Annotations for intersection typechecking

Joshua Dunfield

MPI-SWS

Kaiserslautern and Saarbrücken, Germany

29 June 2012

ITRS ’12, Dubrovnik
Outline

- Overview
  - Elaborating unrestricted ∧
  - Annotations: overview
  - Contextual typing annotations
  - New annotations
  - Summary
Context

• This paper came out of a recent project:
  • Encode language features using intersection and union
  • Elaborate away intersections and unions
• But ultimately applicable to older work on type refinements, $\land$, $\lor$
Background

- $\land (+ \lor)$ with refinement restriction
  (Freeman & Pfenning, Davies & Pfenning, Dunfield & Pfenning)
- Use types to express (more) precise properties
- Contextual typing annotations
Background

• $\land (\land \lor \lor)$ with refinement restriction 
  (Freeman & Pfenning, Davies & Pfenning, 
  Dunfield & Pfenning)
  • Use types to express (more) precise properties
  • Contextual typing annotations

• $\land + \lor$ without refinement restriction 
  (ICFP 2012)
  • Use types to encode language features
  • Merges lead to new type annotation forms
Background

- $\land (+ \lor)$ with refinement restriction
  (Freeman & Pfenning, Davies & Pfenning,
  Dunfield & Pfenning)
  - Use types to express (more) precise properties
  - Contextual typing annotations

- $\land + \lor$ \textbf{without} refinement restriction
  (ICFP 2012)
  - Use types to encode language features
  - Merges lead to new type annotation forms
  (this paper)
Refinement restriction:

- $A \land B$ is a well-formed type only if $A$ and $B$ refine the same base type
  
  $(\text{even} \rightarrow \text{even}) \land (\text{odd} \rightarrow \text{odd})$ ✓ both refine \text{bits} $\rightarrow$ \text{bits}

  $\text{even} \land \text{odd}$ ✓ both refine \text{bits}

  $\text{int} \land (\text{int} \rightarrow \text{int})$ ✗ no common base type

- With this restriction:
  - More is decidable
  - Compilation can ignore refinements and $\land$: compilation is independent of refined typechecking
Some restricted systems

- $\land + \text{refinement types}$  
  (Freeman & Pfenning, Davies & Pfenning)

- $\land + \lor + \text{refinement types} + \text{indexed types}$  
  (Dunfield & Pfenning)
Some restricted systems

- $\land +$ refinement types
  (Freeman & Pfenning, Davies & Pfenning)

- $\land + \lor +$ refinement types + indexed types
  (Dunfield & Pfenning)

- But only implicitly enforced: can’t compile programs that don’t follow the restriction
Without the restriction:

- Forsythe-style encodings, e.g. records as $\wedge$ of one-field records
- Operator overloading
- Heterogeneous containers (via $\vee$)
- Entangles typechecking with compilation
Without the restriction:

- Forsythe-style encodings, e.g. records as $\land$ of one-field records
- Operator overloading
- Heterogeneous containers (via $\lor$)
- Entangles typechecking with compilation … but that’s just elaboration!
Without the restriction:

- Forsythe-style encodings, e.g. records as \( \land \) of one-field records
- Operator overloading
- Heterogeneous containers (via \( \lor \))
- Entangles typechecking with compilation  
  … but that’s just elaboration!

Next section: Elaborating \( \land \) (ICFP 2012).

Not the focus of this paper, but provides context.
Overview

Elaborating unrestricted ∧

- Annotations: overview
- Contextual typing annotations
- New annotations
- Summary
Elaboration for $\land$, $\lor$

- Elaborate $\land$ to product and $\lor$ to disjoint sum

\[
\begin{align*}
\text{int} \land (\text{int} \to \text{int}) & \quad \longrightarrow \quad \text{int} \times (\text{int} \to \text{int}) \\
\text{type in source program} & \quad \text{elaborated type in target program}
\end{align*}
\]

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

- Implicit $\land$-elimination becomes explicit $\ast$-elimination

\[
\begin{align*}
e : A_1 \land A_2 & \quad \leadsto \quad M \\
\land E_1 & \quad e : A_1 \quad \leadsto \quad \text{proj}_1 \ M \\
M : A_1 \ast A_2 & \quad \ast E_1 \\
\text{(proj}_1 \ M) : A_1
\end{align*}
\]
Elaboration for $\land$, $\lor$: Merge

- We can form the type $\text{int} \land (\text{int} \rightarrow \text{int})$, but is it inhabited?
Elaboration for $\land$, $\lor$: Merge

- We can form the type $\text{int} \land (\text{int} \rightarrow \text{int})$, but is it inhabited?
- Depends on the term language!
Elaboration for $\land$, $\lor$: Merge

- We can form the type $\text{int} \land (\text{int} \rightarrow \text{int})$, but is it inhabited?
- Depends on the term language!
- For us, the answer will be yes, but only because of a merge construct

$$e_1 ,, e_2$$
Elaboration for $\land$, $\lor$: Merge

- We can form the type $\text{int} \land (\text{int} \rightarrow \text{int})$, but is it inhabited?
- Depends on the term language!
- For us, the answer will be yes, but only because of a merge construct

\[
e_1 , , e_2
\]

- Above type inhabited: $0 , , (\lambda x. x + 3)$
Elaboration for $\land$, $\lor$: Merge

- We can form the type $\text{int} \land (\text{int} \to \text{int})$, but is it inhabited?
- Depends on the term language!
- For us, the answer will be yes, but only because of a merge construct

$$e_1 \& e_2$$

- Above type inhabited: $0\&(\lambda x. x + 3)$
- Generalization of the merge construct in Forsythe (Reynolds 1988, 1996)
Merge rule

\[
\begin{align*}
\Gamma & \vdash e_k : \Lambda \\
\hline
\Gamma & \vdash e_1 \,,
\,,
\, e_2 : \Lambda \\
\hline
(\exists k \in \{1, 2\})
\end{align*}
\]
Merge rule

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

Example:

\[
\begin{aligned}
\cdot \vdash 0 : \text{int} \\
\cdot \vdash \lambda x. x + 3 : \text{int} \rightarrow \text{int} \\
\cdot \vdash 0”, (\lambda x. x + 3) : \text{int} \\
\cdot \vdash 0”, (\lambda x. x + 3) : \text{int} \rightarrow \text{int} \\
\cdot \vdash 0”, (\lambda x. x + 3) : \text{int} \land (\text{int} \rightarrow \text{int})
\end{aligned}
\]
Merge rule

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

Example:

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

- Order irrelevant: \((\lambda x. x + 3) ;, 0\) also OK
- Not an introduction form for \(\wedge\)
Summary of elaboration and merge

\[
\frac{\Gamma \vdash e_k : \Lambda}{\Gamma \vdash e \,, \, e_2 : \Lambda} (\exists k \in \{1, 2\})
\]

- Merge \(e_1 \,, \, e_2\):
  a way to introduce unrestricted \(\land\)s

- For more, including operational semantics, see the ICFP paper [http://arxiv.org/abs/1206.5386](http://arxiv.org/abs/1206.5386)

- **Key point for this talk:**
  The merge lets you write **two terms** in one position
✓ Overview
✓ Elaborating unrestricted ∧
☞ Annotations: overview
  • Contextual typing annotations
  • New annotations
  • Summary
Annotation forms

- In languages based on ordinary $\lambda$-calculus:
  \[ e : A \quad \text{or} \quad \lambda x : A . e \]

- Without intersection types:
  - (Sub)terms $\iff$ (sub)derivations
  - Annotations $\iff$ subderivations
Annotation forms

- In languages with this introduction rule:

\[
\frac{D_1 \quad D_2}{e : A_1 \quad e : A_2 \quad \land I} \quad e : A_1 \land A_2
\]

- One subterm \(\implies\) multiple subderivations
- One annotation \(\implies\) multiple subderivations
Why we need multiple annotations

- Assume a type \( \text{bits} \) of bitstrings, refined by \( \text{odd} \) and \( \text{even} \), denoting bitstrings of odd and even parity. Appending a one, \( x \cdot 1 \), should flip the parity:

\[
(\lambda x. x \cdot 1) : (\text{odd} \rightarrow \text{even}) \land (\text{even} \rightarrow \text{odd})
\]

- Different assumptions in branches:

\[
\begin{align*}
\text{\textcolor{green}{x : odd}} \vdash x \cdot 1 : \text{even} & \quad \text{\textcolor{green}{x : even}} \vdash x \cdot 1 : \text{odd} \\
\therefore (\lambda x. x \cdot 1) : (\text{odd} \rightarrow \text{even}) & \quad \therefore (\lambda x. x \cdot 1) : (\text{even} \rightarrow \text{odd}) \\
\therefore (\lambda x. x \cdot 1) : (\text{odd} \rightarrow \text{even}) \land (\text{even} \rightarrow \text{odd}) & \quad \end{align*}
\]

- No single type for the use of \( x \) in \( x \cdot 1 \)
Design space

\[ e : \Lambda \]
\[ \lambda x : \Lambda. \ e \]
\[ \lambda x : \text{odd}\mid\text{even}. \ x \cdot 1 \quad \text{(Reynolds 1988, Pierce 1991)} \]
\[ (x : \text{odd, even}) \cdot 1 \quad \text{(Davies 2005)} \]
Design space

\[ e : \Lambda \]
\[ \lambda x : \Lambda . e \]
\[ \lambda x : \text{odd|even}. x \cdot 1 \] (Reynolds 1988, Pierce 1991)
\[ (x : \text{odd, even}) \cdot 1 \] (Davies 2005)
\[ (x : (x:\text{odd} \vdash \text{odd}, x:\text{even} \vdash \text{even})) \cdot 1 \] “Contextual typing annotation” (Dunfield & Pfenning 2004)
Design space

\[ e : \Lambda \]

\[ \lambda x : \Lambda. e \]

\[ \lambda x : \text{odd|even}. x \cdot 1 \quad (\text{Reynolds 1988, Pierce 1991}) \]

\[ (x : \text{odd, even}) \cdot 1 \quad (\text{Davies 2005}) \]

\[ (x : (x:\text{odd} \vdash \text{odd}, x:\text{even} \vdash \text{even})) \cdot 1 \quad \text{“Contextual typing annotation”} \]

\[ ((x \cdot 1) : (x:\text{odd} \vdash \text{even}, x:\text{even} \vdash \text{odd})) \quad \ldots \text{highly general} \]
✓ Overview
✓ Elaborating unrestricted ∧
✓ Annotations: overview

Contextual typing annotations

• New annotations
• Summary
Contextual Typing Annotations

\[(e : (\Gamma_1 \vdash A_1, \ldots, \Gamma_n \vdash A_n))\]

- List of typings \(\Gamma_k \vdash A_k\)
- Choose typing \(\Gamma_k \vdash A_k\) if \(\Gamma\) supports \(\Gamma_k\)
- Example:
  \[
  (((x \cdot 1) : (x:\text{odd} \vdash \text{even}, \ x:\text{even} \vdash \text{odd}))
  \]
  - If checking under \(\ldots, x:\text{odd}\), use 1st typing
  - If checking under \(\ldots, x:\text{even}\), use 2nd typing
- Supports subtyping: suppose \(\text{even} \leq \text{bits}\) (and \(\text{even} \not\leq \text{list}\)):
  \[
  (((\text{id } x) : (x:\text{list} \vdash \text{list}, \ x:\text{bits} \vdash \text{bits}))
  \]
  - If checking under \(\ldots, x:\text{even}\): use 2nd typing
Rules for contextual typing annotations

\[
\begin{align*}
\Gamma \vdash \Gamma(x) \leq B_0 & \quad (\Gamma_0 \vdash A_0) \preceq (\Gamma \vdash A) \\
\Gamma & \vdash (x:B_0, \Gamma_0 \vdash A_0) \preceq (\Gamma \vdash A)
\end{align*}
\]

\[-pvar\]

\[
\begin{align*}
(\Gamma_0 \vdash A_0) \preceq (\Gamma \vdash A) & \quad \Gamma \vdash e : \Lambda \\
\Gamma & \vdash (e : \ldots, (\Gamma_0 \vdash A_0), \ldots) : A
\end{align*}
\]

\[\text{ctx-anno}\]

“Contextual sub-typing” \((\Gamma_0 \vdash A_0) \preceq (\Gamma \vdash A)\) if:

- \(\Gamma_0\) is supported by \(\Gamma\), and
- \(A_0 = A\) (for now)
Mechanisms of contextual typing annotations

one context supporting another

\[
\frac{(\Gamma_0 \vdash A_0) \preceq (\Gamma \vdash A)}{\Gamma \vdash \left( e : \ldots, (\Gamma_0 \vdash A_0), \ldots \right) : A}
\]

ordinary annotation

ordinary annotation

Contextuality: Types \( A_0, \ldots \) guarded by contexts \( \Gamma_0, \ldots \)

Ordinary annotation: Checking against annotated type

Multiplicity: Several annotations on one term
✓ Overview
✓ Elaborating unrestricted ∧
✓ Annotations: overview
✓ Contextual typing annotations
▷ New annotations
  • Summary
Multiplicity

one context supporting another

\[
\frac{(\Gamma_0 \vdash A_0) \lesssim (\Gamma \vdash \Lambda)}{(\Gamma \vdash (e : \ldots, (\Gamma_0 \vdash A_0), \ldots) : \Lambda)\text{ ctx-anno}}
\]

ordinary annotation

- **Multiplicity**: Several annotations on one term
- **Merge** \( e_1 , , e_2 \): Several terms in one position

Therefore:

- **Multiplicity**: Merge with different annotations

\[
(e : \ldots) , , \ldots, (e : (\Gamma_0 \vdash A_0)) , , \ldots
\]
Multiplicity

One context supporting another

\[ (\Gamma_0 \vdash A_0) \preceq (\Gamma \vdash A) \]

Ordinary annotation

\[ (\Gamma \vdash e : A) \]

Context-annotation

\[ \Gamma \vdash (e : \ldots, (\Gamma_0 \vdash A_0), \ldots) : A \]

Multiplicity

- **Multiplicity**: Several annotations
- **Merge** \( e_1, e_2 \): Several terms

Concretely:

\[ \Gamma \vdash (\chi : \text{even}) : \text{even} \]

\[ \Gamma \vdash (\chi : \text{even}), (\chi : \text{odd}) : \text{even} \]

Ordinary annotation
Ordinary annotation

one context supporting another

\[
\frac{(\Gamma_0 \vdash A_0) \preceq (\Gamma \vdash A)}{\Gamma \vdash (e : \ldots, (\Gamma_0 \vdash A_0), \ldots) : A}
\]

ordinary annotation

ctx-anno

multiplicity

- Ordinary annotation rule

\[
\frac{\Gamma \vdash e : A}{\Gamma \vdash (e : A) : A}
\]

right-anno

Concretely:

\[
\frac{\Gamma \vdash x : \text{even}}{\Gamma \vdash (x : \text{even}) : \text{even}}
\]

right-anno

\[
\frac{\Gamma \vdash (x : \text{even}) ,\ldots, (x : \text{odd}) : \text{even}}{\Gamma \vdash (x : \text{even}) ,\ldots, (x : \text{odd}) : \text{even}}
\]

ordinary annotation
Contextuality

one context supporting another

\[(\Gamma_0 \vdash A_0) \preceq (\Gamma \vdash A)\]

ordinary annotation

\[\Gamma \vdash (e : \ldots, (\Gamma_0 \vdash A_0), \ldots) : A\]

ctx-anno

multiplicity

- Contextuality

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

left-anno

\[\Gamma \vdash (x : A \gg e) : B\]

“left-hand” annotation

Example:

\[\Gamma \vdash x \cdot 1 : \text{even}\]

right-anno

\[\Gamma \vdash (x \cdot 1 : \text{even}) : \text{even}\]

left-anno

\[\Gamma \vdash x : \text{odd} \gg (x \cdot 1 : \text{even}) : \text{even}\]

left-hand annotation

ordinary annotation
A theorem

**Theorem** (Encoding Contextual Typing Annotations) 
If $\Gamma \vdash e : A$ with rule ctx-anno available
then $\Gamma \vdash \text{trans}(e') : A$ without applying rule ctx-anno.
A theorem

**Theorem** (Encoding Contextual Typing Annotations)
If $\Gamma \vdash e : A$ with rule ctx-anno available
then $\Gamma \vdash \text{trans}(e') : A$ without applying rule ctx-anno.

Therefore:

- **Ordinary annotations** $x : A$
- + **Left-hand annotations** $x : A > : > e$
- + **Merges** $e_1 , , e_2$
- $\geq$ **Contextual typing annotations**
Contextual Modal Types (Nanevski et al.)

- $A [\Psi]$ represents data of type $A$ closed under context $\Psi$
- Annotation with a contextual type is like ordinary annotation and left-hand annotation:

  $$(x \cdot 1) : \text{even}[x:\text{odd}]$$
Contextual Modal Types (Nanevski et al.)

- $A[\Psi]$ represents data of type $A$ closed under context $\Psi$
- Annotation with a contextual type is like ordinary annotation and left-hand annotation:
  \[
  (x \cdot 1) : \text{even}[x:\text{odd}]
  \]

- Unrestricted intersection gives us multiplicity:
  \[
  \lambda x. \text{let } r = (y \cdot 1) : \text{even}[y:\text{odd}] \land \text{odd}[y:\text{even}] \text{ in } \\
  r[x/y]
  \]
- Much more powerful than needed for annotation
Which is better?

- A (type system/typechecker/compiler) “engineering” question
- It depends on what you have already
- Given contextual types and unrestricted $\land$, use them
- Given merge and unrestricted $\land$, add left-hand annotations
“For more, see the paper”

- Typechecking is actually **bidirectional** (Section 3.1)
- The approach can be extended to indexed types with index variables (Section 4)
- For more on merges and elaboration, see ICFP 2012
✓ Overview
✓ Elaborating unrestricted ∧
✓ Annotations: overview
✓ Contextual typing annotations
✓ New annotations
☞ Summary
Summary

• Exploring the design space of annotations in intersection type systems
• New(ish) mechanisms: merges and left-hand annotations
• Some idea of relative power:

\[
\text{Contextual types} \geq \text{Ordinary annotations} + \text{Left-hand annotations} + \text{Merges} \geq \text{Contextual typing annotations}
\]
Thank you

- And thanks to
  - the ITRS reviewers
  - Neelakantant R. Krishnaswami
  - Frank Pfenning

- Further reading: