Untangling Typechecking of Intersections and Unions

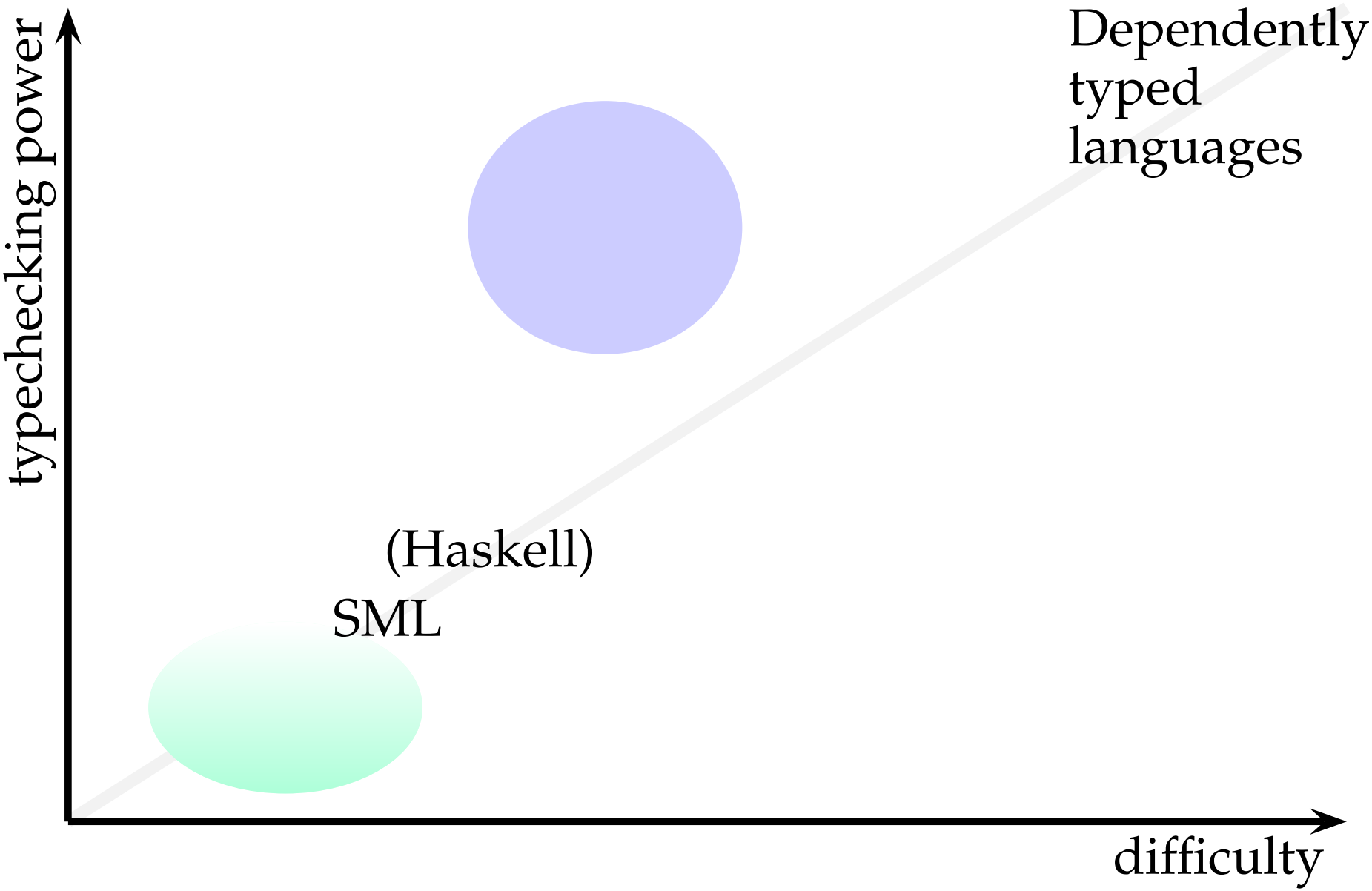
Joshua Dunfield
McGill University
Montréal, Canada

9 July 2010
ITRS ’10
Edinburgh

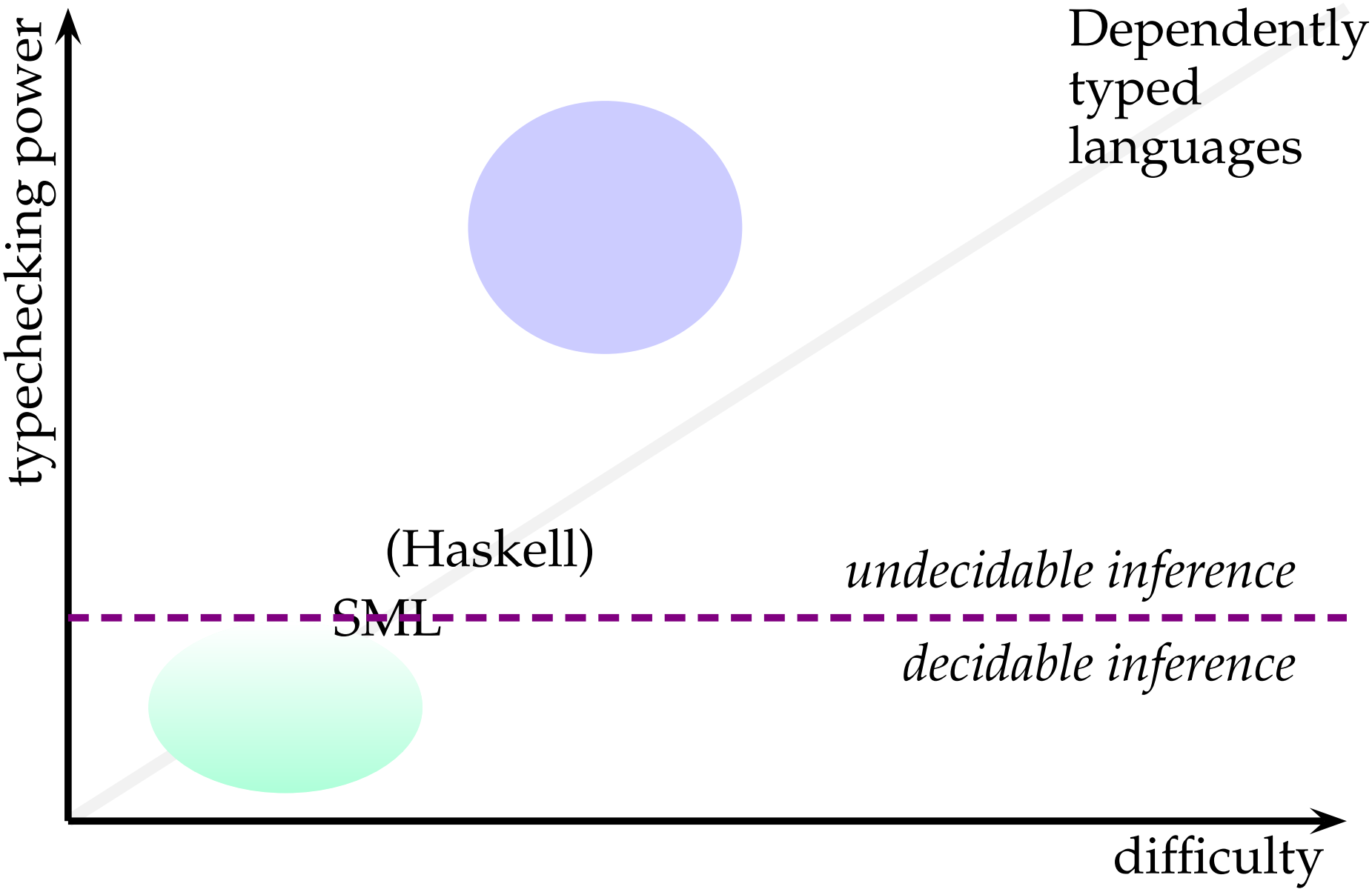
Outline

- ☞ **Background**
  - Bidirectional Typing
  - Intersections and Unions
  - Let-Normal Typechecking
  - Related Work

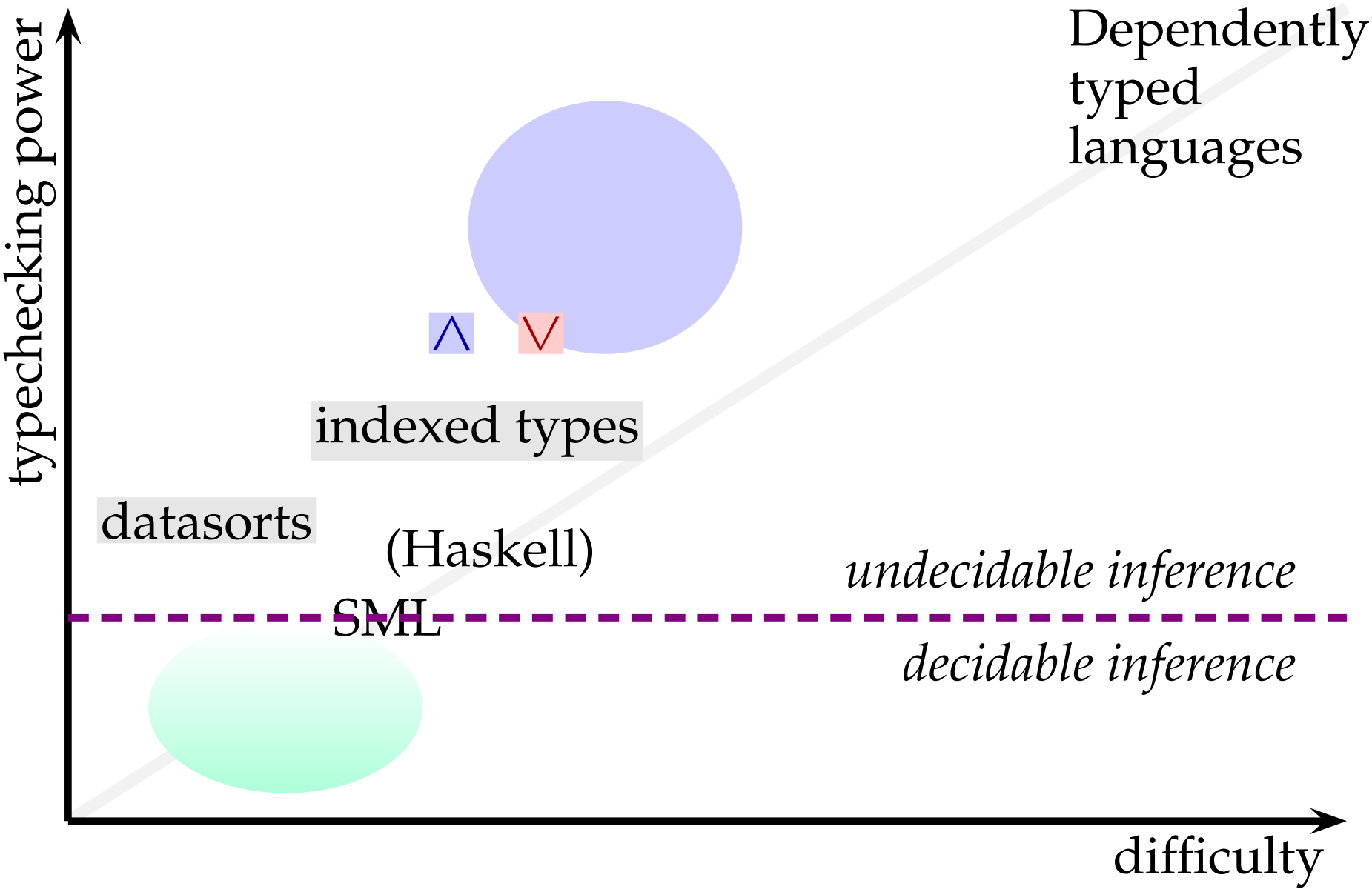
Motivation: Verifying Program Properties



Motivation: Verifying Program Properties



Motivation: Verifying Program Properties



Motivation

- Express atomic properties with datasort refinements, indexed types, ...
- **Combine** properties with intersections $\wedge$ and unions $\vee$
- Inference undecidable
  - Use **bidirectional** typechecking

Background

- Decidable, sound (under **call-by-value**) typechecking
- Typing **intersections**: (Davies & Pfenning '00)
  - Elim rules straightforward
  - Intro-rule **value** restriction
  - No distributivity
- Typing **unions**: (Dunfield & Pfenning '03, '04)
  - Intro rules straightforward
  - (No distributivity)
  - Elim-rule **evaluation context** restriction
    - This (ITRS) paper is about making typechecking with this restriction practical

Language

$$
\begin{array}{rrl}
\text{Types} & A, B, C ::= & 1 \mid A \to B \\
& & \mid A \wedge B \\
& & \mid A \vee B \mid \bot \\
\text{Terms} & e ::= & x \mid u \mid \lambda x.\, e \mid e_1\, e_2 \mid \mathsf{fix}\, u.\, e \\
\text{Values} & v ::= & x \mid () \mid \lambda x.\, e \\
\text{Eval. contexts} & \mathcal{E} ::= & [\,] \mid \mathcal{E}\ e \mid v\ \mathcal{E}
\end{array}
$$

$$
\begin{array}{rl}
\text{Small-step} & \mathcal{E}[(\lambda x.\, e)\, v] \mapsto \mathcal{E}[\,[v/x]e\,] \\
\text{rules} & \mathcal{E}[\mathsf{fix}\, u.\, e] \mapsto \mathcal{E}[[(\mathsf{fix}\, u.\, e)\ /\ u]\, e]
\end{array}
$$

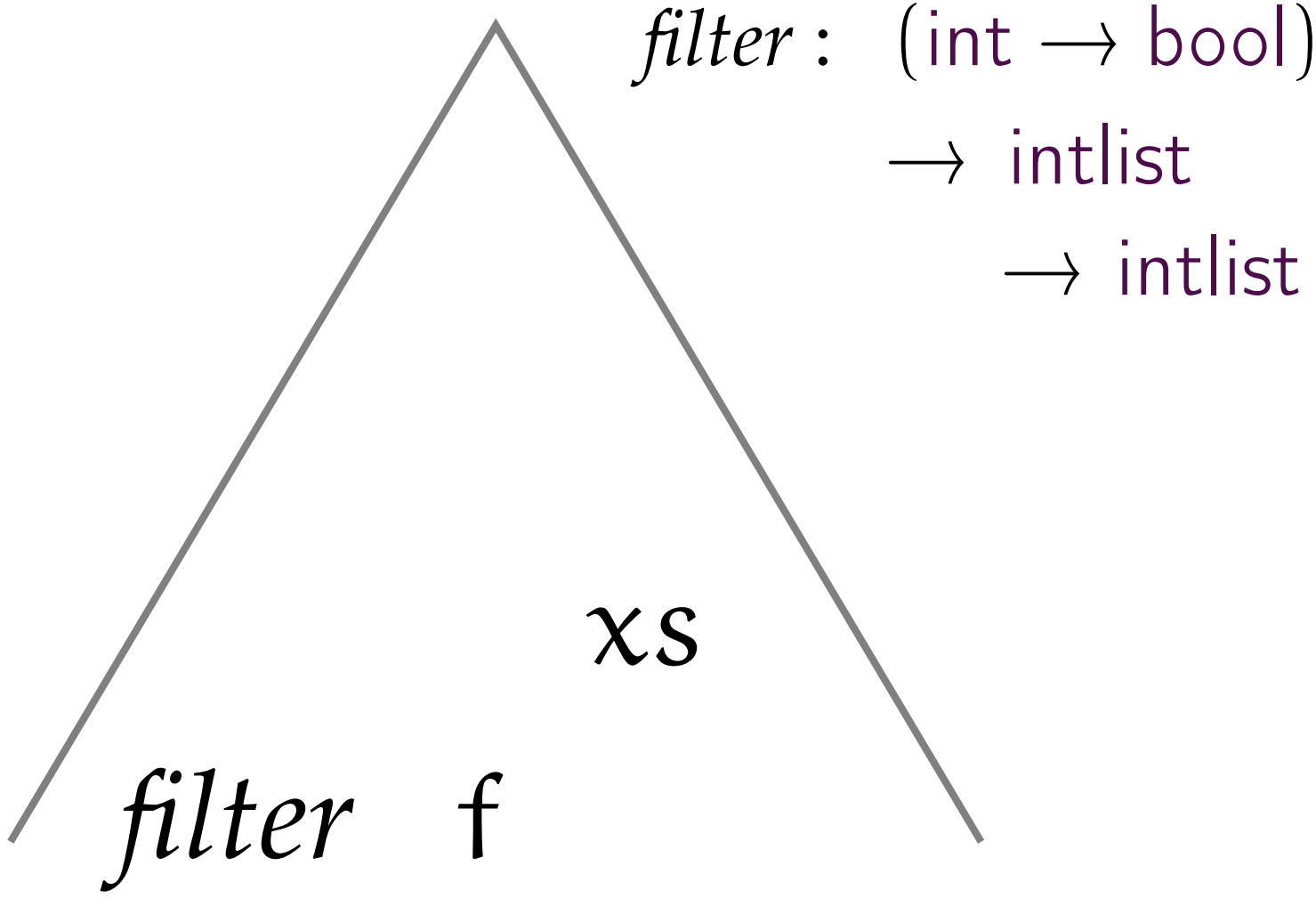
- Thesis adds $*$, datatypes $\delta(i)$, index quantifiers, ...
- Implementation adds SML pattern matching

- ✓ Background
- ☞ **Bidirectional Typing**
  - Intersections and Unions
  - Let-Normal Typechecking
  - Related Work

Bidirectional Typing

Synthesis $\Gamma^{\textbf{input}} \vdash e^{\textbf{input}} \Uparrow A^{\textbf{output}}$

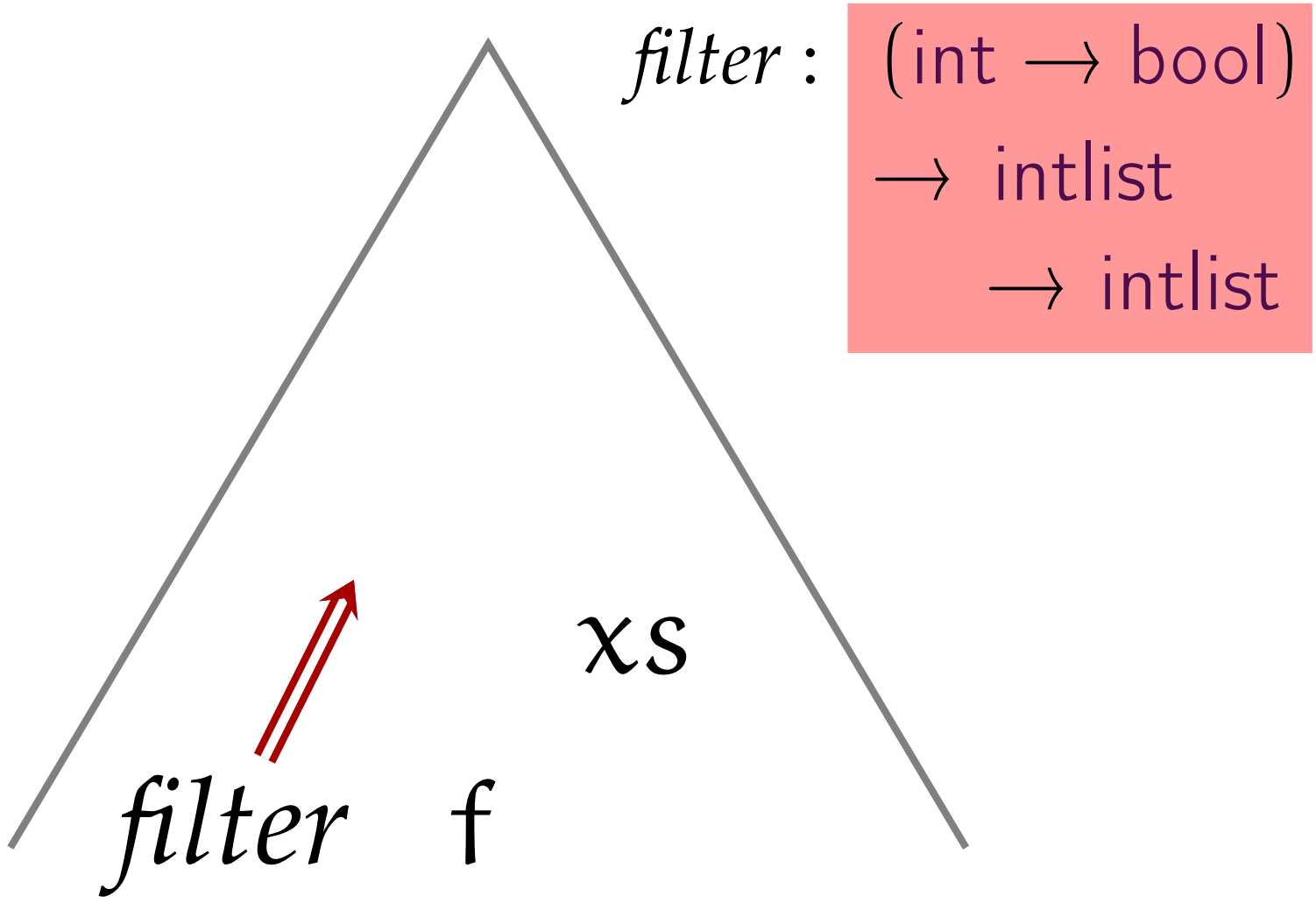
Checking $\Gamma^{\textbf{input}} \vdash e^{\textbf{input}} \Downarrow A^{\textbf{input}}$



Bidirectional Typing

Synthesis $\Gamma^{\textbf{input}} \vdash e^{\textbf{input}} \Uparrow A^{\textbf{output}}$

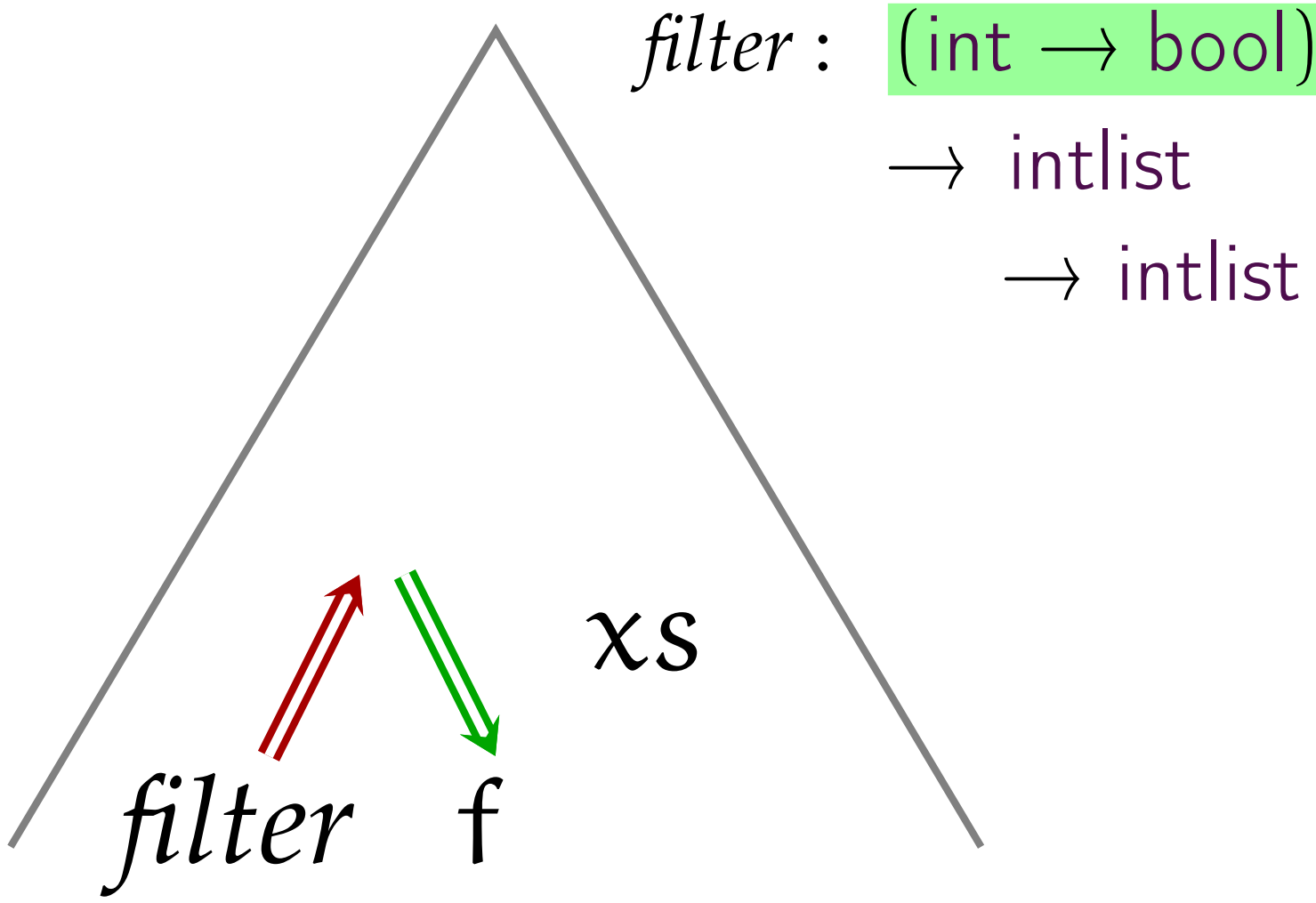
Checking $\Gamma^{\textbf{input}} \vdash e^{\textbf{input}} \Downarrow A^{\textbf{input}}$



Bidirectional Typing

Synthesis $\Gamma^{\textbf{input}} \vdash e^{\textbf{input}} \Uparrow A^{\textbf{output}}$

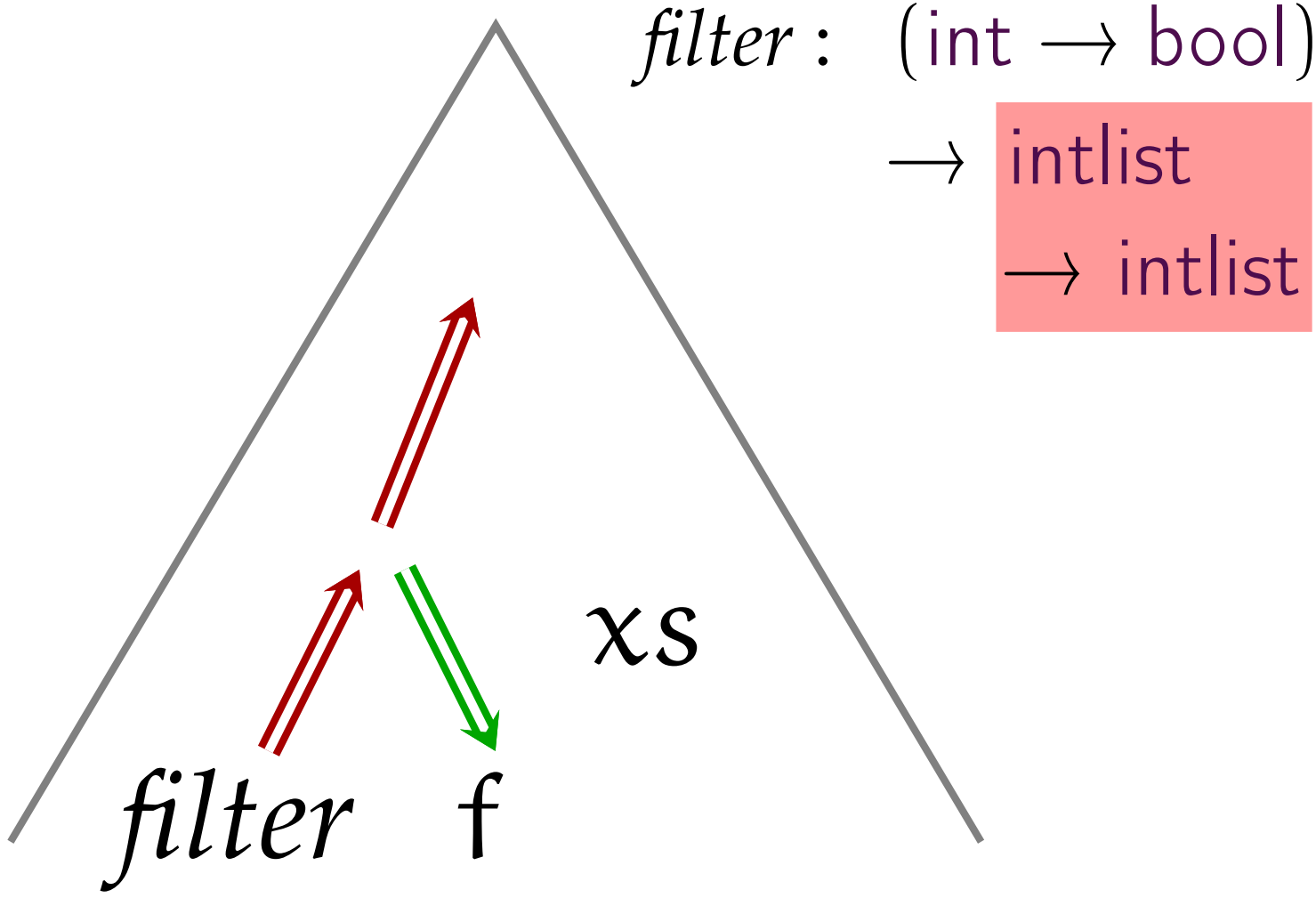
Checking $\Gamma^{\textbf{input}} \vdash e^{\textbf{input}} \Downarrow A^{\textbf{input}}$



Bidirectional Typing

Synthesis $\Gamma^{\textbf{input}} \vdash e^{\textbf{input}} \Uparrow A^{\textbf{output}}$

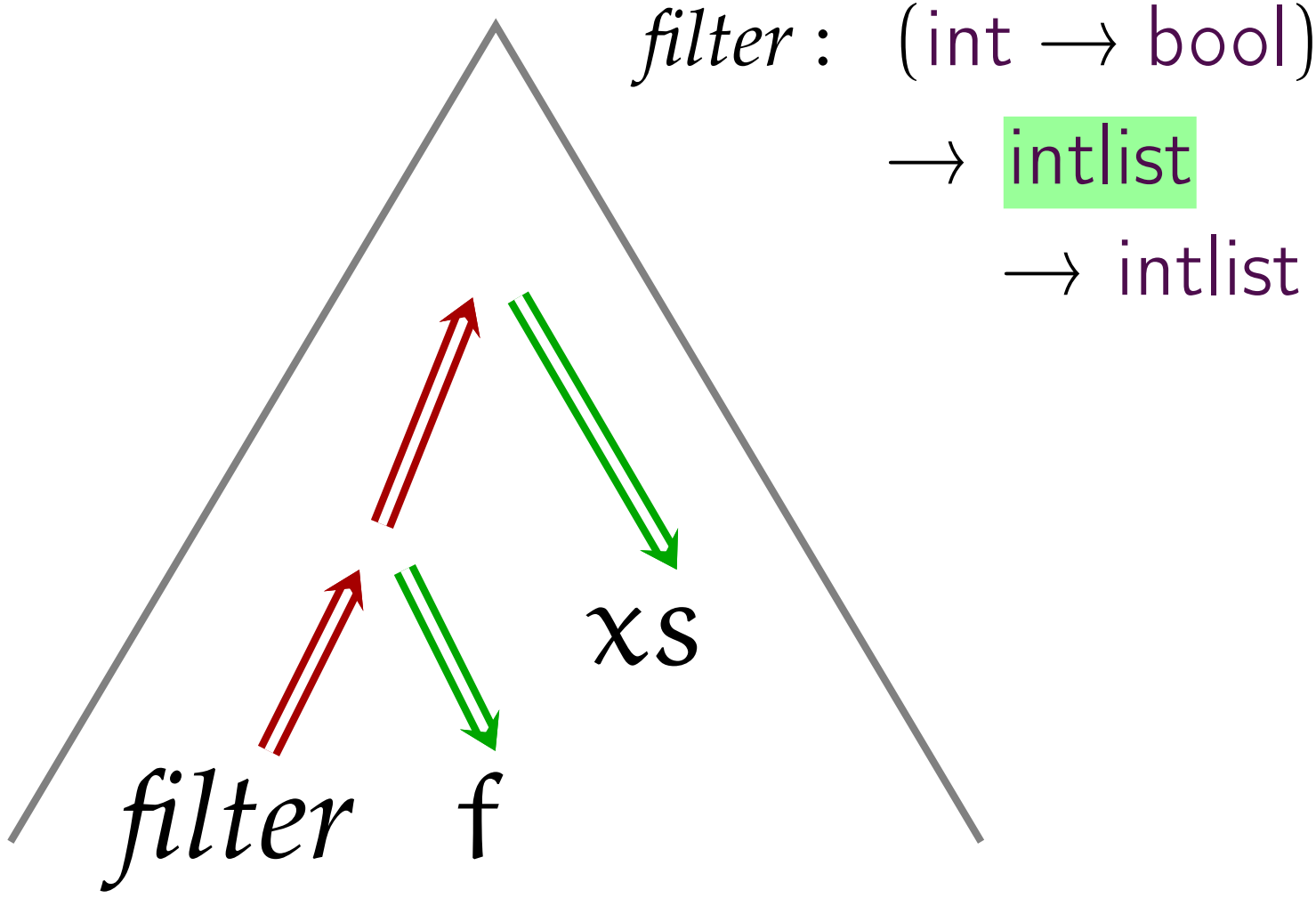
Checking $\Gamma^{\textbf{input}} \vdash e^{\textbf{input}} \Downarrow A^{\textbf{input}}$



Bidirectional Typing

Synthesis $\Gamma^{\textbf{input}} \vdash e^{\textbf{input}} \Uparrow A^{\textbf{output}}$

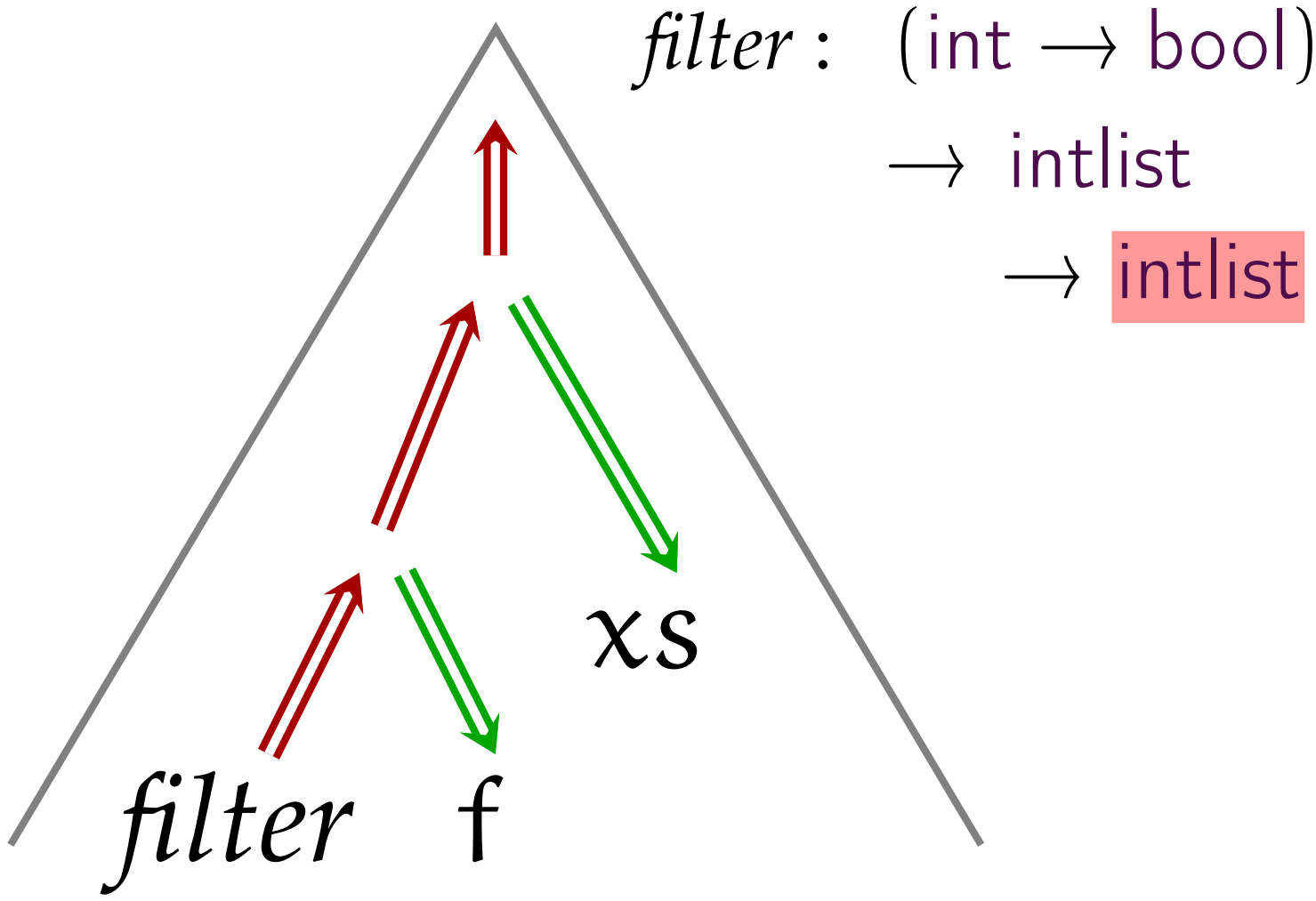
Checking $\Gamma^{\textbf{input}} \vdash e^{\textbf{input}} \Downarrow A^{\textbf{input}}$



Bidirectional Typing

Synthesis $\Gamma^{\textbf{input}} \vdash e^{\textbf{input}} \Uparrow A^{\textbf{output}}$

Checking $\Gamma^{\textbf{input}} \vdash e^{\textbf{input}} \Downarrow A^{\textbf{input}}$



Bidirectional Typing

- Using an assumption:

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

Bidirectional Typing

- Using an assumption:

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

Bidirectional Typing: $\rightarrow$

- 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: $\rightarrow$

- 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: $\rightarrow$

- 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: $\rightarrow$

- 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: $\rightarrow$

- 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: $\rightarrow$

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

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

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

- e_1 usually a variable or another application

Change of Direction

- In syntax-directed rules,
  intros checks, elims synthesize
- Other rules:
  - Checking a synthesizing form: subsumption

$$\frac{\Gamma \vdash e \Uparrow A' \qquad \Gamma \vdash A' \leq A}{\Gamma \vdash e \Downarrow A}\ \mathsf{sub}$$

    - Subtyping

$$\frac{}{1 \leq 1} \qquad \frac{B_1 \leq A_1 \qquad A_2 \leq B_2}{A_1 \to A_2 \leq B_1 \to B_2}$$

  - Synthesizing a checked term: annotation

$$\frac{\Gamma \vdash e \Downarrow A}{\Gamma \vdash (e : A) \Uparrow A}\ \mathsf{anno}$$

Synthesizing and Checking

- Synthesizing terms: x u $e_1\ e_2$ $(e : A)$ (elim)
- Checked terms: $\lambda x.\ e$ $\mathsf{fix}\ u.\ e$ (intro)
- Synthesizing positions: e_1 e_2
- Checked positions: e_1 e_2 $\lambda x.$ e $\mathsf{fix}\ u.$ e $($ e $: A)$
- Annotations needed only where a checked term is used in a synthesizing position, i.e. on the function part of a redex

  $$(\lambda x.\ e_1)\ e_2$$

  and on function declarations (fix is a redex)

- ✓ Background
- ✓ Bidirectional Typing
- ☞ **Intersections and Unions**
  - Let-Normal Typechecking
  - Related Work

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 unsound with mutable references [Davies & Pf. '00]

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

  - $g = (\lambda x.\ \mathsf{ref}\ 0) \Downarrow (1 \to \mathsf{int\ ref}) \wedge (1 \to \mathsf{even\ ref})$, *evenf* : $\mathsf{even} \to 1$
  - $\mathsf{let}\ r = (g : 1 \to (\mathsf{int\ ref} \wedge \mathsf{even\ ref}))\,()\ \mathsf{in}$

    $r := 5;$ at type $\mathsf{int\ ref}$

    evenf $!r$ at type $\mathsf{even\ ref}$: 

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** on $\mathcal{E}[e']$

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

- Example unsound (in cbv w/ effects) elim rule: multiple e's anywhere:

$$f : (A \to A \to C) \wedge (B \to B \to C),$$
$$rand : 1 \to (A \vee B) \qquad \vdash f\ (rand\,())\ (rand\,()) \Uparrow C$$

- Also unsound: one e' anywhere (could be under λ)

The Tridirectional Rule

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

- Assume data constructors $\texttt{None} : \mathsf{none}$ and $\texttt{Some}(n) : \mathsf{some}$
- $\mathit{map}\ f\ (\texttt{Some}(n)) = \texttt{Some}(f\ n)$ and $\mathit{map}\ f\ \texttt{None} = \texttt{None}$;
  $\mathit{map} : (\mathsf{int} \to \mathsf{int}) \to ((\mathsf{some} \to \mathsf{some}) \wedge (\mathsf{none} \to \mathsf{none}))$
- $\mathit{filter} : \mathsf{int} \to (\mathsf{some} \vee \mathsf{none})$
- $(\mathit{map}\ f)\ (\mathit{filter}\ n)$
- $(\mathit{map}\ f) \Uparrow (\mathsf{some} \to \mathsf{some}) \wedge (\mathsf{none} \to \mathsf{none})$;
  need an $\to$ to apply $\to$E (with argument *filter*), so...
- Need to case-split with $\vee$E before eliminating the $\wedge$,
  but $(\mathit{filter}\ n) : \mathsf{some} \vee \mathsf{none}$ not in eval. position.

$$\frac{\mathit{map}\ f \Uparrow \qquad x : (\mathsf{s} \to \mathsf{s}) \wedge (\mathsf{n} \to \mathsf{n}) \vdash x\ (\mathit{filter}\ n) \Downarrow \mathsf{some} \vee \mathsf{none}}{(\mathit{map}\ f)\ (\mathit{filter}\ n) \Downarrow \mathsf{some} \vee \mathsf{none}}\ \text{direct}$$

Linear Variables and Left Rules

- Distinguish $\vee$E/direct variables from ordinary variables
- Add (unordered) linear context $\Delta ::= \cdot \mid \Delta, \overline{\mathbf{x}}{:}A$
- Allows proof of decidability
- $\Gamma; \Delta \vdash \ldots$ sound and complete wrt $\Gamma' \vdash \ldots$

$$\frac{\Gamma; \Delta, \overline{\mathbf{x}}{:}A \vdash e \Downarrow C \qquad \Gamma; \Delta, \overline{\mathbf{x}}{:}B \vdash e \Downarrow C}{\Gamma; \Delta, \overline{\mathbf{x}}{:}A \vee B \vdash e \Downarrow C} \vee\mathbb{L}$$

$$\frac{\Gamma; \Delta_1 \vdash e' \Uparrow A' \qquad \Gamma; \Delta_2, \overline{\mathbf{x}}{:}A' \vdash \mathcal{E}[\overline{\mathbf{x}}] \Downarrow C}{\Gamma; \Delta_1, \Delta_2 \vdash \mathcal{E}[e'] \Downarrow C} \text{direct}\mathbb{L}$$

$$\frac{\Gamma; \Delta_1 \vdash e_1 \Uparrow A{\to}B \qquad \Gamma; \Delta_2 \vdash e_2 \Downarrow A}{\Gamma; \Delta_1, \Delta_2 \vdash e_1 e_2 \Uparrow B} {\to}\text{E} \qquad \text{etc.}$$

Left Rule for the Empty Type

- $\vee\mathbb{L}$ (binary union) had 2 premises;
  $\bot\mathbb{L}$ (union of nothing) has 0:

$$\frac{}{\Gamma;\Delta,\overline{\mathbf{x}}{:}\bot \vdash e \Downarrow C}\ \bot\mathbb{L}$$

The Story So Far

- To typecheck* intersections and unions:
  - Bidirectional typechecking:

$$\ldots \vdash e : A \implies \begin{array}{ll} \ldots \vdash e \Uparrow A & e \text{ synthesizes } A \\ \ldots \vdash e \Downarrow A & e \text{ checks against } A \end{array}$$

  - Tridirectional typechecking uses **evaluation contexts**:

$$\mathsf{direct}\mathbb{L} \quad \vee\mathbb{L} \quad \bot\mathbb{L}$$

- Implementation challenges

* in a **call-by-value** language with effects

The Story So Far

- To typecheck* intersections and unions:
  - Bidirectional typechecking:

$$\ldots \vdash e : A \implies \begin{array}{ll} \ldots \vdash e \Uparrow A & e \text{ synthesizes } A \\ \ldots \vdash e \Downarrow A & e \text{ checks against } A \end{array}$$

  - Tridirectional typechecking uses **evaluation contexts**:

$$\mathsf{direct}\mathbb{L} \quad \vee\mathbb{L} \quad \bot\mathbb{L}$$

- Implementation challenges

* in a **call-by-value** language with effects

First Implementation Challenge

- directℒ has a conclusion of this form:

$$\ldots \vdash \mathcal{E}[e'] \ldots$$

  So we need to choose e' (equivalently, choose $\mathcal{E}$).

- Example: $f\ (g\ x)$

$\mathcal{E}_1 = []\ (g\ x)$ $\quad f\ (g\ x) = \mathcal{E}_1[f]$

$\mathcal{E}_2 = f\ ([]\ x)$ $\quad f\ (g\ x) = \mathcal{E}_2[g]$

$\mathcal{E}_3 = f\ (g\ [])$ $\quad f\ (g\ x) = \mathcal{E}_3[x]$

$\mathcal{E}_4 = f\ []$ $\quad f\ (g\ x) = \mathcal{E}_4[g\ x]$

$\mathcal{E}_5 = []$ $\quad f\ (g\ x) = \mathcal{E}_5[f\ (g\ x)]$

- Possible sequences of evaluation contexts:
  $\langle\rangle, \langle\mathcal{E}_1\rangle, \langle\mathcal{E}_2\rangle, \ldots, \langle\mathcal{E}_1;\mathcal{E}_2\rangle, \langle\mathcal{E}_1;\mathcal{E}_3\rangle, \langle\mathcal{E}_1;\mathcal{E}_5\rangle, \langle\mathcal{E}_1;\mathcal{E}_2;\mathcal{E}_3\rangle, \ldots$

First Implementation Challenge

- directL has a conclusion of this form:

$$\ldots \vdash \mathcal{E}[e'] \ldots$$

  So we need to choose e' (equivalently, choose $\mathcal{E}$).

- Is all this flexibility necessary?

> - **Conjecture**: The correspondence between tri-directional checking and bi-directional checking of let-normal form is exact if we add the left rules to the typing judgment.
>
> —Frank Pfenning, ITRS '02 invited talk

- The answer is **yes** (with some peculiar definitions!)

Other Implementation Challenges

- Exponential typechecking time **in practice** (PLPV '07)
  - Future work…

- ✓ Background
- ✓ Bidirectional Typing
- ✓ Intersections and Unions
- ☞ **Let-Normal Typechecking**
- Related Work

Restricting Tridirectionality

- Rules like direct$\mathbb{L}$ are impractical:
  Must guess which eval. context

$$\frac{\Gamma;\Delta_1 \vdash e' \Uparrow A \qquad \Gamma;\Delta_2, \overline{x}{:}A \vdash \mathcal{E}[\overline{x}] \Downarrow C}{\Gamma;\Delta_1,\Delta_2 \vdash \mathcal{E}[e'] \Downarrow C}\ \text{direct}\mathbb{L}$$

- Example: $f\ (g\ x)$

$\mathcal{E}_1 = []\ (g\ x)$ $\quad f\ (g\ x) = \mathcal{E}_1[f]$

$\mathcal{E}_2 = f\ ([]\ x)$ $\quad f\ (g\ x) = \mathcal{E}_2[g]$

$\mathcal{E}_3 = f\ (g\ [])$ $\quad f\ (g\ x) = \mathcal{E}_3[x]$

$\mathcal{E}_4 = f\ []$ $\quad f\ (g\ x) = \mathcal{E}_4[g\ x]$

$\mathcal{E}_5 = []$ $\quad f\ (g\ x) = \mathcal{E}_5[f\ (g\ x)]$

- Let-normal system: analyze (bind to linear variable)
  f, g, x, $g\ x$, finally $f(g\ x)$

What Is (Traditional) Let-Normal Form?

- 1. Explicitly sequence computation
- 2. Name intermediate results
- ~~3. Name computations~~ —in CPS, not let-normal
- Example:

$$f\ x\ (g\ y) \quad \hookrightarrow \quad \mathsf{let}\ a = f\ x\ \mathsf{in}\ \mathsf{let}\ b = g\ y\ \mathsf{in}\ a\ b$$

- A-normal form [Moggi88]
- Used in compilation, program analysis

My Let-Normal Form

- Let-bindings are syntactic markers
  to guide typechecking:
  - Let-bind every term that could be
    the e' in the $\mathcal{E}[e']$ in $\mathsf{direct}\mathbb{L}$
  - Drop $\mathsf{direct}\mathbb{L}$ and have a rule let instead
- **Not** computational…

Let-Normal Typechecking

- Bind every synthesizing subterm, in order:

$$f\ (g\ x) \quad \hookrightarrow \quad \text{let } \overline{\mathbf{f}} = f \text{ in let } \overline{\mathbf{g}} = g \text{ in let } \overline{\mathbf{x}} = x \text{ in}$$
$$\text{let } \overline{\mathbf{y}} = \overline{\mathbf{g}}\ \overline{\mathbf{x}} \text{ in}$$
$$\text{let } \overline{\mathbf{z}} = \overline{\mathbf{f}}\ \overline{\mathbf{y}} \text{ in } \overline{\mathbf{z}}$$

- Replace $\mathsf{direct}\mathbb{L}$ with let:

$$\frac{\Gamma;\Delta_1 \vdash e' \Uparrow A \qquad \Gamma;\Delta_2, \overline{\mathbf{x}}{:}A \vdash \mathcal{E}[\overline{\mathbf{x}}] \Downarrow C}{\Gamma;\Delta_1,\Delta_2 \vdash \mathcal{E}[e'] \Downarrow C}\ \mathsf{direct}\mathbb{L}$$

$$\frac{\Gamma;\Delta_1 \vdash e' \Uparrow A \qquad \Gamma;\Delta_2, \overline{\mathbf{x}}{:}A \vdash e_2 \Downarrow C}{\Gamma;\Delta_1,\Delta_2 \vdash \text{let } \overline{\mathbf{x}} = e' \text{ in } e_2 \Downarrow C}\ \mathsf{let}$$

- Translation + let $\simeq$ **Restricted strategy for** $\mathsf{direct}\mathbb{L}$

The Easy Part

$\boxed{e \hookrightarrow L + e'}$ "e translates to bindings L with result e'"

$$\frac{}{x \hookrightarrow (\overline{\mathbf{x}} = x) + \overline{\mathbf{x}}} \qquad \frac{e \hookrightarrow L + e'}{\lambda x.\, e \hookrightarrow \cdot + \lambda x.\, (L \text{ in } e')}$$

$$\frac{}{u \hookrightarrow (\overline{\mathbf{x}} = u) + \overline{\mathbf{x}}} \qquad \frac{e \hookrightarrow L + e'}{\text{fix } u.\, e \hookrightarrow \cdot + \text{fix } u.\, (L \text{ in } e')}$$

- Can read $(\overline{\mathbf{x}} = x) + \overline{\mathbf{x}}$ as $\text{let } \overline{\mathbf{x}} = x \text{ in } \overline{\mathbf{x}}$
- Can read $\cdot + \lambda x.\, (L \text{ in } e')$ as $\lambda x.\, \text{let} \cdots L \cdots \text{in } e'$

A Question of Order

- "Bind every synthesizing subterm, in order"
- Suppose $\mathit{loop} : A \to \bot$, and $\text{☠} = () \; ()$
- $(\mathsf{fix}\; u.\; \text{☠})\; (\mathit{loop}\; y)$

  $? \hookrightarrow ?$ $\quad \mathsf{let}\; \overline{\mathbf{x}} = \mathit{loop}\; \mathsf{in}\; \mathsf{let}\; \overline{\mathbf{y}} = y\; \mathsf{in}\; \mathsf{let}\; \overline{\mathbf{a}} = \overline{\mathbf{x}}\; \overline{\mathbf{y}}\; \mathsf{in}\; (\mathsf{fix}\; u.\; \text{☠})\; \overline{\mathbf{a}}$

$$\frac{}{\Gamma; \overline{\mathbf{a}}{:}\bot \vdash (\mathsf{fix}\; u.\; \text{☠})\; \overline{\mathbf{a}} \Downarrow C}\; \bot\mathbb{L}$$

  $(\mathsf{fix}\; u.\; \text{☠})\; (\mathit{loop}\; y) \mapsto \cdots \mapsto \text{☠}(\mathit{loop}\; y) \not\mapsto$

- Unlike the tridirectional system,
  this $? \hookrightarrow ?$ translation "jumps" $\mathsf{fix}\; u.\; \text{☠}$ to bind $\overline{\mathbf{a}}$

- The tridirectional system can turn a **synthesizing subterm** into a value by binding the subterm to a linear variable but can't turn $\mathsf{fix}\ u.\ e$ into a value
- So we need to translate $e_1\ e_2$ differently, depending on whether e_1 is a value, could **become** some $\overline{\mathbf{x}}$ (which is a value), or could never be a value

e_1 **is** a value	e_2 in eval. position	OK to jump e_1
e_1 **eventually** a value	after $\mathsf{direct}\mathbb{L}$ ing, e_2 in eval. position	OK to jump e_1
e_1 **never** a value	e_2 never in eval. position	Don't jump e_1

- The tridirectional system can turn a **synthesizing subterm** into a value by binding the subterm to a linear variable but can't turn $\mathsf{fix}\ u.\ e$ into a value
- So we need to translate $e_1\ e_2$ differently, depending on whether e_1 is a value, could **become** some $\overline{\mathbf{x}}$ (which is a value), or could never be a value

e_1 **is** a value	e_2 in eval. position	OK to jump e_1
"pre-value" $\check{e}_1$		
e_1 **eventually** a value	after $\mathsf{direct}\mathbb{L}$ ing, e_2 in eval. position	OK to jump e_1
e_1 **never** a value	e_2 never in eval. position	Don't jump e_1
"anti-value" $\hat{e}_1$		

Most of the Translation

$\boxed{e \hookrightarrow L + e'}$ "e translates to bindings L with result e'"

$$\frac{}{x \hookrightarrow (\overline{\mathbf{x}} = x) + \overline{\mathbf{x}}} \qquad \frac{e \hookrightarrow L + e'}{\lambda x.\, e \hookrightarrow \cdot + \lambda x.\, (L \text{ in } e')}$$

$$\frac{}{u \hookrightarrow (\overline{\mathbf{x}} = u) + \overline{\mathbf{x}}} \qquad \frac{e \hookrightarrow L + e'}{\text{fix } u.\, e \hookrightarrow \cdot + \text{fix } u.\, (L \text{ in } e')}$$

$$\frac{\hat{e}_1 \hookrightarrow L_1 + e_1' \qquad e_2 \hookrightarrow L_2 + e_2'}{\hat{e}_1 e_2 \hookrightarrow L_1, \overline{\mathbf{x}} = e_1' (L_2 \text{ in } e_2') + \overline{\mathbf{x}}} \qquad \frac{\check{e}_1 \hookrightarrow L_1 + e_1' \qquad e_2 \hookrightarrow L_2 + e_2'}{\check{e}_1 e_2 \hookrightarrow L_1, L_2, \overline{\mathbf{x}} = e_1' e_2' + \overline{\mathbf{x}}}$$

Most of the Translation

$\boxed{e \hookrightarrow L + e'}$ "e translates to bindings L with result e'"

$$\frac{}{x \hookrightarrow (\overline{x} = x) + \overline{x}} \qquad \frac{e \hookrightarrow L + e'}{\lambda x.\, e \hookrightarrow \cdot + \lambda x.\, (L \text{ in } e')}$$

$$\frac{}{u \hookrightarrow (\overline{x} = u) + \overline{x}} \qquad \frac{e \hookrightarrow L + e'}{\text{fix } u.\, e \hookrightarrow \cdot + \text{fix } u.\, (L \text{ in } e')}$$

$$\frac{\hat{e}_1 \hookrightarrow L_1 + e_1' \qquad e_2 \hookrightarrow L_2 + e_2'}{\hat{e}_1 e_2 \hookrightarrow L_1, \overline{x} = e_1'(L_2 \text{ in } e_2') + \overline{x}} \qquad \frac{\check{e}_1 \hookrightarrow L_1 + e_1' \qquad e_2 \hookrightarrow L_2 + e_2'}{\check{e}_1 e_2 \hookrightarrow L_1, L_2, \overline{x} = e_1' e_2' + \overline{x}}$$

$$\frac{(\text{fix } u.\, \text{☠}) \hookrightarrow \cdot + (\text{fix } u.\, \text{☠}) \qquad (\mathit{loop}\ y) \hookrightarrow (\overline{x} = \mathit{loop}, \overline{y} = y, \overline{a} = \overline{x}\,\overline{y}) + \overline{a}}{(\text{fix } u.\, \text{☠})(\mathit{loop}\ y) \hookrightarrow \cdot, \overline{z} = (\text{fix } u.\, \text{☠})(\text{let } \overline{x} = \mathit{loop} \text{ in let } \overline{y} = y \text{ in let } \overline{a} = \overline{x}\ \overline{y} \text{ in } \overline{a}) + \overline{z}}$$

The Last Piece

$\boxed{e \hookrightarrow L + e'}$ "e translates to bindings L with result e'"

$$\frac{}{x \hookrightarrow (\overline{\mathbf{x}} = x) + \overline{\mathbf{x}}} \qquad \frac{e \hookrightarrow L + e'}{\lambda x.\, e \hookrightarrow \cdot + \lambda x.\, (L \text{ in } e')}$$

$$\frac{}{u \hookrightarrow (\overline{\mathbf{x}} = u) + \overline{\mathbf{x}}} \qquad \frac{e \hookrightarrow L + e'}{\text{fix } u.\, e \hookrightarrow \cdot + \text{fix } u.\, (L \text{ in } e')}$$

$$\frac{\hat{e}_1 \hookrightarrow L_1 + e_1' \qquad e_2 \hookrightarrow L_2 + e_2'}{\hat{e}_1 e_2 \hookrightarrow L_1, \overline{\mathbf{x}} = e_1' (L_2 \text{ in } e_2') + \overline{\mathbf{x}}} \qquad \frac{\check{e}_1 \hookrightarrow L_1 + e_1' \qquad e_2 \hookrightarrow L_2 + e_2'}{\check{e}_1 e_2 \hookrightarrow L_1, L_2, \overline{\mathbf{x}} = e_1' e_2' + \overline{\mathbf{x}}}$$

$$\frac{e \hookrightarrow L + e' \qquad e \text{ not a value}}{(e : As) \hookrightarrow L, \overline{\mathbf{x}} = (e' : As) + \overline{\mathbf{x}}} \qquad \frac{v \hookrightarrow L + e'}{(v : As) \hookrightarrow L, {\sim}\overline{\mathbf{x}} = (e' : As) + \overline{\mathbf{x}}}$$

Let-Normal Soundness

Original program e $\quad\hookrightarrow\quad$ 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 of applying direct$\mathbb{L}$; any restricted strategy should be **sound**...
- Proof sketch:
  - Define an inverse translation $\hookleftarrow$
  - Show that if the translation produces let $\overline{\mathbf{x}} = e_1$ in e_2, in $\hookleftarrow(e_2)$, the variable $\overline{\mathbf{x}}$ is in eval. position, so in $[\hookleftarrow(e_1)/\overline{\mathbf{x}}]\ (\hookleftarrow(e_2))$, the term e_1 is in eval. position
  - Apply direct$\mathbb{L}$ in place of let

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 of applying direct$\mathbb{L}$; is this restricted strategy **complete**?

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 of applying direct$\mathbb{L}$; is this restricted strategy **complete**?
- Yes! Each step preserves typing:
  - mark e with lets wherever direct$\mathbb{L}$ is used;
  - add lets wherever direct$\mathbb{L}$ could have been used;
  - move lets into "canonical positions"
- Repeat until the term has the same lets as e': it **is** e'

Two-Fisted Tridirectionality

$x : (A_1 \to B) \wedge (A_2 \to B), y : A_1 \vee A_2; \cdot \vdash x\ y \Downarrow B.$

$x : (A_1 \to B) \wedge (A_2 \to B), \ldots; \overline{y}{:}A_1 \vdash x\ \overline{y} \Downarrow B$

$x : (A_1 \to B) \wedge (A_2 \to B), \ldots; \overline{y}{:}A_2 \vdash x\ \overline{y} \Downarrow B$

$$\dfrac{y \Uparrow A_1 \vee A_2 \qquad \dfrac{\dfrac{x \Uparrow A_1 \to B \qquad \vdots}{\ldots; \overline{y}{:}A_1 \vdash x\ \overline{y} \Downarrow B} \to E \qquad \dfrac{x \Uparrow A_2 \to B \qquad \vdots}{\ldots; \overline{y}{:}A_2 \vdash x\ \overline{y} \Downarrow B} \to E}{\ldots; \overline{y} : A_1 \vee A_2 \vdash x\ \overline{y} \Downarrow B} \vee \mathbb{L}}{x : (A_1 \to B) \wedge (A_2 \to B), y : A_1 \vee A_2; \cdot \vdash x\ y \Downarrow B} \text{direct}\mathbb{L}$$

- In the tridirectional system,
  we get to synthesize a type for x **twice**,
  once in each subderivation of $\vee\mathbb{L}$

Let-Normal Constraint

$$\dfrac{\dots;\cdot \vdash x \Uparrow \bigwedge(A_2 \to B) \qquad \dots;\overline{x}{:}\bigwedge(A_2\to B) \vdash (\mathsf{let}\ \overline{y} = y\ \mathsf{in}\ \mathsf{let}\ \overline{z} = \overline{x}\ \overline{y}\ \mathsf{in}\ \overline{z}) \Downarrow B}{\mathsf{let}}$$

$$\dfrac{\overline{y}{:}A_1 \vdash \dots \qquad \overline{y}{:}A_2 \vdash \dots}{\vdots}\ \bigvee\mathbb{L}$$

$$(A_1 \to B)$$

$$(A_1 \to B)$$

$$(A_1 \to B)$$

$$x : \bigwedge(A_2 \to B), y : A_1 \vee A_2; \cdot \vdash \quad \mathsf{let}\ \overline{x} = x\ \mathsf{in}\ \mathsf{let}\ \overline{y} = y\ \mathsf{in}\ \mathsf{let}\ \overline{z} = \overline{x}\ \overline{y}\ \mathsf{in}\ \overline{z} \quad \Downarrow B$$

- In the let-normal system,
  we get **one** chance to synthesize a type for x,
  outside the $\bigvee\mathbb{L}$ on $\overline{y}$

Principal Types

- **Traditional:** A type τ is the principal type of e if τ represents all types σ such that $e : \sigma$.
  - Hindley-Milner: σs polymorphic instances of τ
  - With subtyping: σs are supertypes of τ

Principal Types

- **Traditional:** A type τ is the principal type of e if τ represents all types σ such that $e : \sigma$.
  - Hindley-Milner: σs polymorphic instances of τ
  - With subtyping: σs are supertypes of τ

- **Bidirectionally**: Principal **synthesis**: e principally synthesizes A if $e \Uparrow A$ and, for all B such that $e \Uparrow B$, can derive $e \Uparrow B$ from $e \Uparrow A$
  - Example: if $x : (C_1 \to C_2) \wedge (D_1 \to D_2)$ is in Γ then x principally synthesizes $x : (C_1 \to C_2) \wedge (D_1 \to D_2)$

$$\frac{x \Uparrow (C_1 \to C_2) \wedge (D_1 \to D_2)}{x \Uparrow C_1 \to C_2} \wedge E_1 \qquad \frac{x \Uparrow (C_1 \to C_2) \wedge (D_1 \to D_2)}{x \Uparrow D_1 \to D_2} \wedge E_2$$

When Does Principal Synthesis Hold?

- e principally synthesizes A if $e \Uparrow A$ and,
  for all B such that $e \Uparrow B$, can derive $e \Uparrow B$ from $e \Uparrow A$
- For **variables** (the principal type is $\Gamma(x)$ or $\Delta(\overline{\mathbf{x}})$)
- Not for **applications**
  $f : (A \to B_1) \wedge (A \to B_2), y : A; \cdot \vdash f\ y \Uparrow B_1$
  $f : (A \to B_1) \wedge (A \to B_2), y : A; \cdot \vdash f\ y \Uparrow B_2$
  but $f\ y \not\Uparrow B_1 \wedge B_2$ (we don't have distributivity)
  nor $f\ y \not\Downarrow B_1 \wedge B_2$ (value restriction in $\wedge$I)

No principal synthesis for $(v : As)$

- Because of contextual typing annotations (omitted in this talk), some terms of the form $(v : As)$ do **not** enjoy principal synthesis
- Yet $(v : As)$ is a value and a synthesizing form...

A Compromise

- Bind $(v : As)$ to a "slack variable" $\sim\overline{\mathbf{x}} = (v' : As)$, which acts like a variable in the tridirectional system

$$\frac{\Gamma; \Delta, \sim\overline{\mathbf{x}} = v \vdash e_2 \Downarrow C}{\Gamma; \Delta \vdash \mathbf{let}\ \sim\overline{\mathbf{x}} = v\ \mathbf{in}\ e_2 \Downarrow C}\ \mathsf{let}\sim$$

$$\frac{\Gamma; \Delta_1 \vdash v \Uparrow A \qquad \Gamma; \Delta_2, \overline{\mathbf{x}}{:}A \vdash e \Downarrow C}{\Gamma; \Delta_1, \Delta_2, \sim\overline{\mathbf{x}} = v \vdash e \Downarrow C}\ \sim\mathsf{var}$$

The Full Translation

$\boxed{e \hookrightarrow \mathrm{L} + e'}$ "e translates to bindings L with result e'"

$$\frac{}{x \hookrightarrow (\overline{\mathbf{x}} = x) + \overline{\mathbf{x}}} \qquad \frac{e \hookrightarrow \mathrm{L} + e'}{\lambda x.\, e \hookrightarrow \cdot + \lambda x.\, (\mathrm{L} \text{ in } e')}$$

$$\frac{}{u \hookrightarrow (\overline{\mathbf{x}} = u) + \overline{\mathbf{x}}} \qquad \frac{e \hookrightarrow \mathrm{L} + e'}{\mathrm{fix}\, u.\, e \hookrightarrow \cdot + \mathrm{fix}\, u.\, (\mathrm{L} \text{ in } e')}$$

$$\frac{\hat{e}_1 \hookrightarrow \mathrm{L}_1 + e_1' \qquad e_2 \hookrightarrow \mathrm{L}_2 + e_2'}{\hat{e}_1 e_2 \hookrightarrow \mathrm{L}_1, \overline{\mathbf{x}} = e_1' (\mathrm{L}_2 \text{ in } e_2') + \overline{\mathbf{x}}} \qquad \frac{\check{e}_1 \hookrightarrow \mathrm{L}_1 + e_1' \qquad e_2 \hookrightarrow \mathrm{L}_2 + e_2'}{\check{e}_1 e_2 \hookrightarrow \mathrm{L}_1, \mathrm{L}_2, \overline{\mathbf{x}} = e_1' e_2' + \overline{\mathbf{x}}}$$

$$\frac{e \hookrightarrow \mathrm{L} + e' \qquad e \text{ not a value}}{(e : As) \hookrightarrow \mathrm{L}, \overline{\mathbf{x}} = (e' : As) + \overline{\mathbf{x}}} \qquad \frac{v \hookrightarrow \mathrm{L} + e'}{(v : As) \hookrightarrow \mathrm{L}, {\sim}\overline{\mathbf{x}} = (e' : As) + \overline{\mathbf{x}}}$$

- ✓ Background
- ✓ Bidirectional Typing
- ✓ Intersections and Unions
- ✓ Let-Normal Typechecking
- ☞ **Related Work**

Related Work: Xi's DML

- [Xi & Pfenning '99]: Bidirectional system with $\mathfrak{i}$, indexed types with $\forall/\exists$; used let-normal form (to handle $\exists$), but typechecking was incomplete:

$$\mathit{map}\ (\lambda x.\, e) \quad \hookrightarrow_{\mathrm{Xi}} \quad \mathsf{let}\ \overline{\mathbf{x}} = \mathit{map}\ \mathsf{in}$$
$$\mathsf{let}\ \overline{\mathbf{y}} = \lambda x.\, e\ \mathsf{in}$$
$$\ldots$$

Related Work: Xi's DML

- [Xi & Pfenning '99]: Bidirectional system with $\mathfrak{i}$, indexed types with $\forall/\exists$; used let-normal form (to handle $\exists$), but typechecking was incomplete:

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

  - $\lambda x.\, e$ does not synthesize

Related Work: Datasorts, $\wedge$, $\vee$

- [Davies '97/'05, Davies & Pf. '00]: Bidirectional system with δ, $\wedge$
- [Xi & Pfenning '99]: Bidirectional system with $\mathfrak{i}$, $\forall$, $\exists$
- [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
- [Pierce & Turner '00]: Bidirectional typechecking

Summary

- Tridirectional typing for decidable typechecking with intersections and unions (Dunfield & Pf. '03, '04)

Summary

- Tridirectional typing for decidable typechecking with intersections and unions (Dunfield & Pf. '03, '04)

> - **Conjecture**: The correspondence between tri-directional checking and bi-directional checking of let-normal form is exact if we add the left rules to the typing judgment.
>
> —Frank Pfenning, ITRS '02 invited talk

Summary

- Tridirectional typing for decidable typechecking with intersections and unions (Dunfield & Pf. '03, '04)

> - **Conjecture**: The correspondence between tri-directional checking and bi-directional checking of let-normal form is exact if we add the left rules to the typing judgment.
>
> —Frank Pfenning, ITRS '02 invited talk

- Typechecking after converting to a (very particular) let-normal form is sound and complete wrt the tridirectional system

Future Work

- Better proofs
  - Mechanized
  - Non-syntactic methods?

Advertisement

- Related ongoing work:
  - First-class polymorphism in bidirectional typing, with subtyping
    - Higher-rank polymorphism (ML Workshop '09)
    - Impredicative polymorphism (ongoing):
      - $\wedge$ = glb, $\vee$ = lub;
        evades undecidability (?)
  - Multiply-indexed types (expressed with $\wedge$)
- Unrelated ongoing work:
  - Beluga: proof assistant with HOAS
    & contextual data (IJCAR, next week)

Thank you!

- Further reading:
  `http://www.cs.cmu.edu/~joshuad/research.html`

- Acknowledgments:
  - Frank Pfenning
  - The ITRS reviewers