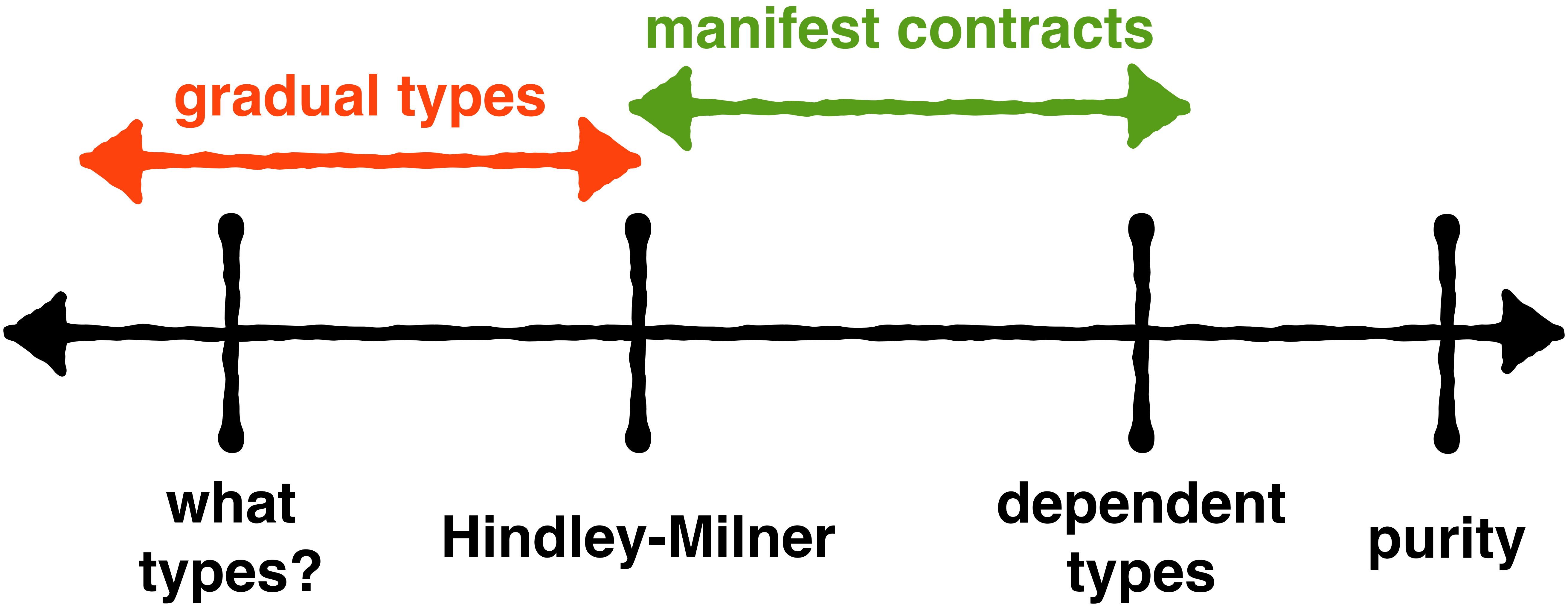
Combining **Manifest Contracts** with **State**

Michael Greenberg
Pomona College
HOPE 2015 / 2015-08-30



manifest contracts
gradual types
what
types?
Hindley-Milner
dependent
types
purity

What are contracts?

Specifications

written in **code**

checked **dynamically**

(First-order) contracts

assert(n≥0)

sqrt : {x:Float | x≥0} → Float

sqrt : {x:Float | x≥0} →
{y:Float | abs(y^2-x) ≤ ϵ}

Higher-order contracts

$$f{:}(\{x{:}\mathrm{Int} \mid x \geq 0\} \rightarrow \{x{:}\mathrm{Int} \mid x \geq 0\}) \rightarrow \{y{:}\mathrm{Int} \mid f\ y = y\}$$

You give a function *f* on Nats, I return a fixpoint of *f*

- If you don't get a fixpoint, oops—you blame me
- If *f* is called with a negative number, oops—you blame me
- If *f* returns a negative, oops—I blame you

"even-odd rule"
—*Findler and Felleisen 2002*

Subset types + dependency

checked dynamically!

$$T ::= \{x{:}B \mid e\}$$

$$\mid\ (x{:}T_1) \rightarrow T_2$$

Casts

$$<T_1 \Rightarrow T_2>^{\ell}\ e$$

I know e has type T_1

Treat it as type T_2

If I'm wrong, blame ℓ

Casts between refinements

$\langle\{x{:}Int \mid true\} \Rightarrow \{x{:}Int \mid x \geq 0\}\rangle^{\ell}\ 7 \longmapsto^{*} 7$

$\langle\{x{:}Int \mid true\} \Rightarrow \{x{:}Int \mid x \geq 0\}\rangle^{\ell}\ -1 \longmapsto^{*} \text{blame}\ \ell$

$$\langle\{x{:}B \mid e_1\} \Rightarrow \{x{:}B \mid e_2\}\rangle^{\ell}\ v$$

$$\equiv$$

$$\text{if } e_2[{}^{v}/_{x}] \text{ then } v \text{ else blame } \ell$$

Types for constants

$ty(7) = \{x{:}Int \mid x{=}7\}$
$ty(\div) = Int \to \{y{:}Int \mid y \neq 0\} \to Int$

$5 \div 0$ is ill typed!
$5 \div (\langle \ldots \Rightarrow \{y{:}Int \mid y \neq 0\}\rangle^{\ell}\ 0) \longmapsto^{*}$ blame ℓ

Casts between functions

$<T_1 \to T_2 \Rightarrow U_1 \to U_2>^{\ell} f$

...is a **value** a/k/a **function proxy**.

Casts between functions

$$(\langle T_1 \rightarrow T_2 \Rightarrow U_1 \rightarrow U_2 \rangle^{\ell}\ f)\ v \longmapsto$$

$$\langle T_2 \Rightarrow U_2 \rangle^{\ell}\ (f\ (\langle U_1 \Rightarrow T_1 \rangle^{\ell}\ v))$$

Just add state!

As seen in DTHF 2012!

Extend types…

$$
\begin{aligned}
T ::= &\ \{x{:}B \mid e\} \\
\mid &\ (x{:}T_1) \rightarrow T_2 \\
\mid &\ \mathrm{Ref}\ T
\end{aligned}
$$

Extend expressions…

$$
\begin{aligned}
e ::= \;& \ldots \\
\mid \;& \text{ref } e \\
\mid \;& !e \\
\mid \;& e_1 := e_2
\end{aligned}
$$

Extend values…

$$v ::= \ldots$$

$$\quad\mid\ \gamma$$

$$\gamma ::= \text{loc}$$

$$\quad\mid\ \langle \text{Ref}\, T_1 \Rightarrow \text{Ref}\, T_2 \rangle^{\ell}\ \gamma$$

Extend semantics (reads)…

$$!(\langle \mathrm{Ref}\, T_1 \Rightarrow \mathrm{Ref}\, T_2 \rangle^{\ell}\ \gamma)$$

$$\longmapsto$$

$$\langle T_1 \Rightarrow T_2 \rangle^{\ell}\ !\gamma$$

Extend semantics (writes)…

$$(\langle \mathrm{Ref}\ T_1 \Rightarrow \mathrm{Ref}\ T_2 \rangle^{\ell}\ \gamma) := v$$

$$\longmapsto$$

$$\gamma := \langle T_2 \Rightarrow T_1 \rangle^{\ell}\ v$$

Scoping

Recursion

Semantics

Proofs

Locations aren’t always in scope.

let nonReentrant =

$\Lambda\alpha\beta.\lambda f : (\alpha \to \beta).$ let inside = ref false in

$\lambda x{:}\{x{:}\alpha \mid \text{not !inside}\}.$

inside := true;

let y = f ($\langle\ldots\Rightarrow\alpha\rangle^{\ell}$ x) in

inside := false;

y

let nonReentrant :

$\forall\alpha\beta.\ (\alpha \to \beta) \to \{x{:}\alpha \mid \text{not !inside}\} \to \beta$ =

$\Lambda\alpha\beta.\lambda f : (\alpha \to \beta).$ let inside = ref false in

$\lambda x{:}\{x{:}\alpha \mid \text{not !inside}\}.$

inside ≔ true;

let y = f ($\langle \ldots \Rightarrow \alpha \rangle^{\ell}$ x) in

inside ≔ false;

y

scope?!

Recursion

initialization?

Ref {x:Int | x ≤ !y}

Ref {y:Int | y ≥ !x}

```
let f = ref λx:Int. x in
let g = <…⇒Ref {x:Int→Int | x 0 = 0}>ℓ f in
  g := <…⇒{x:Int→Int | x 0 = 0}>ℓ'
         (λx:Int. (<…⇒Int→Int>ℓ'' !g) x);
  !g
```

Semantics

```
let x = ref 0 in
let y = <Ref Int⇒Ref {z:Int |        z ≥ 0}>ℓ x in
  y := 5;
  !y;
✗ !y
```

$$\Gamma \vdash v : \{x{:}B \mid e\}$$

implies

$$e[{}^{v}/_{x}] \longmapsto^{*} \text{true}$$

Proofs

- Axiomatization, LR, bisimulation
- Type conversion relation

Scoping

Recursion

Semantics

Proofs

Solutions?

Scoping: contextual typing annotations?

$$\frac{(\Gamma_0 \vdash A_0) \lesssim (\Gamma \vdash A) \qquad \Gamma \vdash e \downarrow A}{\Gamma \vdash (e : (\Gamma_0 \vdash A_0), As) \uparrow A} \text{(ctx-anno)}$$

Dunfield and Pfenning 2004, “Tridirectional typechecking”
Thanks, reviewer 1!

let nonReentrant :

$$\forall \alpha \beta.\ (\alpha \rightarrow \beta) \rightarrow \{x{:}\alpha \mid \text{not !inside}\} \rightarrow \beta = \ldots$$

let nonReentrant :

$$\forall \alpha\beta.\ (\alpha \to \beta) \xrightarrow{\exists \text{inside.}} \{x{:}\alpha \mid \text{not } !\text{inside}\} \to \beta = \ldots$$

Scoping: effects

let nonReentrant :

new(inside)

$\forall \alpha \beta.\ (\alpha \to \beta) \to \{x{:}\alpha \mid \text{not !inside}\} \to \beta = \ldots$

Recursion, semantics: effects

$$\frac{\Gamma, x{:}B \vdash e : \mathrm{Bool}, \emptyset}{\Gamma \vdash \mathrm{Ref}\ \{x{:}B \mid e\}}$$

$$\frac{\Gamma, x{:}B \vdash e : \mathrm{Bool}, \xi' \qquad \xi' \prec \xi}{\Gamma \vdash \mathrm{Ref}\ \{x{:}B \mid e\} : *, \xi}$$

Information-flow control

$\langle\{x{:}B|e_1\}\Rightarrow\{x{:}B|e_2\}\rangle^{\ell}\ v, pc$

$\longmapsto$

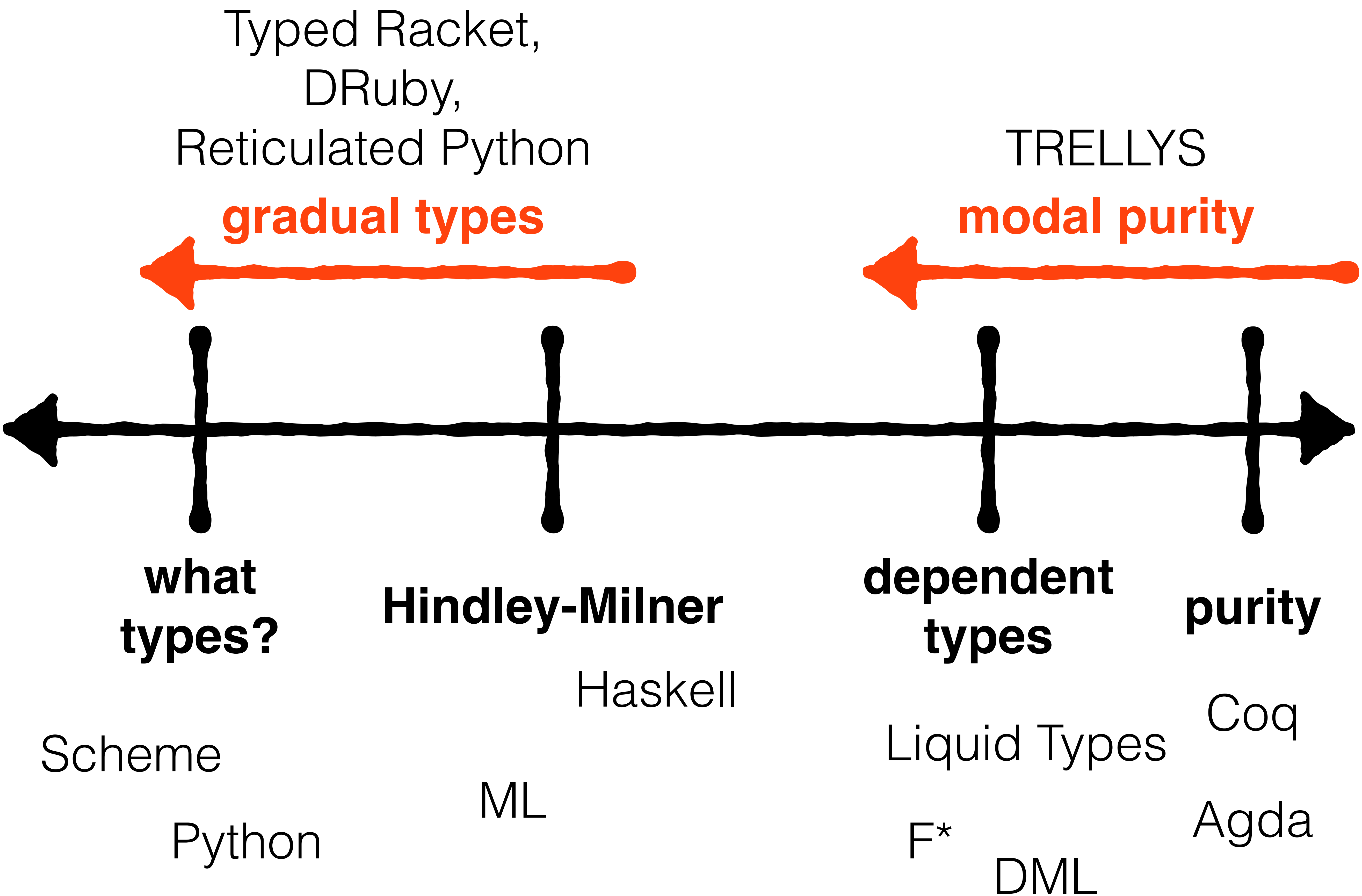
if $e_2[{}^{v}/_{x}]$ then v else blame ℓ,

$pc \sqcup$ CTC

Other ideas?

- Proofs?!
- Can we borrow from work on lock ordering? Something substructural?
- Split pure/impure contracts using a monadic framework?
- Borrow ideas from transactional memory for IO? Cf. Avi Shinnar's thesis

Appendix


Typed Racket,
DRuby,
Reticulated Python
gradual types
TRELLYS
modal purity
what
types?
Hindley-Milner
dependent
types
purity
Haskell
Coq
Liquid Types
Scheme
ML
Agda
Python
F*
DML

What are contracts for?

"Well-typed expressions do not go wrong"

—Robin Milner, "A Theory of Type Polymorphism in Programming"

What's "wrong"?

- Applying a boolean
- Conditioning on a lambda

What are contracts for?

- Contracts expand our notion of wrong
  - Division by zero, square root of negatives
  - Incomplete pattern matches
  - Array indexing

Dynamic by default

- Type refinement systems, dependent types
  static checking by default
- Manifest contracts
  dynamic checking by default
  static checking as an optimization

Stateful contracts, take 2

refine any type!

$$T ::= \{x{:}T \mid e\}$$
$$\mid\ (x{:}T_1) \rightarrow T_2$$
$$\mid\ \mathrm{Ref}\ T$$