Tri-Directional Type Checking

Frank Pfenning
(joint work with Joshua Dunfield)
Carnegie Mellon University

Invited Talk

Workshop on Intersection Types
and Related Systems (ITRS'02)

Copenhagen, Denmark, July 26, 2002

Warning: Work in progress

Acknowledgments: Rowan Davies

Outline

- **Introduction**
- Guiding Principles
- Atomic Subtyping
- Intersection Types
- Union Types
- [Dependent Types]
- Conclusion

Why Aren't Most Programs Verified?

- Difficulty of expressing a precise specification
- Difficulty of proving correctness
- Difficulty of co-evolving program, specification, and proof
- Problems exacerbated by poorly designed languages

Why Are Most Programs Type-Checked?

- Ease of expressing types
- Ease of checking types
- Ease of co-evolving programs and types
- Most useful in properly designed languages

A Continuum?

- Types as a *minimal* requirement for meaningful programs
- Specifications as a *maximal* requirement for correct programs
- Suprisingly few intermediate points have been investigated
- Many errors are caught by simple type-checking
- But many errors also escape simple type-checking

A Research Program

- Designing systems for statically verifying program properties
- Evaluation along the following dimensions:
  - Elegance, generality, brevity (ease of expression)
  - Practicality of verification (ease of checking)
  - Explicitness (ease of understanding and evolution)
  - Support for modularity
- Some of these involve trade-offs

Influences

- Traditional static program analysis
  emphasis there on automation and efficiency improvements
- Traditional type systems
  emphasis there on inference and generality

The Basic Idea

- ML (cbv, funs, datatypes, effects) as a host language
- Data structures via datatypes
- Invariants on data structures via subtypes of datatypes
- Extend to full language via type constructors
  intersection, universal, union, empty,
  [universal dependent, existential dependent]
  (modal, linear, temporal, . . . — future work)

Outline

- Introduction
- **Guiding Principles**
- Atomic Subtyping
- Intersection Types
- Union Types
- [Dependent Types]
- Conclusion

Key Foundational Issue

- **Question:** What are the guiding principles in the design of type (refinement) systems to express and verify program properties?
- **My Answer:** Martin-Löf's method of judgments and derivations
- Proof-theoretic rather than model-theoretic

> *The meaning of a proposition is determined by [. . .] what counts as a verification of it.* — Per Martin-Löf, 1983

Central Technical Issues

- Design questions
  - Rules for typing expressions
  - Rules for subtyping
  - Mechanism for type-checking
- Meta-theorems
  - **Adequacy** for data representation
  - **Preservation** of types under evaluation
  - **Progress** from any well-typed configuration
  - **Decidability** of type-checking

Static Judgments

- A type $\qquad$ A *is a type* (elided in this talk)
- $M : A$ $\qquad$ M *has type* A
- Hypotheses $x_1{:}A_1, \ldots, x_n{:}A_n$, x_i distinct (write Γ)
- $\Gamma \vdash M$ val $\qquad$ M *is a value* (write V)
- Defining properties for hypothetical judgments
  - Hypothesis rules

$$\frac{}{\Gamma, x{:}A, \Gamma' \vdash x : A} \qquad \frac{}{\Gamma, x{:}A, \Gamma' \vdash x \text{ val}}$$

  - Substitution principle (theorem)

    If $\Gamma \vdash V : A$ and $\Gamma, x{:}A, \Gamma' \vdash N : C$ then
    $\Gamma, \Gamma' \vdash [V/x]N : C$

Computation Judgments

- $M \longrightarrow_\beta M'$ $\qquad$ M *beta-reduces to* M'
- E ctx $\qquad$ E *is an evaluation context*, hole [ ]

$$\frac{}{[\,]\ \text{ctx}}$$

- $E[M]$ replaces hole in E by M
- $M \longrightarrow M'$ $\qquad$ M *reduces to* M'
- Closure rule

$$\frac{M \longrightarrow_\beta M'}{E[M] \longrightarrow E[M']}$$

Principles of Computation

- Progress principle (theorem)

  If $\vdash M : A$ *then either* $\vdash M$ *val or* $M \longrightarrow M'$.

- Preservation principle (theorem)

  If $\vdash M : A$ *and* $M \longrightarrow M'$ *then* $\vdash M' : A$

- Note restriction to closed terms

Outline

- Introduction
- Guiding Principles
- **Atomic Subtyping**
- Intersection Types
- Union Types
- [Dependent Types]
- Conclusion

Function Types $A \to B$

- Introduction and elimination rules

$$\frac{\Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x.\, M : A \to B} \to\! I \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B} \to\! E$$

- Values

$$\frac{}{\Gamma \vdash \lambda x.\, M \ \mathsf{val}}$$

- Computation (introduction followed by elimination)

$$(\lambda x.\, M)\ V \longrightarrow_\beta [V/x]M$$

$$\frac{E\ \mathsf{ctx}}{E\ M\ \mathsf{ctx}} \qquad \frac{V\ \mathsf{val} \qquad E\ \mathsf{ctx}}{V\ E\ \mathsf{ctx}}$$

Mechanisms for Type-Checking

- Type synthesis (Church)
  - Given Γ, M, synthesize unique [principal] A with $\Gamma \vdash M : A$ or fail
  - Requires type labels $\lambda x{:}A.\, M$
  - Does not generalize well to intersection and related types
- Type assignment (Curry)
  - Given Γ, M, A succeed if $\Gamma \vdash M : A$, otherwise fail
  - Very general (traditional for intersection types)
  - Often undecidable

Bi-Directional Type-Checking

- Based on two judgments, combining synthesis with analysis
- $\Gamma \vdash M \uparrow A$ *Given* Γ, M, *synthesize* A *with* $\Gamma \vdash M : A$
- $\Gamma \vdash M \downarrow A$ *Given* Γ, M, A, *analyze if* $\Gamma \vdash M : A$
- Hypothesis rule

$$\frac{}{\Gamma, x : A, \Gamma' \vdash x \uparrow A}$$

- New expression $(M : A)$
- Mutual dependencies (revisit later)

$$\frac{\Gamma \vdash M \uparrow A}{\Gamma \vdash M \downarrow A}\uparrow\downarrow \qquad \frac{\Gamma \vdash M \downarrow A}{\Gamma \vdash (M : A) \uparrow A}\downarrow\uparrow$$

- Several substitution principles (elided)

Bi-Directional Type-Checking of Functions

- Introduction forms are analyzed

$$\frac{\Gamma, x{:}A \vdash M \downarrow B}{\Gamma \vdash \lambda x.\, M \downarrow A \rightarrow B} \rightarrow I$$

- Elimination forms are synthesized

$$\frac{\Gamma \vdash M \uparrow A \rightarrow B \qquad \Gamma \vdash N \downarrow A}{\Gamma \vdash M\ N \uparrow B} \rightarrow E$$

- Read '$\uparrow$' and '$\downarrow$' as ':' to obtain type assignment rules
- No type annotations in normal forms. E.g., for any A,

$$\vdash \lambda f.\, \lambda x.\, f(f x) \downarrow (A \rightarrow A) \rightarrow (A \rightarrow A)$$

- Annotate redexes, e.g.,

$$\vdash (\lambda x.\, x : \mathsf{bits} \rightarrow \mathsf{bits})\ (\epsilon\mathbf{110}) \uparrow \mathsf{bits}$$

Definitions

- Internalize substitution principle

$$\frac{\Gamma \vdash M \uparrow A \qquad \Gamma, x{:}A \vdash N \downarrow C}{\Gamma \vdash \mathsf{let}\ x = M\ \mathsf{in}\ N\ \mathsf{end} \downarrow C}$$

- Computation

$$\mathsf{let}\ x = V\ \mathsf{in}\ N\ \mathsf{end} \longrightarrow_{\beta} [V/x]N \qquad\qquad \frac{E\ \mathsf{ctx}}{\mathsf{let}\ x = E\ \mathsf{in}\ N\ \mathsf{end\ ctx}}$$

- In practice, use definitions instead of redices

$$\vdash \mathsf{let}\ f = (\lambda x.\, x : \mathsf{bits} \to \mathsf{bits})\ \mathsf{in}\ f\ (\epsilon\mathbf{110})\ \mathsf{end} \downarrow \mathsf{bits}$$

or $$\vdash \mathsf{let}\ f : \mathsf{bits} \to \mathsf{bits} = \lambda x.\, x\ \mathsf{in}\ f\ (\epsilon\mathbf{110})\ \mathsf{end} \downarrow \mathsf{bits}$$

Remarks on Judgmental Method

- Specification is open-ended
- Constructs are defined orthogonally
- Proofs of meta-theoretic properties (e.g., progress, preservation) decomposes along the same lines
- Logical connections
  - $\Gamma \vdash M \downarrow A$ without $\downarrow\uparrow$ coercions characterizes normal natural deductions of A with the subformula property
  - This is in fact the origin of the rules
  - Judgment is analytic in Γ, M, and A: any derivation mentions only constituent terms and types of Γ, M, A

Adding Data Types

- Proceed by example: bit strings and natural numbers
- For general case, see [Dunfield'02] [Davies'02]
- Introduction forms

$$\frac{}{\Gamma \vdash \epsilon \downarrow \mathsf{bits}} \qquad \frac{\Gamma \vdash M \downarrow \mathsf{bits}}{\Gamma \vdash M\ \mathbf{0} \downarrow \mathsf{bits}} \qquad \frac{\Gamma \vdash M \downarrow \mathsf{bits}}{\Gamma \vdash M\ \mathbf{1} \downarrow \mathsf{bits}}$$

- ϵ represents empty string, $\mathbf{0}$ and $\mathbf{1}$ are postfix operators.
- For example: $\ulcorner 0 \urcorner = \epsilon$, $\ulcorner 6 \urcorner = \epsilon\,\mathbf{1}\,\mathbf{1}\,\mathbf{0}$.
- Elimination form

$$\frac{\Gamma \vdash M \uparrow \mathsf{bits} \quad \Gamma \vdash N_e \downarrow C \quad \Gamma, x{:}\mathsf{bits} \vdash N_0 \downarrow C \quad \Gamma, y{:}\mathsf{bits} \vdash N_1 \downarrow C}{\Gamma \vdash \mathsf{case}\, M \;\mathsf{of}\; \epsilon \Rightarrow N_e \mid x\,\mathbf{0} \Rightarrow N_0 \mid y\,\mathbf{1} \Rightarrow N_1 \downarrow C}$$

Computation on Data Types

- Rules for computation, values, evaluation contexts straightforward
- Need recursion for interesting functions

$$\frac{\Gamma, u{:}A \vdash M \downarrow A}{\Gamma \vdash \mathsf{fix}\, u.\, M \downarrow A} \qquad \mathsf{fix}\, u.\, M \longrightarrow_\beta [\mathsf{fix}\, u.\, M/u]M$$

- No new values or evaluation contexts
- Orthogonal to other constructs in this form
- Technical complication: u stands for a term, not a value
- Treat explicitly or restrict syntax to fix $u.\, \lambda x.\, M$

Data Structure Invariants and Subtyping

- Example: natural numbers as bit strings without leading zeroes.
- Intuition: need positive numbers, at least internally

$$\begin{array}{lrcl} \textit{Natural Numbers} & \mathsf{nat} & ::= & \epsilon \mid \mathsf{pos} \\ \textit{Positive Numbers} & \mathsf{pos} & ::= & \mathsf{pos}\,\mathbf{0} \mid \mathsf{nat}\,\mathbf{1} \end{array}$$

- Capture systematically and orthogonally to everything before via
  - typing rules
  - subtyping rules

Typing Natural Numbers

- New rules (ignore redundancy):

$$\frac{}{\Gamma \vdash \epsilon \downarrow \mathsf{nat}} \qquad \text{(no } \epsilon \downarrow \mathsf{pos}\text{)}$$

$$\frac{\Gamma \vdash M \downarrow \mathsf{pos}}{\Gamma \vdash M\, \mathbf{0} \downarrow \mathsf{nat}} \qquad \frac{\Gamma \vdash M \downarrow \mathsf{pos}}{\Gamma \vdash M\, \mathbf{0} \downarrow \mathsf{pos}}$$

$$\frac{\Gamma \vdash M \downarrow \mathsf{nat}}{\Gamma \vdash M\, \mathbf{1} \downarrow \mathsf{nat}} \qquad \frac{\Gamma \vdash M \downarrow \mathsf{nat}}{\Gamma \vdash M\, \mathbf{1} \downarrow \mathsf{pos}}$$

$$\frac{\Gamma \vdash M \uparrow \mathsf{nat} \quad \Gamma \vdash N_e \downarrow C \quad \Gamma, x{:}\mathsf{pos} \vdash N_0 \downarrow C \quad \Gamma, y{:}\mathsf{nat} \vdash N_1 \downarrow C}{\Gamma \vdash \mathsf{case}\, M \text{ of } \epsilon \Rightarrow N_e \mid x\, \mathbf{0} \Rightarrow N_0 \mid y\, \mathbf{1} \Rightarrow N_1 \downarrow C}$$

$$\frac{\Gamma \vdash M \uparrow \mathsf{pos} \quad \text{(no } N_\epsilon \downarrow C\text{)} \quad \Gamma, x{:}\mathsf{pos} \vdash N_0 \downarrow C \quad \Gamma, y{:}\mathsf{nat} \vdash N_1 \downarrow C}{\Gamma \vdash \mathsf{case}\, M \text{ of } \epsilon \Rightarrow N_e \mid x\, \mathbf{0} \Rightarrow N_0 \mid y\, \mathbf{1} \Rightarrow N_1 \downarrow C}$$

Subtyping Judgment

- New judgment $A \leq B$ $\qquad$ A is a subtype of B
- $A \leq B$ if every value of type A also has type B
- Reflexivity rule ($\sim$ hypothesis rule)

$$\frac{}{A \leq A}$$

- Transitivity principle (theorem, $\sim$ substitution principle)

If $A \leq B$ and $B \leq C$ then $A \leq C$.

- Subsumption rule, replaces $\uparrow\downarrow$

$$\frac{\Gamma \vdash M \uparrow A \qquad \Gamma \vdash A \leq C}{\Gamma \vdash M \downarrow C}$$

Subtyping of Data Types

- From the typing rules:

$$\frac{}{\mathsf{pos} \leq \mathsf{nat}} \qquad \frac{}{\mathsf{nat} \leq \mathsf{bits}} \qquad \frac{}{\mathsf{pos} \leq \mathsf{bits}}$$

- In general, a lattice
- Example of need for subsumption rule

$$x{:}\mathsf{pos} \vdash x \downarrow \mathsf{nat}$$

$$\text{since} \quad x{:}\mathsf{pos} \vdash x \uparrow \mathsf{pos}$$

$$\text{and} \quad \mathsf{pos} \leq \mathsf{nat}$$

- Subtyping of functions

$$\frac{B_1 \leq A_1 \qquad A_2 \leq B_2}{A_1 \to A_2 \leq B_1 \to B_2}$$

Summary of Atomic Subtyping

- Type assignment $\Gamma \vdash M : A$
- Bi-directional system $\Gamma \vdash M \downarrow A$, $\Gamma \vdash M \uparrow A$
- Values V, evaluation contexts $E[\,]$, reduction $M \longrightarrow M'$
- Subtyping $A \leq B$
- All judgments are analytic and therefore decidable
- Can express data structure invariants recognizable by finite-state tree automata (regular tree languages)
- Cannot express, e.g., lengths of lists or depths of trees

Outline

- Introduction
- Guiding Principles
- Atomic Subtyping
- **Intersection Types**
- Union Types
- [Dependent Types]
- Conclusion

Limitations of Atomic Subtyping

- Problem: consider $\mathit{shiftl} = \lambda x.\, x\, \mathbf{0}$.

$$\vdash \lambda x.\, x\, \mathbf{0} \;\downarrow\; \mathsf{bits} \rightarrow \mathsf{bits}$$

$$\vdash \lambda x.\, x\, \mathbf{0} \;\downarrow\; \mathsf{nat} \rightarrow \mathsf{bits}$$

$$\vdash \lambda x.\, x\, \mathbf{0} \;\downarrow\; \mathsf{pos} \rightarrow \mathsf{pos}$$

- These may all be needed, but cannot be expressed simultaneously

- Especially troublesome for recursive functions

$$\begin{array}{ll} \not\vdash & \mathsf{fix}\; \mathit{inc}.\, \lambda n.\; \mathsf{case}\; n \\ & \qquad\qquad\qquad \mathsf{of}\; \epsilon \Rightarrow \epsilon\, \mathbf{1} \\ & \qquad\qquad\qquad\;\; \mid x\, \mathbf{0} \Rightarrow x\, \mathbf{1} \\ & \qquad\qquad\qquad\;\; \mid x\, \mathbf{1} \Rightarrow (\mathit{inc}\; x)\, \mathbf{0} \qquad \%\; \mathit{inc}\, x : \mathsf{pos}? \\ \;\downarrow & \mathsf{nat} \rightarrow \mathsf{nat} \end{array}$$

Intersection Types $A \wedge B$

- Introduction and elimination forms

$$\frac{\Gamma \vdash V \downarrow A \qquad \Gamma \vdash V \downarrow B}{\Gamma \vdash V \downarrow A \wedge B} \wedge I$$

$$\frac{\Gamma \vdash M \uparrow A \wedge B}{\Gamma \vdash M \uparrow A} \wedge E_1 \qquad \frac{\Gamma \vdash M \uparrow A \wedge B}{\Gamma \vdash M \uparrow B} \wedge E_2$$

- Subject of judgment identical in premises and conclusion
- $A \wedge B$ a *property type* (refinement type)
- bits, $A \to B$, $A \times B$, 1 are *constructor types*
- Elimination rules are **not** redundant with bi-directionality
- Value restriction is necessary for type preservation with effects [Davies & Pf, ICFP'00]

Subtyping Intersection Types

- Right and left rules ($\sim$ sequent calculus)

$$\frac{A \leq B \qquad A \leq C}{A \leq B \wedge C} \wedge R$$

$$\frac{A \leq C}{A \wedge B \leq C} \wedge L_1 \qquad\qquad \frac{B \leq C}{A \wedge B \leq C} \wedge L_2$$

- Easily justified by our meaning explanation
- Transitivity remains admissible
- Distributivity

$$\left[\frac{}{(A \to B) \wedge (A \to C) \leq A \to (B \wedge C)}\right]$$

would disturb orthogonality and is unsound with effects [Davies & Pf, ICFP'00]

Example: External vs Internal Invariants

- Reconsider example

$$
\begin{array}{rcl}
inc & = & \text{fix } inc.\, \lambda n.\ \text{case } n \\
 & & \qquad\qquad\qquad\quad \text{of } \epsilon \Rightarrow \epsilon\,\mathbf{1} \\
 & & \qquad\qquad\qquad\quad\ \mid x\,\mathbf{0} \Rightarrow x\,\mathbf{1} \\
 & & \qquad\qquad\qquad\quad\ \mid x\,\mathbf{1} \Rightarrow (inc\ x)\,\mathbf{0} \qquad \%\ inc\ x : \text{pos(!)}
\end{array}
$$

- Then

$$\vdash inc \downarrow (\text{bits} \rightarrow \text{bits}) \wedge (\text{nat} \rightarrow \text{pos})$$

$$(\text{bits} \rightarrow \text{bits}) \wedge (\text{nat} \rightarrow \text{pos}) \leq \text{nat} \rightarrow \text{nat}$$

$$(\text{bits} \rightarrow \text{bits}) \wedge (\text{nat} \rightarrow \text{pos}) \leq \text{pos} \rightarrow \text{pos}$$

- But

$$\not\vdash inc \downarrow \text{nat} \rightarrow \text{nat}$$

cannot be checked directly

Summary of Intersection Types

- Property types without term constructors
- Logically motivated subtyping
- Value restriction for soundness with effects
- No distributivity law for soundness with effects
- In practice may need to ascribe more explicit types
- Intersection orthogonal to all other types and constructor

Refinement Restriction

- System is cleanest with refinement restriction
- Segregate system explicitly into types (constructor types) and sorts (property types)
- Only sorts of similar structure may be compared or intersected
- Conservative over ML, including effects
- No further consideration in this talk (see [Freeman & Pf'91] [Freeman'94] [Davies'97])

Universal Type $\top$

- Introduction and elimination rules

$$\frac{}{\Gamma \vdash V \downarrow \top}\ \top I \qquad\qquad \text{(no } \top E \text{ rule)}$$

- Subtyping rules

$$\frac{}{A \leq \top}\ \top R \qquad\qquad \text{(no } \top L \text{ rule)}$$

- Value restriction necessary for progress theorem, otherwise, e.g.

$$[\vdash (\epsilon\ \epsilon) \downarrow \top]$$

- Confirms value restriction

- Useful for unreachable code, e.g.

$$\begin{array}{rcl} casenat_C & : & (\mathsf{nat} \to C \to (\mathsf{pos} \to C) \to (\mathsf{nat} \to C)) \\ & & \wedge\ (\mathsf{pos} \to \top \to (\mathsf{pos} \to C) \to (\mathsf{nat} \to C)) \end{array}$$

Outline

- Introduction
- Guiding Principles
- Atomic Subtyping
- Intersection Types
- **Union Types**
- [Dependent Types]
- Conclusion

Union Types $A \vee B$

- Another property type, not constructor type
- Introduction rules

$$\frac{\Gamma \vdash M \downarrow A}{\Gamma \vdash M \downarrow A \vee B} \vee I_1 \qquad \frac{\Gamma \vdash M \downarrow B}{\Gamma \vdash M \downarrow A \vee B} \vee I_2$$

- Problem: how do we write the elimination rule?
- A fundamental problem in natural deduction! [Prawitz'65] [Girard]
- Subtyping is straightforward ($\sim$ sequent calculus!)

$$\frac{A \leq B}{A \leq B \vee C} \vee R_1 \qquad \frac{A \leq C}{A \leq B \vee C} \vee R_2$$

$$\frac{A \leq C \qquad B \leq C}{A \vee B \leq C} \vee L$$

The Substitution Approach

- Due to [MacQueen, Plotkin, Sethi'86] and [Barbanera, Dezani-Ciancaglini, De'Liguoro'95]
- Union elimination

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

- Note uniformity of N in the two branches
- Does not satisfy type preservation:
  Different copies of M can reduce differently in $[M/x]N$
- Too general, even for pure calculus
- Undecidable

Towards a Solution

- First idea: require exactly one occurrence of x in N
- Second idea: account for bi-directionality
- Union elimination: for N linear in x,

$$\frac{\Gamma \vdash M \uparrow A \vee B \quad \Gamma, x{:}A \vdash N \downarrow C \quad \Gamma, x{:}B \vdash N \downarrow C}{\Gamma \vdash [M/x]N \downarrow C}$$

- Still not sound with effects(?)

Further Towards a Solution

- Third idea: require N to be an evaluation context.

$$\frac{\Gamma \vdash M \uparrow A \vee B \quad \Gamma, x{:}A \vdash E[x] \downarrow C \quad \Gamma, x{:}B \vdash E[x] \downarrow C}{\Gamma \vdash E[M] \downarrow C} \vee E$$

- Restores progress and preservation
- Much more restrictive than Barbanera et al.
- Setting and goals are different

Example

- Use

$$\begin{aligned}\Gamma_0 \;=\;& f : (B_1 \to C_1) \wedge (B_2 \to C_2),\\ & g : A \to (B_1 \vee B_2),\\ & x : A\end{aligned}$$

- Show $\Gamma_0 \vdash f\ (g\ x) \downarrow C_1 \vee C_2$
- Using evaluation context $f\ [\,]$

$$\dfrac{\Gamma_0 \vdash g\ x \uparrow B_1 \vee B_2 \qquad \dfrac{\Gamma_0, y{:}B_1 \vdash f\ y \downarrow C_1}{\Gamma_0, y{:}B_1 \vdash f\ y \downarrow C_1 \vee C_2} \qquad \dfrac{\Gamma_0, y{:}B_2 \vdash f\ y \downarrow C_2}{\Gamma_0, y{:}B_2 \vdash f\ y \downarrow C_2 \vee C_2}}{\Gamma_0 \vdash f\ (g\ x) \downarrow C_1 \vee C_2}$$

Empty Type $\bot$

- Zero-ary case of disjunction
- Introduction and elimination rules

$$\text{(no } \bot I \text{ rule)} \qquad \frac{\Gamma \vdash M \uparrow \bot}{\Gamma \vdash E[M] \downarrow C}\, \bot E$$

- Restriction to evaluation contexts critical
- Counterexample: for $abort : \mathsf{nat} \to \bot$,

$$((\epsilon\ \epsilon)\ (abort\ \epsilon)) \downarrow C$$

  for any C, but violates progress.

- Note: $(\epsilon\ \epsilon)\ [\,]$ is not an evaluation context!
- Subtyping

$$\text{(no } \bot R \text{ rule)} \qquad \frac{}{\bot \leq C}\, \bot L$$

Another Problem

- System is not yet general enough
- Example: use

$$\begin{array}{rcl}\Gamma_1 & = & f : \mathsf{nat} \to (B_1 \to C_1) \wedge (B_2 \to C_2), \\ & & h : \mathsf{nat} \to \mathsf{nat} \\ & & g : A \to (B_1 \vee B_2), \\ & & x : A\end{array}$$

- Show $\Gamma_1 \vdash f\ (h\ \epsilon)\ (g\ x) \downarrow C_1 \vee C_2$?
- Problem $f\ (h\ \epsilon)$ [] is not an evaluation context!

Solution

- Add "unary disjunction" rule

$$\frac{\Gamma \vdash M \uparrow A \qquad \Gamma, x{:}A \vdash E[x] \downarrow C}{\Gamma \vdash E[M] \downarrow C}$$

- Realizes a substitution principle that is normally admissible
- Also form of analytic cut
- Now

$$\frac{\Gamma_1 \vdash h\ \epsilon \uparrow \mathsf{nat} \qquad \dfrac{\Gamma_1, n{:}\mathsf{nat} \vdash f\ n \uparrow D \quad \Gamma_1, n{:}\mathsf{nat}, k{:}D \vdash k\ (g\ x) \downarrow C_1 \vee C_2}{\Gamma_1, n{:}\mathsf{nat} \vdash f\ n\ (g\ x) \downarrow C_1 \vee C_2}}{\Gamma_1 \vdash f\ (h\ \epsilon)\ (g\ x) \downarrow C_1 \vee C_2}$$

for

$$\begin{array}{rcl} \Gamma_1 & = & f : \mathsf{nat} \to (B_1 \to C_1) \wedge (B_2 \to C_2), \\ & & h : \mathsf{nat} \to \mathsf{nat}, \\ & & g : A \to (B_1 \vee B_2), \\ & & x : A \\ D & = & (B_1 \to C_1) \wedge (B_2 \to C_2) \end{array}$$

Summary of Tri-Directional Rules

- Binary case (union elimination)

$$\frac{\Gamma \vdash M \uparrow A \vee B \quad \Gamma, x{:}A \vdash E[x] \downarrow C \quad \Gamma, x{:}B \vdash E[x] \downarrow C}{\Gamma \vdash E[M] \downarrow C} \vee E$$

- Unary case (substitution)

$$\frac{\Gamma \vdash M \uparrow A \qquad \Gamma, x{:}A \vdash E[x] \downarrow C}{\Gamma \vdash E[M] \downarrow C}$$

- Zeroary case (contradiction)

$$\frac{\Gamma \vdash M \uparrow \bot}{\Gamma \vdash E[M] \downarrow C} \bot E$$

- Note: unary case is **not** general cut, but analytic!

Some Theorems

- Progress Theorem

  If $\vdash M : A$ *then either* $\vdash M$ *val or* $M \longrightarrow M'$.

- Preservation Theorem

  If $\vdash M : A$ *and* $M \longrightarrow M'$ *then* $\vdash M' : A$

- Tri-directional type-checking is decidable.
- Critical lemmas are substitution and various inversion properties
- Example: Determinacy

  If $\vdash V : A \vee B$ *then* $\vdash V : A$ *or* $\vdash V : B$

- Hold with and without mutable references

Tri-Directional Checking and Let-Normal Form

- Tri-directionality allows us to check the term in evaluation order
- Appears related to bi-directional checking after translation to let-normal form (2/3-continuation passing style, A-normal form)
- For example,

$$
\begin{array}{rcl}
f\ (h\ \epsilon)\ (g\ x) & \mapsto & \text{let } n = h\ \epsilon \text{ in} \\
& & \text{let } k = f\ n \text{ in} \\
& & \text{let } y = g\ x \text{ in} \\
& & \text{let } z = k\ y \text{ in} \\
& & z \text{ end end end end}
\end{array}
$$

Left Rules for Type-Checking

- Also considered by Barbanera et al. (there: admissible)
- The following left rules are sound, but not admissible

$$\frac{\Gamma, x{:}A, \Gamma' \vdash M \downarrow C}{\Gamma, x{:}A \wedge B, \Gamma' \vdash M \downarrow C}\ \wedge L_1 \qquad \frac{\Gamma, x{:}B, \Gamma' \vdash M \downarrow C}{\Gamma, x{:}A \wedge B, \Gamma' \vdash M \downarrow C}\ \wedge L_2$$

$$\frac{\Gamma, x{:}A, \Gamma' \vdash M \downarrow C \quad \Gamma, x{:}B, \Gamma' \vdash M \downarrow C}{\Gamma, x{:}A \vee B, \Gamma' \vdash M \downarrow C}\ \vee L$$

$$\frac{}{\Gamma, x{:}\bot, \Gamma' \vdash M \downarrow C}\ \bot L$$

- **Conjecture**: The correspondence between tri-directional checking and bi-directional checking of let-normal form is exact if we add the left rules to the typing judgment.

Related Work on This Correspondence

- [Sabry & Felleisen'94]
  Is Continuation-Passing Useful for Data Flow Analysis?

- [Damian & Danvy'00]
  Syntactic Accidents in Program Analysis

- [Palsberg & Wand'02]
  CPS Transformation of Flow Information

Connections to Commuting Conversions

- Under the coercion interpretation,
  - $\wedge \mapsto \times$ (product type)
  - $\top \mapsto 1$ (unit type)
  - $\vee \mapsto +$ (disjoint sum type)
  - $\perp \mapsto 0$ (void type)
- Different ways to apply contextual rules corresponds to certain commuting conversions on disjoint sum and void types
- These different versions are identified by CPS transformation [deGroote99,deGroote01]

Alternative Methods for Type Checking for Unions

- [Pierce'91]

  $$\mathit{case}\ M\ \text{of}\ x \Rightarrow N \quad \text{for} \quad [M/x]N$$

  determines where $\vee L$ rule can be applied. No effects. Note difference in operational semantics between two sides.

- [Wells, Dimock, Muller, Turbak'99]
  Virtual terms copied to establish bijection between valid terms and typing derivations. Designed as intermediate language only, for expressing flow information.

- [Palsberg & Pavlopoulou'00]
  Disjunction only in subtyping (not typing), designed for flow information.

Summary of Tri-Directional Checking

- Tri-directional type-checking combines
  - Synthesis ($\Gamma \vdash M \uparrow A$, given Γ, M, generates all A)
  - Analysis ($\Gamma \vdash M \downarrow A$, given Γ, M, A, verify)
  - Contextual rules (visit subterm in evaluation order)
- **Theorem**: Preservation and progress hold for call-by-value (even in the presence of effects)
- **Theorem**: Type checking is decidable (judgments are analytic on terms and types)
- **Theorem**: Conservative extension of various fragments (orthogonal definition of constructor types ($\rightarrow$, $\times$, 1, $+$, 0) and property types ($\wedge$, $\top$, $\vee$, $\bot$))

Practicality for Intersection Types

- Bi-directional checking is practical for $\wedge$, $\top$ in SML [Davies'97]
- Good tradeoff between verbosity, expressive power, and efficiency of type-checking
- Implements refinement restriction (conservative over ML)
- Property complexity determines efficiency
- Infeasible examples exist [Reynolds'96]
- Use of unions only for data types and pattern matching

Adding Union Types in Implementation

- Conjecture practicality with some efficiency improvements
  - Focusing strategy for subtyping [Davies & Pf'00]
  - Focusing strategy for typing
  - Lazy splitting of $A \vee B$
  - Memoization during multiple traversals
  - Algorithmic conservativity?
- Infeasible examples exist
- Anticipate sparing use of unions outside data types

Outline

- Introduction
- Guiding Principles
- Atomic Subtyping
- Intersection Types
- Union Types
- [**Dependent Types**]
- Conclusion

Universal and Existential Dependent Types

- Many important data structure invariants cannot be expressed, for example
  - Lists of length n
  - Closed terms in de Bruijn form
  - Height invariant on balanced trees
- Extend simple types to integrate indexed types (list(i)), universal dependent types ($\Pi a.\, A$), and existential dependent types ($\Sigma a.\, A$) [Xi'98, Xi & Pf'98,99]
- Prior work suffered from a lack of intersections
- Ad hoc treatment of existential dependent types

Index Domain

- New hypotheses $a{:}\gamma$ for index variables a
- New hypotheses $i \doteq j$ for index terms i, j.
- New judgment $\Gamma \vdash i : \gamma$ for index domain
- Generalize subtyping $\Gamma \vdash A \leq B$
- New subtyping for indexed data types δ, δ'

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

Example: Lists

- Introduction

$$\frac{}{\Gamma \vdash \mathbf{nil} \downarrow \mathsf{list}(0)} \qquad \frac{\Gamma \vdash M \downarrow \mathsf{bits} \quad \Gamma \vdash L \downarrow \mathsf{list}(n)}{\Gamma \vdash \mathbf{cons}(M, L) \downarrow \mathsf{list}(n+1)}$$

- Elimination

$$\frac{\begin{array}{l} \Gamma \vdash L \uparrow \mathsf{list}(n) \\ \Gamma, n \doteq 0 \vdash N_1 \downarrow C \\ \Gamma, x{:}\mathsf{bits}, a{:}\mathit{nat}, n \doteq a+1, l{:}\mathsf{list}(a) \vdash N_1 \downarrow C \end{array}}{\Gamma \vdash \mathsf{case}\ L\ \mathsf{of}\ \mathbf{nil} \Rightarrow N_1 \mid \mathbf{cons}(x, l) \Rightarrow N_2 \downarrow C}$$

Example Types

- Definite

$$append \;:\; \Pi n{:}\mathit{nat}.\, \Pi k{:}\mathit{nat}.\, \mathsf{list}(n) \to \mathsf{list}(k) \to \mathsf{list}(n+k)$$

- Indefinite

$$\begin{aligned} hd \;&:\; (\mathsf{list}(0) \to \bot) \wedge (\Pi n{:}\mathit{nat}.\, \mathsf{list}(n+1) \to \mathsf{list}(n)) \\ tl \;&:\; \Pi n{:}\mathit{nat}.\, \mathsf{list}(n) \to (\mathsf{list}(n-1) \vee \mathsf{list}(0)) \\ filter0 \;&:\; \Pi n{:}\mathit{nat}.\, \mathsf{list}(n) \to \Sigma k{:}\mathit{nat}.\, \mathsf{list}(k) \end{aligned}$$

- Existential types are not “optional” like unions!

Universal Dependent Types

- Universal dependent type as property type
- Universal introduction

$$\frac{\Gamma, a{:}\gamma \vdash M \downarrow A}{\Gamma \vdash M \downarrow \Pi a{:}\gamma.\, A}\,\Pi I$$

- Universal elimination

$$\frac{\Gamma \vdash M \uparrow \Pi a{:}\gamma.\, A \qquad \Gamma \vdash i : \gamma}{\Gamma \vdash M \uparrow [i/a]A}\,\Pi E$$

- Subtyping

$$\frac{\Gamma, b{:}\gamma \vdash A \leq B}{\Gamma \vdash A \leq \forall b{:}\gamma.\, B}\,\forall R \qquad\qquad \frac{\Gamma \vdash [i/a]A \leq B \qquad \Gamma \vdash i : \gamma}{\Gamma \vdash \forall a{:}\gamma.\, A \leq B}\,\forall L$$

Existential Dependent Types

- Existential dependent types as property type
- Existential introduction

$$\frac{\Gamma \vdash M \downarrow [i/a]A \quad \Gamma \vdash i : a}{\Gamma \vdash M \downarrow \Sigma a{:}\gamma.\, A}\ \Sigma I$$

- Existential elimination (requires contextual form)

$$\frac{\Gamma \vdash M \uparrow \Sigma a{:}\gamma.\, A \quad \Gamma, a{:}\gamma, x{:}A \vdash E[x] \downarrow C}{\Gamma \vdash E[M] \downarrow C}\ \Sigma E$$

- Subtyping

$$\frac{\Gamma \vdash A \leq [i/b]B \quad \Gamma \vdash i : \gamma}{\Gamma \vdash A \leq \Sigma b{:}\gamma.\, B}\ \Sigma R \qquad \frac{\Gamma, a{:}\gamma \vdash A \leq B}{\Gamma \vdash \Sigma a{:}\gamma.\, A \leq B}\ \Sigma L$$

Summary: Dependent Types

- Definition orthogonal to other constructs
- Meta-theoretic analysis carries over
- For type-checking, collect equational constraints in index domain
- For decidability, constraint domain must be decidable in the presence of universal and existential variables
- Example: Presburger arithmetic
- Existential types are critical (e.g., filter)
- Clean formulation only with contextual rules

Outline

- Introduction
- Guiding Principles
- Atomic Subtyping
- Intersection Types
- Union Types
- [Dependent Types]
- **Conclusion**

Other Related Work

- Intersection types (many)
- Forsythe [Reynolds'88] [Reynolds'96]
- Intersections and explicit polymorphism [Pierce'91] [Pierce'97]
- Soft types (many)
- Shape analysis and software model checking (many)

Future Work: Parametric Polymorphism

- ML-style polymorphism via refinement restriction
- Bi-directionality for full parametric polymorphism requires subtyping
- Value restriction on $\forall I$ for soundness with effects [Davies & Pf'00]
- Subtyping undecidable [Wells'95][Tiuryn & Urzyczyn'96] even without distributivity [Chrząszcz'98]
- Conjecture predicative part with universes decidable
- Combine with local inference? [Pierce & Turner'97]

Other Future Work

- General case of data types (mostly done)
- Precise relationship to logic, CPS, commuting conversions
- Version for call-by-name, lazy evaluation
- Translation to monadic meta-language to encapsulate effects
- Sequential pattern matching with union and existential
- Apply where types express effects or resources(!)

Summary

- Refinement types to statically verify program invariants
- System constructed orthogonally based on judgments
- Conservativity with respect to fragments
- Bi-directional checking for intersection and universal types
- Tri-directional checking for union and existential types
- Type-checking in evaluation order
- Sound with effects through value and evaluation context restrictions
- Preliminary examples indicate it may be practical

Intersections are Unsound with Effects

- Counterexample

$$
\begin{array}{rll}
\text{let} & x = \mathsf{ref}\,(\epsilon\,\mathbf{1}) & : \mathsf{nat\ ref} \wedge \mathsf{pos\ ref} \\
\text{in} & & \\
& x := \epsilon; & \%\ \textit{use}\ x : \mathsf{nat\ ref} \\
& !\,x & \%\ \textit{use}\ x : \mathsf{pos\ ref} \\
\text{end} & : \mathsf{pos} &
\end{array}
$$

  evaluates to ϵ which does not have type pos.

- Analogous counterexample with parametric polymorphism:

$$
\begin{array}{rll}
\text{let} & x = \mathsf{ref}\,(\lambda y.\,y) & : \forall\alpha.\,(\alpha \to \alpha)\,\mathsf{ref} \\
\text{in} & & \\
& x := (\lambda y.\,\epsilon); & \%\ \textit{use}\ x : (\mathsf{nat} \to \mathsf{nat})\,\mathsf{ref} \\
& (!\,x)\,(\epsilon\,\mathbf{1}) & \%\ \textit{use}\ x : (\mathsf{pos} \to \mathsf{pos})\,\mathsf{ref} \\
\text{end} & : \mathsf{pos} &
\end{array}
$$

Distributivity is Unsound with Effects

- Recall distributivity

$$\left[\frac{}{(A \to B) \wedge (A \to C) \leq A \to (B \wedge C)}\right]$$

- Counterexample:

$$\vdash \lambda u.\, \mathsf{ref}\,(\epsilon\, \mathbf{1}) \quad : \quad (\mathsf{unit} \to \mathsf{nat\ ref}) \wedge (\mathsf{unit} \to \mathsf{pos\ ref})$$

by distributivity and subsumption:

$$\vdash \lambda u.\, \mathsf{ref}\,(\epsilon\, \mathbf{1}) \quad : \quad \mathsf{unit} \to (\mathsf{nat\ ref} \wedge \mathsf{pos\ ref})$$

$$\vdash (\lambda u.\, \mathsf{ref}\,(\epsilon\, \mathbf{1}))\,\langle\,\rangle \quad : \quad \mathsf{nat\ ref} \wedge \mathsf{pos\ ref}$$

- In a program:

$$\begin{array}{ll} \mathsf{let} & x = (\lambda u.\, \mathsf{ref}\,(\epsilon\, \mathbf{1}))\,\langle\,\rangle \quad : \mathsf{nat\ ref} \wedge \mathsf{pos\ ref} \\ \mathsf{in} & \ldots\ \mathsf{end} \qquad\qquad\qquad\qquad \textit{\% as on previous slide} \end{array}$$