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:
    \[ + : (\text{int} \times \text{int} \rightarrow \text{int}) \wedge (\text{real} \times \text{real} \rightarrow \text{real}) \wedge \ldots \]

- 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
  
  \[ \nu : A \land B \] means \( \nu \) has type \( A \) **and** type \( B \)

  with these typing rules:

  \[
  \begin{align*}
  &e : A_1 && e : A_2 \\
  \hline
  &e : A_1 \land A_2 \\
  \end{align*}
  \]

  \( \land I \)

  \[
  \begin{align*}
  &e : A_1 \land A_2 \\
  \hline
  &e : A_k \\
  \end{align*}
  \]

  \( \land E_k \)

- Can form \( A \land B \) from arbitrary \( A \) and \( B \)
  
  (unlike previous work on type refinements!)
What is an intersection type?

• Something conjunctive, like set intersection

\[ \nu : A \land B \] means \( \nu \) has type \( A \) and type \( B \)

with these typing rules:

\[
\begin{align*}
& \frac{e : A_1 \quad e : A_2}{e : A_1 \land A_2} \quad \land I \\
& \frac{e : A_1 \land A_2}{e : A_k} \quad \land E_k
\end{align*}
\]

• Can form \( A \land B \) 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 $\ast$

intro has **multiple witnesses**

$\frac{e_1 : A_1 \quad e_2 : A_2}{(e_1, e_2) : A_1 \ast A_2}$ $\ast I$

elim has **explicit eliminations**

$\frac{e : A_1 \ast A_2}{\text{proj}_k e : A_k}$ $\ast E_k$

Intersection $\wedge$

has **one witness**

$\frac{e : A_1 \quad e : A_2}{e : A_1 \wedge A_2}$ $\wedge I$

is implicitly eliminated

$\frac{e : A_1 \wedge A_2}{e : A_k}$ $\wedge E_k$
It’s not a product type

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

\[ \text{int} \land (\text{int} \rightarrow \text{int}) \]

uninhabited because no \( \nu \) is both an integer and a function

- All products inhabited:

\[ (0, \lambda x. x + 3) : \text{int} \ast (\text{int} \rightarrow \text{int}) \]

- \( \ldots \)
It’s not a product type

- Not all intersections [of inhabited types] are inhabited:
  \[
  \text{int} \land (\text{int} \to \text{int})
  \]
  uninhabited because no \( v \) is both an integer and a function

- All products inhabited:
  \[
  (0, \lambda x. x + 3) : \text{int} \ast (\text{int} \to \text{int})
  \]

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

\[
\Gamma \vdash e_k : A \\
\Gamma \vdash e_1, e_2 : A \\
\therefore \quad (\exists k \in \{1, 2\})
\]
Merge construct

\[ \Gamma \vdash e_k : A \]
\[ \Gamma \vdash e_1, e_2 : A \quad (\exists k \in \{1, 2\}) \]

Example:

\[ \cdot \vdash 0 : \text{int} \quad \cdot \vdash \lambda x. x + 3 : \text{int} \rightarrow \text{int} \]
\[ \cdot \vdash 0, (\lambda x. x + 3) : \text{int} \quad \cdot \vdash 0, (\lambda x. x + 3) : \text{int} \rightarrow \text{int} \]
\[ \cdot \vdash 0, (\lambda x. x + 3) : \text{int} \land (\text{int} \rightarrow \text{int}) \]
\[ \land I \]
Merge construct

\[ \Gamma \vdash e_k : A \]
\[ \Gamma \vdash e_1,, e_2 : A \quad (\exists k \in \{1, 2\}) \]

Example:

\[ \cdot \vdash 0 : \text{int} \]
\[ \cdot \vdash 0,, (\lambda x. x+3) : \text{int} \]
\[ \cdot \vdash 0,, (\lambda x. x+3) : \text{int} \rightarrow \text{int} \]
\[ \cdot \vdash (\lambda x. x+3) : \text{int} \rightarrow \text{int} \]
\[ \vdash 0,, (\lambda x. x+3) : \text{int} \land \text{(int} \rightarrow \text{int}) \]

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

Product $\ast$

- intro has **multiple witnesses**
- $e_1 : A_1$  $e_2 : A_2$
- $(e_1, e_2) : A_1 \ast A_2$

Intersection $\land$

- has one witness, with **two parts**
- $e_1, e_2 : A_1$
- $e_1, e_2 : A_2$
- $(e_1, e_2) : A_1 \land A_2$
**Elaborating** ∧, ∨

- Elaborate ∧ to product and ∨ to disjoint sum

\[
\text{int} \land (\text{int} \rightarrow \text{int}) \quad \longrightarrow \quad \text{int} \star (\text{int} \rightarrow \text{int})
\]

  type in source program  elaborated type in target program

- Old idea (Pierce, or earlier?), but never fully worked out

- Implicit ∧-elimination becomes explicit ∗-elimination

\[
e : A_1 \land A_2 \quad \leftrightarrow \quad M \\
\quad \quad \text{∧E}_1
\]
\[
e : A_1 \quad \leftrightarrow \quad \text{proj}_1 M
\]
\[
M : A_1 \star A_2 \quad \rightarrow \quad \text{∗E}_1
\]
\[
(\text{proj}_1 M) : A_1
\]
Overview

Program

Source language
$\rightarrow, \land, \lor$

Result

$\nu : A \xleftarrow{\text{elaborate}} W : T$

$nondeterministic evaluation (cbv + merge)$

Target language
$\rightarrow, *, +$

$\epsilon : A \xrightarrow{\text{elaborate}} M : T$

$\text{standard evaluation (cbv)}$
✓ Motivation
✓ Overview
☞ Source semantics
  • Target semantics
  • Elaboration
  • Applications
  • Related work
  • Summary & future work
Source: syntax

Source types \( A, B, C \ ::= \top \mid A \to B \mid A \land B \mid A \lor B \)

Typing contexts \( \Gamma ::= \cdot \mid \Gamma, x : A \)

Source expressions \( e ::= x \mid () \mid \lambda x. e \mid e_1 e_2 \mid \text{fix } x. e \)
\[ \mid e_1,\ldots, e_2 \]

Source values \( v ::= x \mid () \mid \lambda x. e \mid v_1,\ldots, v_2 \)
Source: dynamic semantics

\[
\begin{align*}
& e \leadsto e' \quad \text{Source expression } e \text{ steps to } e' \\
& e_1 \leadsto e'_1, \quad e_2 \leadsto e'_2, \quad \lambda x. e \leadsto [v/x]e \\
& \overset{\text{fix } x.}{\text{Unmerge:}} e_1, e_2 \leadsto e_1, e_1, e_2 \leadsto e_2 \\
& \overset{\text{Merge:}}{\text{Split:}} e \leadsto e, e
\end{align*}
\]
Warning: Do not run

\[
\begin{align*}
\text{e}_1,, \text{e}_2 &\rightsquigarrow \text{e}_1 & \text{e}_1,, \text{e}_2 &\rightsquigarrow \text{e}_2 \\
\text{e}_1 &\rightsquigarrow \text{e'}_1 & \text{e}_2 &\rightsquigarrow \text{e'}_2 \\
\text{e}_1,, \text{e}_2 &\rightsquigarrow \text{e'}_1,, \text{e}_2 & \text{e}_1,, \text{e}_2 &\rightsquigarrow \text{e}_1,, \text{e'}_2
\end{align*}
\]

• Therefore:

\[
(0,, (\lambda x. x + 3)) \; 5 \rightsquigarrow 0 \; 5
\]
Warning: Do not run

\[
\begin{align*}
\quad & e_1,, e_2 \leadsto e_1 & \quad & e_1,, e_2 \leadsto e_2 \\
\quad & e_1 \leadsto e'_1 & \quad & e_2 \leadsto e'_2 \\
\hline
\quad & e_1,, e_2 \leadsto e'_1,, e_2 & \quad & e_1,, e_2 \leadsto e_1,, e'_2 \\
\end{align*}
\]

- Therefore:

\[
(0,, (\lambda x. x + 3)) 5 \leadsto 0 5 \not\leadsto \quad \text{ill-typed}
\]

- Every \( e : A \) is a value, or there exists some \( e' \) such that \( e \leadsto e' \) and \( e' : A \).

\[
(0,, (\lambda x. x + 3)) 5 \leadsto (\lambda x. x + 3) 5 \leadsto 5 + 3
\]
✓ Motivation
✓ Overview
✓ Source semantics
陴 Target semantics
  • Elaboration
  • Applications
  • Related work
  • Summary & future work
Target language: cbv + products + sums

Target types
\[ T ::= \text{unit} \mid T \to T \mid T \times T \mid T + T \]

Typing contexts
\[ G ::= \cdot \mid G, x : T \]

Target terms \( M, N \) ::= \( x \mid () \mid \lambda x.\ M \mid M \, N \mid \text{fix} \, x.\ M \mid (M_1, M_2) \mid \text{proj}_k \, M \mid \text{inj}_k \, M \mid \text{case} \, M \, \text{of} \, \text{inj}_1 \, x_1 \Rightarrow N_1 \mid \text{inj}_2 \, x_2 \Rightarrow N_2 \]

Target values \( W \) ::= \( x \mid () \mid \lambda x.\ M \mid (W_1, W_2) \mid \text{inj}_k \, W \)

\[
\begin{align*}
G \vdash M : (T_1 \times T_2) & \quad \quad \quad \quad \quad \quad M \mapsto M' \\
G \vdash (\text{proj}_k \, M) : T_k & \quad \quad \quad \quad \quad \quad \text{proj}_k \, M' \mapsto \text{proj}_k \, M' \\
\text{proj}_k \, (W_1, W_2) & \mapsto W_k
\end{align*}
\]
✓ Motivation
✓ Overview
✓ Source semantics
✓ Target semantics

Elaboration
  • Applications
  • Related work
  • Summary & future work
Elaboration

\[ \Gamma \vdash e : A \rightsquigarrow M \] Source expr. e elaborates to target term M

\[
\frac{\Gamma, x : A \vdash e : B \rightsquigarrow M}{\Gamma \vdash \lambda x. e : A \rightarrow B \rightsquigarrow \lambda x. M} \quad \rightarrow I
\]

\[
\frac{\Gamma \vdash e_1 : A \rightarrow B \rightsquigarrow M_1}{\Gamma \vdash e_2 : A \rightsquigarrow M_2} \quad \rightarrow E
\]

\[
\frac{\Gamma \vdash e_k : A \rightsquigarrow M}{\Gamma \vdash e_1,, e_2 : A \rightsquigarrow M} \quad \text{merge}_k
\]

\[
\frac{\Gamma \vdash e : A_1 \rightsquigarrow M_1}{\Gamma \vdash e : A_2 \rightsquigarrow M_2} \quad \wedge I
\]

\[
\frac{\Gamma \vdash e : A_1 \wedge A_2 \rightsquigarrow (M_1, M_2)}{\Gamma \vdash e : A_k \rightsquigarrow \text{proj}_k M} \quad \wedge E_k
\]

- If \( e : A \rightsquigarrow M \) then \( M : |A| \), where \(|-|\) replaces \( \wedge \) with \(*\)
Incoherence

- Depending on which part the typechecker chooses,
  
  $3, 4 : \text{int}$

  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 \( \cdot \vdash e : A \rightarrow M \) and \( M \leftrightarrow M' \)

then \( \exists e' \text{ such that } e \rightarrow^* e' \) and \( \cdot \vdash e' : A \rightarrow M' \).
Consistency

Program $e : A \xrightarrow{\text{elaborate}} M : T$

zero or more

$e' : A$

If $\vdash e : A \rightarrow M$ and $M \rightarrow M'$

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

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

* **zero** steps: $\text{proj}_1 (W_1, W_2) \rightarrow W_1$

* $\geq 2$ steps: $(v_1, \lambda x. x) \text{v} \leftrightarrow (\lambda x. x) W \rightarrow W$
  but $(v_1, \lambda x. x) \text{v} \rightsquigarrow (\lambda x. x) \text{v} \rightsquigarrow \text{v}$
Consistency, multi-step

Source language: $\rightarrow, \land, \lor$

Program: $e : A$ $\xleftarrow{\text{elaborate}} M : T$

Target language: $\rightarrow, *, +$

Result: $\nu : A$ $\xleftarrow{\text{elaborate}} W : T$

Consistency (multi-step)

If $\cdot \vdash e : A \leftrightarrow M$ and $M \rightarrow^* W$

then there exists $\nu$ such that $e \sim^* \nu$ and $\cdot \vdash \nu : A \leftrightarrow W$. 
✓ Motivation
✓ Overview
✓ Source semantics
✓ Target semantics
✓ Elaboration
☞ Applications
  • Related work
  • Summary & future work
StardustML*: indexed types, refinement types, first-class polymorphism, ...

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

\[ e : A \quad \rightarrow \quad M : T \quad \rightarrow \quad binary \]

\[ v : A \quad \rightarrow \quad W : T \quad \rightarrow \quad SML/NJ \]

elaborate

nondeterministic evaluation (cbv + merge)

standard evaluation (cbv)

machine execution

\[ \rightarrow, \wedge, \vee \]
Some applications

• Overloading:

\[ + : (\text{int} \times \text{int} \rightarrow \text{int}) \land (\text{real} \times \text{real} \rightarrow \text{real}) \]
\[ + = \text{Int.}+.\text{Real.}+ \]

• Records:
  Multi-field records as the intersection/merge of single-field records (Reynolds)

• Heterogeneous data: convenience with safety
Union types for heterogeneous data

```ml
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 $\land$ and $\lor$: see paper
- Pierce (1991): idea of compiling $\land$ to $\ast$
- Forsythe (Reynolds 1988, ’96): first practical language with $\land$; used a limited, coherent merge
- $\lambda$&-calculus (Castagna et al. 1995): $\lambda$-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 $\land$ and $\lor$ ... 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:

  \[3, 4 : \text{int}\]

  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:

  \[r, \{\text{fld}=e\}\]

- Need a notion of type difference to “subtract” the behaviour to be overridden
Thank you

- And thanks to
  - the ICFP reviewers
  - Adam Megacz

- Further reading:
  - [http://www.cs.cmu.edu/~joshuad/intcomp/](http://www.cs.cmu.edu/~joshuad/intcomp/) (Twelf proofs)