

# Type assignment for intersections and unions in call-by-value languages

Joshua Dunfield and Frank Pfenning

Triple Project

Carnegie Mellon University

8 April 2003

*FOSSACS '03*, Warsaw, Poland

# Outline

- Motivation
- Language
- Datasort refinements
- Index refinements
- Intersections
- Indefinite types
- Related work
- Conclusion & future work

# Motivation

- Conventional type systems: ML, Java, ...
  - Tractable and easy to use
  - Limited expressive power
- Refined type systems:
  - Tractable and easy to use
  - Better expressive power
- Fully dependent type systems: Nuprl, ...
  - Undecidable
  - Extremely expressive

# Push the envelope

The goal of the Triple Project: create *refined type systems*.

- The present work: Persistent refinements of algebraic datatypes

# Type Assignment

- A *type assignment* system *a la* Curry
  - Terms do not contain types
  - Undecidable
- A *bidirectional* system
  - Terms contain some type annotations
  - Two judgments: *inference*  $e \uparrow A$ , *checking*  $e \downarrow A$ .
- This paper: Type assignment only

# Property Types

- Some types *build* values
  - Example: tuples
- Others talk *about* values
  - Example: parametric polymorphism  $\forall \alpha. \alpha \rightarrow \alpha$
  - $\forall$  describing behavior of the identity fn
- We call the latter *property types*

# Property Types

- In our system:
  - $\delta(i)$ : Datasort and index refinement
  - $\wedge$ : Intersection:  $v : A \wedge B$  means  $v$  has type  $A$  *and* type  $B$
  - $\top$ : Greatest type (0-ary  $\wedge$ )
  - $\Pi$ : Universal quantifier over indices (infinitary  $\wedge$ )
  - $\vee$ : Union:  $v : A \vee B$  means  $v$  has type  $A$  *or* type  $B$
  - $\perp$ : Empty type (0-ary  $\vee$ )
  - $\Sigma$ : Existential quantifier over indices (infinitary  $\vee$ )

# Language

# Language + Initial Typing

$A, B, C, D ::= 1 \mid A \rightarrow B$

$e ::= x \mid f \mid () \mid \lambda x. e \mid e_1(e_2) \mid \text{fix } f. e$

$$\frac{B_1 \leq A_1 \quad A_2 \leq B_2}{A_1 \rightarrow A_2 \leq B_1 \rightarrow B_2} (\rightarrow) \quad \frac{}{1 \leq 1} (1)$$

$$\frac{\Gamma(x) = A}{\Gamma \vdash x : A} (\text{var}) \quad \frac{\Gamma(f) = A}{\Gamma \vdash f : A} (\text{fixvar}) \quad \frac{\Gamma \vdash e : A \quad A \leq B}{\Gamma \vdash e : B} (\text{sub})$$

$$\frac{\Gamma \vdash e_1 : A \rightarrow B \quad \Gamma \vdash e_2 : A}{\Gamma \vdash e_1(e_2) : B} (\rightarrow E) \quad \frac{\Gamma, f:A \vdash e : A}{\Gamma \vdash \text{fix } f. e : A} (\text{fix})$$

$$\frac{\Gamma, x:A \vdash e : B}{\Gamma \vdash \lambda x. e : A \rightarrow B} (\rightarrow I) \quad \frac{}{\Gamma \vdash () : 1} (1I)$$

# cbv Semantics

Values  $v ::= x \mid () \mid \lambda x. e$

Evaluation contexts  $E ::= [] \mid E(e) \mid v(E)$

$$\frac{e' \mapsto_{\mathcal{R}} e''}{E[e'] \mapsto E[e'']}$$

$$(\lambda x. e) v \mapsto_{\mathcal{R}} [v/x] e$$

$$\text{fix } f. e \mapsto_{\mathcal{R}} [\text{fix } f. e / f] e$$

# Key Properties

A practical type system should satisfy

- Preservation: If  $\vdash e : A$  and  $e \mapsto e'$  then  $\vdash e' : A$ .
- Progress: If  $\vdash e : A$  then  $e$  is a value or  $\exists e'. e \mapsto e'$ .

# Datasort Refinements

- a.k.a. refinement types (Freeman, Pfenning, Davies)
- Refine an algebraic datatype by a datasort  $\delta$
- Example: Lists of integers

Nil : 1  $\rightarrow$  list

Nil : 1  $\rightarrow$  even

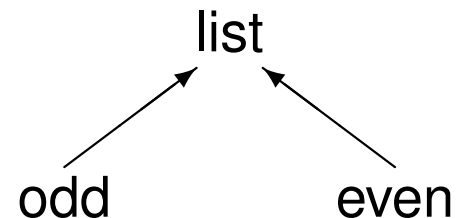
Cons : int \* list  $\rightarrow$  list

Cons : (int \* odd  $\rightarrow$  even)

$\wedge$  (int \* even  $\rightarrow$  odd)

$\wedge$  (int \* list  $\rightarrow$  list)

$\delta$  :



- Intersections essential

# Index Refinements

- a.k.a. dependent types restricted to a decidable constraint domain (Xi & Pfenning)
- Refine an algebraic datatype by an index
- Indices drawn from any decidable constraint domain, here  $\mathcal{N}$

- Example: Lists indexed by their length

$\text{Nil} : 1 \rightarrow \text{list}$

$\text{Nil} : 1 \rightarrow \text{list}(0)$

$\text{Cons} : \text{int} * \text{list} \rightarrow \text{list}$

$\text{Cons} : \prod a:\mathcal{N}. \text{int} * \text{list}(a) \rightarrow \text{list}(a + 1)$

- Example:

$\text{append} : \prod a:\mathcal{N}. \prod b:\mathcal{N}. \text{list}(a) * \text{list}(b) \rightarrow \text{list}(a+b)$

- Universal quantifier  $\prod$  essential
- Existential quantifier  $\sum$  also essential

# Intersection Types

# Typing Intersections

- Typing:

$$\frac{\Gamma \vdash e : A_1 \wedge A_2}{\Gamma \vdash e : A_1} (\wedge E_1) \quad \frac{\Gamma \vdash e : A_1 \wedge A_2}{\Gamma \vdash e : A_2} (\wedge E_2)$$

$$\frac{\Gamma \vdash e : A_1 \quad \Gamma \vdash e : A_2}{\Gamma \vdash e : A_1 \wedge A_2} (\wedge I) \quad \text{☠}$$

- $(\wedge I)$  unsound with mutable references.



$$\frac{\Gamma \vdash v : A_1 \quad \Gamma \vdash v : A_2}{\Gamma \vdash v : A_1 \wedge A_2} (\wedge I)$$

- The fix: a *value restriction* [Davies & Pfenning '00].

# Indefinite Types

# Indefinite Types: Motivation

$filter : (int \rightarrow bool) \rightarrow list \rightarrow list$

$filter : \prod \alpha:\mathcal{N}. (int \rightarrow bool) \rightarrow list(\alpha) \rightarrow list(\_)$

$filter : \prod \alpha:\mathcal{N}. (int \rightarrow bool) \rightarrow list(\alpha) \rightarrow (\sum b:\mathcal{N}. list(b))$

- $\sum b:\mathcal{N}. B$  quantifies existentially over  $b$  in  $B$
- [Xi & Pfenning]

# Indefinite Types: Motivation

- Definitely need  $\Sigma$
- The binary analogue of  $\Sigma$  is the union type  $\vee$
- Need  $\vee$ ? Possibly just a convenience
- Issues similar for both, but easier to talk about  $\vee$

# Union Types

- Introduction rules straightforward:

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash e : A \vee B} (\vee I_1)$$

$$\frac{\Gamma \vdash e : B}{\Gamma \vdash e : A \vee B} (\vee I_2)$$

- Elimination rule?

# Rough Road

- Recall: for  $\wedge$ , introduction rule ( $\wedge I$ ) was tricky
- Duality suggests ( $\vee E$ ) will be interesting too

# ∨ Elimination

- Want to reason by cases: For  $x : A \vee B$ , show for  $x : A$  and for  $x : B$ .
- Substitution approach (e.g. [MacQueen et al.'86]), reminiscent of natural deduction:

$$\frac{\Gamma, x:A \vdash e : C \quad \Gamma, x:B \vdash e : C}{\Gamma \vdash e' : A \vee B \quad \Gamma \vdash [e'/x] e : C}$$

- Suppose  $\Gamma \vdash h : (A \rightarrow C) \wedge (B \rightarrow C)$ ,  $\Gamma \vdash v : A \vee B$

$$\frac{\Gamma \vdash v : A \vee B \quad \Gamma, x:A \vdash h x : C \quad \Gamma, x:B \vdash h x : C}{\Gamma \vdash h v : C}$$

# ∨ Elimination

$$\frac{\Gamma \vdash e' : A \quad \Gamma, x:A \vdash e : C \quad \Gamma, x:B \vdash e : C}{\Gamma \vdash [e'/x] e : C} \quad \text{☠}$$

- Consider a nondeterministic choice  $e_1 \oplus e_2$ :

$$e_1 \oplus e_2 \mapsto e_1 \quad e_1 \oplus e_2 \mapsto e_2$$

- Suppose  $\Gamma \vdash g : (A \rightarrow A \rightarrow C) \wedge (B \rightarrow B \rightarrow C)$ ,  
 $\Gamma \vdash y : A, \Gamma \vdash z : B$ .

$$\frac{\Gamma \vdash y \oplus z : A \vee B \quad \Gamma, x:A \vdash g \ x \ x : C \quad \Gamma, x:B \vdash g \ x \ x : C}{\Gamma \vdash g (y \oplus z) (y \oplus z) : C}$$

- $g (y \oplus z) (y \oplus z) \mapsto g \ y \ (y \oplus z) \mapsto g \ y \ z$
- Unsound whenever a term evaluates to more than one value

# ∨ Elimination

$$\frac{\Gamma \vdash e' : A \quad \Gamma, x:A \vdash e : C \quad \Gamma, x:B \vdash e : C}{\Gamma \vdash [e'/x] e : C} \quad \text{☠}$$

- Restrict to exactly one occurrence of  $e'$ ?
  - Can fail if  $e'$  inside a  $\lambda$  (see paper)
- Restrict  $e'$  to a value? [van Bakel '99]
  - Sound—but still must guess occurrences of  $e'$

# ∨ Elimination: Our Solution

$$\frac{\Gamma \vdash e' : A \quad \Gamma, x:A \vdash E[x] : C}{\Gamma \vdash E[e'] : C} \text{ (}\vee\text{E)}$$

- Exactly one  $e'$ , *in evaluation position*
- So  $e'$  is the next thing to be evaluated
- Remainder of computation (E) on the result of  $e'$

# The Empty Type $\perp$

- $\perp$  is the empty or void type

$$\frac{}{\perp \leq A} (\perp L)$$

- Elimination rule: by analogy with  $(\forall E)$

$$\frac{\Gamma \vdash e' : \perp}{\Gamma \vdash E[e'] : C} (\perp E)$$

# A Generalization?

- Must we restrict the rules to an evaluation context?
- Or could we also allow cases where  $e'$  must be evaluated exactly once but might not be in evaluation position?

$$\frac{\Gamma \vdash e' : \perp \quad e' \text{ eval'd exactly once in } e}{\Gamma \vdash e : C}$$



- ... let  $\omega = (\text{fix } f. f)$ . Then  $\omega : \perp$ .

$$\frac{\Gamma \vdash \omega : \perp}{\Gamma \vdash ( () () )\omega : C}$$

- But  $( () () )$  is not a value and does not step

# Type Safety

# Type Safety

$$\frac{\Gamma \vdash e' : A \vee B \quad \begin{array}{l} \Gamma, x:A \vdash E[x] : C \\ \Gamma, x:B \vdash E[x] : C \end{array}}{\Gamma \vdash E[e'] : C} \text{ (}\vee\text{E)} \quad \checkmark$$

- Preservation and progress hold with  $\delta, i, \wedge, \top, \Pi, \vee, \perp, \Sigma$
- Lemma: Values always have definite types.
  - $\not\vdash v : \perp$
  - If  $\vdash v : A \vee B$  then  $\vdash v : A$  or  $\vdash v : B$
  - If  $\vdash v : \Sigma a:\mathcal{N}. A$  then  $\vdash v : [i/a] A$  for some  $i$
- Above: a special case of the actual lemma
- (Fully written out for  $\delta, i, \wedge, \Pi$  in [Dunfield '02])

# Closely Related Work

Davies '97, '00\*

$\delta, \wedge; \uparrow\downarrow$

Xi '98, '99\*, '00

$i, \Pi, \Sigma; \uparrow\downarrow$

Dunfield '02

$\delta, i, \wedge, \Pi; \uparrow\downarrow$

This paper\*

$\delta, i, \wedge, \Pi, \top, \vee, \Sigma, \perp; :$

Ongoing work

$\delta, i, \wedge, \Pi, \top, \vee, \Sigma, \perp; \uparrow\downarrow$

\* & Pfenning

# Related Work

- [Coppo et al.'81]:  $\wedge$  can characterize normal forms (termination); hence undecidable
- [Reynolds '96]: FORSYTHE with  $\wedge$  (& type annotations)
- [Pierce '91]: Language with  $\wedge$ ,  $\vee$ , syntactic markers
- [Barbanera et al.'95]: With unrestricted  $\vee$ -elim rule, types not preserved under  $\beta$ -reduction
- [Wells et al.'02], [Palsberg & Pavlopoulou '01]:  $\wedge$ ,  $\vee$  for control flow analysis

# Conclusion


# Summary

- Goal: express more invariants
- Two especially useful refinements already developed:
  - Datasort refinements
  - Index refinements
- Refinements motivate  $\wedge$ ,  $\vee$ ,  $\Pi$ ,  $\Sigma$
- $(\wedge I)$ : Value restriction
- $(\vee E)$ : Eval. context restriction
- Type safety
- The present system: Undecidable, hence impractical

# Future Work

- Bidirectional formulation ✓
- Let-normal form ✓
- Mutable references
- Implementation

# Acknowledgments

- *National Science Foundation*: CCR-0204248: Type Refinements; 0121633: ITR/SY+SI: Language Technology for Trustless Software Dissemination; Graduate Research Fellowship
- Feedback on the paper: Brigitte Pientka and the anonymous referees
- Feedback on the talk: Aleksey Kliger, Stephen Magill, Jonathan Moody, Frank Pfenning
- “”: Henrik Christian Grove, available at a CTAN mirror near you

**The End**

# Intersections: Subtyping

- Subtyping:

$$\frac{A \leq B_1 \quad A \leq B_2}{A \leq B_1 \wedge B_2} (\wedge R)$$

$$\frac{A_1 \leq B}{A_1 \wedge A_2 \leq B} (\wedge L_1) \quad \frac{A_2 \leq B}{A_1 \wedge A_2 \leq B} (\wedge L_2)$$

- Distributivity?

$$\frac{}{(A \rightarrow B) \wedge (A \rightarrow B') \leq A \rightarrow (B \wedge B')} \text{ ☠}$$

- Again unsound with mutable references. [Davies & Pfenning '00]

# Unions: Subtyping

- Subtyping for  $\vee$  just the dual of  $\wedge$ :

$$\frac{A_1 \leq B \quad A_2 \leq B}{A_1 \vee A_2 \leq B} (\vee L)$$

$$\frac{A \leq B_1}{A \leq B_1 \vee B_2} (\vee R_1) \quad \frac{A \leq B_2}{A \leq B_1 \vee B_2} (\vee R_2)$$

# Formulating Refinements

- $P ::= \perp \mid i \doteq j \mid \dots$   
 $\Gamma ::= \cdot \mid \Gamma, x:A \mid \Gamma, a:\gamma \mid \Gamma, P$   
 $e ::= \dots \mid c(e) \mid \text{case } e \text{ of } ms$   
 $v ::= \dots \mid c(v)$   
 $E ::= \dots \mid c(E) \mid \text{case } E \text{ of } ms$

- Atomic subtyping

$$\frac{\delta_1 \preceq \delta_2 \quad \bar{\Gamma} \vdash i \doteq j}{\Gamma \vdash \delta_1(i) \leq \delta_2(j)} \quad (\delta)$$

- 

$$\frac{\bar{\Gamma} \vdash c : A \rightarrow \delta(i) \quad \Gamma \vdash e : A}{\Gamma \vdash c(e) : \delta(i)} \quad (\delta I)$$

# Formulating Refinements

- Atomic subtyping

$$\frac{\delta_1 \preceq \delta_2 \quad \bar{\Gamma} \vdash i \doteq j}{\Gamma \vdash \delta_1(i) \leq \delta_2(j)} (\delta)$$

# Evaluation Order

# Evaluation Order

- Binary and 0-ary indefinite type elim rules:

$$\frac{\Gamma, x:A \vdash E[x] : C \quad \Gamma \vdash e' : A \vee B \quad \Gamma, x:B \vdash E[x] : C}{\Gamma \vdash E[e'] : C} (\vee E) \quad \frac{\Gamma \vdash e' : \perp}{\Gamma \vdash E[e'] : C} (\perp E)$$

- A corresponding *unary* rule?

$$\frac{\Gamma \vdash e' : A \quad \Gamma, x:A \vdash E[x] : C}{\Gamma \vdash E[e'] : C} \text{ (direct)}$$

- Actually a derivable rule: ( $\vee E$ ) with  $A \vee A$
- But not derivable in a bidirectional system

# Evaluation Order

$$\frac{\Gamma \vdash e' : A \quad \Gamma, x:A \vdash E[x] : C}{\Gamma \vdash E[e'] : C} \text{ (direct)}$$

- Rule needed so that  $\Pi$ 's can be instantiated with existential  $\Sigma$  index variables
- Problem first recognized by Xi
- Formulated a translation to let-normal form
- Added “good” programs, but removed them too
- Current work: a sound, complete let-normal translation