Elaborating intersection and union types

Joshua Dunfield

MPI-SWS

Kaiserslautern and Saarbrücken, Germany

10 September 2012

ICFP 2012, Copenhagen

Outline

- ☞ **Motivation**
- Overview
- Source semantics
- Target semantics
- Elaboration
- Applications
- Related work
- Summary & future work

Motivation

- Type systems are great, but...
  - designing them is hard
  - implementing them is hard
- Each new type system feature is a burden
- This paper is about encoding type system features
  using **intersection and union types**,
  then **elaborating** them away

(and the real motivation)

- Intersection types are fun!

Approach

- Encode type system features **as intersections and unions**
  - Operator overloading:

$$+ \;:\; (\mathsf{int} * \mathsf{int} \longrightarrow \mathsf{int}) \wedge (\mathsf{real} * \mathsf{real} \longrightarrow \mathsf{real}) \wedge \;\ldots$$

- Need unrestricted intersection and union types
  and a **merge construct**
- **Elaborate** intersections and unions
  into simple products and sums

- **This talk** is about intersection types;
  for union types, see the paper

- ✓ Motivation
- ☞ **Overview**
  - Source semantics
  - Target semantics
  - Elaboration
  - Applications
  - Related work
  - Summary & future work

What is an intersection type?

- Something **conjunctive**, like set intersection

  $v : A \wedge B$ means v has type A **and** type B

  with these typing rules:

$$\frac{e : A_1 \qquad e : A_2}{e : A_1 \wedge A_2}\ \wedge\mathrm{I} \qquad \frac{e : A_1 \wedge A_2}{e : A_k}\ \wedge\mathrm{E}_k$$

- Can form $A \wedge B$ from arbitrary A and B
  (unlike previous work on type refinements!)

What is an intersection type?

- Something **conjunctive**, like set intersection

  $v : A \wedge B$ means v has type A **and** type B

  with these typing rules:

$$\frac{e : A_1 \qquad e : A_2}{e : A_1 \wedge A_2}\ \wedge\mathrm{I} \qquad \frac{e : A_1 \wedge A_2}{e : A_k}\ \wedge\mathrm{E}_k$$

- Can form $A \wedge B$ from arbitrary A and B
  (unlike previous work on type refinements!)
- Is it conjunction?
  - Product types correspond to conjunction

Is intersection a product type?

	Product $*$	Intersection $\wedge$
intro	has **multiple witnesses**	has **one witness**
	$\dfrac{e_1 : A_1 \qquad e_2 : A_2}{(e_1, e_2) : A_1 * A_2}\ *\mathrm{I}$	$\dfrac{e : A_1 \qquad e : A_2}{e : A_1 \wedge A_2}\ \wedge\mathrm{I}$
elim	has **explicit eliminations**	is **implicitly eliminated**
	$\dfrac{e : A_1 * A_2}{\mathbf{proj}_k\, e : A_k}\ *\mathrm{E}_k$	$\dfrac{e : A_1 \wedge A_2}{e : A_k}\ \wedge\mathrm{E}_k$

It's not a product type

- Not all intersections [of inhabited types] are inhabited:

$$\mathsf{int} \wedge (\mathsf{int} \to \mathsf{int})$$

  uninhabited because no v is both an integer and a function

- All products inhabited:

$$(0,\ \lambda x.\, x + 3)\ :\ \mathsf{int} * (\mathsf{int} \to \mathsf{int})$$

- ...

It's not a product type

- Not all intersections [of inhabited types] are inhabited:

$$\mathsf{int} \wedge (\mathsf{int} \to \mathsf{int})$$

  uninhabited because no v is both an integer and a function

- All products inhabited:

$$(0,\ \lambda x.\, x + 3)\ :\ \mathsf{int} * (\mathsf{int} \to \mathsf{int})$$

- But we can make all intersections inhabited by adding a **merge construct**

Merge construct

$$\frac{\Gamma \vdash e_k : A}{\Gamma \vdash e_1 ,, e_2 : A} \; (\exists k \in \{1, 2\})$$

Merge construct

$$\frac{\Gamma \vdash e_k : A}{\Gamma \vdash e_1 ,, e_2 : A}\ (\exists k \in \{1, 2\})$$

Example:

$$\dfrac{\dfrac{\cdot \vdash 0 : \mathsf{int}}{\cdot \vdash 0 ,, (\lambda x.\, x + 3) : \mathsf{int}} \qquad \dfrac{\cdot \vdash \lambda x.\, x + 3 : \mathsf{int} \to \mathsf{int}}{\cdot \vdash 0 ,, (\lambda x.\, x + 3) : \mathsf{int} \to \mathsf{int}}}{\cdot \vdash 0 ,, (\lambda x.\, x + 3) \ : \ \mathsf{int} \wedge (\mathsf{int} \to \mathsf{int})}\ \wedge \mathrm{I}$$

Merge construct

$$\frac{\Gamma \vdash e_k : A}{\Gamma \vdash e_1 ,, e_2 : A}\ (\exists k \in \{1,2\})$$

Example:

$$\frac{\dfrac{\cdot \vdash 0 : \mathsf{int}}{\cdot \vdash 0 ,, (\lambda x.\, x+3) : \mathsf{int}} \qquad \dfrac{\cdot \vdash \lambda x.\, x+3 : \mathsf{int} \to \mathsf{int}}{\cdot \vdash 0 ,, (\lambda x.\, x+3) : \mathsf{int} \to \mathsf{int}}}{\cdot \vdash 0 ,, (\lambda x.\, x+3) \ : \ \mathsf{int} \wedge (\mathsf{int} \to \mathsf{int})}\ \wedge\mathrm{I}$$

- Order irrelevant: $(\lambda x.\, x+3) ,, 0$ also OK
- **Not** an introduction form for $\wedge$
- Generalization of the merge construct in Forsythe (Reynolds 1988, 1996)

Merge construct

	Product $*$	Intersection $\wedge$
intro	has **multiple witnesses**	has one witness, with **two parts**

$$\frac{e_1 : A_1 \qquad e_2 : A_2}{(e_1\,,\ e_2) : A_1 * A_2}\ *\mathrm{I}$$

$$\frac{e_1 ,, e_2 : A_1 \qquad e_1 ,, e_2 : A_2}{e_1 ,, e_2 : A_1 \wedge A_2}\ \wedge\mathrm{I}$$

Elaborating $\wedge$, $\vee$

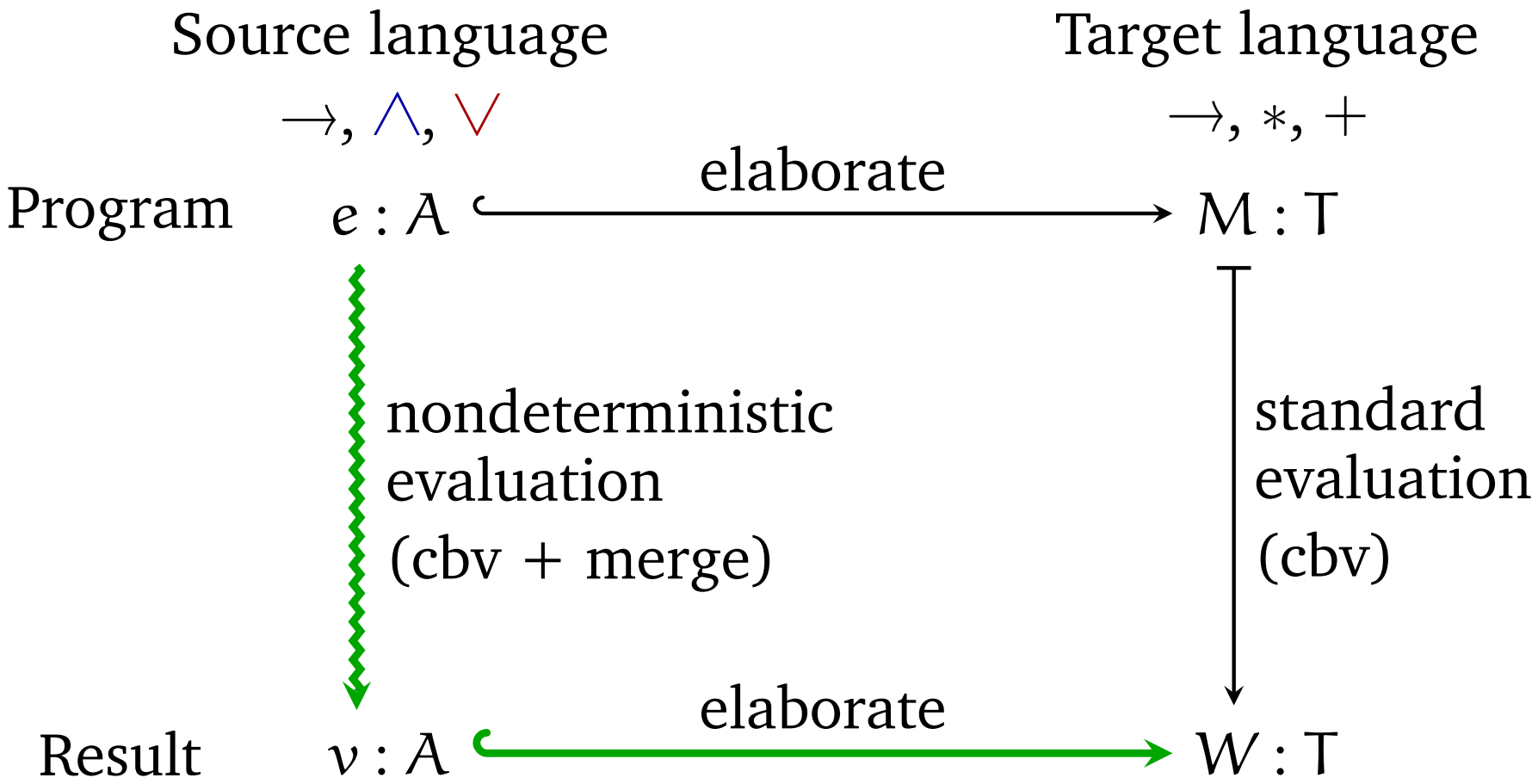
- Elaborate $\wedge$ to product and $\vee$ to disjoint sum

$$\underbrace{\mathsf{int} \wedge (\mathsf{int} \to \mathsf{int})}_{\text{type in source program}} \longrightarrow \underbrace{\mathsf{int} * (\mathsf{int} \to \mathsf{int})}_{\text{elaborated type in target program}}$$

- Old idea (Pierce, or earlier?), but never fully worked out
- Implicit $\wedge$-elimination becomes explicit $*$-elimination

$$\frac{e : A_1 \wedge A_2 \hookrightarrow M}{e : A_1 \hookrightarrow \mathbf{proj}_1\ M}\ \wedge E_1 \qquad \frac{M : A_1 * A_2}{(\mathbf{proj}_1\ M) : A_1}\ *E_1$$

Overview


Source language
$\to, \wedge, \vee$
Target language
$\to, *, +$
elaborate
Program
$e : A$
$M : T$
nondeterministic
evaluation
(cbv + merge)
standard
evaluation
(cbv)
elaborate
Result
$v : A$
$W : T$

- ✓ Motivation
- ✓ Overview
- ☞ **Source semantics**
  - Target semantics
  - Elaboration
  - Applications
  - Related work
  - Summary & future work

Source: syntax

$$
\begin{array}{rrcl}
\text{Source types} & A, B, C & ::= & \top \mid A \to B \mid A \wedge B \mid A \vee B \\
\text{Typing contexts} & \Gamma & ::= & \cdot \mid \Gamma, x : A \\
\text{Source expressions} & e & ::= & x \mid () \mid \lambda x.\, e \mid e_1\, e_2 \mid \mathbf{fix}\, x.\, e \\
 & & \mid & e_1 ,, e_2 \\
\text{Source values} & v & ::= & x \mid () \mid \lambda x.\, e \mid v_1 ,, v_2
\end{array}
$$

Source: dynamic semantics

$\boxed{e \rightsquigarrow e'}$ Source expression e steps to e'

$$\frac{e_1 \rightsquigarrow e_1'}{e_1 e_2 \rightsquigarrow e_1' e_2} \qquad \frac{e_2 \rightsquigarrow e_2'}{v_1 e_2 \rightsquigarrow v_1 e_2'} \qquad \frac{}{(\lambda x.\, e) v \rightsquigarrow [v/x]e}$$

$$\frac{}{\mathbf{fix}\ x.\, e \rightsquigarrow [(\mathbf{fix}\ x.\, e)/x]e}$$

Unmerge:

$$\frac{}{e_1 ,, e_2 \rightsquigarrow e_1} \qquad \frac{}{e_1 ,, e_2 \rightsquigarrow e_2}$$

Merge:

$$\frac{e_1 \rightsquigarrow e_1'}{e_1 ,, e_2 \rightsquigarrow e_1' ,, e_2} \qquad \frac{e_2 \rightsquigarrow e_2'}{e_1 ,, e_2 \rightsquigarrow e_1 ,, e_2'}$$

Split:

$$\frac{}{e \rightsquigarrow e ,, e}$$

Warning: Do not run

$$\frac{}{e_1 ,, e_2 \rightsquigarrow e_1} \qquad \frac{}{e_1 ,, e_2 \rightsquigarrow e_2}$$

$$\frac{e_1 \rightsquigarrow e_1'}{e_1 ,, e_2 \rightsquigarrow e_1' ,, e_2} \qquad \frac{e_2 \rightsquigarrow e_2'}{e_1 ,, e_2 \rightsquigarrow e_1 ,, e_2'}$$

- Therefore:

$$\big(0 ,, (\lambda x.\, x + 3)\big)\ 5 \ \rightsquigarrow\ 0\ 5$$

Warning: Do not run

$$\frac{}{e_1 ,, e_2 \rightsquigarrow e_1} \qquad \frac{}{e_1 ,, e_2 \rightsquigarrow e_2}$$

$$\frac{e_1 \rightsquigarrow e_1'}{e_1 ,, e_2 \rightsquigarrow e_1' ,, e_2} \qquad \frac{e_2 \rightsquigarrow e_2'}{e_1 ,, e_2 \rightsquigarrow e_1 ,, e_2'}$$

- Therefore:

$$\big(0 ,, (\lambda x.\, x + 3)\big)\ 5 \ \rightsquigarrow\ 0\ 5 \ \not\rightsquigarrow$$

ill-typed

- Every $e : A$ is a value, or
  there exists some e' such that $e \rightsquigarrow e'$ and $e' : A$.

$$\big(0 ,, (\lambda x.\, x + 3)\big)\ 5 \ \rightsquigarrow\ (\lambda x.\, x + 3)\ 5 \ \rightsquigarrow\ 5 + 3$$

- ✓ Motivation
- ✓ Overview
- ✓ Source semantics
- ☞ **Target semantics**
  - Elaboration
  - Applications
  - Related work
  - Summary & future work

Target language: cbv + products + sums

$$\begin{array}{rrl}
\text{Target types} & T & ::= \mathsf{unit} \mid T \to T \mid T * T \mid T + T \\
\text{Typing contexts} & G & ::= \cdot \mid G, x : T \\
\text{Target terms} & M, N & ::= x \mid () \mid \lambda x.\, M \mid M\, N \mid \mathbf{fix}\ x.\, M \\
 & & \quad \mid (M_1, M_2) \mid \mathbf{proj}_k\ M \\
 & & \quad \mid \mathbf{inj}_k\ M \mid \mathbf{case}\ M\ \mathbf{of}\ \mathbf{inj}_1\ x_1 \Rightarrow N_1 \\
 & & \quad\quad \| \ \mathbf{inj}_2\ x_2 \Rightarrow N_2 \\
\text{Target values} & W & ::= x \mid () \mid \lambda x.\, M \mid (W_1, W_2) \mid \mathbf{inj}_k\ W
\end{array}$$

$$\frac{G \vdash M : (T_1 * T_2)}{G \vdash (\mathbf{proj}_k\ M) : T_k} \qquad \frac{M \mapsto M'}{\mathbf{proj}_k\ M' \mapsto \mathbf{proj}_k\ M'}$$

$$\frac{}{\mathbf{proj}_k\ (W_1, W_2) \mapsto W_k}$$

- ✓ Motivation
- ✓ Overview
- ✓ Source semantics
- ✓ Target semantics
- ☞ **Elaboration**
  - Applications
  - Related work
  - Summary & future work

Elaboration

$\boxed{\Gamma \vdash e : A \hookrightarrow M}$ Source expr. e elaborates to target term M

$$\frac{\Gamma, x : A \vdash e : B \hookrightarrow M}{\Gamma \vdash \lambda x.\, e : A \to B \hookrightarrow \lambda x.\, M} \to\mathrm{I} \qquad \frac{\Gamma \vdash e_1 : A \to B \hookrightarrow M_1 \quad \Gamma \vdash e_2 : A \hookrightarrow M_2}{\Gamma \vdash e_1\, e_2 : B \hookrightarrow M_1\, M_2} \to\mathrm{E}$$

$$\frac{\Gamma \vdash e_k : A \hookrightarrow M}{\Gamma \vdash e_1 ,, e_2 : A \hookrightarrow M}\ \mathsf{merge}_k$$

$$\frac{\Gamma \vdash e : A_1 \hookrightarrow M_1 \quad \Gamma \vdash e : A_2 \hookrightarrow M_2}{\Gamma \vdash e : A_1 \wedge A_2 \hookrightarrow (M_1, M_2)} \wedge\mathrm{I} \qquad \frac{\Gamma \vdash e : A_1 \wedge A_2 \hookrightarrow M}{\Gamma \vdash e : A_k \hookrightarrow \mathbf{proj}_k\, M} \wedge\mathrm{E}_k$$

- If $e : A \hookrightarrow M$ then $M : |A|$, where $|{-}|$ replaces $\wedge$ with $*$

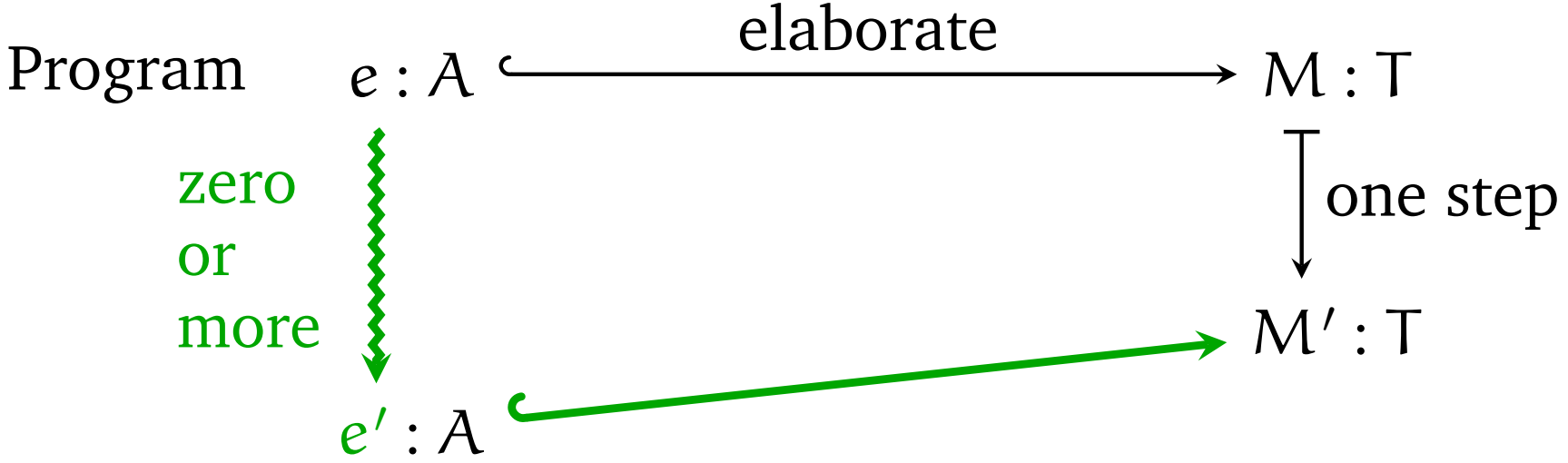
Incoherence

- Depending on which part the typechecker chooses,

$$3 ,, 4 \ : \ \mathsf{int}$$

  can elaborate to either 3 or 4
- Sound—3 and 4 both have type int
- But **incoherent**:
  evaluation depends on the whim of the typechecker!

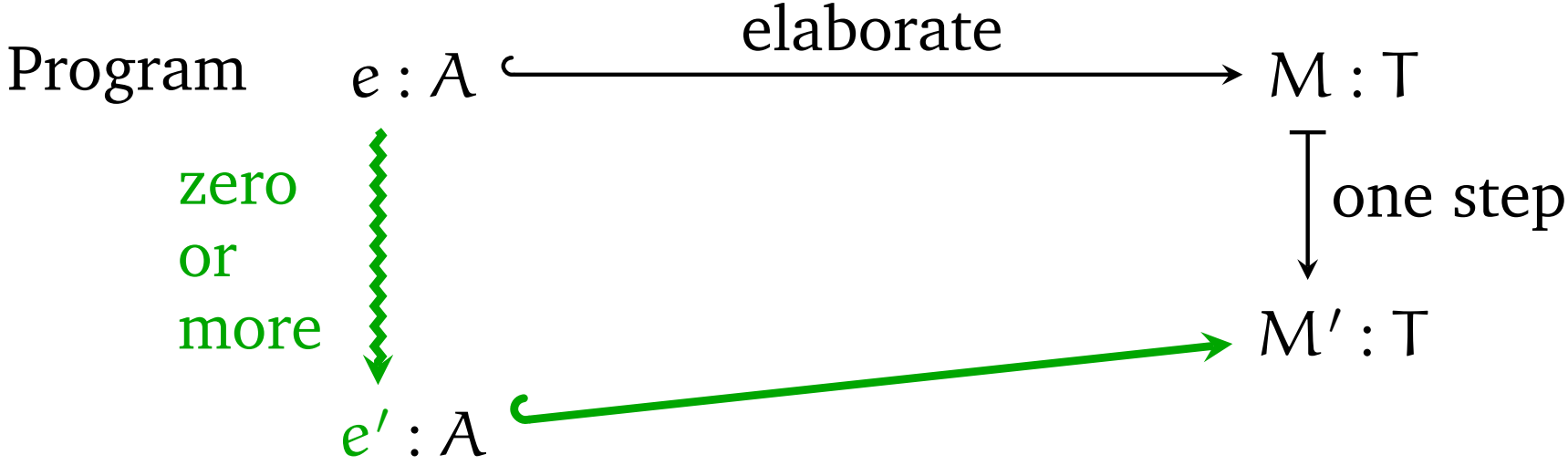
Consistency



If $\cdot \vdash e : A \hookrightarrow M$ and $M \mapsto M'$

then $\exists e'$ such that $e \rightsquigarrow^* e'$ and $\cdot \vdash e' : A \hookrightarrow M'$.

Consistency



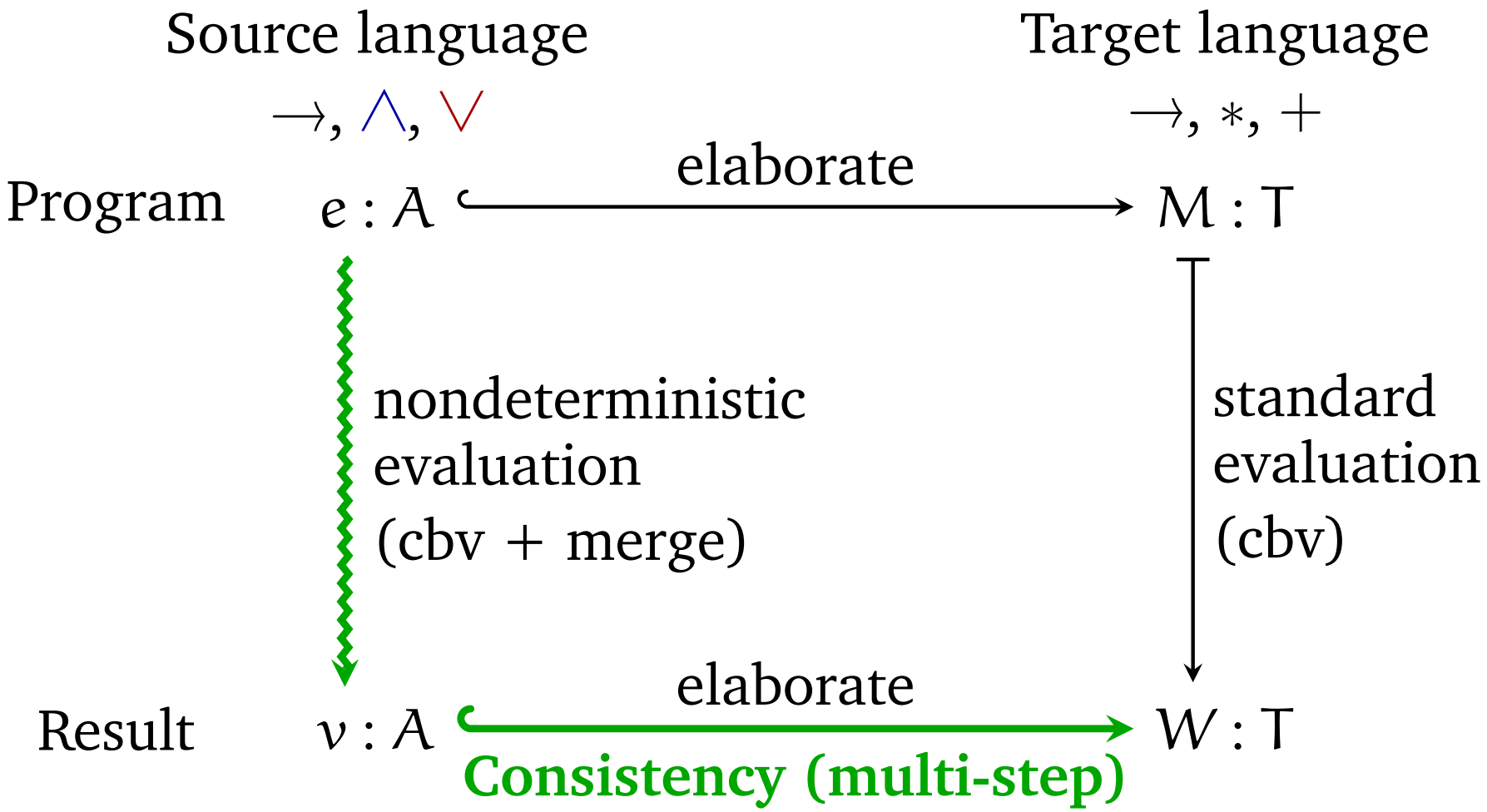
If $\cdot \vdash e : A \hookrightarrow M$ and $M \mapsto M'$

then $\exists e'$ such that $e \rightsquigarrow^* e'$ and $\cdot \vdash e' : A \hookrightarrow M'$.

Note that $e \rightsquigarrow^* e'$ not always one step:

- **zero** steps: $\mathbf{proj}_1\ (W_1, W_2) \mapsto W_1$
- $\geq$**2** steps: $(v_1 ,, \lambda x.\, x)\, v \hookrightarrow (\lambda x.\, x)\, W \mapsto W$
  but $(v_1 ,, \lambda x.\, x)\, v \rightsquigarrow (\lambda x.\, x)\, v \rightsquigarrow v$

Consistency, multi-step

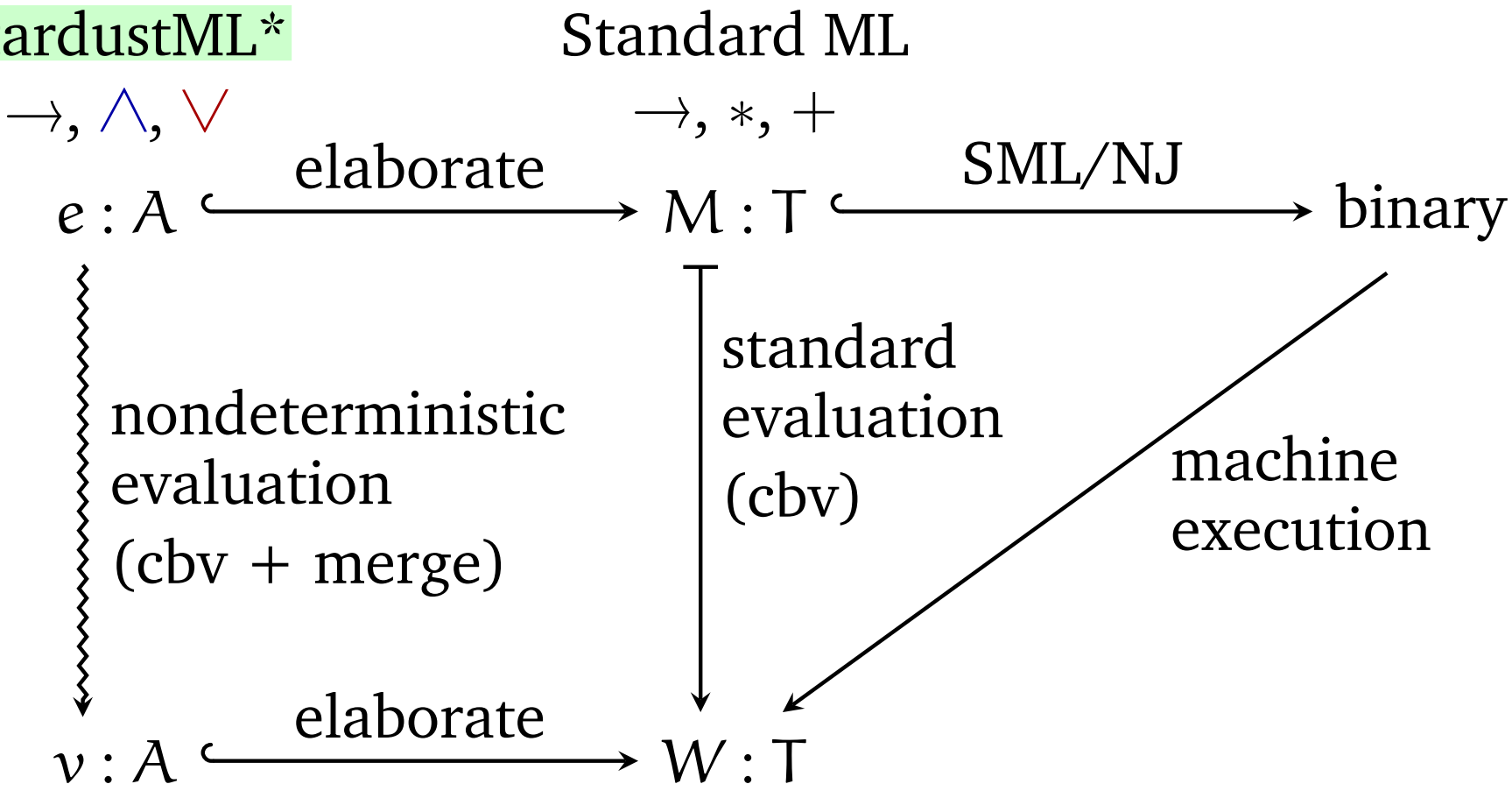


If $\cdot \vdash e : A \hookrightarrow M$ and $M \mapsto^* W$

then there exists v such that $e \rightsquigarrow^* v$ and $\cdot \vdash v : A \hookrightarrow W$.

- ✓ Motivation
- ✓ Overview
- ✓ Source semantics
- ✓ Target semantics
- ✓ Elaboration
- ☞ **Applications**
  - Related work
  - Summary & future work

Implementation



* StardustML: indexed types, refinement types,
first-class polymorphism, ...

Some applications

- **Overloading:**

$$+ \;:\; (\mathsf{int} * \mathsf{int} \longrightarrow \mathsf{int}) \wedge (\mathsf{real} * \mathsf{real} \longrightarrow \mathsf{real})$$

$$+ \;=\; \texttt{Int.+},, \texttt{Real.+}$$

- **Records:**
  Multi-field records as the intersection/merge of single-field records (Reynolds)
- **Heterogeneous data:** convenience with safety

Union types for heterogeneous data

```
type dyn = int ∨ real ∨ string

val toString : dyn → string
fun toString x =
  (Int.toString ,,
   (fn s ⇒ s : string) ,,
   Real.toString) x

val hetListToString : dyn list → string
fun hetListToString xs = case xs of
    nil  ⇒ "nil"
  | h::t ⇒ (toString h) ^ "::"
         ^ (hetListToString t)

hetListToString [1, 2, "what", 3.14159, 4, "why"]
  = "1::2::what::3.14159::4::why::nil"
```

- ✓ Motivation
- ✓ Overview
- ✓ Source semantics
- ✓ Target semantics
- ✓ Elaboration
- ✓ Applications
- ☞ **Related work**
- Summary & future work

Related work

- Early work on $\wedge$ and $\vee$: see paper
- Pierce (1991): idea of compiling $\wedge$ to $*$
- **Forsythe** (Reynolds 1988, '96): first practical language with $\wedge$; used a limited, coherent merge
- $\lambda\&$-calculus (Castagna et al. 1995): λ-bodies are merges; type-based dynamic semantics, without elaboration
- Flow types (Turbak et al.):
  internal virtual tuples and virtual sums
- Type refinements
  (Pfenning, Freeman, Davies, Dunfield, . . .)

- ✓ Motivation
- ✓ Overview
- ✓ Source semantics
- ✓ Target semantics
- ✓ Elaboration
- ✓ Applications
- ✓ Related work
- ☞ **Summary & future work**

Summary

- Various language features can be encoded with $\wedge$ and $\vee$ . . . which can be elaborated away
- Elaboration produces terms consistent with the (impractical) dynamic semantics of the source language
- The current system lacks coherence

\+ In the paper:

  - union types
  - subtyping
  - bidirectional typing (inference is undecidable)
  - implementation

Future work

- Current approach is **incoherent**:

$$3 ,, 4 \ : \ \mathsf{int}$$

  could elaborate to either 3 or 4

  - Runtime behaviour still sound, but unpredictable
- Solution (?): a merge that **prefers the second part**, allowing e.g. functional record update:

$$r ,, \{\mathtt{fld} = e\}$$

  - Need a notion of **type difference** to "subtract" the behaviour to be overridden

Thank you

- And thanks to
  - the ICFP reviewers
  - Adam Megacz
  - Yan Chen, Matthew A. Hammer, Scott Kilpatrick, Neelakantan R. Krishnaswami, Viktor Vafeiadis

- Further reading:
  `http://www.cs.cmu.edu/~joshuad/papers/intcomp/` (paper)
  `http://www.cs.cmu.edu/~joshuad/intcomp/` (Twelf proofs)