = check
- principla judgment of elim = synth

Step 5: Based on assigned arrows, propogate information "naturally"

Step 6: Remaining unresolved choices = check

https://www.youtube.com/watch?v=pvgwR2aVJCo