Sums of Uncertainty: Refinements go gradual

Khurram A. Jafery  Joshua Dunfield

University of British Columbia

POPL 2017
Gradual typing

I’m last in the session, so I’ll keep this brief.

Traditionally, gradual typing is about

- migrating *incrementally* (gradually) from dynamically typed code to statically typed code.
Gradual typing

I’m last in the session, so I’ll keep this brief.

Traditionally, gradual typing is about

- migrating **incrementally** (gradually) from dynamically typed code to statically typed code.

Joshua is from CMU…
Gradual typing

I’m last in the session, so I’ll keep this brief.

Traditionally, gradual typing is about

- migrating incrementally (gradually) from dynamically typed code to statically typed code.

Joshua is from CMU...

...I lost him at “dynamically typed”.
Gradual typing

Traditionally, gradual typing is about

- migrating incrementally (gradually) from less precisely statically typed code to more precisely statically typed code
Gradual typing

Traditionally, gradual typing is about

- migrating incrementally (gradually) from less precisely statically typed code to more precisely statically typed code

Joshua’s from CMU, but now he’s interested.
Gradual typing

Traditionally, gradual typing is about

- migrating **incrementally** (gradually) from **less precisely** statically typed code (like SML) to **more precisely** statically typed code

Joshua’s from CMU, but now he’s interested.
Gradual typing

Traditionally, gradual typing is about

- migrating incrementally (gradually) from less precisely statically typed code (like SML) to more precisely statically typed code (like refined SML)

Joshua’s from CMU, but now he’s interested.
Traditionally, gradual typing is about

- migrating \textit{incrementally} (gradually)
  from \textit{less precisely} statically typed code (like SML)
  to \textit{more precisely} statically typed code (like \textit{refined} SML)

Wait, isn’t that the same as gradual refinement types?
Gradual typing runs rampant

Traditionally, gradual typing is about

- migrating incrementally (gradually)
  from less precisely statically typed code (like SML)
  to more precisely statically typed code (like refined SML)

Wait, isn’t that the same as gradual refinement types?

No, that paper has what are now called refinement types, which we used to call index refinements.
Gradual typing runs rampant

Traditionally, gradual typing is about

- migrating *incrementally* (gradually)
  from *less precisely* statically typed code (like SML)
  to *more precisely* statically typed code (like refined SML)

Wait, isn’t that the same as gradual refinement types?

No, that paper has what are now called refinement types, which we used to call index refinements.

Our paper has (a simplified form of) what were once called refinement types, which we now call datasort refinements.
Standard ML: dynamically typed?

datatype nat = Zero | Succ of nat

case x : nat of
    Zero ⇒ ...
  | Succ y ⇒ ...
Standard ML: dynamically typed?

datatype nat = Zero | Succ of nat

    case x : nat of
        Zero ⇒ ...
    | Succ y ⇒ ...

But the Definition requires compilers to accept \textbf{nonexhaustive} matches:

    case x : nat of
        Succ y ⇒ ...
Standard ML: dynamically typed?

```plaintext
datatype nat = Zero | Succ of nat

case x : nat of
    Zero ⇒ ...
  | Succ y ⇒ ...
```

But the Definition requires compilers to accept **nonexhaustive** matches:

```plaintext
case x : nat of
    Succ y ⇒ ...
```

If \( x = \text{Zero} \), then the exception \text{Match} is raised.

This nonexhaustive match is fine, **if** we know that \( x \) will never be \( \text{Zero} \).
Datasort refinements [Freeman & Pfenning 1991, Davies 2005, …] push the knowledge that $x$ is not Zero into the type system.

```ml
  case $x$ : nonzero of
        Succ $y$ ⇒ ...
```

This is exhaustive, because $x$ has datasort `nonzero`. 
Datasorts refine ML datatypes

\[
\text{datatype } \text{nat} = \text{Zero} \mid \text{Succ of nat}
\]

- sum type: \text{Succ or Zero}
- recursive type: \text{datatype nat} = \text{Zero} \mid \text{Succ of nat}
Datasorts

Datasorts refine ML datatypes

datatype nat = Zero | Succ of nat

▶ sum type: Succ or Zero

▶ recursive type: datatype nat = Zero | Succ of nat

datasort zero = Zero

datasort nonzero = Succ of nat
Datasorts refine ML datatypes

datatype \texttt{nat} = \texttt{Zero} | \texttt{Succ of nat}

- sum type: \texttt{Succ} or \texttt{Zero}
- recursive type: datatype \texttt{nat} = \texttt{Zero} | \texttt{Succ of nat}

datasort \texttt{zero} = \texttt{Zero}
datasort \texttt{nonzero} = \texttt{Succ of nat}

This paper: gradual, refined sum types.
Static sums

The usual type-theoretic sum type:

datatype $A_1 + A_2 =$

  inj$_1$ of $A_1 |$ inj$_2$ of $A_2$

Elimination form: two-armed case($e,\text{inj}_1 x_1.e_1,\text{inj}_2 x_2.e_2$)

Case expressions over $+, +_1, +_2$ never raise Match exceptions.
Static sums

The usual type-theoretic sum type:

```plaintext
datatype \( A_1 + A_2 = \)
    inj_1 \ of \( A_1 \) \| \ inj_2 \ of \( A_2 \)
```

Elimination form: two-armed case\((e, \ inj_1 \ x_1.e_1, \ inj_2 \ x_2.e_2)\)

Subscript sums \( A_1 +_1 A_2 \) and \( A_1 +_2 A_2 \), corresponding to datasort refinements:

```plaintext
datasort \( A_1 +_1 A_2 = \ inj_1 \ of \ A_1 \)
datasort \( A_1 +_2 A_2 = \ inj_2 \ of \ A_2 \)
```

Elimination form: one-armed case\((e, \ inj_k \ x_k.e_k)\).

\( x : (\text{Int} +_1 \text{Bool}) \vdash \text{case}(x, \ inj_1 \ x_1.x_1) : \text{Int} \)
Static sums

The usual type-theoretic sum type:

\[
\text{datatype } A_1 + A_2 = \text{inj}_1 \text{ of } A_1 \mid \text{inj}_2 \text{ of } A_2
\]

Elimination form: two-armed case(e, inj_1 x_1.e_1, inj_2 x_2.e_2)

Subscript sums \( A_1 +_1 A_2 \) and \( A_1 +_2 A_2 \), corresponding to datasort refinements:

\[
\text{datasort } A_1 +_1 A_2 = \text{inj}_1 \text{ of } A_1
\]
\[
\text{datasort } A_1 +_2 A_2 = \text{inj}_2 \text{ of } A_2
\]

Elimination form: one-armed case(e, inj_k x_k.e_k).

\[
\begin{align*}
x : (\text{Int} +_1 \text{Bool}) \vdash \text{case}(x, \text{inj}_1 x_1.x_1) : \text{Int}
\end{align*}
\]

Case expressions over \(+, +_1, +_2\) never raise Match exceptions.
Dynamic sum

The **dynamic** sum type, corresponding to Standard ML:

\[
\text{datatype } A_1 +? A_2 = \\
\text{ inj}_1 \text{ of } A_1 \mid \text{ inj}_2 \text{ of } A_2
\]

+? allows two-armed case\((e, \text{ inj}_1 x_1.e_1, \text{ inj}_2 x_2.e_2)\).

But +? **also** allows one-armed case\((e, \text{ inj}_k x_k.e_k)\), which may raise a Match exception.
Gradual sums

| Standard ML | + | +₁ | +₂ | possible |
| refined SML | + | +₁ | +₂ | impossible |
Gradual sums

<table>
<thead>
<tr>
<th>Standard ML</th>
<th>+</th>
<th>+?</th>
<th>possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ refined SML</td>
<td>+</td>
<td>+1</td>
<td>+2</td>
</tr>
</tbody>
</table>

= Gradual sums + +1 +2 +? possible iff +? used in annotations
Road map

Source bidirectional type system

Source type assignment system

Target type system with run-time casts

Steps to

type-directed translation

matchfail-free

matchfail-free

varying precision

static sublanguage

(no +?)
Road map

Source bidirectional type system

Source type assignment system

Target type system with run-time casts

\[ e \xrightarrow{\Rightarrow} A \xleftarrow{\Leftarrow} e : A \rightarrow M : T \rightarrow M' : T \]

steps to

type-directed translation

\[ e^S \xrightarrow{\Rightarrow} A^S \xleftarrow{\Leftarrow} M : T \rightarrow M' : T \]

varying precision

static sublanguage
(no + ?)

matchfail-free
matchfail-free
Road map

Source bidirectional type system

Source type assignment system

Target type system with run-time casts

steps to

type-directed translation

matchfail-free    matchfail-free

 varying precision

static sublanguage (no +?)
Road map

Source bidirectional type system

Source type assignment system

Target type system with run-time casts

steps to

type-directed translation

\[ e \xrightarrow{\text{varying precision}} A \xleftarrow{\text{A}} e : A \xrightarrow{\text{M}} M : T \xleftarrow{\text{M}} M' : T \]

\[ e^S \xrightarrow{\text{static sublanguage}} A^S \xleftarrow{\text{(no +?)}} \]

matchfail-free  matchfail-free
Road map

Source bidirectional type system

Source type assignment system

Target type system with run-time casts

\( e \xrightarrow{\Rightarrow} A \leftrightarrow e : A \xrightarrow{\Leftarrow \Rightarrow} M : T \xrightarrow{\text{steps to}} M' : T \)

type-directed translation

\( M : T \xrightarrow{\text{matchfail-free}} M' : T \)

varying precision

\( e^S \xrightarrow{\Rightarrow} A^S \leftrightarrow e^S : A^S \xrightarrow{\Leftarrow \Rightarrow} M : T \xrightarrow{\text{matchfail-free}} M' : T \)

static sublanguage (no +?)

bidirectional

type system
Road map

Source bidirectional type system

Source type assignment system

Target type system with run-time casts

steps to

type-directed translation

varying precision

static sublanguage (no +?)

matchfail-free
Road map

Source bidirectional type system

Source type assignment system

Target type system with run-time casts

e \xLongrightarrow{\text{varying precision}} A

\xleftrightarrow{\text{type-directed translation}}

e : A

\xrightarrow{\text{steps to}}

M : T

\xleftarrow{\text{matchfail-free}}

M' : T

M : T

\xleftarrow{\text{matchfail-free}}

M' : T

static sublanguage (no +?)
Road map

Source bidirectional type system

\[ e \xrightarrow{\Rightarrow} A \xleftarrow{\Leftarrow} e : A \]

Source type assignment system

\[ e : A \xrightarrow{\text{type-directed translation}} M : T \xleftarrow{\text{steps to}} M' : T \]

Target type system with run-time casts

\[ M : T \xrightarrow{\text{matchfail-free}} M' : T \]

Varying precision

\[ e^S \xrightarrow{\Rightarrow} A^S \]

Static sublanguage (no +?)

\[ M : T \xrightarrow{\text{matchfail-free}} M' : T \]
Road map

Source bidirectional type system

e \xrightarrow{\text{varying precision}} A \xleftarrow{\text{A}}

Source type assignment system

e : A \xrightarrow{\text{type-directed translation}} M : T \xleftarrow{\text{steps to}} M' : T

Target type system with run-time casts

\[ M : T \xrightarrow{\text{matchfail-free}} M' : T \]

static sublanguage (no +?)

\[ e^S \xrightarrow{\text{A}} A^S \xleftarrow{\text{A}} \]
Source type assignment

Design introduction and elimination rules:

- How are the static sums $+, +_1, +_2$ introduced and eliminated?
- How is the dynamic sum $+?$ introduced and eliminated?
Static sums

Design introduction and elimination rules for $+_1$, $+_2$:

$$
\Gamma \vdash e : A_k \\
\Gamma \vdash (\text{inj}_k e) : (A_1 +_k A_2) \quad +_k\text{Intro}
$$

$$
\Gamma \vdash e : (A_1 +_k A_2) \quad \Gamma, x_k : A_k \vdash e_k : B \\
\Gamma \vdash \text{case}(e, \text{inj}_k x_k.e_k) : B \quad +_k\text{Elim}
$$
Static sums

Design **introduction** and **elimination** rules for $+_1$, $+_2$:

\[
\frac{\Gamma \vdash e : A_k}{\Gamma \vdash (\text{inj}_k e) : (A_1 +_k A_2)} \quad +_k\text{Intro}
\]

\[
\frac{\Gamma \vdash e : (A_1 +_k A_2) \quad \Gamma, x_k : A_k \vdash e_k : B}{\Gamma \vdash \text{case}(e, \text{inj}_k x_k.e_k) : B} \quad +_k\text{Elim}
\]

**Introduction rule for $+$ via subtyping:**

$(\text{inj}_k e) : (A_1 + A_2)$ because $(A_1 +_k A_2) \leq (A_1 + A_2)$.

\[
\frac{\Gamma, x_1 : A_1 \vdash e_1 : B \quad \Gamma, x_2 : A_2 \vdash e_2 : B}{\Gamma \vdash \text{case}(e, \text{inj}_1 x_1.e_1, \text{inj}_2 x_2.e_2) : B} \quad +\text{Elim}
\]

(two-armed elimination for $+_k$ possible via subtyping)
Dynamic sum

Design **introduction** and **elimination** rules for $+$?:

$$
\Gamma \vdash e : A_k \\
\Gamma \vdash (\text{inj}_k \ e) : (A_1 \ +? \ A_2) \\
+?\text{Intro}
$$
Dynamic sum

Design **introduction** and **elimination** rules for \(+\): \[\Gamma \vdash e : A_k \quad \frac{\Gamma \vdash (\text{inj}_k e) : (A_1 + ? A_2)}{\text{+?Intro}}\]

\[\Gamma \vdash e : (A_1 + ? A_2) \quad \Gamma, x_k : A_k \vdash e_k : B \quad \frac{\Gamma \vdash \text{case}(e, \text{inj}_k x_k.e_k) : B}{\text{+?Elim-one-arm}}\]

\[\Gamma \vdash e : (A_1 + ? A_2) \quad \Gamma, x_1 : A_1 \vdash e_1 : B \quad \Gamma, x_2 : A_2 \vdash e_2 : B \quad \frac{\Gamma \vdash \text{case}(e, \text{inj}_1 x_1.e_1, \text{inj}_2 x_2.e_2) : B}{\text{+?Elim-two-arm}}\]
Varying precision

Given a typing derivation, we want to

► Replace more precise types, like $A +_1 B$, with the less precise type $A + ? B$
Varying precision

Given a typing derivation, we want to

- Replace **more precise** types, like $A +_1 B$, with the **less precise** type $A + ? B$
- Replace **less precise** types $A + ? B$ with **more precise** types $A + B$ or $A + k B$
Varying precision

Given a typing derivation, we want to

- Replace **more precise** types, like \( A +_1 B \),
  with the **less precise** type \( A +? B \)
- Replace **less precise** types \( A +? B \)
  with **more precise** types \( A + B \) or \( A +_k B \)

Replacer an annotation \( A +_1 B \) with \( A +? B \) preserves typing
(varying precision—gradual guarantee)

Replacer an annotation \( A +? B \) with a **more precise** annotation
do not always preserve typing.
Defining precision

First, $\sqsubseteq$ on sum constructors

$+\text{?}, +, +_1, +_2$:

- dynamic
- static

$\sqsubseteq$

$+_1$ $+_1$ $+_2$
Defining precision

First, \( \sqsubseteq \) on sum constructors

\( +?, +, +_1, +_2 \):

- dynamic
  - \( +? \)
  \( \sqcup \)
- static
  - \( +_1 \)
  - \( + \)
  - \( +_2 \)

Extend \( \sqsubseteq \) pointwise:

if \( A' \sqsubseteq A \) and \( B' \sqsubseteq B \) then...

\[
A + ? B =\]
\[
A' +_1 B' \quad A' + B' \quad A' +_2 B'
\]
Defining precision

First, $\sqsubseteq$ on sum constructors $+\?, +, +1, +2$:

Extend $\sqsubseteq$ pointwise:

if $A' \sqsubseteq A$ and $B' \sqsubseteq B$ then...

Other constructors covariant (similar to $\sqsubseteq$ in refinement types):

dynamic
Subsumption

- The usual subsumption rule:

\[ \Gamma \vdash e : A \quad A \leq B \]

\[ \Gamma \vdash e : B \]
Subsumption

- The usual subsumption rule:

\[
\Gamma \vdash e : A \quad A \leq B \\
\Gamma \vdash e : B
\]

- In a land of imprecision: “kinda $A$”, “kinda $B$”

\[
\Gamma \vdash e : A' \quad A \sqsubseteq A' \quad A \leq B \quad B \sqsubseteq B' \\
\Gamma \vdash e : B'
\]

- These 3 premises = directed consistency $A' \rightsquigarrow B'$

\[
A' \quad B' \\
\sqsubseteq \quad \sqsubseteq \\
A \quad \leq \quad B
\]
Subsumption

- The usual subsumption rule:

\[ \Gamma \vdash e : A \quad A \leq B \]
\[ \Gamma \vdash e : B \]

- In a land of imprecision: “kinda A”, “kinda B”

\[ \Gamma \vdash e : A' \quad A' \sqsubseteq A' \quad A \leq B \quad B \sqsubseteq B' \]
\[ \Gamma \vdash e : B' \]

- These 3 premises = **directed consistency** \( A' \rightsquigarrow B' \)

\[
\begin{array}{c}
A' \\
\sqsubset \sqsubset \\
A \leq B
\end{array}
\begin{array}{c}
B' \\
\sqsubset \sqsubset \\
B \sqsubseteq B'
\end{array}
\]
Subsumption

- The usual subsumption rule:

\[
\Gamma \vdash e : A \quad A \leq B \\
\hline
\Gamma \vdash e : B
\]

- In a land of imprecision: “kinda A”, “kinda B”

\[
\Gamma \vdash e : A' \quad A \sqsubseteq A' \quad A \leq B \quad B \sqsubseteq B' \\
\hline
\Gamma \vdash e : B'
\]

- These 3 premises = directed consistency \( A' \sim B' \)

\[
A' \quad B' \\
\sqsubseteq \quad \sqsubseteq \\
A \quad \leq \quad B
\]
Subsumption

- The usual subsumption rule:

\[
\Gamma \vdash e : A \quad A \leq B \\
\hline
\Gamma \vdash e : B
\]

- In a land of imprecision: “kinda A”, “kinda B”

\[
\Gamma \vdash e : A' \quad A \subseteq A' \quad A \leq B \quad B \subseteq B' \\
\hline
\Gamma \vdash e : B'
\]

- These 3 premises = directed consistency \( A' \rightsquigarrow B' \)

\[
\begin{array}{cc}
A' & B' \\
\sqsubseteq & \sqsubseteq \\
\hline
A \leq B
\end{array}
\]
Subsumption

- The usual subsumption rule:

\[
\Gamma \vdash e : A \quad A \leq B
\]

\[
\Gamma \vdash e : B
\]

- In a land of imprecision: “kinda A”, “kinda B”

\[
\Gamma \vdash e : A' \quad A \sqsubseteq A' \quad A \leq B \quad B \sqsubseteq B'
\]

\[
\Gamma \vdash e : B'
\]

- These 3 premises = directed consistency \( A' \leadsto B' \)

\[
A' \quad B' \\
\sqsubseteq \quad \sqsubseteq \\
A \quad \leq \quad B
\]
Subsumption

- The usual subsumption rule:

\[ \Gamma \vdash e : A \quad A \leq B \]

\[ \Gamma \vdash e : B \]

- In a land of imprecision: “kinda \( A \)”, “kinda \( B \)"

\[ \Gamma \vdash e : A' \quad A' \sqsubseteq A' \quad A \leq B \quad B \sqsubseteq B' \]

\[ \Gamma \vdash e : B' \]

- These 3 premises = \textit{directed consistency} \( A' \rightsquigarrow B' \)

\[ \begin{array}{c}
  A' \\
  \sqsubset \\
  A \\
  \leq \\
  B \\
\end{array} \]

- Is directed consistency transitive?
Road map

Source bidirectional type system

Source type assignment system

Target type system with run-time casts

steps to
type-directed translation

matchfail-free
matchfail-free

Source

\[ e \overset{\Rightarrow}{\leftrightarrow} A \]

Source type assignment system

\[ e : A \]

Target type system with run-time casts

\[ M : T \overset{\Leftarrow}{\rightarrow} M' : T \]

Varying precision

\[ e^S \overset{\Rightarrow}{\leftrightarrow} A^S \]

Static sublanguage (no +?)

\[ M : T \overset{\Leftarrow}{\rightarrow} M' : T \]
Road map

Source bidirectional type system

e ⇔ A

varying precision

eS ⇔ AS

static sublanguage (no +?)

Source type assignment system

e : A

type-directed translation

Target type system with run-time casts

M : T

steps to

M' : T

matchfail-free

matchfail-free
Bidirectional typing: why?

Some past answers:

- to handle features beyond Damas–Milner (Pierce & Turner 2000; Dunfield & Pfenning 2004; Dunfield & Krishnaswami 2013; …)
- for better (earlier) type error messages
Bidirectional typing: why?

Some past answers:

▶ to handle features beyond Damas–Milner
  (Pierce & Turner 2000;
  Dunfield & Pfenning 2004;
  Dunfield & Krishnaswami 2013; …)

▶ for better (earlier) type error messages

Here:

▶ to make typing **more predictable**, 
  by avoiding **unnecessary imprecision**.
Bidirectional typing in one slide

- **Organize** the flow of information from type annotations:
  - Given $\Gamma$, $e$, and a known type $A$, **check** $e$:
    $$\Gamma \vdash e \iff A$$
  - Given $\Gamma$ and $e$, **synthesize** a type for $e$:
    $$\Gamma \vdash e \Rightarrow A$$
  - The type $A$ in the checking judgment $e \iff A$ is a **goal**.
Bidirectional typing

Frank Pfenning’s recipe:
intro rules check, elim rules synthesize.

\[
\begin{align*}
\Gamma, x : A_1 & \vdash e \iff A_2 \quad \text{Chk} \rightarrow \text{Intro} \\
\Gamma & \vdash \lambda x . e \iff A_1 \rightarrow A_2 \\
\Gamma & \vdash e_1 \Rightarrow (A \rightarrow B) \quad \Gamma & \vdash e_2 \iff A \\
\Gamma & \vdash e_1 \ e_2 \Rightarrow B \quad \text{Syn} \rightarrow \text{Elim}
\end{align*}
\]

- **Chk→Intro:**
  The type \(A_1 \rightarrow A_2\) must flow from an annotation.

- **Syn→Elim:** The type \(A \rightarrow B\) must flow from an annotation, perhaps via \(\Gamma\).
Bidirectional typing

Frank Pfenning’s recipe: intro rules check, elim rules synthesize.

\[
\frac{\Gamma, x : A_1 \vdash e \leftrightarrow A_2}{\Gamma \vdash \lambda x. e \leftrightarrow A_1 \rightarrow A_2} \quad \text{Chk} \rightarrow \text{Intro}
\]

\[
\frac{\Gamma \vdash e_1 \Rightarrow (A \rightarrow B)}{\Gamma \vdash e_1 e_2 \Rightarrow B} \quad \Gamma \vdash e_2 \leftrightarrow A \quad \text{Syn} \rightarrow \text{Elim}
\]

- **Chk→Intro:**
  The type \( A_1 \rightarrow A_2 \) must flow from an annotation.

- **Syn→Elim:** The type \( A \rightarrow B \) must flow from an annotation, perhaps via \( \Gamma \).
Bidirectional typing

The subsumption rule:

\[ \Gamma \vdash e \Rightarrow A' \quad A' \leadsto B' \]

\[ \Gamma \vdash e \Leftarrow B' \]

\[ A' \leadsto B' \]

\[ A \preceq B \]
Bidirectional typing

The subsumption rule:

\[
\frac{\Gamma \vdash e \Rightarrow A' \quad A' \rightsquigarrow B'}{\Gamma \vdash e \Leftrightarrow B'}
\]

Subformula property:
Every type synthesized or checked flows from a type annotation.
Road map

Source bidirectional type system

Source type assignment system

type-directed translation

target type system with run-time casts

steps to

matchfail-free

matchfail-free

Target type system with run-time casts

Source type assignment system

Steps to target type system with run-time casts
Road map

Source bidirectional type system

\[ e \xrightarrow{\Rightarrow} A \]

Source type assignment system

\[ e : A \xleftarrow{\Rightarrow} M : T \xrightarrow{\text{steps to}} M' : T \]

Target type system with run-time casts

\[ \text{type-directed translation} \]

Source type assignment system

\[ e^S \xrightarrow{\Rightarrow} A^S \xleftarrow{\Rightarrow} M : T \xrightarrow{\text{matchfail-free}} M' : T \]

Static sublanguage (no `+`?)
Target language

- Target sum types include only **static** sums: +, +₁, +₂
- Casts between sums:

  \[ \langle +₁ \leftarrow + \rangle (\text{inj}_1 \, ν) \quad \text{will step to} \quad \text{inj}_1 \, ν \]

  \[ \langle +₂ \leftarrow + \rangle (\text{inj}_1 \, ν) \quad \text{will step to} \quad \text{matchfail} \]
Type-directed translation: add casts

Where directed consistency $\rightsquigarrow$ is used, translation adds a cast from $A'$ to $B'$

$$
\Gamma \vdash e : A' \leftrightarrow M \quad A' \rightsquigarrow B' \hookrightarrow C
$$

$$
\Gamma \vdash e : B' \hookrightarrow C[M]
$$

$$
\begin{align*}
A' & \rightsquigarrow B' \\
\square & \quad \square \\
A & \leq B
\end{align*}
$$
**Type-directed translation: add casts**

Where **directed consistency** $\rightsquigarrow$ is used, translation adds a cast from $A'$ to $B'$

$$
\frac{
\Gamma \vdash e : A' \rightsquigarrow M \quad A' \rightsquigarrow B' \hookrightarrow C \\
}{\Gamma \vdash e : B' \hookrightarrow C[M]}
$$

$A' \rightsquigarrow B' \\
\sqsubseteq \sqsubseteq \\
A \leq B \\

(Unit + ? Unit) \rightsquigarrow (Unit +_2 Unit) \\

$$
\frac{
\Gamma \vdash x : (Unit + ? Unit) \hookrightarrow x \hookrightarrow \langle +_2 \leftarrow + \rangle [] \\
}{\Gamma \vdash x : B' \hookrightarrow \langle +_2 \leftarrow + \rangle x}
$$
Type-directed translation: add casts

Where **directed consistency** ⇝ is used, translation adds a cast from \( A' \) to \( B' \)

\[
\begin{align*}
\Gamma \vdash e &: A' \hookrightarrow M & A' &\aabla B' \hookrightarrow C \\
\frac{}{\Gamma \vdash e &: B' \hookrightarrow C[M]} & A' &\aabla B' \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash x &: (\text{Unit} + ? \text{Unit}) \hookrightarrow x &\hookrightarrow \langle +_2 \leftarrow + \rangle[] \\
\frac{}{\Gamma \vdash x &: B' \hookrightarrow \langle +_2 \leftarrow + \rangle x} & (\text{Unit} + ? \text{Unit}) &\aabla (\text{Unit} + _2 \text{Unit}) \\
\end{align*}
\]

\[
\begin{align*}
\text{Unit} + ? \text{Unit} &\aabla \text{Unit} + _2 \text{Unit} \\
\text{Unit} + _2 \text{Unit} &\leq \text{Unit} + _2 \text{Unit}
\end{align*}
\]
Metatheory

Source
bidirectional
type system
Thm. 1

$e \xleftrightarrow{A}$

Source
type assignment
system

$e : A \xrightarrow{\text{Thm. 2}} \xleftarrow{\text{Thm. 3}} M : T$

Target type system
with run-time casts

steps to
type safety (Thms. 6,7)

$M : T \xrightarrow{\text{Thm. 9}} \xleftarrow{\text{Thm. 10}} M' : T$

varying
precision
Thm. 4

$e^S \xleftrightarrow{A^S} e^S$

static sublanguage
(no \(\oplus\)?)

$M : T \xrightarrow{\text{matchfail-free}} \xleftarrow{\text{matchfail-free}} M' : T$

$M : T \xrightarrow{\text{matchfail-free}} \xleftarrow{\text{matchfail-free}} M' : T$
Metatheory

Source bidirectional type system

Thm. 1

Source type assignment system

Thm. 2

Target type system with run-time casts

steps to

type safety (Thms. 6,7)

\( e \leftrightarrow A \)

Thm. 3

\( e : A \)

Thm. 9
type-directed translation

\( M : T \)

Thm. 5

\( M' : T \)

\( e^S \leftrightarrow A^S \)

Thm. 5

\( M : T \)

matchfail-free

Thm. 10

\( M' : T \)

matchfail-free

Thm. 8

static sublanguage (no +?)

Thm. 4

varying precision
Metatheory

Source bidirectional type system
Thm. 1

Source type assignment system
Thm. 2

Target type system with run-time casts
Thm. 9

steps to type safety (Thms. 6,7)

Thm. 3

e \ll A \leftrightarrow e : A \rightarrow M : T \rightarrow M' : T

Thm. 5

e^S \ll A^S \leftrightarrow e : A \rightarrow M : T \rightarrow M' : T

Thm. 4

varying precision

Source

Thm. 10

matchfail-free

Thm. 8

matchfail-free

Target type system

static sublanguage (no +?)

29
Metatheory

Source bidirectional type system (Thm. 1)

\(e \iff A\)

Source type assignment system (Thm. 2)

\(e : A\)

Target type system with run-time casts

\(M : T \iff M' : T\)

steps to type safety (Thms. 6, 7)

\(M : T \iff M' : T\)

matchfail-free (Thm. 10)

matchfail-free (Thm. 8)

Target type system with run-time casts

\(M : T \iff M' : T\)

Thm. 9

type-directed translation

Thm. 3

varying precision (Thm. 4)

Thm. 5

static sublanguage (no + ?)

\(e^S \iff A^S\)
Metatheory

Source bidirectional type system (Thm. 1)

Source type assignment system (Thm. 2)

Target type system with run-time casts (steps to type safety (Thms. 6,7))

Type-directed translation (Thm. 9)

Varying precision (Thm. 4)

Type safety (Thms. 6,7)

Static sublanguage (no + ?) (Thm. 10)

Matchfail-free (Thm. 8)

\( e \rightarrow\leftarrow A \rightarrow\leftarrow e : A \rightarrow\leftarrow M : T \rightarrow\leftarrow M' : T \)
Metatheory

Source bidirectional type system
Thm. 1

Source type assignment system

Varying precision
Thm. 3

$
\begin{align*}
e & \Rightarrow A \\
e : A & \Rightarrow M : T \\
M' : T & \Rightarrow \text{matchfail-free (Thm. 8)} \\
M : T & \Rightarrow \text{matchfail-free (Thms. 6,7)} \\
\end{align*}
$

Target type system with run-time casts

Type-directed translation

Thm. 9

Steps to type safety (Thms. 6,7)

Static sublanguage (no +?)

Thm. 4

Thm. 2

Thm. 5

Thm. 10
**Metatheory**

Source bidirectional type system

Thm. 1

\[ e \iff A \]

Thm. 2

Thm. 3

Target type system with run-time casts

steps to type safety (Thms. 6,7)

Source type assignment system

\[ e : A \]

Thm. 9

type-directed translation

\[ M : T \]

M'D : T

Thm. 10

matchfail-free

Thm. 8

matchfail-free

static sublanguage (no + ? )
Metatheory

Source bidirectional type system (Thm. 1)

$e \Leftrightarrow A$

Source type assignment system (Thm. 2)

$e : A \leftrightarrow M : T$

Target type system with run-time casts (Thms. 6, 7)

$M \rightarrow M'$

Varying precision (Thm. 4)

$e^S \Leftrightarrow A^S$

Type-directed translation (Thm. 3)

$e^S : A^S \rightarrow M : T$

Step to type safety (Thms. 6, 7)

$M \rightarrow M'$

Static sublanguage (no +?)

$M : T \rightarrow M' : T$

Matchfail-free (Thm. 10)

$M : T \rightarrow M' : T$

Matchfail-free (Thm. 8)
Metatheory

Source bidirectional type system (Thm. 1)

Source type assignment system

Target type system with run-time casts

steps to type safety (Thms. 6,7)

Type-directed translation (Thm. 9)

Thm. 2

Thm. 3

Thm. 5

static sublanguage (no `+` )

Thm. 10

matchfail-free

Thm. 8

matchfail-free

Thm. 4

varying precision

Thm. 4
Metatheory

Gradual guarantee (Siek et al. 2015)

- Thm. 4: Varying precision
- Thm. 5: Static soundness and completeness
- Thm. 15: Dynamic soundness and completeness
- Thm. 11: Translation preserves precision
- Thm. 12: Stepping preserves precision
- Thm. 13: Precision respects convergence
Metatheory

Gradual guarantee (Siek et al. 2015)

- Thm. 4: Varying precision
- Thm. 5: Static soundness and completeness
- Thm. 15: Dynamic soundness and completeness
- Thm. 11: Translation preserves precision
- Thm. 12: Stepping preserves precision
- Thm. 13: Precision respects convergence
Related work

Refinements:

- Datasort refinements:
  \( A \sqsubseteq \tau \) says refinement (sort) \( A \) refines type \( \tau \).
  Kind of like \( A' \sqsubseteq A \)—but sorts and types cannot be mixed:
  varying precision cannot even be stated.

- Bidirectionality makes type-checking practical
Related work

Refinements:

- Datasort refinements:
  Freeman & Pfenning 1991, Davies 2005, …
  \( A \sqsubseteq \tau \) says refinement (sort) \( A \) refines type \( \tau \).
  Kind of like \( A' \sqsubseteq A \)—but sorts and types cannot be mixed:
  varying precision cannot even be stated.
- Bidirectionality makes type-checking practical

Gradual typing:

- Consistency (Siek and Taha 2006, …)
- Consistent subtyping (Siek and Taha 2007, …)
- Blame (Wadler & Findler 2009, …)
- Subformula property (Garcia & Cimini 2015)
What’s next?

- Implement the bidirectional system and translation
- Add more types (intersection, μ, ∀)
- Evaluate run-time efficiency
What’s next?

- Implement the bidirectional system and translation
- Add more types (intersection, $\mu$, $\forall$)
- Evaluate run-time efficiency

- Unify and generalize
  
  (1) classic gradual typing, and
  (2) gradual sums

  through a new type constructor, guided by ideas from abstracting gradual typing (Garcia et al. 2016)
Conclusion

- Guided by type-theoretic intuition, we combined static sums and dynamic sums into a gradual type system
- The subformula property of bidirectional typing controls imprecision
- The system enjoys the gradual guarantee

Paper and proofs:  arxiv.org/abs/1611.02392
Conclusion

- Guided by type-theoretic intuition, we combined static sums and dynamic sums into a gradual type system
- The subformula property of bidirectional typing controls imprecision
- The system enjoys the gradual guarantee

Paper and proofs:  arxiv.org/abs/1611.02392