Sums of Uncertainty: Refinements go gradual

Khurram A. Jafery   Jana 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.

Jana 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.

Jana is from CMU. . .

. . . I lost her at "dynamically typed".

Gradual typing

~~Traditionally,~~ gradual typing is about

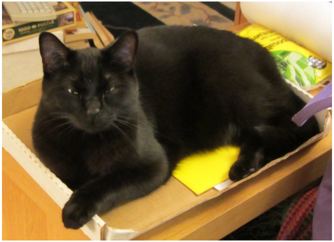
- migrating **incrementally** (gradually)
  from **less precisely** statically typed code
  to **more precisely** statically typed code

Gradual typing

~~Traditionally,~~ gradual typing is about

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

Jana’s from CMU, but now she’s interested.

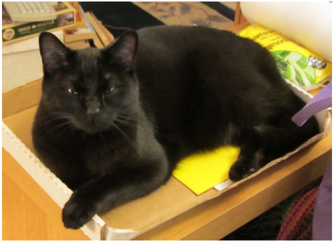


Gradual typing

~~Traditionally,~~ gradual typing is about

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

Jana's from CMU, but now she's interested.

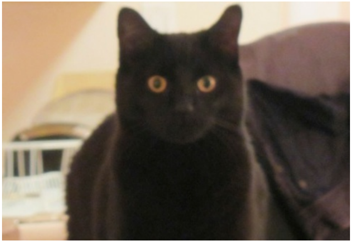


Gradual typing

~~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)

Jana's from CMU, but now she's interested.



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?

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?

$$\mathsf{datatype\ nat} \ = \ \mathtt{Zero} \mid \mathtt{Succ}\ \mathsf{of\ nat}$$

$$\begin{array}{l} \mathsf{case}\ x : \mathsf{nat}\ \mathsf{of} \\ \quad \mathtt{Zero} \Rightarrow \mathtt{...} \\ \ \mid \mathtt{Succ}\ y \Rightarrow \mathtt{...} \end{array}$$

Standard ML: dynamically typed?

$\texttt{datatype nat} \;=\; \texttt{Zero} \mid \texttt{Succ of nat}$

$$
\begin{array}{l}
\texttt{case}\ x : \texttt{nat}\ \texttt{of} \\
\quad\ \ \texttt{Zero} \Rightarrow \texttt{...} \\
\quad \mid \texttt{Succ}\ y \Rightarrow \texttt{...}
\end{array}
$$

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

$$
\begin{array}{l}
\texttt{case}\ x : \texttt{nat}\ \texttt{of} \\
\quad \texttt{Succ}\ y \Rightarrow \texttt{...}
\end{array}
$$

Standard ML: dynamically typed?

$\texttt{datatype nat} = \texttt{Zero} \mid \texttt{Succ of nat}$

$$\begin{aligned} &\texttt{case}\ x : \texttt{nat}\ \texttt{of} \\ &\quad \texttt{Zero} \Rightarrow \ldots \\ &\ \mid \texttt{Succ}\ y \Rightarrow \ldots \end{aligned}$$

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

$$\begin{aligned} &\texttt{case}\ x : \texttt{nat}\ \texttt{of} \\ &\quad \texttt{Succ}\ y \Rightarrow \ldots \end{aligned}$$

If $x = \texttt{Zero}$, then the exception Match is raised.

This nonexhaustive match is fine,
if we know that x will never be Zero.

Refined Standard ML

Datasort refinements [Freeman & Pfenning 1991, Davies 2005, . . .]
push the knowledge that x is not `Zero` into the type system.

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

This **is** exhaustive, because x has **datasort** nonzero.

Datasorts

Datasorts refine ML datatypes

`datatype nat = Zero | Succ of nat`

- sum type : `Succ` **or** `Zero`
- recursive type: `datatype nat = Zero | Succ of nat`

Datasorts

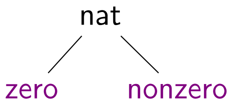
Datasorts refine ML datatypes

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



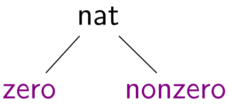
Datasorts

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



This paper: gradual, refined **sum types**.

Static sums

The usual type-theoretic sum type:

$\texttt{datatype}\ A_1 + A_2 \ =$
$\quad \texttt{inj}_1\ \texttt{of}\ A_1 \mid \texttt{inj}_2\ \texttt{of}\ A_2$

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

Static sums

The usual type-theoretic sum type:

$\mathtt{datatype}\ A_1 + A_2 \ =$
$\quad \mathsf{inj}_1\ \mathsf{of}\ A_1 \mid \mathsf{inj}_2\ \mathsf{of}\ A_2$

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

Subscript sums $A_1 +_1 A_2$ and $A_1 +_2 A_2$,
corresponding to datasort refinements:

$\mathtt{datasort}\ A_1 +_1 A_2 \ = \ \mathsf{inj}_1\ \mathsf{of}\ A_1$

$\mathtt{datasort}\ A_1 +_2 A_2 \ = \ \mathsf{inj}_2\ \mathsf{of}\ A_2$

Elimination form: **one**-armed $\mathsf{case}(e, \mathsf{inj}_k\, x_k.e_k)$.

$$x : (\mathsf{Int} +_1 \mathsf{Bool}) \vdash \mathsf{case}(x, \mathsf{inj}_1\, x_1.x_1) : \mathsf{Int}$$

Static sums

The usual type-theoretic sum type:

$\mathtt{datatype}\ A_1 + A_2 =$
$\quad \mathtt{inj}_1\ \mathtt{of}\ A_1 \mid \mathtt{inj}_2\ \mathtt{of}\ A_2$

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

Subscript sums $A_1 +_1 A_2$ and $A_1 +_2 A_2$, corresponding to datasort refinements:

$\mathtt{datasort}\ A_1 +_1 A_2 = \mathtt{inj}_1\ \mathtt{of}\ A_1$

$\mathtt{datasort}\ A_1 +_2 A_2 = \mathtt{inj}_2\ \mathtt{of}\ A_2$

Elimination form: **one**-armed $\mathsf{case}(e, \mathsf{inj}_k\, x_k.e_k)$.

$$x : (\mathsf{Int} +_1 \mathsf{Bool}) \vdash \mathsf{case}(x, \mathsf{inj}_1\, x_1.x_1) : \mathsf{Int}$$

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

Dynamic sum

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

$\texttt{datatype}\ A_1 +^? A_2\ =$
$\quad \texttt{inj}_1\ \texttt{of}\ A_1 \mid \texttt{inj}_2\ \texttt{of}\ A_2$

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

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

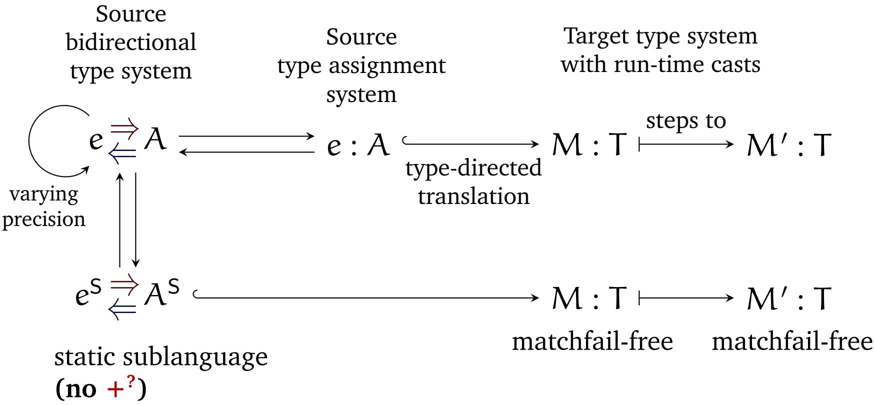
Gradual sums

					match failures are…
Standard ML				$+^?$	possible
refined SML	$+$	$+_1$	$+_2$		impossible

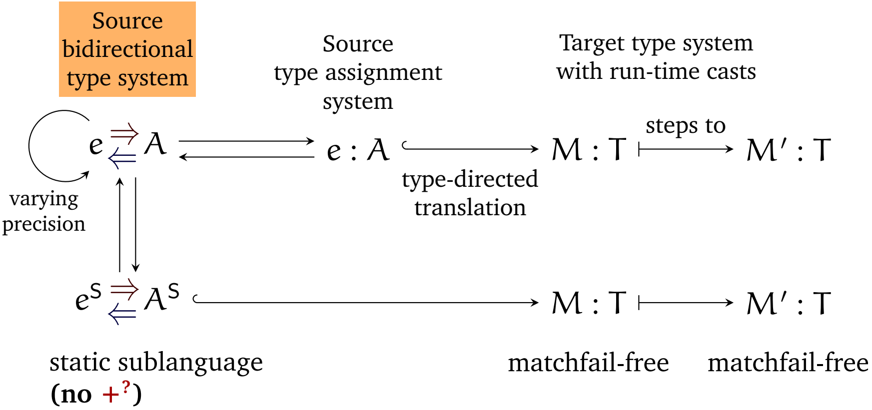
Gradual sums

						match failures are...
	Standard ML				$+^?$	possible
+	refined SML	$+$	$+_1$	$+_2$		impossible
=	**Gradual sums**	$+$	$+_1$	$+_2$	$+^?$	**possible** iff $+^?$ used in annotations

Road map

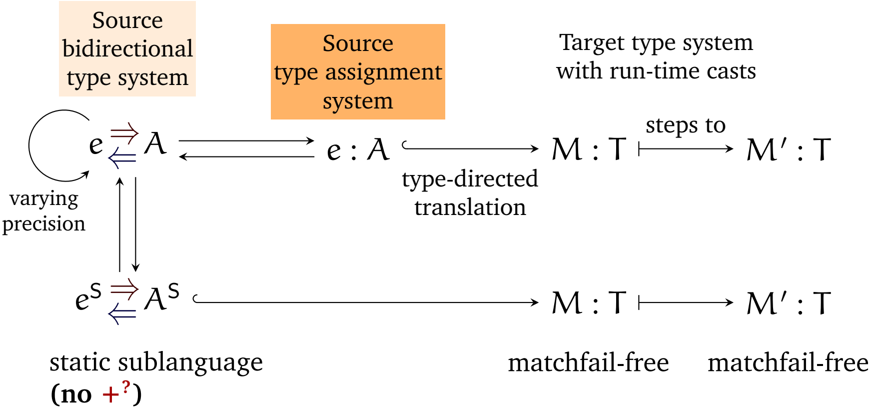

Source bidirectional type system
Source type assignment system
Target type system with run-time casts
$e \Rightarrow A$ / $e \Leftarrow A$
$e : A$
$M : T$
steps to
$M' : T$
type-directed translation
varying precision
$e^S \Rightarrow A^S$ / $e^S \Leftarrow A^S$
$M : T$
$M' : T$
matchfail-free
matchfail-free
static sublanguage
(no $+^?$)

Road map



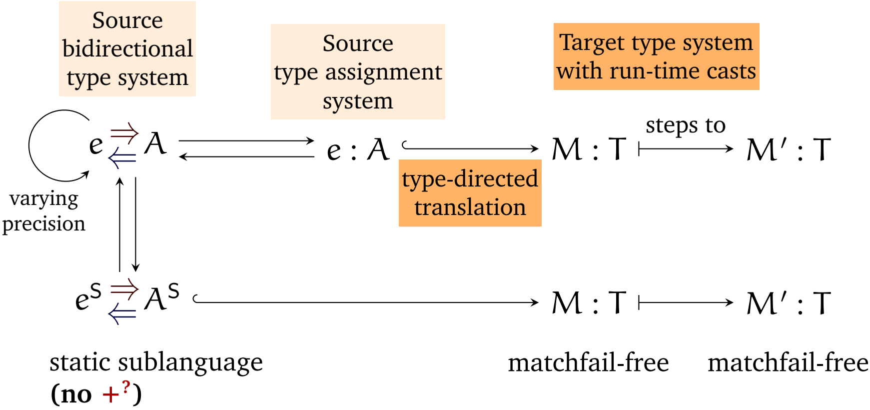
Source bidirectional type system
Source type assignment system
Target type system with run-time casts
$e \rightrightarrows A$
$e : A$
type-directed translation
$M : T$
steps to
$M' : T$
varying precision
$e^S \rightrightarrows A^S$
$M : T$
$M' : T$
static sublanguage
(no $+^?$)
matchfail-free
matchfail-free

Road map

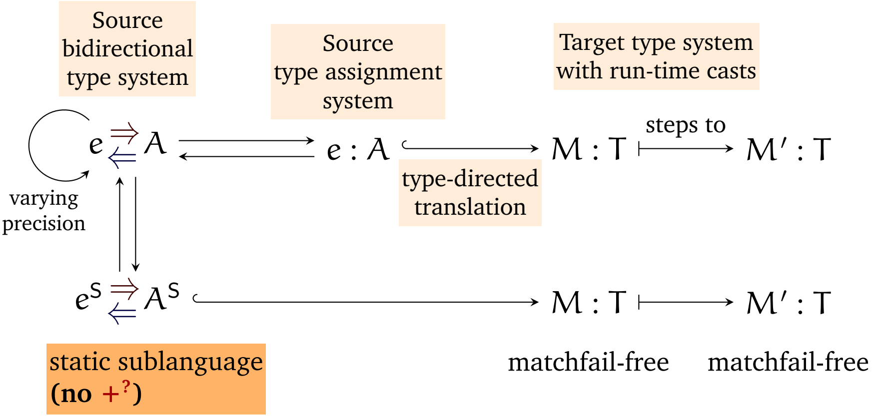


Source bidirectional type system
Source type assignment system
Target type system with run-time casts
$e \rightrightarrows A$
$e : A$
$M : T$
steps to
$M' : T$
varying precision
type-directed translation
$e^S \rightrightarrows A^S$
$M : T$
$M' : T$
static sublanguage
(no $+^?$)
matchfail-free
matchfail-free

Road map

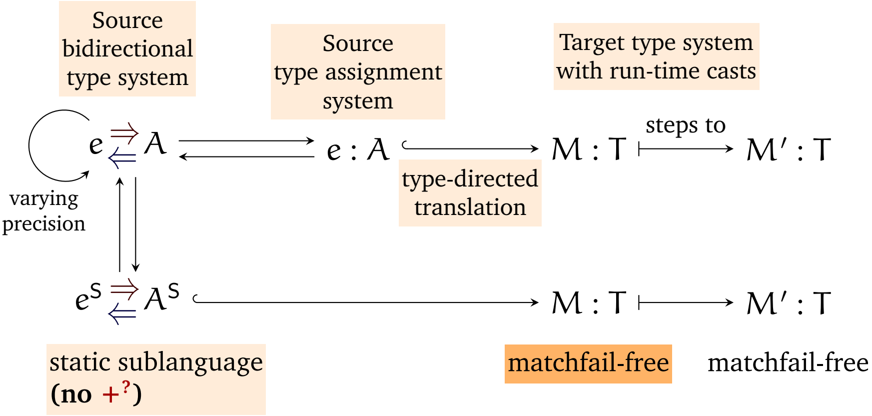

Source bidirectional type system
Source type assignment system
Target type system with run-time casts
steps to
type-directed translation
varying precision
$e \rightrightarrows A$
$e : A$
$M : T$
$M' : T$
$e^S \rightrightarrows A^S$
static sublanguage
(no $+^?$)
matchfail-free
matchfail-free

Road map



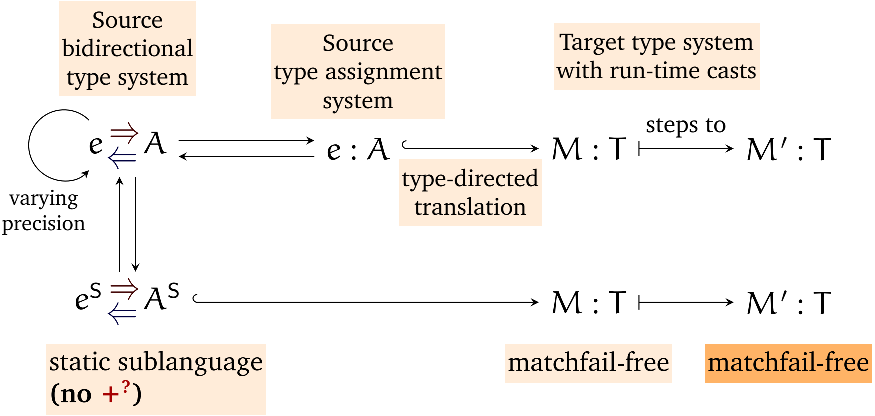
Source bidirectional type system
Source type assignment system
Target type system with run-time casts
$e \rightrightarrows A$
$e : A$
$M : T$
steps to
$M' : T$
varying precision
type-directed translation
$e^S \rightrightarrows A^S$
$M : T$
$M' : T$
static sublanguage (no $+^?$)
matchfail-free
matchfail-free

Road map

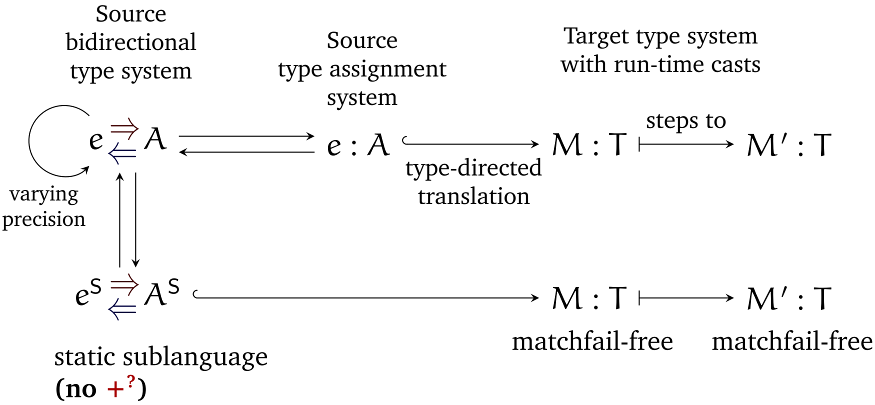


Source bidirectional type system
Source type assignment system
Target type system with run-time casts
$e \rightleftarrows A$
$e : A$
$M : T$
steps to
$M' : T$
varying precision
type-directed translation
$e^S \rightleftarrows A^S$
$M : T$
$M' : T$
static sublanguage (no $+^?$)
matchfail-free
matchfail-free

Road map

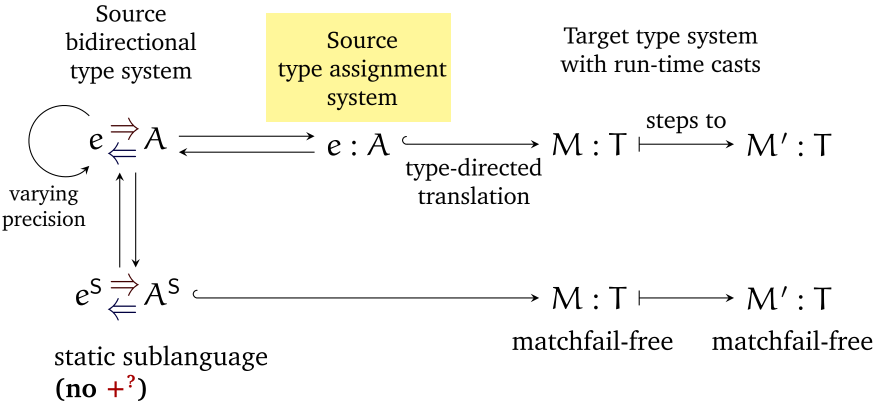

Source bidirectional type system
Source type assignment system
Target type system with run-time casts
e ⇒⇐ A
e : A
M : T
steps to
M′ : T
varying precision
type-directed translation
e^S ⇒⇐ A^S
M : T
M′ : T
static sublanguage (no +?)
matchfail-free
matchfail-free

Road map



Source bidirectional type system
Source type assignment system
Target type system with run-time casts
e ⇒ A ⇐
e : A
M : T
steps to
M′ : T
varying precision
type-directed translation
e^S ⇒ A^S ⇐
M : T
M′ : T
matchfail-free
matchfail-free
static sublanguage
(no +?)

Road map



Source bidirectional type system
Source type assignment system
Target type system with run-time casts
varying precision
$e \Rightarrow A$ / $e \Leftarrow A$
$e : A$
type-directed translation
$M : T$
steps to
$M' : T$
$e^S \Rightarrow A^S$ / $e^S \Leftarrow A^S$
$M : T$
$M' : T$
matchfail-free
matchfail-free
static sublanguage
(no $+^?$)

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 $+_1$, $+_2$:

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

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

Static sums

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

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

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

Introduction rule for $+$ via subtyping:
$(\mathsf{inj}_k\, e) : (A_1 + A_2)$ because $(A_1 +_k A_2) \leq (A_1 + A_2)$.

$$\frac{\Gamma \vdash e : (A_1 + A_2) \qquad \begin{array}{c}\Gamma, x_1 : A_1 \vdash e_1 : B \\ \Gamma, x_2 : A_2 \vdash e_2 : B\end{array}}{\Gamma \vdash \mathsf{case}(e, \mathsf{inj}_1\, x_1.e_1, \mathsf{inj}_2\, x_2.e_2) : B}\ +\text{Elim}$$

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

Dynamic sum

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

$$\frac{\Gamma \vdash e : A_k}{\Gamma \vdash (\mathtt{inj}_k\, e) : (A_1 +^? A_2)}\ +^?\text{Intro}$$

Dynamic sum

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

$$\frac{\Gamma \vdash e : A_k}{\Gamma \vdash (\mathsf{inj}_k\, e) : (A_1 +^? A_2)}\ +^?\text{Intro}$$

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

$$\frac{\Gamma \vdash e : (A_1 +^? A_2) \qquad \begin{array}{c}\Gamma, x_1 : A_1 \vdash e_1 : B \\ \Gamma, x_2 : A_2 \vdash e_2 : B\end{array}}{\Gamma \vdash \mathsf{case}(e, \mathsf{inj}_1\, x_1.e_1, \mathsf{inj}_2\, x_2.e_2) : B}\ +^?\text{Elim-two-arm}$$

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 $A +_k B$

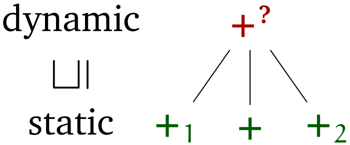
Replacing an annotation $A +_1 B$ with $A +^? B$ preserves typing (varying precision—gradual guarantee)

Replacing an annotation $A +^? B$ with a **more precise** annotation does not always preserve typing.

Defining precision

First, $\sqsubseteq$ on **sum constructors**

$+^?$, $+$, $+_1$, $+_2$:



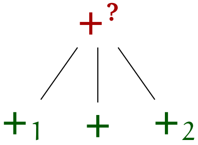
Defining precision

First, $\sqsubseteq$ on **sum constructors** $+^?$, $+$, $+_1$, $+_2$:

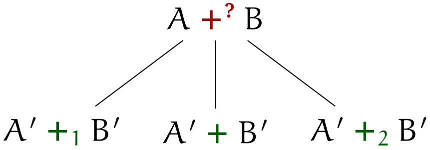
dynamic

$\sqsupseteq$

static

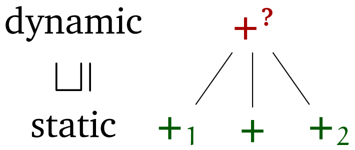


Extend $\sqsubseteq$ pointwise:
if $A' \sqsubseteq A$ and $B' \sqsubseteq B$ then...

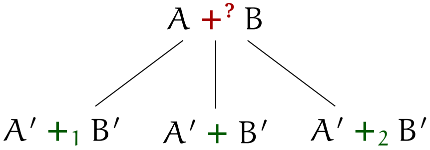


Defining precision

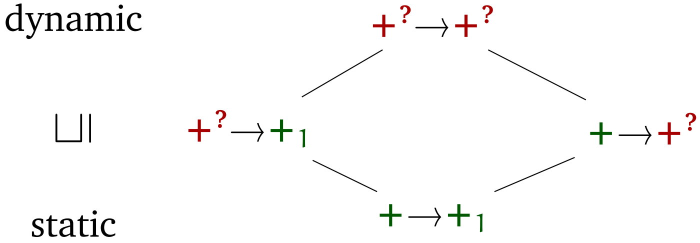
First, $\sqsubseteq$ on **sum constructors** $+^?$, $+$, $+_1$, $+_2$:



Extend $\sqsubseteq$ pointwise:
if $A' \sqsubseteq A$ and $B' \sqsubseteq B$ then...



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



Subsumption

- The usual subsumption rule:

$$\frac{\Gamma \vdash e : A \qquad A \leq B}{\Gamma \vdash e : B}$$

Subsumption

- The usual subsumption rule:

$$\frac{\Gamma \vdash e : A \qquad A \leq B}{\Gamma \vdash e : B}$$

- In a land of imprecision: "kinda A", "kinda B"

$$\frac{\Gamma \vdash e : A' \qquad A \sqsubseteq A' \qquad A \leq B \qquad B \sqsubseteq B'}{\Gamma \vdash e : B'}$$

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

$$\begin{array}{ccc} A' & & B' \\ \sqsupseteq & & \sqsupseteq \\ A & \leq & B \end{array}$$

Subsumption

- The usual subsumption rule:

$$\frac{\Gamma \vdash e : A \qquad A \leq B}{\Gamma \vdash e : B}$$

- In a land of imprecision: "kinda A", "kinda B"

$$\frac{\Gamma \vdash e : A' \qquad A \sqsubseteq A' \qquad A \leq B \qquad B \sqsubseteq B'}{\Gamma \vdash e : B'}$$

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

$$\begin{array}{ccc} A' & & B' \\ \sqsupseteq & & \sqsupseteq \\ A & \leq & B \end{array}$$

Subsumption

- The usual subsumption rule:

$$\frac{\Gamma \vdash e : A \qquad A \leq B}{\Gamma \vdash e : B}$$

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

$$\frac{\Gamma \vdash e : A' \qquad A \sqsubseteq A' \qquad A \leq B \qquad B \sqsubseteq B'}{\Gamma \vdash e : B'}$$

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

$$\begin{array}{ccc} A' & & B' \\ \sqcup\!\mid & & \sqcup\!\mid \\ A & \leq & B \end{array}$$

Subsumption

- The usual subsumption rule:

$$\frac{\Gamma \vdash e : A \qquad A \leq B}{\Gamma \vdash e : B}$$

- In a land of imprecision: "kinda A", "kinda B"

$$\frac{\Gamma \vdash e : A' \qquad A \sqsubseteq A' \qquad A \leq B \qquad B \sqsubseteq B'}{\Gamma \vdash e : B'}$$

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

$$\begin{array}{ccc} A' & & B' \\ \sqcup\!\vert & & \sqcup\!\vert \\ A & \leq & B \end{array}$$

Subsumption

- The usual subsumption rule:

$$\frac{\Gamma \vdash e : A \qquad A \leq B}{\Gamma \vdash e : B}$$

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

$$\frac{\Gamma \vdash e : A' \qquad A \sqsubseteq A' \qquad A \leq B \qquad B \sqsubseteq B'}{\Gamma \vdash e : B'}$$

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

$$\begin{array}{ccc} A' & & B' \\ \sqsupseteq & & \sqsupseteq \\ A & \leq & B \end{array}$$

Subsumption

- The usual subsumption rule:

$$\frac{\Gamma \vdash e : A \qquad A \leq B}{\Gamma \vdash e : B}$$

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

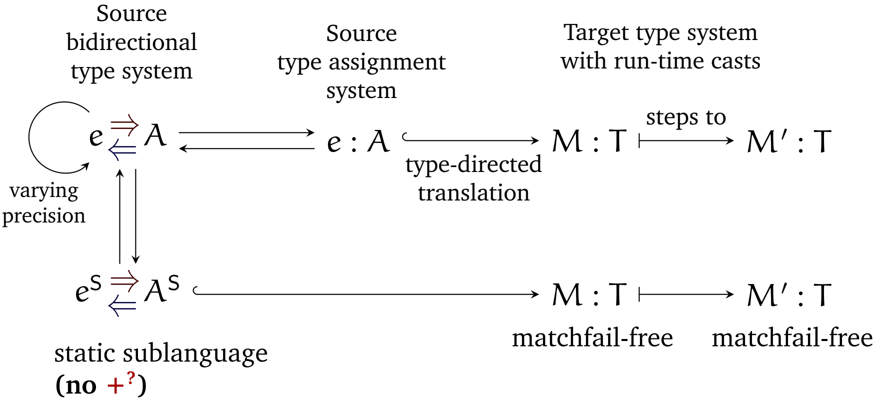
$$\frac{\Gamma \vdash e : A' \qquad A \sqsubseteq A' \qquad A \leq B \qquad B \sqsubseteq B'}{\Gamma \vdash e : B'}$$

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

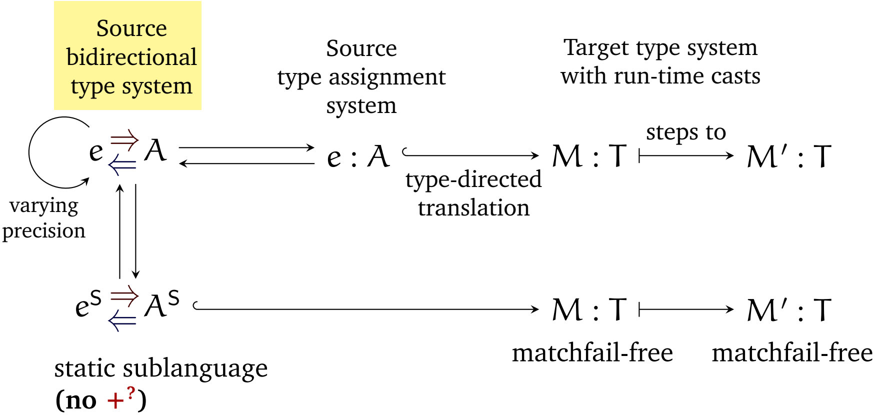
$$\begin{array}{ccc} A' & & B' \\ \sqsupseteq & & \sqsupseteq \\ A & \leq & B \end{array}$$

- Is directed consistency transitive?

Road map


Source
bidirectional
type system
Source
type assignment
system
Target type system
with run-time casts
e ⇒ ⇐ A
e : A
M : T
steps to
M′ : T
type-directed
translation
varying
precision
e^S ⇒ ⇐ A^S
M : T
M′ : T
matchfail-free
matchfail-free
static sublanguage
(no +?)

Road map



Source
bidirectional
type system
Source
type assignment
system
Target type system
with run-time casts
e ⇒ ⇐ A
e : A
M : T
steps to
M′ : T
type-directed
translation
varying
precision
e^S ⇒ ⇐ A^S
M : T
M′ : T
matchfail-free
matchfail-free
static sublanguage
(no +?)

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 Γ, e, and a known type A,
    check e:
    $$\Gamma \vdash e \Leftarrow A$$
  - Given Γ and e,
    synthesize a type for e:
    $$\Gamma \vdash e \Rightarrow A$$
- The type A in the checking judgment $e \Leftarrow A$ is a **goal**.

Bidirectional typing

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

$$\frac{\Gamma, x : A_1 \vdash e \Leftarrow A_2}{\Gamma \vdash \lambda x.\, e \Leftarrow A_1 \rightarrow A_2}\ \textsf{Chk}{\rightarrow}\textsf{Intro}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow (A \rightarrow B) \qquad \Gamma \vdash e_2 \Leftarrow A}{\Gamma \vdash e_1\, e_2 \Rightarrow B}\ \textsf{Syn}{\rightarrow}\textsf{Elim}$$

- 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 Γ.

Bidirectional typing

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

$$\frac{\Gamma, x : A_1 \vdash e \Leftarrow A_2}{\Gamma \vdash \lambda x.\, e \Leftarrow A_1 \to A_2}\ \textsf{Chk}{\to}\textsf{Intro}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow (A \to B) \qquad \Gamma \vdash e_2 \Leftarrow A}{\Gamma \vdash e_1\, e_2 \Rightarrow B}\ \textsf{Syn}{\to}\textsf{Elim}$$

- Chk→Intro:
  The type $A_1 \to A_2$ must flow from an annotation.
- Syn→Elim: The type $A \to B$ must flow from an annotation, perhaps via Γ.

Bidirectional typing

The subsumption rule:

$$\frac{\Gamma \vdash e \Rightarrow A' \qquad A' \rightsquigarrow B'}{\Gamma \vdash e \Leftarrow B'}$$

$$\begin{array}{ccc} A' & \rightsquigarrow & B' \\ \sqsupseteq & & \sqsupseteq \\ A & \leq & B \end{array}$$

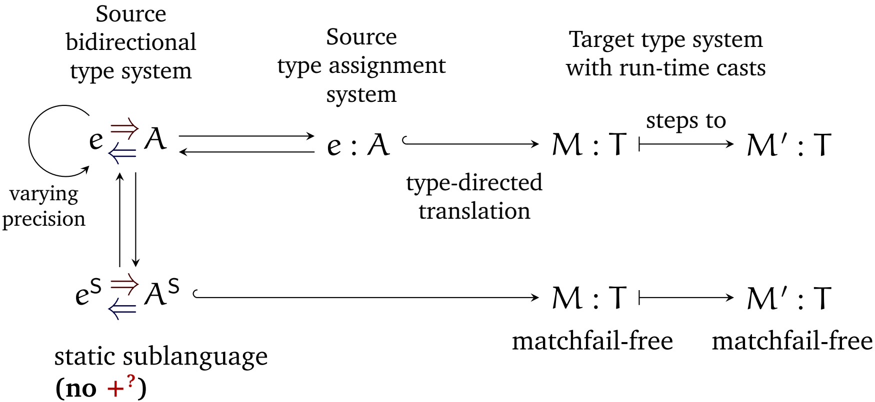
Bidirectional typing

The subsumption rule:

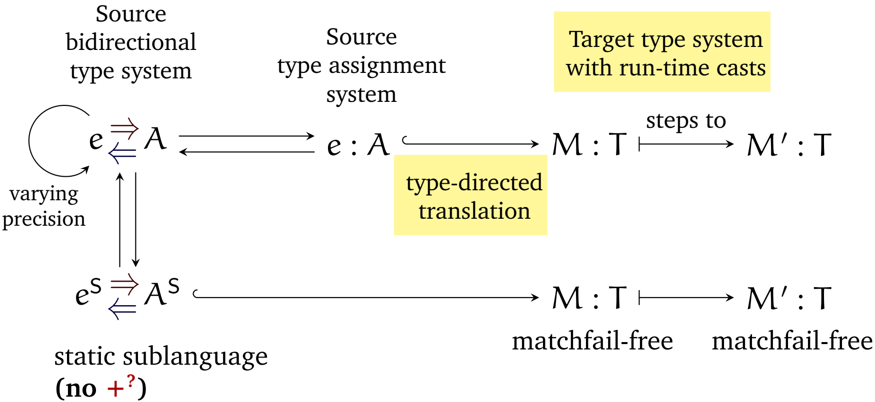
$$\frac{\Gamma \vdash e \Rightarrow A' \qquad A' \rightsquigarrow B'}{\Gamma \vdash e \Leftarrow B'} \qquad \begin{array}{ccc} A' & \rightsquigarrow & B' \\ \sqsupseteq & & \sqsupseteq \\ A & \leq & B \end{array}$$

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

Road map


Source
bidirectional
type system
Source
type assignment
system
Target type system
with run-time casts
e ⇒ ⇐ A
e : A
M : T
steps to
M′ : T
varying
precision
type-directed
translation
e^S ⇒ ⇐ A^S
M : T
M′ : T
matchfail-free
matchfail-free
static sublanguage
(no +?)

Road map


Source
bidirectional
type system
Source
type assignment
system
Target type system
with run-time casts
e ⇒⇐ A
e : A
M : T
steps to
M′ : T
varying
precision
type-directed
translation
e^S ⇒⇐ A^S
M : T
M′ : T
matchfail-free
matchfail-free
static sublanguage
(no +?)

Target language

- Target sum types include only **static** sums: $+$, $+_1$, $+_2$
- Casts between sums:

$$\langle +_1 \Leftarrow + \rangle(\mathtt{inj}_1\, v) \quad \text{will step to } \mathtt{inj}_1\, v$$

$$\langle +_2 \Leftarrow + \rangle(\mathtt{inj}_1\, v) \quad \text{will step to } \mathtt{matchfail}$$

Type-directed translation: add casts

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

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

$$\begin{array}{ccc} A' & \rightsquigarrow & B' \\ \sqsupseteq & & \sqsupseteq \\ A & \leq & B \end{array}$$

Type-directed translation: add casts

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

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

$$\begin{array}{ccc} A' & \rightsquigarrow & B' \\ \sqsubseteq & & \sqsubseteq \\ A & \leq & B \end{array}$$

$$\frac{\Gamma \vdash x : (\mathsf{Unit} +^{?} \mathsf{Unit}) \hookrightarrow x \qquad \begin{array}{l}(\mathsf{Unit} +^{?} \mathsf{Unit}) \rightsquigarrow (\mathsf{Unit} +_2 \mathsf{Unit}) \\ \hookrightarrow \langle +_2 \Leftarrow + \rangle [\,] \end{array}}{\Gamma \vdash x : B' \hookrightarrow \langle +_2 \Leftarrow + \rangle x}$$

Type-directed translation: add casts

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

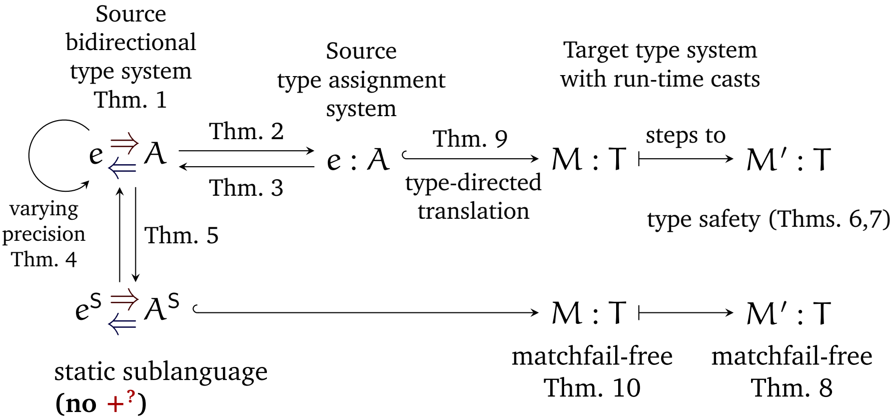
$$\frac{\Gamma \vdash e : A' \hookrightarrow M \qquad A' \rightsquigarrow B' \hookrightarrow \mathcal{C}}{\Gamma \vdash e : B' \hookrightarrow \mathcal{C}[M]}$$

$$\begin{array}{ccc} A' & \rightsquigarrow & B' \\ \sqsubseteq & & \sqsubseteq \\ A & \leq & B \end{array}$$

$$\frac{\Gamma \vdash x : (\mathsf{Unit} +^{?} \mathsf{Unit}) \hookrightarrow x \qquad \begin{array}{l}(\mathsf{Unit} +^{?} \mathsf{Unit}) \rightsquigarrow (\mathsf{Unit} +_2 \mathsf{Unit}) \\ \hookrightarrow \langle +_2 \Leftarrow + \rangle[\,]\end{array}}{\Gamma \vdash x : B' \hookrightarrow \langle +_2 \Leftarrow + \rangle x}$$

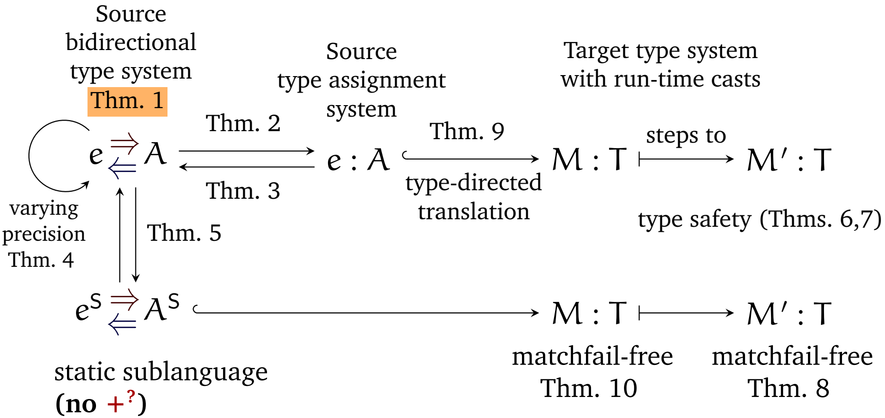
$$\begin{array}{ccc} \mathsf{Unit} +^{?} \mathsf{Unit} & \rightsquigarrow & \mathsf{Unit} +_2 \mathsf{Unit} \\ \sqsubseteq & & \sqsubseteq \\ \mathsf{Unit} +_2 \mathsf{Unit} & \leq & \mathsf{Unit} +_2 \mathsf{Unit} \end{array}$$

Metatheory



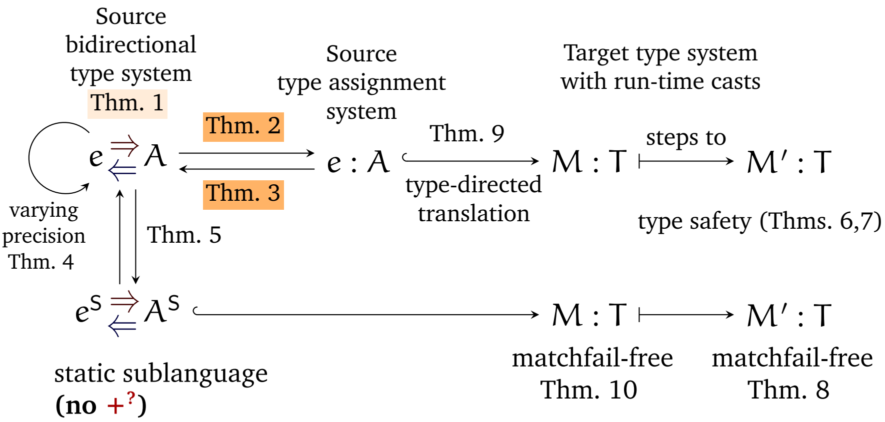
Source bidirectional type system Thm. 1
Source type assignment system
Target type system with run-time casts
varying precision Thm. 4
Thm. 2
Thm. 3
Thm. 5
Thm. 9
type-directed translation
steps to
type safety (Thms. 6,7)
static sublanguage
(no +?)
matchfail-free Thm. 10
matchfail-free Thm. 8

Metatheory

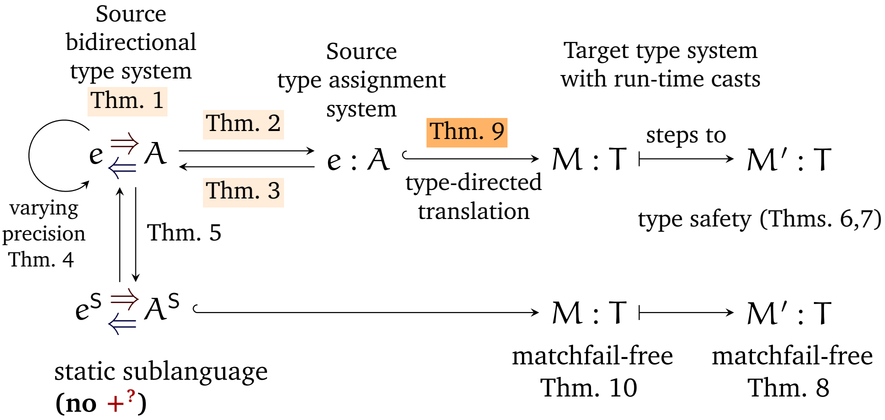


Source
bidirectional
type system
Thm. 1
$e \overset{\Rightarrow}{\Leftarrow} A$
varying
precision
Thm. 4
Thm. 2
Thm. 3
Source
type assignment
system
$e : A$
Thm. 5
Thm. 9
type-directed
translation
Target type system
with run-time casts
$M : T$
steps to
$M' : T$
type safety (Thms. 6,7)
$e^S \overset{\Rightarrow}{\Leftarrow} A^S$
$M : T$
$M' : T$
matchfail-free
Thm. 10
matchfail-free
Thm. 8
static sublanguage
(no $+^?$)

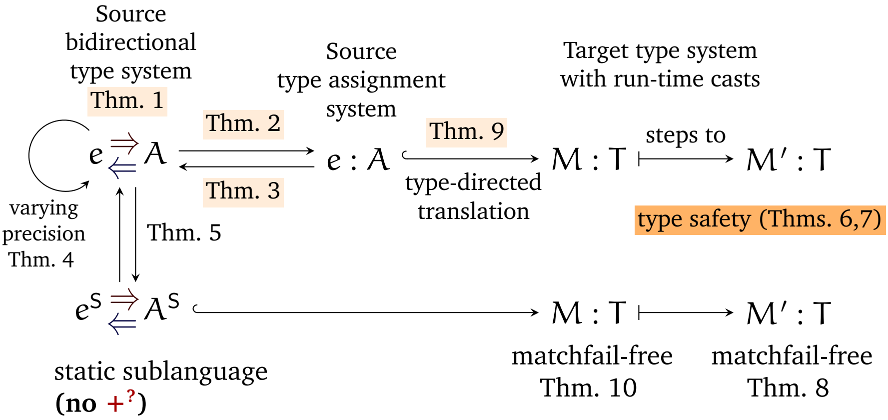
Metatheory


Source
bidirectional
type system
Thm. 1
Thm. 2
Source
type assignment
system
Target type system
with run-time casts
Thm. 9
steps to
Thm. 3
type-directed
translation
type safety (Thms. 6,7)
varying
precision
Thm. 4
Thm. 5
matchfail-free
Thm. 10
matchfail-free
Thm. 8
static sublanguage
(no +?)

Metatheory

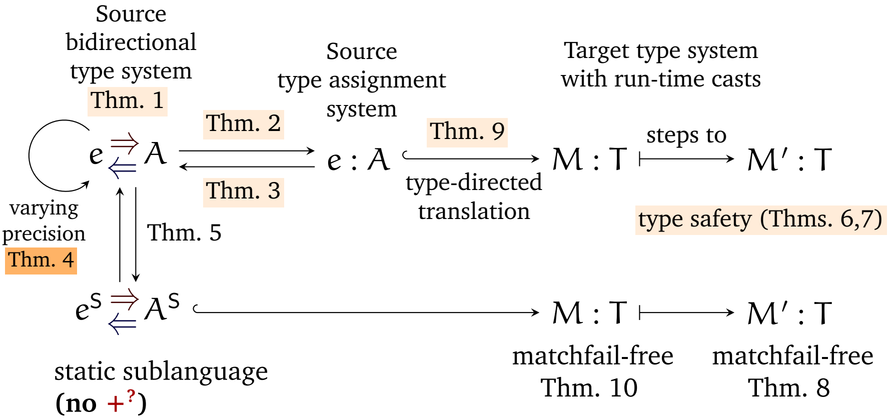

Source
bidirectional
type system
Thm. 1
Source
type assignment
system
Target type system
with run-time casts
Thm. 2
Thm. 3
Thm. 9
steps to
$e \Rightarrow A$ / $e \Leftarrow A$
$e : A$
$M : T$
$M' : T$
type-directed
translation
type safety (Thms. 6,7)
varying
precision
Thm. 4
Thm. 5
$e^S \Rightarrow A^S$ / $e^S \Leftarrow A^S$
$M : T$
$M' : T$
matchfail-free
Thm. 10
matchfail-free
Thm. 8
static sublanguage
(no $+^?$)

Metatheory

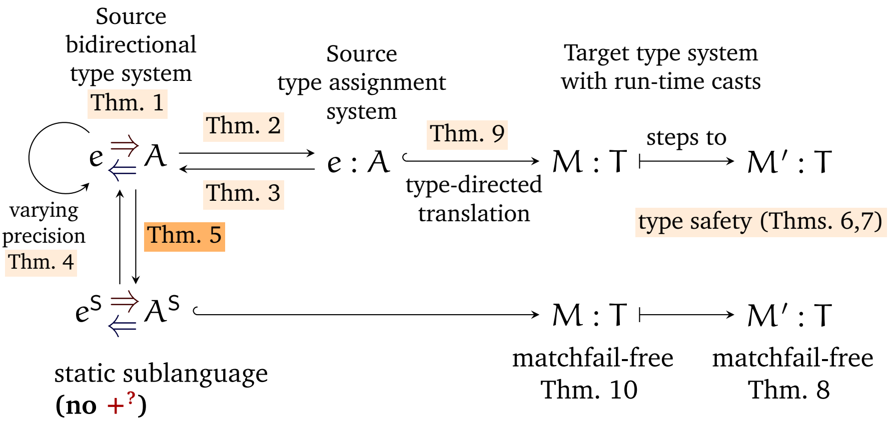


Source
bidirectional
type system
Thm. 1
Thm. 2
Source
type assignment
system
Target type system
with run-time casts
Thm. 9
steps to
e ⇒ ⇐ A
e : A
M : T
M′ : T
Thm. 3
type-directed
translation
type safety (Thms. 6,7)
varying
precision
Thm. 4
Thm. 5
e^S ⇒ ⇐ A^S
M : T
M′ : T
matchfail-free
Thm. 10
matchfail-free
Thm. 8
static sublanguage
(no +?)

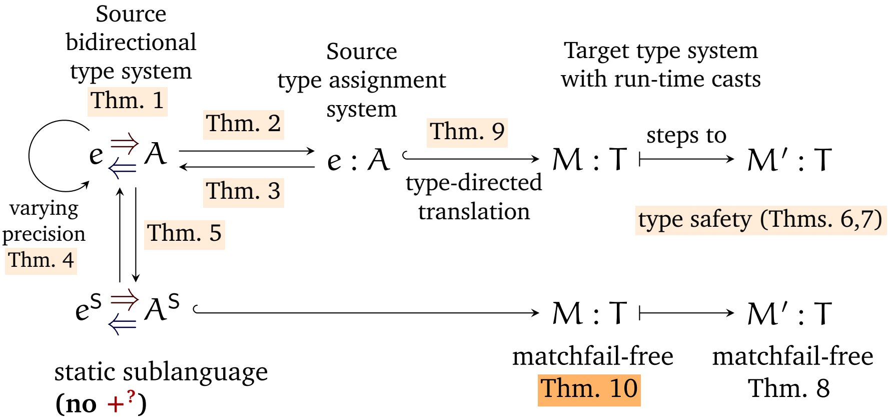
Metatheory


Source
bidirectional
type system
Thm. 1
Source
type assignment
system
Target type system
with run-time casts
Thm. 2
Thm. 9
steps to
$e \rightrightarrows A$
$e : A$
$M : T$
$M' : T$
Thm. 3
type-directed
translation
type safety (Thms. 6,7)
varying
precision
Thm. 4
Thm. 5
$e^S \rightrightarrows A^S$
$M : T$
$M' : T$
matchfail-free
Thm. 10
matchfail-free
Thm. 8
static sublanguage
(no $+^?$)

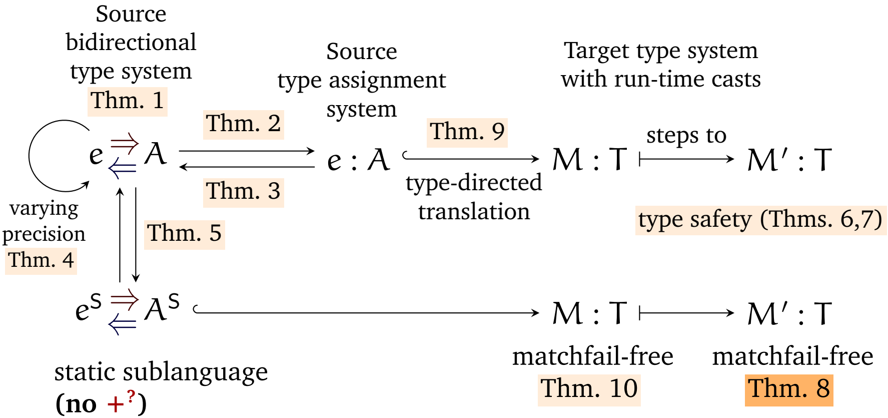
Metatheory


Source
bidirectional
type system
Thm. 1
Source
type assignment
system
Target type system
with run-time casts
Thm. 2
Thm. 3
Thm. 9
steps to
type-directed
translation
type safety (Thms. 6,7)
varying
precision
Thm. 4
Thm. 5
$e \Rightarrow A$
$e \Leftarrow A$
$e : A$
$M : T$
$M' : T$
$e^S \Rightarrow A^S$
$e^S \Leftarrow A^S$
matchfail-free
Thm. 10
matchfail-free
Thm. 8
static sublanguage
(no $+^?$)

Metatheory


Source bidirectional type system
Thm. 1
Thm. 2
Source type assignment system
Target type system with run-time casts
Thm. 9
steps to
e ⇒ ⇐ A
e : A
M : T
M′ : T
Thm. 3
type-directed translation
type safety (Thms. 6,7)
varying precision
Thm. 4
Thm. 5
e^S ⇒ ⇐ A^S
M : T
M′ : T
matchfail-free
Thm. 10
matchfail-free
Thm. 8
static sublanguage
(no +?)

Metatheory


Source bidirectional type system
Thm. 1
Thm. 2
Source type assignment system
Target type system with run-time casts
Thm. 9
steps to
e ⇒ ⇐ A
e : A
M : T
M′ : T
Thm. 3
type-directed translation
type safety (Thms. 6,7)
varying precision
Thm. 4
Thm. 5
e^S ⇒ ⇐ A^S
M : T
M′ : T
matchfail-free
Thm. 10
matchfail-free
Thm. 8
static sublanguage
(no +?)

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:
  Freeman & Pfenning 1991, Davies 2005, …
  $A \sqsubset \tau$ says refinement (**sort**) A refines **type** τ.
  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, ...
  $A \sqsubset \tau$ says refinement (**sort**) A refines **type** τ.
  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, $\forall$)
- Evaluate run-time efficiency

What's next?

- Implement the bidirectional system and translation
- Add more types (intersection, μ, $\forall$)
- 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`