Unifying Principles of Type Refinements

Joshua Dunfield

27 April 2007

Carnegie Mellon University

Thesis Committee:

Frank Pfenning, chair

Jonathan Aldrich

Robert Harper

Benjamin Pierce, Univ. of Pennsylvania

Outline

- ☞ **Introduction**
  - Thesis Statement
  - Contributions
  - Bi- and Tridirectional Typechecking
  - Let-Normal Typechecking
  - Implementation
  - Related Work
  - Summary
  - Future Work

Making Software More Reliable

- Many answers...
- Tools (not methods)
  - Static typing to check invariants

Conventional Static Typing

- ML, Haskell, ...
- "Well typed programs don't go wrong"
- Types express **properties**:
  - $insert$: tree $*$ key $\rightarrow$ tree
  - $map : \forall\alpha, \beta.(\alpha \rightarrow \beta) \rightarrow \alpha$ list $\rightarrow \beta$ list
- Types can be inferred;
  type annotations are documentation
  (except module interfaces)

Conventional Static Typing

- ML, Haskell, ...
- "Well typed programs don't go wrong"
- Types express **properties**:
  - $\mathit{insert} : \mathsf{tree} * \mathsf{key} \to \mathsf{tree}$
  - $\mathit{map} : \forall \alpha, \beta . (\alpha \to \beta) \to \alpha\ \mathsf{list} \to \beta\ \mathsf{list}$
- Types can be inferred;
  type annotations are documentation
  (except module interfaces)
- How to get more expressive power?
  - Extend conventional static typing

Setting

- core Standard ML (approximately):
  - First class functions
  - Primitive types int, real, string, ...
  - Algebraic datatypes

$$\textbf{datatype}\ \mathsf{list} = \mathsf{Nil} \mid \mathsf{Cons}\ \textbf{of}\ \mathsf{int} * \mathsf{list}$$

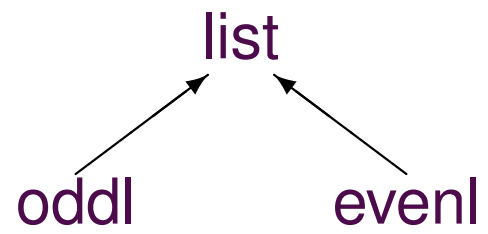
$$\textbf{datatype}\ \mathsf{tree} = \mathsf{Empty} \mid \mathsf{Tree}\ \textbf{of}\ \mathsf{key} * \mathsf{tree} * \mathsf{tree}$$

Datasort Refinements

- a.k.a. refinement types [Freeman+Pf. '91, Davies '97,'05]
- Refine an algebraic datatype by a datasort δ
- Example: Lists of integers

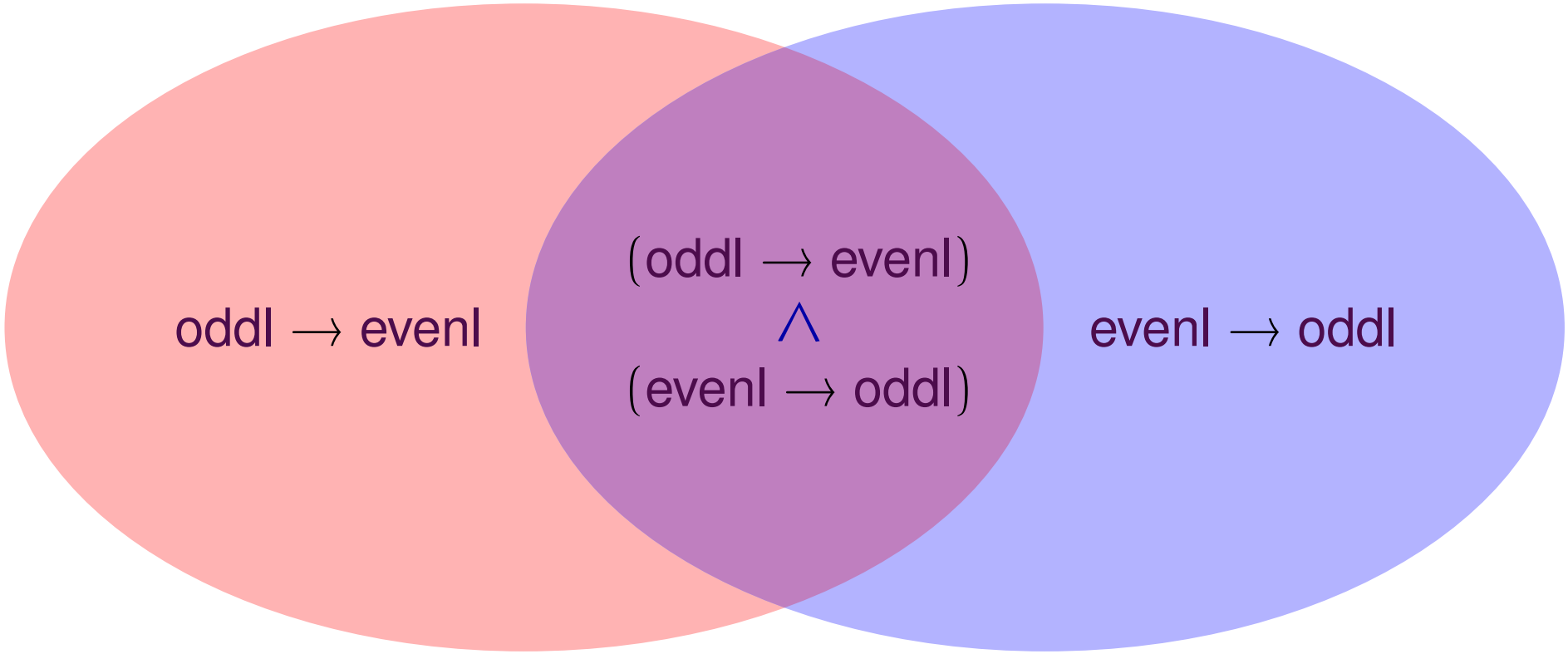
$$\begin{array}{ll} \mathsf{Nil} : \mathsf{list} & \mathsf{Nil} : \mathsf{evenl} \\ \mathsf{Cons} : \mathsf{int} * \mathsf{list} \to \mathsf{list} & \mathsf{Cons} : (\mathsf{int} * \mathsf{oddl} \to \mathsf{evenl}) \\ & \quad \wedge (\mathsf{int} * \mathsf{evenl} \to \mathsf{oddl}) \\ & \quad \wedge (\mathsf{int} * \mathsf{list} \to \mathsf{list}) \end{array}$$

δ :



- Intersections $v : A \wedge B$: v has type A and type B

Intersection Types ~ Set Intersection



If

$tail : (\mathsf{oddl} \to \mathsf{evenl}) \wedge (\mathsf{evenl} \to \mathsf{oddl})$

$\mathrm{od} : \mathsf{oddl}, \qquad ev : \mathsf{evenl}$

then

$tail(\mathrm{od}) : \mathsf{evenl}$ and $tail(ev) : \mathsf{oddl}$

Datasort Refinements

$tail \ : \ (\mathsf{evenl} \rightarrow \mathsf{oddl}) \wedge (\mathsf{oddl} \rightarrow \mathsf{evenl}) \wedge (\mathsf{list} \rightarrow \mathsf{list})$

fun *tail* xs =

  case xs **of** Nil $\Rightarrow$ **raise** *Error*

  | Cons(h, t) $\Rightarrow$ t

- First conjunct ($\mathsf{evenl} \rightarrow \mathsf{oddl}$):
  - Assume xs : evenl
  - Show **raise** *Error* : oddl and t : oddl
- Second conjunct ($\mathsf{oddl} \rightarrow \mathsf{evenl}$):
  - Assume xs : oddl
  - Show t : evenl

Index Refinements

- Dependent types restricted to a decidable constraint domain [Xi & Pfenning '99]
- Refine an algebraic datatype by an index
- Indices drawn from any decidable constraint domain
- Example: Lists indexed by their length

$$\mathsf{Nil} : \mathsf{list} \qquad\qquad \mathsf{Nil} : \mathsf{list}(0)$$

$$\mathsf{Cons} : \mathsf{int} * \mathsf{list} \to \mathsf{list} \qquad \mathsf{Cons} : \Pi a{:}\mathcal{N}.\, \mathsf{int} * \mathsf{list}(a) \to \mathsf{list}(a+1)$$

$$\mathit{append} : \Pi a{:}\mathcal{N}.\, \Pi b{:}\mathcal{N}.\, \mathsf{list}(a) * \mathsf{list}(b) \to \mathsf{list}(a+b)$$

$$\mathit{filter} : (\mathsf{int} \to \mathsf{bool}) \to \Pi a{:}\mathcal{N}.\, \mathsf{list}(a) \to (\Sigma b{:}\mathcal{N}.\, (b \leq a) \otimes \mathsf{list}(b))$$

- Universal quantifier Π, existential quantifier Σ

Index Refinements + Union Types

- Unions $v : A \vee B$: v has type A or type B
- Not tagged!
- Choice function:

  $$\mathit{choose} \;:\; \Pi a{:}\mathcal{N}.\, \Pi b{:}\mathcal{N}.\, \mathsf{list}(a) * \mathsf{list}(b) \to (\mathsf{list}(a) \vee \mathsf{list}(b))$$

  fun *choose* $(xs, ys) =$

  case *random_bit* () **of** true $\Rightarrow xs$

  | false $\Rightarrow ys$

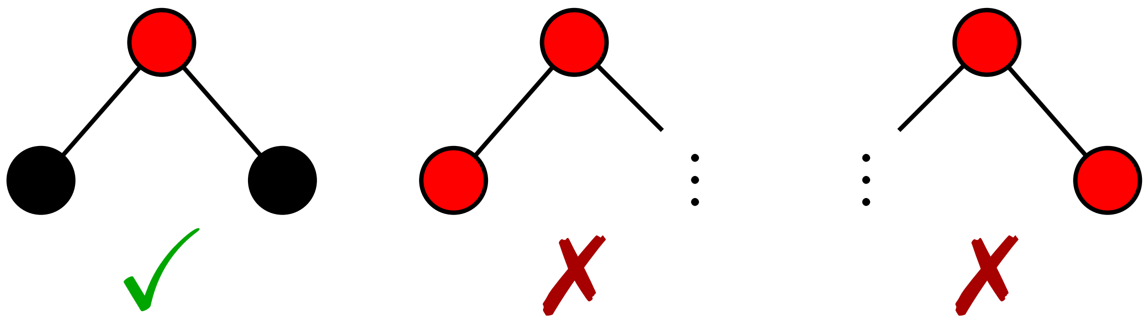
- Assume xs:list(a), ys:list(b)
- Show body checks against list$(a) \vee$ list(b):
  - Show $xs : \mathsf{list}(a) \vee \mathsf{list}(b) \impliedby xs : \mathsf{list}(a)$ ✓
  - Show $ys : \mathsf{list}(a) \vee \mathsf{list}(b) \impliedby ys : \mathsf{list}(a)$ ✗

    $ys : \mathsf{list}(b)$ ✓

Red-black trees

datatype tree = Empty
| Red **of** key ∗ tree ∗ tree
| Black **of** key ∗ tree ∗ tree

- Invariants:
  (1) Empty nodes are considered black
  (2) The children of a red node must be black



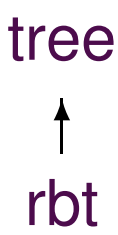
  (3) For every node t, there exists $bh(t)$ s.t. the number of black nodes on **all** paths from t to its leaves is $bh(t)$

Use datasort for invariants (1), (2)

tree

| Empty : tree
| Red : key $*$ tree $*$ tree $\longrightarrow$ tree
| Black : key $*$ tree $*$ tree $\longrightarrow$ tree

Use datasort for invariants (1), (2)



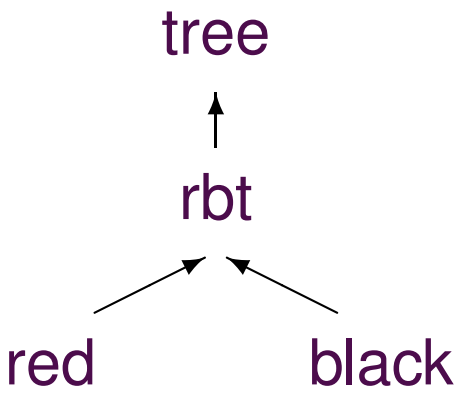
Empty : tree

Red : key $*$ tree $*$ tree $\longrightarrow$ tree

Black : key $*$ tree $*$ tree $\longrightarrow$ tree

- Add rbt to represent valid red-black trees

Use datasort for invariants (1), (2)



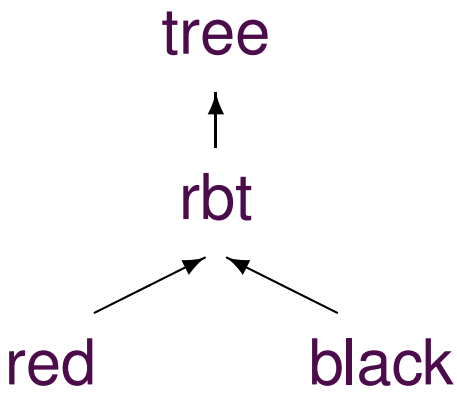
Empty : tree

Red : key $*$ tree $*$ tree $\longrightarrow$ tree

Black : key $*$ tree $*$ tree $\longrightarrow$ tree

- Add rbt to represent valid red-black trees
- Add red, black to distinguish colors

Use datasort for invariants (1), (2)



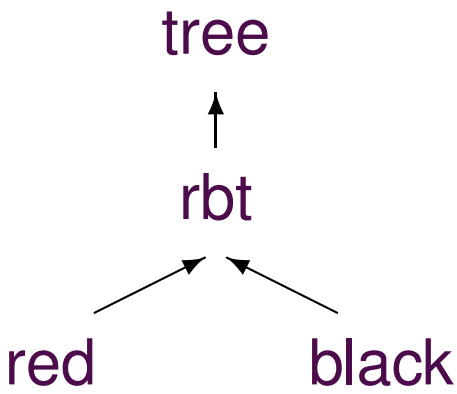
Empty : tree $\wedge$ black

Red : key $*$ tree $*$ tree $\longrightarrow$ tree

Black : key $*$ tree $*$ tree $\longrightarrow$ tree

- Add rbt to represent valid red-black trees
- Add red, black to distinguish colors
- Invariant (1): Empty nodes are considered black

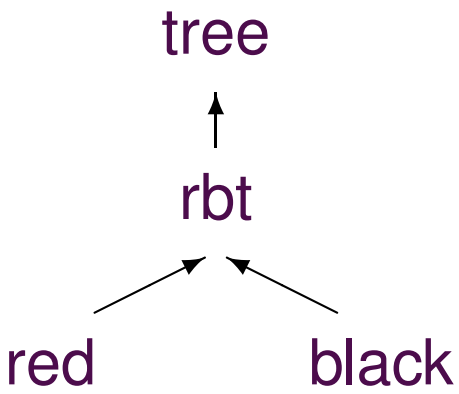
Use datasort for invariants (1), (2)



$$\begin{aligned} \text{Empty} &: \text{tree} \wedge \text{black} \\ \text{Red} &: \text{key} * \text{tree} * \text{tree} \longrightarrow \text{tree} \\ &\quad \wedge \text{key} * \text{black} * \text{black} \longrightarrow \text{red} \\ \text{Black} &: \text{key} * \text{tree} * \text{tree} \longrightarrow \text{tree} \\ &\quad \wedge \text{key} * \text{rbt} * \text{rbt} \longrightarrow \text{black} \end{aligned}$$

- Add rbt to represent valid red-black trees
- Add red, black to distinguish colors
- Invariant (1): Empty nodes are considered black
- Invariant (2): The children of a red node must be black

Use index for invariant (3)



Empty : black

Red :

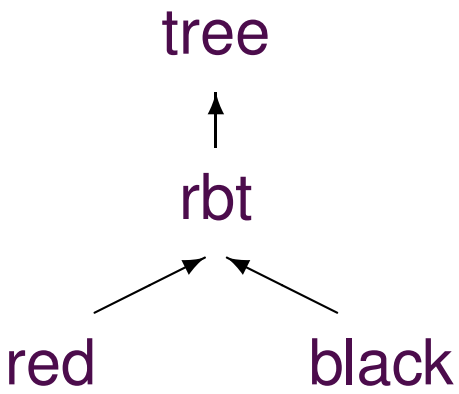
$$\text{key} * \text{tree} \quad * \text{tree} \quad \longrightarrow \text{tree}$$
$$\wedge\ \text{key} * \text{black} \quad * \text{black} \quad \longrightarrow \text{red}$$

Black :

$$\text{key} * \text{tree} \quad * \text{tree} \quad \longrightarrow \text{tree}$$
$$\wedge\ \text{key} * \text{rbt} \quad * \text{rbt} \quad \longrightarrow \text{black}$$

- Invairant (3): For every node t, there exists $bh(\mathrm{t})$ s.t. the number of black nodes on **all** paths from t to its leaves is $bh(\mathrm{t})$

Use index for invariant (3)



Empty : black

Red :

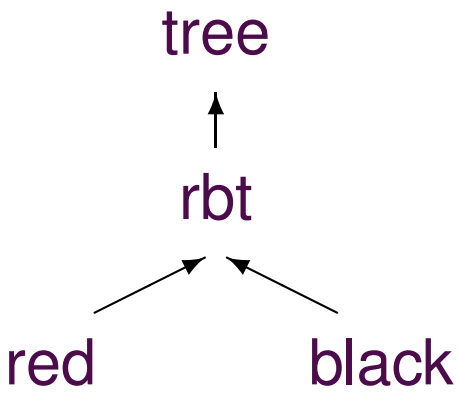
$$\text{key} * \text{tree} \quad * \text{tree} \quad \longrightarrow \text{tree}$$
$$\wedge\ \text{key} * \text{black} \quad * \text{black} \quad \longrightarrow \text{red}$$

Black :

$$\text{key} * \text{tree} \quad * \text{tree} \quad \longrightarrow \text{tree}$$
$$\wedge\ \text{key} * \text{rbt} \quad * \text{rbt} \quad \longrightarrow \text{black}$$

- Invairant (3): For every node t, there exists $bh(\mathrm{t})$ s.t. the number of black nodes on **all** paths from t to its leaves is $bh(\mathrm{t})$
- Index by the black height $bh(\mathrm{t})$, a natural number

Use index for invariant (3)



$$\begin{aligned}
&\text{Empty} \;:\; \mathsf{black}(0)\\
&\text{Red} : \Pi \mathtt{h}{:}\mathcal{N}.\\
&\qquad \mathsf{key} * \mathsf{tree}(\mathtt{h}) * \mathsf{tree}(\mathtt{h}) \rightarrow \mathsf{tree}(\mathtt{h})\\
&\qquad \wedge\, \mathsf{key} * \mathsf{black}(\mathtt{h}) * \mathsf{black}(\mathtt{h}) \rightarrow \mathsf{red}(\mathtt{h})\\
&\text{Black} : \Pi \mathtt{h}{:}\mathcal{N}.\\
&\qquad \mathsf{key} * \mathsf{tree}(\mathtt{h}) * \mathsf{tree}(\mathtt{h}) \rightarrow \mathsf{tree}(\mathtt{h}+1)\\
&\qquad \wedge\, \mathsf{key} * \mathsf{rbt}(\mathtt{h}) * \mathsf{rbt}(\mathtt{h}) \rightarrow \mathsf{black}(\mathtt{h}+1)
\end{aligned}$$

- Invairant (3): For every node t, there exists $bh(\mathtt{t})$ s.t. the number of black nodes on **all** paths from t to its leaves is $bh(\mathtt{t})$
- Index by the black height $bh(\mathtt{t})$, a natural number

- ✓ Introduction
- ☞ **Thesis Statement**
- Contributions
- Bi- and Tridirectional Typechecking
- Let-Normal Typechecking
- Implementation
- Related Work
- Summary
- Future Work

Thesis Statement

A rich type system with datasort and index refinement properties, where such properties are combined through intersection and union types, is a practical means of statically checking interesting properties of functional programs that are difficult or impossible to check in conventional static type systems.

- ✓ Introduction
- ✓ Thesis Statement
- ☞ **Contributions**
  - Bi- and Tridirectional Typechecking
  - Let-Normal Typechecking
  - Implementation
  - Related Work
  - Summary
  - Future Work

Contributions

- Rich yet practical typechecking:

Ch. 2 Type assignment system

Ch. 3 Tridirectional type system

Ch. 4 Full patterns

Ch. 5 Let-normal type system

Ch. 6, 7 Implementation

  - integers
  - dimensions

- ✓ Introduction
- ✓ Thesis Statement
- ✓ Contributions
- ☞ **Bi- and Tridirectional Typechecking**
  - Let-Normal Typechecking
  - Implementation
  - Related Work
  - Summary
  - Future Work

Language

$$
\begin{array}{rrcl}
\text{Types} & A, B, C & ::= & \mathsf{unit} \mid A \to B \mid A * B \mid \delta(i) \\
\text{definite} & & & \mid A \wedge B \mid \Pi a{:}\gamma.\, A \mid P \supset A \mid \top \\
\text{indefinite} & & & \mid A \vee B \mid \Sigma a{:}\gamma.\, A \mid P \parr A \mid \bot \\
\\
\text{Terms} & e & ::= & x \mid u \mid () \mid \lambda x.\, e \mid e_1(e_2) \mid \mathbf{fix}\, u.\, e \\
& & & \mid (e_1, e_2) \mid \mathbf{fst}(e) \mid \mathbf{snd}(e) \\
& & & \mid c(e) \mid \mathbf{case}\ e\ \mathbf{of}\ ms
\end{array}
$$

cbv Semantics

$$\begin{array}{rl} \text{Values} & v ::= x \mid () \mid \lambda x.\, e \mid (v_1, v_2) \mid c(v) \\ \text{Evaluation contexts} & \mathcal{E} ::= [\,] \mid \mathcal{E}(e) \mid v(\mathcal{E}) \\ & \quad \mid (\mathcal{E}, e) \mid (v, \mathcal{E}) \mid \mathbf{fst}(\mathcal{E}) \mid \mathbf{snd}(\mathcal{E}) \\ & \quad \mid c(\mathcal{E}) \mid \mathbf{case}\ \mathcal{E}\ \mathbf{of}\ ms \end{array}$$

$$\frac{e' \ \mapsto_R \ e''}{\mathcal{E}[e'] \ \mapsto \ \mathcal{E}[e'']}$$

$$\begin{array}{rlrl} (\lambda x.\, e)\, v & \mapsto_R [v/x]\, e & \mathbf{fst}(v_1, v_2) & \mapsto_R v_1 \\ \mathbf{fix}\, u.\, e & \mapsto_R [\mathbf{fix}\, u.\, e \,/\, u]\, e & \mathbf{snd}(v_1, v_2) & \mapsto_R v_2 \\ \multicolumn{4}{c}{\mathbf{case}\ c(v)\ \mathbf{of}\ \ldots c(x) \Rightarrow e \ldots \ \mapsto_R \ [v/x]\, e} \end{array}$$

Why Bidirectional Typing?

- Rich language of property types: $\wedge, \Pi, \supset, \top, \vee, \Sigma, \otimes, \bot$
- Pure type assignment: undecidable ($\wedge, \ldots$)
- Bidirectional typing (some annotations): **decidable**

Bidirectional Typing

- Bidirectional typing

$$\text{Synthesis} \quad \Gamma^{+} \vdash e^{+} \uparrow A^{-}$$

$$\text{Checking} \quad \Gamma^{+} \vdash e^{+} \downarrow A^{+}$$

$+$ input

$-$ output

Bidirectional Typing

- Getting started:

$$\frac{\Gamma(x) = A}{\Gamma \vdash x \,:\, A}\ \mathsf{var}$$

Bidirectional Typing

- Getting started:

$$\frac{\Gamma(x) = A}{\Gamma \vdash x \uparrow A}\ \mathsf{var}$$

Bidirectional Typing: $\rightarrow$

- **Principle:**
  Introduction rules **check**,
  elimination rules **synthesize**

$$\frac{\Gamma, x{:}A \vdash e \ : \ B}{\Gamma \vdash \lambda x.\, e \ : \ A \rightarrow B} \rightarrow\!\mathrm{I}$$

$$\frac{\Gamma \vdash e_1 \ : \ A \rightarrow B \qquad \Gamma \vdash e_2 \ : \ A}{\Gamma \vdash e_1 e_2 \ : \ B} \rightarrow\!\mathrm{E}$$

- e_1 usually a variable or another application

Bidirectional Typing: $\rightarrow$

- **Principle:**
  Introduction rules **check**,
  elimination rules **synthesize**

$$\frac{\Gamma, x{:}A \vdash e \ : \ B}{\Gamma \vdash \lambda x.\, e \downarrow A \rightarrow B} \rightarrow\!\mathrm{I}$$

$$\frac{\Gamma \vdash e_1 \ : \ A \rightarrow B \qquad \Gamma \vdash e_2 \ : \ A}{\Gamma \vdash e_1 e_2 \ : \ B} \rightarrow\!\mathrm{E}$$

- e_1 usually a variable or another application

Bidirectional Typing: $\rightarrow$

- **Principle:**
  Introduction rules **check**,
  elimination rules **synthesize**

$$\frac{\Gamma, x{:}A \vdash e \downarrow B}{\Gamma \vdash \lambda x.\, e \downarrow A \rightarrow B} \rightarrow\!\mathrm{I}$$

$$\frac{\Gamma \vdash e_1 \;:\; A \rightarrow B \qquad \Gamma \vdash e_2 \;:\; A}{\Gamma \vdash e_1 e_2 \;:\; B} \rightarrow\!\mathrm{E}$$

- e_1 usually a variable or another application

Bidirectional Typing: $\rightarrow$

- **Principle:**
  Introduction rules **check**,
  elimination rules **synthesize**

$$\frac{\Gamma, x{:}A \vdash e \downarrow B}{\Gamma \vdash \lambda x.\, e \downarrow A \rightarrow B} \rightarrow\!\mathrm{I}$$

$$\frac{\Gamma \vdash e_1 \uparrow A \rightarrow B \qquad \Gamma \vdash e_2 \,:\, A}{\Gamma \vdash e_1 e_2 \,:\, B} \rightarrow\!\mathrm{E}$$

- e_1 usually a variable or another application

Bidirectional Typing: $\rightarrow$

- **Principle:**
  Introduction rules **check**,
  elimination rules **synthesize**

$$\frac{\Gamma, x{:}A \vdash e \downarrow B}{\Gamma \vdash \lambda x.\, e \downarrow A \rightarrow B} \rightarrow\!\mathrm{I}$$

$$\frac{\Gamma \vdash e_1 \uparrow A \rightarrow B \qquad \Gamma \vdash e_2 \downarrow A}{\Gamma \vdash e_1 e_2 \;:\; B} \rightarrow\!\mathrm{E}$$

- e_1 usually a variable or another application

Bidirectional Typing: $\to$

- **Principle:**
  Introduction rules **check**,
  elimination rules **synthesize**

$$\frac{\Gamma, x{:}A \vdash e \downarrow B}{\Gamma \vdash \lambda x.\, e \downarrow A \to B} \to\!\mathrm{I}$$

$$\frac{\Gamma \vdash e_1 \uparrow A \to B \qquad \Gamma \vdash e_2 \downarrow A}{\Gamma \vdash e_1\, e_2 \uparrow B} \to\!\mathrm{E}$$

- e_1 usually a variable or another application

Change of Direction

- For syntax-directed rules, intro checks, elim synthesizes
- Other rules:
  - Subsumption

$$\frac{\Gamma \vdash e \uparrow A' \qquad \Gamma \vdash A' \leq A}{\Gamma \vdash e \downarrow A}\ \mathsf{sub}$$

    - Subtyping

$$\frac{}{\mathsf{unit} \leq \mathsf{unit}} \qquad \frac{B_1 \leq A_1 \qquad A_2 \leq B_2}{A_1 \to A_2 \leq B_1 \to B_2}$$

  - Type annotation

$$\frac{\Gamma \vdash e \downarrow A}{\Gamma \vdash (e : A) \uparrow A}\ \mathsf{anno}$$

Property Types

- $A \to B$, $A * B$, ... realized by specific syntactic forms:
  - $A \to B$ introduced by $\lambda x.\, e$, eliminated by $e_1 e_2$
  - $A * B$ introduced by (e_1, e_2), eliminated by **fst** and **snd**
  - $\delta(i)$ introduced by $c(e)$, eliminated by **case** e **of** ms
- **Property types**
  $A \wedge B$ (intersection), $A \vee B$ (union), ...
  not realized by particular syntactic forms

Intersection Types

- Typing:

$$\frac{\Gamma \vdash e : A_1 \wedge A_2}{\Gamma \vdash e : A_1} \wedge E_1 \qquad \frac{\Gamma \vdash e : A_1 \wedge A_2}{\Gamma \vdash e : A_2} \wedge E_2$$

$$\frac{\Gamma \vdash v : A_1 \qquad \Gamma \vdash v : A_2}{\Gamma \vdash v : A_1 \wedge A_2} \wedge I$$

- Value restriction in $\wedge$I due to Davies & Pf. '00 (mutable references)
- Again: introduction rules check,
  elimination rules synthesize

Intersection Types

- Typing:

$$\frac{\Gamma \vdash e \uparrow A_1 \wedge A_2}{\Gamma \vdash e : A_1} \wedge E_1 \qquad \frac{\Gamma \vdash e : A_1 \wedge A_2}{\Gamma \vdash e : A_2} \wedge E_2$$

$$\frac{\Gamma \vdash v : A_1 \qquad \Gamma \vdash v : A_2}{\Gamma \vdash v : A_1 \wedge A_2} \wedge I$$

- Value restriction in $\wedge$I due to Davies & Pf. '00 (mutable references)
- Again: introduction rules check,
  elimination rules synthesize

Intersection Types

- Typing:

$$\frac{\Gamma \vdash e \uparrow A_1 \wedge A_2}{\Gamma \vdash e \uparrow A_1}\ \wedge E_1 \qquad \frac{\Gamma \vdash e : A_1 \wedge A_2}{\Gamma \vdash e : A_2}\ \wedge E_2$$

$$\frac{\Gamma \vdash v : A_1 \qquad \Gamma \vdash v : A_2}{\Gamma \vdash v : A_1 \wedge A_2}\ \wedge I$$

- Value restriction in $\wedge$I due to Davies & Pf. '00 (mutable references)
- Again: introduction rules check,
  elimination rules synthesize

Intersection Types

- Typing:

$$\frac{\Gamma \vdash e \uparrow A_1 \wedge A_2}{\Gamma \vdash e \uparrow A_1} \wedge E_1 \qquad \frac{\Gamma \vdash e \uparrow A_1 \wedge A_2}{\Gamma \vdash e : A_2} \wedge E_2$$

$$\frac{\Gamma \vdash v : A_1 \qquad \Gamma \vdash v : A_2}{\Gamma \vdash v : A_1 \wedge A_2} \wedge I$$

- Value restriction in $\wedge$I due to Davies & Pf. '00 (mutable references)
- Again: introduction rules check,
  elimination rules synthesize

Intersection Types

- Typing:

$$\frac{\Gamma \vdash e \uparrow A_1 \wedge A_2}{\Gamma \vdash e \uparrow A_1} \wedge E_1 \qquad \frac{\Gamma \vdash e \uparrow A_1 \wedge A_2}{\Gamma \vdash e \uparrow A_2} \wedge E_2$$

$$\frac{\Gamma \vdash v : A_1 \qquad \Gamma \vdash v : A_2}{\Gamma \vdash v : A_1 \wedge A_2} \wedge I$$

- Value restriction in $\wedge$I due to Davies & Pf. '00 (mutable references)
- Again: introduction rules check,
  elimination rules synthesize

Intersection Types

- Typing:

$$\frac{\Gamma \vdash e \uparrow A_1 \wedge A_2}{\Gamma \vdash e \uparrow A_1} \wedge E_1 \qquad \frac{\Gamma \vdash e \uparrow A_1 \wedge A_2}{\Gamma \vdash e \uparrow A_2} \wedge E_2$$

$$\frac{\Gamma \vdash v : A_1 \qquad \Gamma \vdash v : A_2}{\Gamma \vdash v \downarrow A_1 \wedge A_2} \wedge I$$

- Value restriction in $\wedge$I due to Davies & Pf. '00 (mutable references)
- Again: introduction rules check,
  elimination rules synthesize

Intersection Types

- Typing:

$$\frac{\Gamma \vdash e \uparrow A_1 \wedge A_2}{\Gamma \vdash e \uparrow A_1}\,\wedge E_1 \qquad \frac{\Gamma \vdash e \uparrow A_1 \wedge A_2}{\Gamma \vdash e \uparrow A_2}\,\wedge E_2$$

$$\frac{\Gamma \vdash v \downarrow A_1 \qquad \Gamma \vdash v : A_2}{\Gamma \vdash v \downarrow A_1 \wedge A_2}\,\wedge I$$

- Value restriction in $\wedge$I due to Davies & Pf. '00 (mutable references)
- Again: introduction rules check,
  elimination rules synthesize

Intersection Types

- Typing:

$$\frac{\Gamma \vdash e \uparrow A_1 \wedge A_2}{\Gamma \vdash e \uparrow A_1}\,\wedge E_1 \qquad \frac{\Gamma \vdash e \uparrow A_1 \wedge A_2}{\Gamma \vdash e \uparrow A_2}\,\wedge E_2$$

$$\frac{\Gamma \vdash v \downarrow A_1 \qquad \Gamma \vdash v \downarrow A_2}{\Gamma \vdash v \downarrow A_1 \wedge A_2}\,\wedge I$$

- Value restriction in $\wedge$I due to Davies & Pf. '00 (mutable references)
- Again: introduction rules check,
  elimination rules synthesize

Intersection Types

- Subtyping: $$\frac{A \leq B_1 \qquad A \leq B_2}{A \leq B_1 \wedge B_2}\ \wedge R$$

$$\frac{A_1 \leq B}{A_1 \wedge A_2 \leq B}\ \wedge L_1 \qquad \frac{A_2 \leq B}{A_1 \wedge A_2 \leq B}\ \wedge L_2$$

- Distributivity?

$$\frac{}{(A \to B) \wedge (A \to B') \leq A \to (B \wedge B')}$$

  - Unsound with mutable references [Davies & Pf. '00]

Union Types $A \vee B$

- Subtyping dual to $\wedge$
- Introduction rules

$$\frac{\Gamma \vdash e \ : \ A}{\Gamma \vdash e \ : \ A \vee B} \vee I_1 \qquad \frac{\Gamma \vdash e \ : \ B}{\Gamma \vdash e \ : \ A \vee B} \vee I_2$$

- Elimination rule: **contextual rule** $\mathcal{E}[e']$

$$\frac{\Gamma \vdash e' \ : \ A \vee B \qquad \begin{array}{c} \Gamma, x{:}A \vdash \mathcal{E}[x] \ : \ C \\ \Gamma, x{:}B \vdash \mathcal{E}[x] \ : \ C \end{array}}{\Gamma \vdash \mathcal{E}[e'] \ : \ C} \vee E$$

- Several alternative elim rules are unsound (Ch. 2)

Union Types $A \vee B$

- Subtyping dual to $\wedge$
- Introduction rules

$$\frac{\Gamma \vdash e \,:\, A}{\Gamma \vdash e \downarrow A \vee B}\ \vee I_1 \qquad \frac{\Gamma \vdash e \,:\, B}{\Gamma \vdash e \,:\, A \vee B}\ \vee I_2$$

- Elimination rule: **contextual rule** $\mathcal{E}[e']$

$$\frac{\Gamma \vdash e' \,:\, A \vee B \qquad \begin{array}{c}\Gamma, x{:}A \vdash \mathcal{E}[x] \,:\, C \\ \Gamma, x{:}B \vdash \mathcal{E}[x] \,:\, C\end{array}}{\Gamma \vdash \mathcal{E}[e'] \,:\, C}\ \vee E$$

- Several alternative elim rules are unsound (Ch. 2)

Union Types $A \vee B$

- Subtyping dual to $\wedge$
- Introduction rules

$$\frac{\Gamma \vdash e \downarrow A}{\Gamma \vdash e \downarrow A \vee B}\ \vee I_1 \qquad \frac{\Gamma \vdash e : B}{\Gamma \vdash e : A \vee B}\ \vee I_2$$

- Elimination rule: **contextual rule** $\mathcal{E}[e']$

$$\frac{\Gamma \vdash e' : A \vee B \qquad \begin{array}{c}\Gamma, x{:}A \vdash \mathcal{E}[x] : C \\ \Gamma, x{:}B \vdash \mathcal{E}[x] : C\end{array}}{\Gamma \vdash \mathcal{E}[e'] : C}\ \vee E$$

- Several alternative elim rules are unsound (Ch. 2)

Union Types $A \vee B$

- Subtyping dual to $\wedge$
- Introduction rules

$$\frac{\Gamma \vdash e \downarrow A}{\Gamma \vdash e \downarrow A \vee B}\ \vee I_1 \qquad \frac{\Gamma \vdash e \downarrow B}{\Gamma \vdash e \downarrow A \vee B}\ \vee I_2$$

- Elimination rule: **contextual rule** $\mathcal{E}[e']$

$$\frac{\Gamma \vdash e' : A \vee B \qquad \begin{array}{c}\Gamma, x{:}A \vdash \mathcal{E}[x] : C \\ \Gamma, x{:}B \vdash \mathcal{E}[x] : C\end{array}}{\Gamma \vdash \mathcal{E}[e'] : C}\ \vee E$$

- Several alternative elim rules are unsound (Ch. 2)

Union Types $A \vee B$

- Subtyping dual to $\wedge$
- Introduction rules

$$\frac{\Gamma \vdash e \downarrow A}{\Gamma \vdash e \downarrow A \vee B}\ \vee I_1 \qquad \frac{\Gamma \vdash e \downarrow B}{\Gamma \vdash e \downarrow A \vee B}\ \vee I_2$$

- Elimination rule: **contextual rule** $\mathcal{E}[e']$

$$\frac{\Gamma \vdash e' \uparrow A \vee B \qquad \begin{array}{c}\Gamma, x{:}A \vdash \mathcal{E}[x] \ : \ C \\ \Gamma, x{:}B \vdash \mathcal{E}[x] \ : \ C\end{array}}{\Gamma \vdash \mathcal{E}[e'] \ : \ C}\ \vee E$$

- Several alternative elim rules are unsound (Ch. 2)

Union Types $A \vee B$

- Subtyping dual to $\wedge$
- Introduction rules

$$\frac{\Gamma \vdash e \downarrow A}{\Gamma \vdash e \downarrow A \vee B}\ \vee I_1 \qquad \frac{\Gamma \vdash e \downarrow B}{\Gamma \vdash e \downarrow A \vee B}\ \vee I_2$$

- Elimination rule: **contextual rule** $\mathcal{E}[e']$

$$\frac{\Gamma \vdash e' \uparrow A \vee B \qquad \begin{array}{c}\Gamma, x{:}A \vdash \mathcal{E}[x] \downarrow C \\ \Gamma, x{:}B \vdash \mathcal{E}[x] \downarrow C\end{array}}{\Gamma \vdash \mathcal{E}[e'] \ : \ C}\ \vee E$$

- Several alternative elim rules are unsound (Ch. 2)

Union Types $A \vee B$

- Subtyping dual to $\wedge$
- Introduction rules

$$\frac{\Gamma \vdash e \downarrow A}{\Gamma \vdash e \downarrow A \vee B}\ \vee I_1 \qquad \frac{\Gamma \vdash e \downarrow B}{\Gamma \vdash e \downarrow A \vee B}\ \vee I_2$$

- Elimination rule: **contextual rule** $\mathcal{E}[e']$

$$\frac{\Gamma \vdash e' \uparrow A \vee B \qquad \begin{array}{c}\Gamma, x{:}A \vdash \mathcal{E}[x] \downarrow C \\ \Gamma, x{:}B \vdash \mathcal{E}[x] \downarrow C\end{array}}{\Gamma \vdash \mathcal{E}[e'] \downarrow C}\ \vee E$$

- Several alternative elim rules are unsound (Ch. 2)

More Contextual Rules

- Existential index quantifier $\Sigma a{:}\gamma.\, A$

$$\frac{\Gamma \vdash e' \uparrow \Sigma a{:}\gamma.\, A \qquad \Gamma, a{:}\gamma, x{:}A \vdash \mathcal{E}[x] \downarrow C}{\Gamma \vdash \mathcal{E}[e'] \downarrow C}\ \Sigma E$$

- Empty type $\bot$

$$\frac{\Gamma \vdash e' \uparrow \bot}{\Gamma \vdash \mathcal{E}[e'] \downarrow C}\ \bot E$$

- Common structure of $\vee E$, ΣE, $\bot E$

$$\frac{\Gamma \vdash e' \uparrow \text{indefinite type} \qquad \ldots n \text{ premises} \ldots}{\Gamma \vdash \mathcal{E}[e'] \downarrow C}$$

The Tridirectional Rule

- Common structure of $\vee$E, ΣE, $\bot$E

$$\frac{\Gamma \vdash e' \uparrow \text{indefinite type} \qquad \ldots n \text{ premises} \ldots}{\Gamma \vdash \mathcal{E}[e'] \downarrow C}$$

- Tridirectional rule

$$\frac{\Gamma \vdash e' \uparrow A \qquad \Gamma, x{:}A \vdash \mathcal{E}[x] \downarrow C}{\Gamma \vdash \mathcal{E}[e'] \downarrow C}\ \text{direct}$$

  - Not admissible due to evaluation context restriction

Contextual Typing Annotations

$$(e : (\Gamma_1 \vdash A_1, \ldots, \Gamma_n \vdash A_n))$$

- With $\wedge$ and $\sqcap$,
  the form $(e : A)$ is insufficient
- List of typings $\Gamma_k \vdash A_k$
- Choose typing $\Gamma_k \vdash A_k$ if Γ supports Γ_k
- Safe under α-conversion
- Example:

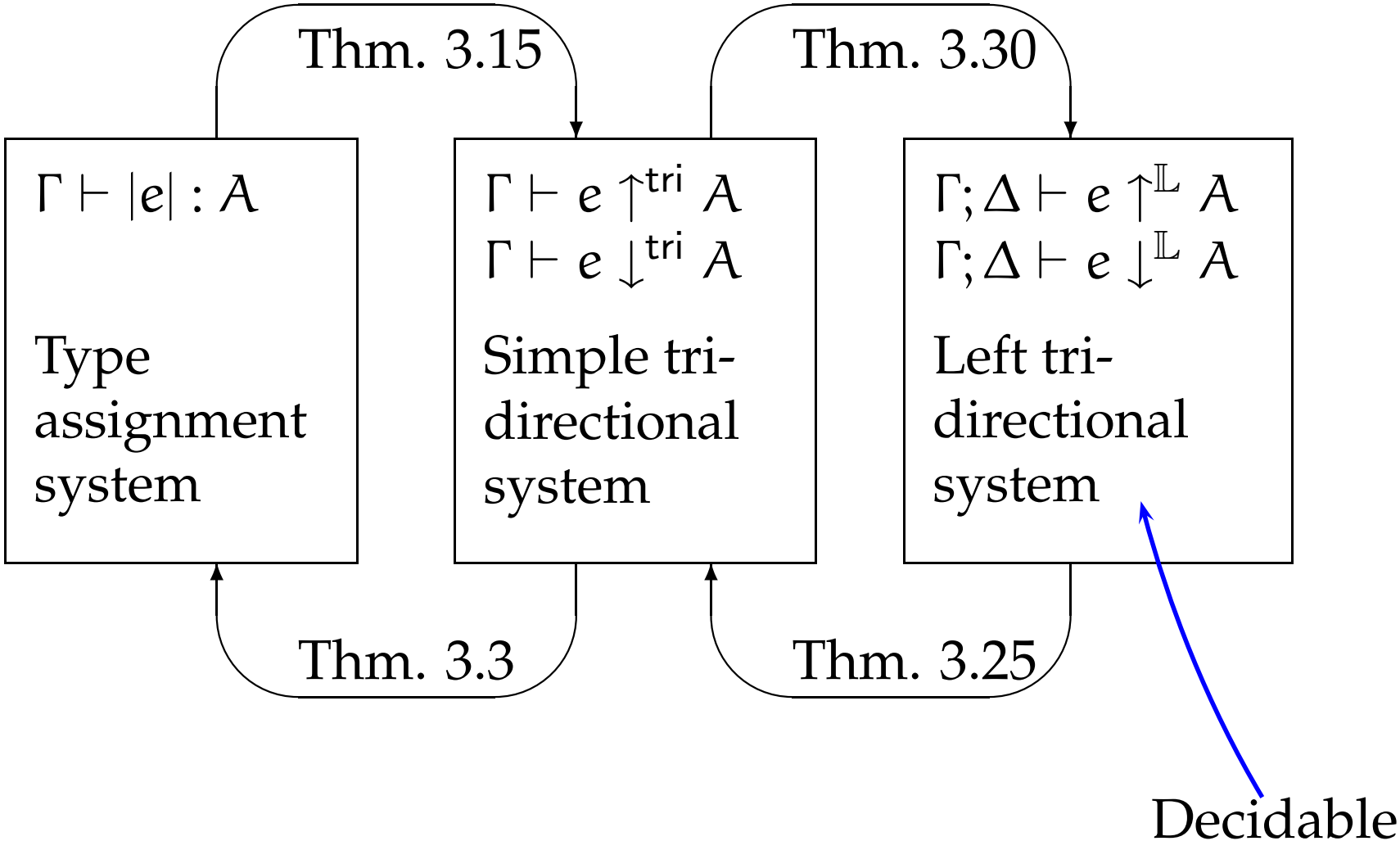
$$((\lambda y.\, \mathsf{Cons}(42, x)) : x{:}\mathsf{evenl} \vdash \mathsf{unit} \to \mathsf{oddl},\ x{:}\mathsf{oddl} \vdash \mathsf{unit} \to \mathsf{evenl})$$

  - If typechecking under $\ldots, x{:}\mathsf{evenl}$, use 1st typing
  - If typechecking under $\ldots, x{:}\mathsf{oddl}$, use 2nd typing
- (This will make let-normal typechecking interesting...)

Theorems

- **Soundness**: If $\vdash e \downarrow A$ or $\vdash e \uparrow A$ then $\vdash |e| : A$.
- **Completeness**: If $\vdash e : A$ then $\vdash e' \downarrow A$ where e' is an annotated version of e.
- **Preservation + Progress** (cbv): Follows from result for type assignment system (Ch. 2)
- **Decidability of Typechecking**: Not immediate (can repeatedly apply direct); shown via left-rule system (inspired by [Barbanera et al. '95])

Bird's-Eye View



Tridirectional Typechecking: Summary

- To express properties of algebraic datatypes:
  - Datasort refinements
  - Index refinements
- To combine properties: $\wedge$, $\vee$, Π, Σ
- Bidirectional typechecking
  + Elim rules for $\vee$, Σ, $\parr$, $\bot$ on evaluation contexts
  = **Tridirectional typechecking**
- Contextual typing annotations
- Preservation + Progress
- Decidable

- ✓ Introduction
- ✓ Thesis Statement
- ✓ Contributions
- ✓ Bi- and Tridirectional Typechecking
- ☞ **Let-Normal Typechecking**
  - Implementation
  - Related Work
  - Summary
  - Future Work

Restricting Tridirectionality

- Rules like direct are impractical:
  Must guess which eval. context

$$\frac{\Gamma \vdash e' \uparrow A \qquad \Gamma, x{:}A \vdash \mathcal{E}[x] \downarrow C}{\Gamma \vdash \mathcal{E}[e'] \downarrow C}\ \text{direct}$$

- For example, $f\ (x, y)$ can be decomposed 4 ways:

$\mathcal{E}_1 = []\ (x, y)$	$f\ (x, y) = \mathcal{E}_1[f]$	Analyze f
$\mathcal{E}_2 = f\ ([], y)$	$f\ (x, y) = \mathcal{E}_2[x]$	Analyze x
$\mathcal{E}_3 = f\ (x, [])$	$f\ (x, y) = \mathcal{E}_3[y]$	Analyze y
$\mathcal{E}_4 = []$	$f\ (x, y) = \mathcal{E}_4[f\ (x, y)]$	Analyze entire term
✗ $\mathcal{E}_✗ = f\ []$	$f\ (x, y) = \mathcal{E}_✗[(x, y)]$	No: $(x, y) \not\uparrow$

- Let-normal system: analyze f, x, y, $f(x, y)$

Let-Normal Form: Related Work

- [Moggi88]
- Compilation, program analysis ("2/3 CPS") [many]
- Xi used let-normal form (for Σ),
  but typechecking was incomplete:

$$map\ (\lambda x.\ e) \quad \hookrightarrow_{\text{Xi}} \quad \textbf{let}\ \overline{x} = map\ \textbf{in}$$
$$\textbf{let}\ \overline{y} = \lambda x.\ e\ \textbf{in}$$
$$\ldots$$

Let-Normal Form: Related Work

- [Moggi88]
- Compilation, program analysis ("2/3 CPS") [many]
- Xi used let-normal form (for Σ),
  but typechecking was incomplete:

$$map\ (\lambda x.\ e) \quad \hookrightarrow_{\chi i} \quad \textbf{let}\ \overline{\mathbf{x}} = map\ \textbf{in}\ \textbf{let}\ \overline{\mathbf{y}} = \lambda x.\ e\ \textbf{in}\ \ldots$$

$$map\ (\textbf{case}\ \ldots\ \textbf{of}\ \ldots) \quad \hookrightarrow_{\chi i} \quad \textbf{let}\ \overline{\mathbf{x}} = map\ \textbf{in}\ \textbf{let}\ \overline{\mathbf{y}} = \textbf{case}\ \ldots\ \textbf{of}\ \ldots\ \textbf{in}\ \ldots$$

- Our form is peculiar, e.g. $\textbf{let}\ \overline{\mathbf{x}} = x$
  - ...and complete

Let-Normal Typechecking

- Translate, naming every synthesizing subterm in order

$$
\begin{array}{lll}
f\ (x, y) & \hookrightarrow & \textbf{let}\ \overline{f} = f\ \textbf{in} \\
 & & \quad \textbf{let}\ \overline{x} = x\ \textbf{in} \\
 & & \quad \textbf{let}\ \overline{y} = y\ \textbf{in} \\
 & & \quad \textbf{let}\ \overline{z} = \overline{f}(\overline{x}, \overline{y})\ \textbf{in}\ \overline{z}
\end{array}
$$

- Typecheck with this rule instead of direct:

$$\frac{\Gamma \vdash e' \uparrow A \qquad \Gamma, \overline{x}{:}A \vdash e \downarrow C}{\Gamma \vdash \textbf{let}\ \overline{x} = e'\ \textbf{in}\ e\ \downarrow\ C}\ \text{let}$$

- Translation + let $\simeq$ **Restricted strategy for** direct

Let-Normal Soundness

Original program e $\hookrightarrow$ Let-normal version e'

- **If** e' is well typed in the let-system
  then e is well typed in the (left) tridirectional system
- Not too difficult:
  Let-normal system is a restricted strategy for applying direct;
  any restricted strategy should be **sound**...

Let-Normal Completeness

Original program e $\hookrightarrow$ Let-normal version e'

- **If** e is well typed in the (left) tridirectional system
  then e' is well typed in the let-system
- Difficult!
  Let-normal system is a restricted strategy for applying direct;
  is this restricted strategy **complete**?

Seek Refreshment

at the Proof Bar:

Proving Completeness

$\mathcal{D} :: \Gamma; \Delta \vdash e \downarrow^{\mathbb{L}} C$

- Add **let**s corresponding to the places direct was used in $\mathcal{D}$, yielding e_1
- This is (almost certainly) not the real translation e' (as in $e \hookrightarrow e'$)
- But it's a start: easy to get $\mathcal{D}_1 :: \Gamma; \Delta \vdash e_1 \downarrow^{\mathsf{let}} C$

Little By Little

$\mathcal{D}_1 :: \Gamma; \Delta \vdash e_1 \downarrow C$

- **Letification**:
  - Add "useless" **let**s for **all** synthesizing terms (except $(v : As)$)
    yielding e_2

Little By Little

$\mathcal{D}_1 :: \Gamma; \Delta \vdash e_1 \downarrow C$

- **Letification**:
  - Add "useless" **let**s for **all** synthesizing terms (except $(v : As)$)
    yielding e_2
- **Slackening**:
  - Add **let**$\sim$s for all non-principal synth. values $(v : As)$,
    yielding e_3

Little By Little

$\mathcal{D}_1 :: \Gamma; \Delta \vdash e_1 \downarrow C$

- **Letification**:
  - Add "useless" **let**s for **all** synthesizing terms
    (except $(v : As)$)
    yielding e_2
- **Slackening**:
  - Add **let**$\sim$s for all non-principal synth. values $(v : As)$,
    yielding e_3
- Now e_3 has all the **let**s that e' does,
  just not in the right places

Little By Little

$\mathcal{D}_3 :: \Gamma; \Delta \vdash e_3 \downarrow C$

- **Letification** got us from e_1 to e_2
- **Slackening** got us from e_2 to e_3
- **Permutation**:

$$\Gamma; \Delta \vdash (), \mathbf{let}\ \overline{y} = y\ \mathbf{in}\ \overline{y}) \downarrow \mathsf{unit} * B$$

The $\overline{y} = y$ is there, but in the wrong place

$$(x, y) \quad \hookrightarrow \quad \mathbf{let}\ \overline{y} = y\ \mathbf{in}\ ((), \overline{y})$$

So we **permute** it to its canonical position:

$$\Gamma; \Delta \vdash \mathbf{let}\ \overline{y} = y\ \mathbf{in}\ ((), \overline{y}) \downarrow \mathsf{unit} * B$$

Little By Little

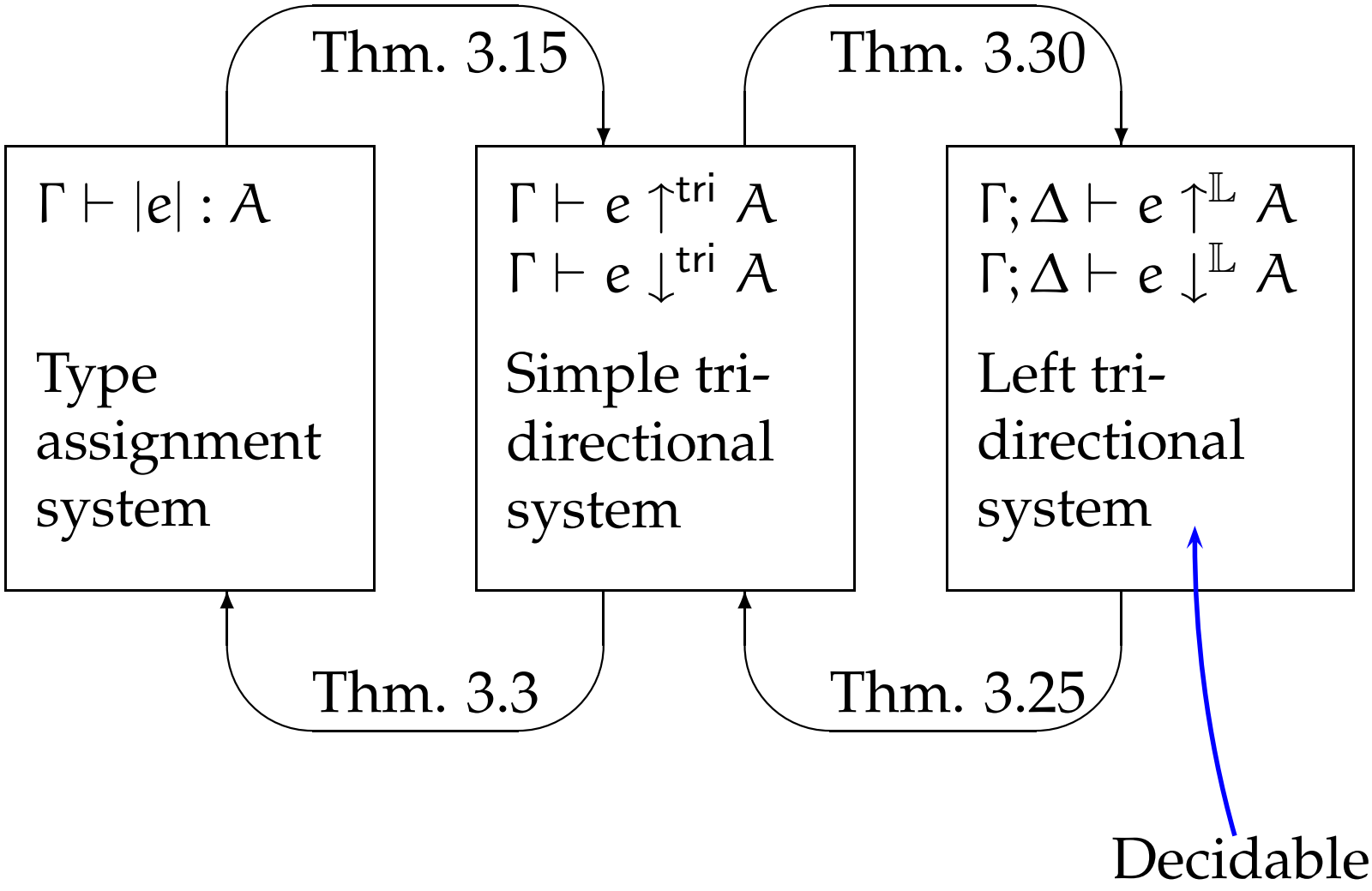
$e \hookrightarrow e'$
$\mathcal{D}_4 :: \Gamma; \Delta \vdash e_4 \downarrow C$

- **Letification** got us from e_1 to e_2
- **Slackening** got us from e_2 to e_3
- **Permutation** got us from e_3 to e_4

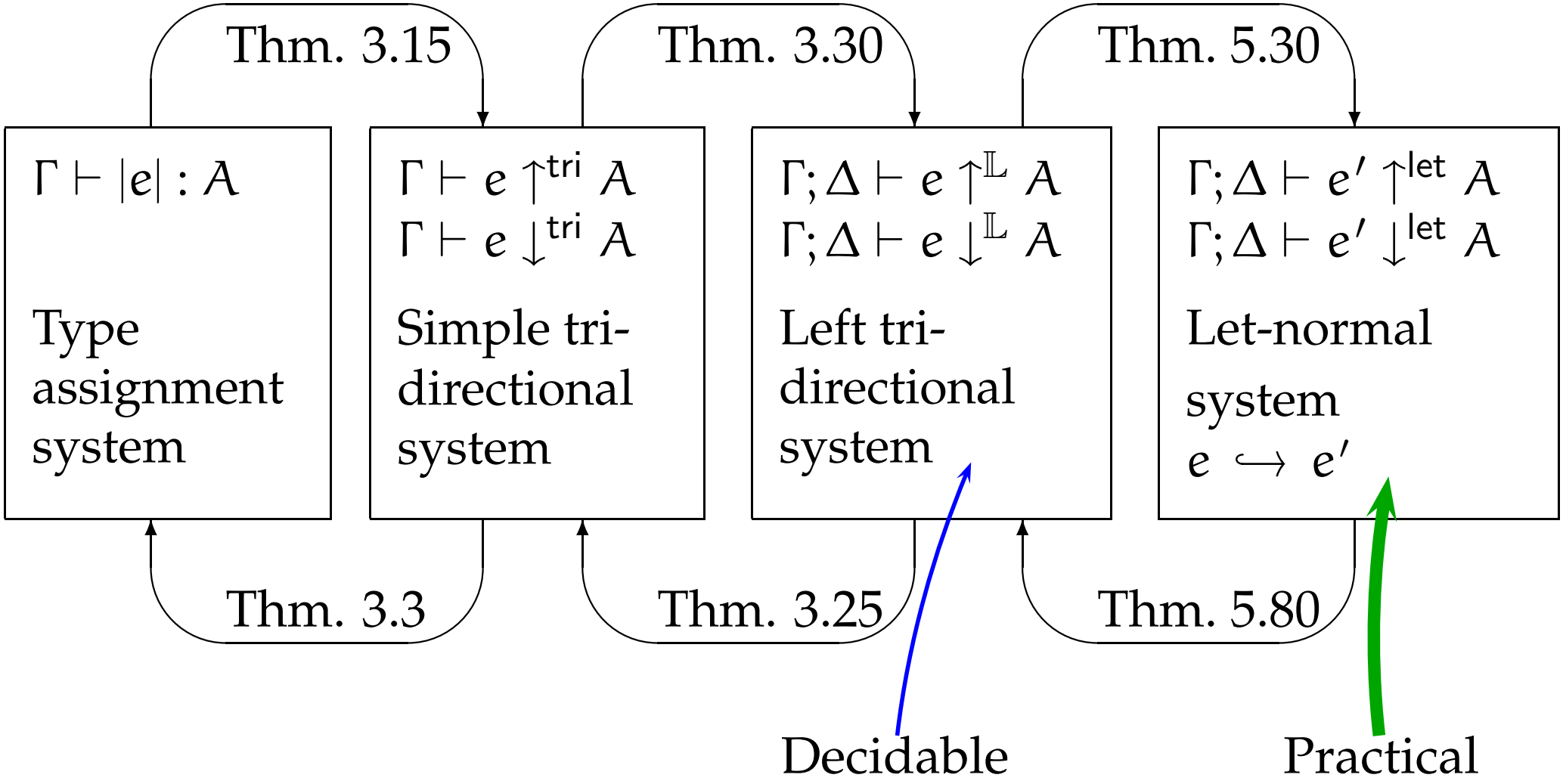
$e_4 = e'$
Above process must be done piecemeal, e.g.:

$$\frac{\Gamma; \Delta \vdash e \downarrow A \qquad \Gamma; \Delta \vdash e \downarrow B}{\Gamma; \Delta \vdash e \downarrow A \wedge B} \wedge I$$

Bird's-Eye View, Revisited



Bird's-Eye View, Letified



- ✓ Introduction
- ✓ Thesis Statement
- ✓ Contributions
- ✓ Bi- and Tridirectional Typechecking
- ✓ Let-Normal Typechecking
- ☞ **Implementation**
  - Related Work
  - Summary
  - Future Work

Implementation: Stardust

- subset of core Standard ML
  - ✗ syntactic sugar
  - ✓ patterns, e.g.

```
Black(e, Red lt, Red (rt as (_,_,Red _)))
```

Index Domains

- Integers $\mathcal{Z}$
  - linear inequalities
- Booleans bool
- Dimensions dim

Integers: Red-Black Trees

- Verify invariants (1)–(3) for insertion
- Deletion: SML/NJ library
  - Couldn't find right refined types, but still found a bug (violated color invariant)
  - Such a bug is very unlikely to be found by testing

Dimension Index Sort dim

- Units `M`, `S`, `KG`
- Multiplication and powers:

```
M*KG    M^2    S^a
```

- Multiplicative identity *1*
- real indexed by dim

Dimensions Example

```
(*[ val power : -all  d:dim- -all  n:int-
                  int(n) → real(d) → real(d^n)
]*)
fun power n x =
  if  n = 0 then
    1.0
  else  if  n < 0 then
    1.0 / power (~n) x
  else
    x * power (n-1) x
```

Dimensions: Invaluable Refinement

Unlike e.g. red-black trees,
our dimension refinements are **not** tied to values:

- For every d, each type real(d) is inhabited by
  exactly the same set of floating-point values:

$$\begin{aligned} 3.1 &\downarrow \mathsf{real}(1) \\ 3.1 * M &\downarrow \mathsf{real}(\mathtt{M}) \\ 3.1 * S &\downarrow \mathsf{real}(\mathtt{S}) \\ 3.1 * M * M &\downarrow \mathsf{real}(\mathtt{M\,\hat{}\,2}) \end{aligned}$$

  All these expressions yield identical values at runtime!

Dimensions: Related Work

- ’70s and ’80s
- Kennedy ’96: Polymorphism, type inference
- Allen ’04: abelian classes in Java

Constraint Solving

- Delegate most work to external tools:
  ICS or CVC Lite
  - Accumulate constraints incrementally
- Solve for existential index variables
- Reduce dimension constraints to integer constraints

(Almost) Ideal Constraint Solver Interface

1. Persistent solver contexts Ω encapsulating $\overline{\Gamma}$
2. $\textsc{assert}(\Omega, P)$ returning

$$\begin{array}{rll} \mathsf{Valid} & \text{if} & \Omega \models P \\ \mathsf{Invalid} & \text{if} & \Omega, P \models \bot \\ \mathsf{Contingent}(\Omega') & \text{if} & \Omega \not\models P \text{ and } \Omega, P \not\models \bot, \\ & & \text{yielding a new context } \Omega' = \Omega, P \end{array}$$

3. $\textsc{valid}(\Omega, P)$ returning

$$\begin{array}{rll} \mathsf{Valid} & \text{if} & \Omega \models P \\ \mathsf{Invalid} & \text{otherwise.} & \end{array}$$

4. $\textsc{declare}(\Omega, a, \gamma)$

ICS

- Developed at SRI (de Moura, Rueß, Shankar, . . .)
- Integers (almost), rationals, bitvectors (not really), functional arrays, . . .
- Provides the desired interface
- Fast

- **Future:**
  - Closed source
  - Deprecated, replaced by Yices (hilarious licensing)
  - ICS a great improvement on its successor?

ICS

- Developed at SRI (de Moura, Rueß, Shankar, . . .)
- Integers (almost), rationals, bitvectors (not really), functional arrays, . . .
- Provides the desired interface
- Fast
  except when it hangs
- **Future:**
  - Closed source
  - Deprecated, replaced by Yices (hilarious licensing)
  - ICS a great improvement on its successor?

CVC Lite

- Barrett, Berezin, [Dill], [Tinelli], . . .
- Integers, rationals, bitvectors, functional arrays, . . .
- Does not provide persistent contexts
- Slower than ICS, but more stable
- **Future:**
  - Deprecated, replaced by CVC 3
  - Open source

Typechecking Time

Input program		Wall-clock seconds ICS	CVC Lite library	CVC Lite stand-alone
redblack	$\delta(i)$ $\wedge$	< 1	< 1	1.2
redblack-full	$\delta(i)$ $\wedge$	3.5	10.0	15.2
redblack-full-bug1	$\delta(i)$ $\wedge$	2.9	8.6	13.3
refine	$\delta(i)$ $\wedge$	< 1	< 1	< 1
bits	$\delta(i)$ $\wedge\vee$	*	10.8	6.9
bits-un	$\delta(i)$ $\wedge\vee$	61.9	*	*
others	$\delta(i)$	< 1	< 1	< 1

Subformula property $\Longrightarrow$ Don't pay for what you don't use

Memoization

- Xi's DML:
  index refinements
  No $\wedge\vee$:
  - don't need to memoize
- Davies' SML-CIDRE:
  datasort refinements, $\wedge$
  No index refinements—no Σ:
  - memoization easy
- Stardust:
  datasort + index, $\wedge\vee$
  - memoization ineffective
    - might work with sophisticated irrelevance

- ✓ Introduction
- ✓ Thesis Statement
- ✓ Contributions
- ✓ Bi- and Tridirectional Typechecking
- ✓ Let-Normal Typechecking
- ✓ Implementation
- ☞ **Related Work**
- Summary
- Future Work

Related Work

- [Davies '97/'05, Davies & Pf. '00]: Bidirectional system with δ, $\wedge$
- [Xi & Pfenning '99]: Bidirectional system with $\imath$, Π, Σ
- [Coppo et al. '81]: $\wedge$ can characterize normal forms (termination); hence full inference undecidable
- [Reynolds '96]: FORSYTHE with $\wedge$ (and type annotations)
- [Pierce '91]: Language with $\wedge$, $\vee$, syntactic markers
- Dependent types ((less) restricted): Cayenne, Epigram, [Chen & Xi '05], [Licata & Harper '05], ...

- ✓ Introduction
- ✓ Thesis Statement
- ✓ Contributions
- ✓ Bi- and Tridirectional Typechecking
- ✓ Let-Normal Typechecking
- ✓ Implementation
- ✓ Related Work
- ☞ **Summary**
- Future Work

Summary of Contributions

- Type system
  - Combination of datasort and index refinements, marker-free intersections, unions, quantifiers
    - Value restriction on $\top$I, $\wedge$I, ΠI
      Evaluation context restriction on $\bot$E, $\vee$E, ΣE
    - Complete annotation mechanism
- Data structure refinements, e.g. bitstrings, red-black trees
- Invaluable refinements, e.g. dimensions
- Bidirectional $\leftrightarrow$ Tridirectional $\leftrightarrow$ Let-Normal
  - + ML patterns (Ch. 4)
- Proof-of-concept typechecker

- ✓ Introduction
- ✓ Thesis Statement
- ✓ Contributions
- ✓ Bi- and Tridirectional Typechecking
- ✓ Let-Normal Typechecking
- ✓ Implementation
- ✓ Related Work
- ✓ Summary
- ☞ **Future Work**

Future Work

- Parametric polymorphism
- Index domains: bit vectors, inductive families, finite sets, regular languages, ...
- Memoization and parallelization
- Refinement-based compilation
- Call-by-name (and -need)
- $\dfrac{\text{Derivation}}{\text{Counterexample}}$ generation
- Suggestion tools

Thesis Statement

A rich type system with datasort and index refinement properties, where such properties are combined through intersection and union types, is a practical means of statically checking interesting properties of functional programs that are difficult or impossible to check in conventional static type systems.

Acknowledgments

- Håkan, Mukesh, Jonathan, Maverick, Kate, Urs, Margaret, Aleksey, Neel, Noam, Tim, . . .
- My committee

- "I have discovered many wonderful people, which this slide is too small to contain"
- My transit systems, and my bikes:

(R.I.P.)

Extra slides

Bidirectional Typing: $*$

- **Principle:**
  Introduction rules **check**,
  elimination rules **synthesize**

$$\frac{\Gamma \vdash e_1 \ : \ A \qquad \Gamma \vdash e_2 \ : \ B}{\Gamma \vdash (e_1, e_2) \ : \ A * B} *\mathrm{I}$$

$$\frac{\Gamma \vdash e \ : \ A * B}{\Gamma \vdash \mathbf{fst}(e) \ : \ A} *\mathrm{E}_1 \qquad \frac{\Gamma \vdash e \ : \ A * B}{\Gamma \vdash \mathbf{snd}(e) \ : \ B} *\mathrm{E}_2$$

- Subtyping:

$$\frac{A_1 \leq B_1 \qquad A_2 \leq B_2}{A_1 * A_2 \leq B_1 * B_2}$$

Bidirectional Typing: $*$

- **Principle:**
  Introduction rules **check**,
  elimination rules **synthesize**

$$\frac{\Gamma \vdash e_1 \ : \ A \qquad \Gamma \vdash e_2 \ : \ B}{\Gamma \vdash (e_1, e_2) \downarrow A * B} *I$$

$$\frac{\Gamma \vdash e \ : \ A * B}{\Gamma \vdash \mathbf{fst}(e) \ : \ A} *E_1 \qquad \frac{\Gamma \vdash e \ : \ A * B}{\Gamma \vdash \mathbf{snd}(e) \ : \ B} *E_2$$

- Subtyping:

$$\frac{A_1 \leq B_1 \qquad A_2 \leq B_2}{A_1 * A_2 \leq B_1 * B_2}$$

Bidirectional Typing: $*$

- **Principle:**
  Introduction rules **check**,
  elimination rules **synthesize**

$$\frac{\Gamma \vdash e_1 \downarrow A \qquad \Gamma \vdash e_2 \downarrow B}{\Gamma \vdash (e_1, e_2) \downarrow A * B} *\mathrm{I}$$

$$\frac{\Gamma \vdash e \ : \ A * B}{\Gamma \vdash \mathbf{fst}(e) \ : \ A} *\mathrm{E}_1 \qquad \frac{\Gamma \vdash e \ : \ A * B}{\Gamma \vdash \mathbf{snd}(e) \ : \ B} *\mathrm{E}_2$$

- Subtyping:

$$\frac{A_1 \leq B_1 \qquad A_2 \leq B_2}{A_1 * A_2 \leq B_1 * B_2}$$

Bidirectional Typing: $*$

- **Principle:**
  Introduction rules **check**,
  elimination rules **synthesize**

$$\frac{\Gamma \vdash e_1 \downarrow A \qquad \Gamma \vdash e_2 \downarrow B}{\Gamma \vdash (e_1, e_2) \downarrow A * B} *\mathrm{I}$$

$$\frac{\Gamma \vdash e \uparrow A * B}{\Gamma \vdash \mathbf{fst}(e) \ : \ A} *\mathrm{E}_1 \qquad \frac{\Gamma \vdash e \ : \ A * B}{\Gamma \vdash \mathbf{snd}(e) \ : \ B} *\mathrm{E}_2$$

- Subtyping:

$$\frac{A_1 \leq B_1 \qquad A_2 \leq B_2}{A_1 * A_2 \leq B_1 * B_2}$$

Bidirectional Typing: $*$

- **Principle:**
  Introduction rules **check**,
  elimination rules **synthesize**

$$\frac{\Gamma \vdash e_1 \downarrow A \qquad \Gamma \vdash e_2 \downarrow B}{\Gamma \vdash (e_1, e_2) \downarrow A * B} *\mathrm{I}$$

$$\frac{\Gamma \vdash e \uparrow A * B}{\Gamma \vdash \mathbf{fst}(e) \uparrow A} *\mathrm{E}_1 \qquad \frac{\Gamma \vdash e \,:\, A * B}{\Gamma \vdash \mathbf{snd}(e) \,:\, B} *\mathrm{E}_2$$

- Subtyping:

$$\frac{A_1 \leq B_1 \qquad A_2 \leq B_2}{A_1 * A_2 \leq B_1 * B_2}$$

Bidirectional Typing: $*$

- **Principle:**
  Introduction rules **check**,
  elimination rules **synthesize**

$$\frac{\Gamma \vdash e_1 \downarrow A \qquad \Gamma \vdash e_2 \downarrow B}{\Gamma \vdash (e_1, e_2) \downarrow A * B} *\mathrm{I}$$

$$\frac{\Gamma \vdash e \uparrow A * B}{\Gamma \vdash \mathbf{fst}(e) \uparrow A} *\mathrm{E}_1 \qquad \frac{\Gamma \vdash e \uparrow A * B}{\Gamma \vdash \mathbf{snd}(e) \ : \ B} *\mathrm{E}_2$$

- Subtyping:

$$\frac{A_1 \leq B_1 \qquad A_2 \leq B_2}{A_1 * A_2 \leq B_1 * B_2}$$

Bidirectional Typing: $*$

- **Principle:**
  Introduction rules **check**,
  elimination rules **synthesize**

$$\frac{\Gamma \vdash e_1 \downarrow A \qquad \Gamma \vdash e_2 \downarrow B}{\Gamma \vdash (e_1, e_2) \downarrow A * B} *\mathrm{I}$$

$$\frac{\Gamma \vdash e \uparrow A * B}{\Gamma \vdash \mathbf{fst}(e) \uparrow A} *\mathrm{E}_1 \qquad \frac{\Gamma \vdash e \uparrow A * B}{\Gamma \vdash \mathbf{snd}(e) \uparrow B} *\mathrm{E}_2$$

- Subtyping:

$$\frac{A_1 \leq B_1 \qquad A_2 \leq B_2}{A_1 * A_2 \leq B_1 * B_2}$$

Type Annotations

- Usual annotation form $(e : A)$ not adequate:
  - Intersection types
    - Annotate with several types $(e : A_1, \ldots, A_n)$ [Pierce '91, Reynolds '96]
    - Typechecker guesses which type to use
  - Index variables
    - Scoping issues (α-conversion fails)

Index Variable Scoping

- $(\lambda x.(\underbrace{\lambda z.\, x}_{\text{Requires annotation}})())$: $\Pi a{:}\mathcal{N}.\, \mathsf{list}(a) \to \mathsf{list}(a)$
- What annotation?
- $((\lambda z.\, x) : \mathsf{unit} \to \mathsf{list}(a))$
  - $\Pi a{:}\mathcal{N}.\, \underbrace{\mathsf{list}(a) \to \mathsf{list}(a)}_{\text{Natural scope of } a}$
  - α-converting should change nothing:
    $(\lambda z.\, x : \mathsf{unit} \to \mathsf{list}(a)) \cdots : \Pi b{:}\mathcal{N}.\, \mathsf{list}(b) \to \mathsf{list}(b)$
    ...but $\mathsf{list}(a)$ is now meaningless

Index Variable Scoping

- Term-level binder $\Lambda a{:}\gamma.\ e$ for Π fails with intersections:

$$\begin{array}{lll} \mathit{rev} & : & (\Pi a{:}\mathcal{N}.\,\mathsf{list}(a) \to \mathsf{list}(a)) \qquad\qquad \text{needs one } \Lambda \\ & & \wedge\ ((\Sigma b{:}\mathcal{N}.\,\mathsf{list}(b)) \to \Sigma c{:}\mathcal{N}.\,\mathsf{list}(c)) \quad \text{needs no } \Lambda\text{s} \end{array}$$

Great Collapsing "Theorem" Disaster of 2004

$$\dfrac{y \uparrow A_1 \vee A_2 \qquad \dfrac{\dfrac{\dfrac{}{v_1 \uparrow A_1 \to B} \quad \vdots}{\ldots; \overline{y}{:}A_1 \vdash v_1\ \overline{y} \downarrow B}\ \to E \qquad \dfrac{\dfrac{}{v_1 \uparrow A_2 \to B} \quad \vdots}{\ldots; \overline{y}{:}A_2 \vdash v_1\ \overline{y} \downarrow B}\ \to E}{\ldots; \overline{y} : A_1 \vee A_2 \vdash v_1\ \overline{y} \downarrow B}\ \vee \mathbb{L}}{y : A_1 \vee A_2; \cdot \vdash v_1\ y \downarrow B}\ \text{direct}\mathbb{L}$$

Great Collapsing "Theorem" Disaster of 2004

$$\dfrac{y \uparrow A_1 \vee A_2 \qquad \dfrac{\dfrac{v_1 \uparrow A_1 \to B \quad \vdots}{\ldots;\overline{y}{:}A_1 \vdash v_1\ \overline{y} \downarrow B}\to E \quad \dfrac{v_1 \uparrow A_2 \to B \quad \vdots}{\ldots;\overline{y}{:}A_2 \vdash v_1\ \overline{y} \downarrow B}\to E}{\ldots;\overline{y} : A_1 \vee A_2 \vdash v_1\ \overline{y} \downarrow B}\vee\mathbb{L}}{y : A_1 \vee A_2;\cdot \vdash v_1\ y \downarrow B}\ \text{direct}\mathbb{L}$$

$$v_1 = \Big((\lambda z.\, e) : (\vdash A_1 \to B), (\vdash A_2 \to B)\Big) \quad \text{and} \quad e \hookrightarrow e'.$$

$$\begin{aligned} v_1\ y \quad \hookrightarrow \quad & \textbf{let}\ \overline{x} = \lambda z.\, e' : (\vdash A_1 \to B), (\vdash A_2 \to B)\ \textbf{in} \\ & \textbf{let}\ \overline{y} = y\ \textbf{in let}\ \overline{a} = \overline{x}\ \overline{y}\ \textbf{in}\ \overline{a} \end{aligned}$$

Great Collapsing "Theorem" Disaster of 2004

$$\dfrac{y \uparrow A_1 \vee A_2 \qquad \dfrac{\dfrac{v_1 \uparrow A_1 \to B \quad \vdots}{\ldots;\overline{y}{:}A_1 \vdash v_1\ \overline{y} \downarrow B}\to E \quad \dfrac{v_1 \uparrow A_2 \to B \quad \vdots}{\ldots;\overline{y}{:}A_2 \vdash v_1\ \overline{y} \downarrow B}\to E}{\ldots;\overline{y} : A_1 \vee A_2 \vdash v_1\ \overline{y} \downarrow B}\vee\mathbb{L}}{y : A_1 \vee A_2;\cdot \vdash v_1\ y \downarrow B}\ \text{direct}\mathbb{L}$$

$$v_1 = \Big((\lambda z.\, e) : (\vdash A_1 \to B), (\vdash A_2 \to B)\Big) \quad \text{and} \quad e \hookrightarrow e'.$$

$$\begin{array}{lll} v_1\ y & \hookrightarrow & \textbf{let}\ \overline{x} = \lambda z.\, e' : (\vdash A_1 \to B), (\vdash A_2 \to B)\ \textbf{in} \\ & & \quad \textbf{let}\ \overline{y} = y\ \textbf{in let}\ \overline{a} = \overline{x}\ \overline{y}\ \textbf{in}\ \overline{a} \end{array}$$

$$\dfrac{\ldots;\cdot \vdash v_1 \uparrow (A_{???} \to B) \qquad \ldots;\overline{x}{:}(A_{???} \to B) \vdash \textbf{let}\ \overline{y} = y\ \textbf{in}\ \ldots \downarrow B}{y : A_1 \vee A_2;\cdot \vdash \textbf{let}\ \overline{x} = (\lambda z.\, e' : \ldots)\ \textbf{in}\ \ldots \downarrow B}\ \text{let}$$

(with $\vdots \quad \vdots$ above $\vee\mathbb{L}$ above $\vdots$)

Great Collapsing "Theorem" Disaster of 2004

$$\dfrac{y \uparrow A_1 \vee A_2 \qquad \dfrac{\dfrac{v_1 \uparrow A_1 \to B \quad \vdots}{\ldots;\overline{y}{:}A_1 \vdash v_1\ \overline{y} \downarrow B}\to E \quad \dfrac{v_1 \uparrow A_2 \to B \quad \vdots}{\ldots;\overline{y}{:}A_2 \vdash v_1\ \overline{y} \downarrow B}\to E}{\ldots;\overline{y} : A_1 \vee A_2 \vdash v_1\ \overline{y} \downarrow B}\vee\mathbb{L}}{y : A_1 \vee A_2;\cdot \vdash v_1\ y \downarrow B}\ \text{direct}\mathbb{L}$$

$$v_1 = \Big((\lambda z.\, e) : (\vdash A_1 \to B), (\vdash A_2 \to B)\Big) \quad \text{and} \quad e \hookrightarrow e'.$$

$$v_1\ y \quad \hookrightarrow \quad \mathbf{let}\ \overline{x} = \lambda z.\, e' : (\vdash A_1 \to B), (\vdash A_2 \to B)\ \mathbf{in}$$
$$\mathbf{let}\ \overline{y} = y\ \mathbf{in}\ \mathbf{let}\ \overline{a} = \overline{x}\ \overline{y}\ \mathbf{in}\ \overline{a}$$

$$\dfrac{\ldots;\cdot \vdash v_1 \uparrow (A_{???} \to B) \qquad \ldots;\overline{x}{:}(A_{???} \to B) \vdash \mathbf{let}\ \overline{y} = y\ \mathbf{in}\ \ldots \downarrow B}{y : A_1 \vee A_2;\cdot \vdash \mathbf{let}\ \overline{x} = (\lambda z.\, e' : \ldots)\ \mathbf{in}\ \ldots \downarrow B}\ \text{let}$$

(the $\vdots$ $\vdots$ / $\vee\mathbb{L}$ / $\vdots$ above the left premise)

No best type for v_1: Completeness fails

Great Collapsing "Theorem" Disaster of 2004

Need **principal synthesis** for let-bound synthesizing **values**.

Principal synthesis: If $v \uparrow C_1$ and $v \uparrow C_2$
then there exists C s.t.
$v \uparrow C$ and $C \uparrow C_1$ and $C \uparrow C_2$.

Example: $\Gamma = x : A \wedge B$ $x \uparrow A$ and $x \uparrow B$
$C = A \wedge B$ $A \wedge B \uparrow A$ and $A \wedge B \uparrow B$

Great Collapsing "Theorem" Disaster of 2004

Need **principal synthesis** for let-bound synthesizing **values**.

Principal synthesis: If $v \uparrow C_1$ and $v \uparrow C_2$
then there exists C s.t.
$v \uparrow C$ and $C \uparrow C_1$ and $C \uparrow C_2$.

Example: $\Gamma = x : A \wedge B$ $x \uparrow A$ and $x \uparrow B$
$C = A \wedge B$ $A \wedge B \uparrow A$ and $A \wedge B \uparrow B$

We only care about **values**: In $e_1\ y$, if e_1 **not** a value,

- can't let-bind y to $\overline{y}$ without doing $e_1 \uparrow$ first...
  - so can't $e_1 \uparrow$ twice inside each branch of $\vee\mathbb{L}$ on $\overline{y}$

Great Collapsing "Theorem" Disaster of 2004

Need **principal synthesis** for let-bound synthesizing **values**.

Principal synthesis: If $\nu \uparrow C_1$ and $\nu \uparrow C_2$
then there exists C s.t.
$\nu \uparrow C$ and $C \uparrow C_1$ and $C \uparrow C_2$.

Example: $\Gamma = x : A \wedge B$ $x \uparrow A$ and $x \uparrow B$
$C = A \wedge B$ $A \wedge B \uparrow A$ and $A \wedge B \uparrow B$

We only care about **values**: In $e_1\ y$, if e_1 **not** a value,

- can't let-bind y to $\overline{y}$ without doing $e_1 \uparrow$ first...
  - so can't $e_1 \uparrow$ twice inside each branch of $\vee\mathbb{L}$ on $\overline{y}$
- Most synthesizing **values** have principal synthesis:

$$x \quad \text{has} \quad \Gamma(x)$$
$$\overline{x} \quad \text{has} \quad \Delta(\overline{x})$$

The Slackers Seize Control

- But type annotations don't have principal synthesis
  $(v : (\vdash A_1 \to B, \vdash A_2 \to B))$
  $(v : (a{:}\mathcal{Z} \vdash \mathsf{list}(a) \to \mathsf{list}(a)))$ ("corner case")
- Solution: Bind $(v : As)$ to a "slack variable"
  $\mathbf{let} \sim\overline{\mathbf{x}} = (v : As)\ \mathbf{in}\ \ldots$

$$\frac{\Gamma; \Delta_1 \vdash v \uparrow A \qquad \Gamma; \Delta_2, \overline{\mathbf{x}}{:}A \vdash e \downarrow C}{\Gamma; \Delta_1, \Delta_2, \sim\overline{\mathbf{x}} = v \vdash e \downarrow C}\ \sim\mathsf{var} \qquad \frac{\Gamma; \Delta, \sim\overline{\mathbf{x}} = v \vdash e \downarrow C}{\Gamma; \Delta \vdash \mathbf{let} \sim\overline{\mathbf{x}} = v\ \mathbf{in}\ e \downarrow C}\ \mathsf{let}\sim$$

- Let-normal system is a restricted strategy for direct$\mathbb{L}$ ...
- Slack variables **remove** the restriction for $(v : As)$ terms

Slacking in Moderation

- Slack variables **remove** the restriction for $(v : As)$ terms
- Restricting direct$\mathbb{L}$ nondeterminism was the whole point! So why bother?

Slacking in Moderation

- Slack variables **remove** the restriction for $(v : As)$ terms
- Restricting $\mathsf{direct}\mathbb{L}$ nondeterminism was the whole point! So why bother?
  - Terms of the form $(v : As)$ are **rare**

Slacking in Moderation

- Slack variables **remove** the restriction for $(v : As)$ terms
- Restricting $\mathsf{direct}\mathbb{L}$ nondeterminism was the whole point! So why bother?
  - Terms of the form $(v : As)$ are **rare**
  - Most annotations are on **fun**s, which become **fix** $u.\ \lambda x.\ e$
  - A **fix** is not a value

Great Collapsing "Theorem" Disaster of ~~2004~~2006

The translation of (e_1, e_2) depends on e_1

$$(x, y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in let } \overline{y} = y \textbf{ in } (\overline{x}, \overline{y})$$

y in eval. pos.

$$(\textbf{case } x \textbf{ of } ms, y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in } (\textbf{case } \overline{x} \textbf{ of } ms, \textbf{let } \overline{y} = y \textbf{ in } \overline{y})$$

Great Collapsing "Theorem" Disaster of ~~2004~~2006

The translation of (e_1, e_2) depends on e_1

$$(x, y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in } \textbf{let } \overline{y} = y \textbf{ in } (\overline{x}, \overline{y})$$

y in eval. pos. — y bound outside tuple

$$(\textbf{case } x \textbf{ of } ms, y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in } (\textbf{case } \overline{x} \textbf{ of } ms, \textbf{let } \overline{y} = y \textbf{ in } \overline{y})$$

Great Collapsing "Theorem" Disaster of ~~2004~~2006

The translation of (e_1, e_2) depends on e_1

$$(x, y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in } \textbf{let } \overline{y} = y \textbf{ in } (\overline{x}, \overline{y})$$

y in eval. pos. — y bound outside tuple

$$(\textbf{case } x \textbf{ of } ms, y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in } (\textbf{case } \overline{x} \textbf{ of } ms, \textbf{let } \overline{y} = y \textbf{ in } \overline{y})$$

y **not** in eval. pos.

Great Collapsing "Theorem" Disaster of ~~2004~~2006

The translation of (e_1, e_2) depends on e_1

$$(x, y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in } \textbf{let } \overline{y} = y \textbf{ in } (\overline{x}, \overline{y})$$

y in eval. pos. — y bound outside tuple

$$(\textbf{case } x \textbf{ of } ms, y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in } (\textbf{case } \overline{x} \textbf{ of } ms, \textbf{let } \overline{y} = y \textbf{ in } \overline{y})$$

y **not** in eval. pos. — y bound **inside** tuple

Great Collapsing "Theorem" Disaster of ~~2004~~2006

The translation of (e_1, e_2) depends on e_1

$$(x, y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in } \textbf{let } \overline{y} = y \textbf{ in } (\overline{x}, \overline{y})$$

y in eval. pos. — y bound outside tuple

$$(\textbf{case } x \textbf{ of } ms, y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in } (\textbf{case } \overline{x} \textbf{ of } ms, \textbf{let } \overline{y} = y \textbf{ in } \overline{y})$$

y **not** in eval. pos. — y bound **inside** tuple

x is a value, so next term is in eval. pos.

case x **of** ms is **not** a value, so next term **not** in eval. pos.

Great Collapsing "Theorem" Disaster of ~~2004~~2006

The translation of (e_1, e_2) depends on e_1

$$(x, y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in } \textbf{let } \overline{y} = y \textbf{ in } (\overline{x}, \overline{y})$$

y in eval. pos. — y bound outside tuple

$$(x(), y) \quad \hookrightarrow$$

$$(\textbf{case } x \textbf{ of } ms, y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in } (\textbf{case } \overline{x} \textbf{ of } ms, \textbf{let } \overline{y} = y \textbf{ in } \overline{y})$$

y **not** in eval. pos. — y bound **inside** tuple

x is a value, so next term is in eval. pos.

case x **of** ms is **not** a value, so next term **not** in eval. pos.

Great Collapsing "Theorem" Disaster of ~~2004~~2006

The translation of (e_1, e_2) depends on e_1

$$(x, y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in } \textbf{let } \overline{y} = y \textbf{ in } (\overline{x}, \overline{y})$$

y in eval. pos. — y bound outside tuple

$$(\overline{a}\,x(), y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in let } \overline{a} = \overline{x}() \textbf{ in } \textbf{let } \overline{y} = y \textbf{ in } (\overline{a}, \overline{y})$$

y in eval. pos.! — y bound outside tuple

$$(\textbf{case } x \textbf{ of } ms, y) \quad \hookrightarrow \quad \textbf{let } \overline{x} = x \textbf{ in } (\textbf{case } \overline{x} \textbf{ of } ms, \textbf{let } \overline{y} = y \textbf{ in } \overline{y})$$

y **not** in eval. pos. — y bound **inside** tuple

x	is a **pre-**value,	so next term is in eval. pos.
$x()$	is a **pre-**value,	so next term **will be** in eval. pos.
case x **of** ms	is **not** a value,	so next term **not** in eval. pos.

Same-Dimension Units

```
(*[  indexconstant   M : dim
     indexconstant   FT : dim
     primitive   val  M : real(M)
     primitive   val  FT : real(FT)     ]*)
     val  metersPerFoot : real(M / FT)
     val  fromFeet : real(FT) → real(M)
     val  toFeet : real(M) → real(FT)       ]*)
val  metersPerFoot = (0.3048 * M) / FT
fun  fromFeet ft = ft * metersPerFoot
fun  toFeet m = m / metersPerFoot

(*[ val readDistance : real → string → real(M)    ]*)
fun  readDistance number units = case  units of
     "m" ⇒ number * M
   | "ft" ⇒ fromFeet (number * FT)
   | _ ⇒ raise  BadInput ("unknown␣distance␣unit:␣"^units)
```

Stardust: Initial Phases

1. Check index sorts (e.g. $\mathsf{int}(3,4)$ ✗)
2. Inject unindexed types

$$\mathsf{int} \implies \Sigma a{:}\mathcal{Z}.\, \mathsf{int}(a)$$

$$\delta \implies \Sigma a{:}\gamma.\, \delta(a)$$

$$\mathsf{real} \implies \mathsf{real}(1) \qquad \text{default index}$$

3. Flatten products
4. Eliminate subset sorts

$$\Pi a{:}\{a{:}\gamma \mid P\}.\, A \implies \Pi a{:}\gamma.\ P \supset A$$

$$\Sigma a{:}\{a{:}\gamma \mid P\}.\, A \implies \Sigma a{:}\gamma.\ P \otimes A$$

$$\text{e.g. } \Sigma a{:}\mathcal{N}.\, \mathsf{int}(a) \implies \Sigma a{:}\mathcal{Z}.\ (a \geq 0) \otimes \mathsf{int}(a)$$

5. Translate to let-normal form

Parametric Polymorphism: Declarative

- As property type

$$\frac{\Gamma \vdash e \uparrow \Pi a{:}\gamma.\, A \qquad \overline{\Gamma} \vdash i : \gamma}{\Gamma \vdash e \uparrow [i/a]\, A}\ \Pi E$$

$$\frac{\Gamma \vdash e \uparrow \forall \alpha.\, B \qquad \Gamma \vdash A\ [\text{mono}]\text{type}}{\Gamma \vdash e \uparrow [A/\alpha]\, B}$$

Greed and Failure

- Generate existential variable
- The greedy method [Cardelli '93]
- $f : \forall\alpha.\ \alpha \to \alpha \to \alpha$
  $x : A, y : B \vdash f\ x\ y \uparrow \cdots$

Greed and Failure

- Generate existential variable
- The greedy method [Cardelli '93]
- $f : \forall\alpha.\ \alpha \to \alpha \to \alpha$
  $x : A, y : B \vdash f\ x\ y \uparrow \cdots$
  - $A \leq B$ and $B \not\leq A$: $y \downarrow A$ ✗

Greed and Failure

- Generate existential variable
- The greedy method [Cardelli '93]
- $f : \forall\alpha.\ \alpha \to \alpha \to \alpha$
  $x : A, y : B \vdash f\ x\ y \uparrow \cdots$
  - $A \leq B$ and $B \not\leq A$: $\quad y \downarrow A$ ✗
  - $A \not\leq B$ and $B \leq A$: $\quad y \downarrow A$ ✓

Greed and Failure

- Generate existential variable
- The greedy method [Cardelli '93]
- $f : \forall\alpha.\ \alpha \to \alpha \to \alpha$
  $x : A, y : B \vdash f\ x\ y \uparrow \cdots$
  - $A \leq B$ and $B \not\leq A$: $\quad y \downarrow A$ ✗
  - $A \not\leq B$ and $B \leq A$: $\quad y \downarrow A$ ✓
  - $A \not\leq B$ and $B \not\leq A$: $\quad y \downarrow A$ ✗

Unionism

- The greedy method seems to fail…

Unionism

- The greedy method seems to fail…
- If only we could get an upper bound of A and B!

Unionism

- The greedy method seems to fail…
- If only we could get an upper bound of A and B!

  $$A \vee B$$

  Instantiating $\alpha = A \vee B$:

  $x : A, y : B \vdash f\ x\ y \uparrow \cdots$ ✓

Typechecker, Phone Home

- Let's not break the subformula property

Typechecker, Phone Home

- Let's not break the subformula property
- Instead, let the user write the union:
  $(*[\quad f : \forall\alpha.\, \forall\beta.\, \alpha \to \beta \to \alpha \vee \beta \quad]*)$

Typechecker, Phone Home

- Let's not break the subformula property
- Instead, let the user write the union:
  $(*[\quad f : \forall\alpha.\, \forall\beta.\, \alpha \to \beta \to \alpha \vee \beta \quad]*)$
  $x : A, y : B \vdash f\ x\ y \uparrow \cdots$
  $\alpha = A$
  $\beta = B$

Typechecker, Phone Home

- Let's not break the subformula property
- Instead, let the user write the union:
  $(*[\quad f : \forall\alpha.\, \forall\beta.\, \alpha \to \beta \to \alpha \vee \beta \quad]*)$
  $x : A, y : B \vdash f\ x\ y \uparrow \quad A \vee B \quad \checkmark$
  $\alpha = A$
  $\beta = B$