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
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 $A$ is a type (elided in this talk)
- $M : A$ $M$ has type $A$
- Hypotheses $x_1 : A_1, \ldots, x_n : A_n$, $x_i$ distinct (write $\Gamma$)
- $\Gamma \vdash M \text{ val}$ $M$ is a value (write $V$)
- Defining properties for hypothetical judgments
  - Hypothesis rules
    \[
    \frac{
    \Gamma, x: A, \Gamma' \vdash x : A
    }{
    \Gamma, x: A, \Gamma' \vdash x \text{ val}
    }\]
  - Substitution principle (theorem)
    If $\Gamma \vdash V : A$ and $\Gamma, x: A, \Gamma' \vdash N : C$ then
    $\Gamma, \Gamma' \vdash [V/x]N : C$
Computations Judgments

- $M \rightarrow_{\beta} M'$ \quad $M$ beta-reduces to $M'$
- $E \text{ctx}$ \quad $E$ is an evaluation context, hole $[]$

\[ [] \text{ctx} \]

- $E[M]$ replaces hole in $E$ by $M$
- $M \rightarrow M'$ \quad $M$ reduces to $M'$
- Closure rule

\[
\frac{M \rightarrow_{\beta} M'}{E[M] \rightarrow E[M']}
\]
Principles of Computation

• Progress principle (theorem)

\[ \text{If } \vdash M : A \text{ then either } \vdash M \text{ val or } M \rightarrow M'. \]

• Preservation principle (theorem)

\[ \text{If } \vdash M : A \text{ and } M \rightarrow M' \text{ then } \vdash M' : A \]

• 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

\[
\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \to B} \quad \text{I} \quad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \quad \text{E}
\]

- Values

\[
\Gamma \vdash \lambda x. M \text{ val}
\]

- Computation (introduction followed by elimination)

\[
(\lambda x. M)\ V \to_\beta [V/x]M
\]

\[
\frac{E \text{ ctx}}{EM \text{ ctx}} \quad \frac{V \text{ val}}{VE \text{ ctx}}
\]
Mechanisms for Type-Checking

- Type synthesis (Church)
  - Given $\Gamma, M$, 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 $\Gamma, M, A$ 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 $\Gamma, M$, synthesize $A$ with $\Gamma \vdash M : A$
- $\Gamma \vdash M \downarrow A$  Given $\Gamma, M, A$, analyze if $\Gamma \vdash M : A$
- Hypothesis rule
  \[
  \Gamma, x : A, \Gamma' \vdash x \uparrow A
  \]
- New expression ($M : A$)
- Mutual dependencies (revisit later)
  \[
  \frac{\Gamma \vdash M \uparrow A}{\Gamma \vdash M \downarrow A} \quad \frac{\Gamma \vdash M \downarrow A}{\Gamma \vdash (M : A) \uparrow A}
  \]
- Several substitution principles (elided)
Bi-Directional Type-Checking of Functions

- Introduction forms are analyzed
  \[ \frac{\Gamma, x:A \vdash M \downarrow B}{\Gamma \vdash \lambda x. M \downarrow A \to B} \rightarrow I \]

- Elimination forms are synthesized
  \[ \frac{\Gamma \vdash M \uparrow A \to B \quad \Gamma \vdash N \downarrow A}{\Gamma \vdash MN \uparrow B} \rightarrow E \]

- Read ‘\( \uparrow \)’ and ‘\( \downarrow \)’ as ‘:\( \cdot \)’ to obtain type assignment rules
- No type annotations in normal forms. E.g., for any \( A \),
  \[ \vdash \lambda f. \lambda x. f(fx) \downarrow (A \to A) \to (A \to A) \]
- Annotate redexes, e.g.,
  \[ \vdash (\lambda x. x : \text{bits} \to \text{bits}) (\epsilon 110) \uparrow \text{bits} \]
Definitions

- Internalize substitution principle

\[
\frac{\Gamma \vdash M \uparrow A \quad \Gamma, x:A \vdash N \downarrow C}{\Gamma \vdash \text{let } x = M \text{ in } N \text{ end} \downarrow C}
\]

- Computation

\[
\frac{E \text{ ctx}}{\text{let } x = E \text{ in } N \text{ end ctx}}
\]

- In practice, use definitions instead of redices

\[\vdash \text{let } f = (\lambda x. x \colon \text{bits} \to \text{bits}) \text{ in } f (\epsilon 110) \text{ end} \downarrow \text{bits}\]

or \[\vdash \text{let } f : \text{bits} \to \text{bits} = \lambda x. x \text{ in } f (\epsilon 110) \text{ end} \downarrow \text{bits}\]
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 $\Gamma$, $M$, and $A$: any derivation mentions only constituent terms and types of $\Gamma$, $M$, $A$
Adding Data Types

- Proceed by example: bit strings and natural numbers
- For general case, see [Dunfield’02] [Davies’02]
- Introduction forms
  \[ \Gamma \vdash M \downarrow \text{bits} \]
  \[ \Gamma \vdash \epsilon \downarrow \text{bits} \]
  \[ \Gamma \vdash M \ 0 \downarrow \text{bits} \]
  \[ \Gamma \vdash M \ 1 \downarrow \text{bits} \]

- \( \epsilon \) represents empty string, 0 and 1 are postfix operators.
- For example: \( \Gamma 0 \downarrow = \epsilon \), \( \Gamma 6 \downarrow = \epsilon 110 \).
- Elimination form
  \[ \Gamma \vdash M \uparrow \text{bits} \]
  \[ \Gamma \vdash N_e \downarrow C \]
  \[ \Gamma, x : \text{bits} \vdash N_0 \downarrow C \]
  \[ \Gamma, y : \text{bits} \vdash N_1 \downarrow C \]
  \[ \Gamma \vdash \text{case } M \text{ of } \epsilon \Rightarrow N_e \ | \ x \ 0 \Rightarrow N_0 \ | \ y \ 1 \Rightarrow N_1 \downarrow C \]
Computation on Data Types

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

\[
\Gamma, u : A \vdash M \Downarrow A \\
\Gamma \vdash \text{fix } u. M \Downarrow A \\
\text{fix } u. M \rightarrow^\beta [\text{fix } u. M/u] M
\]

- 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 \( \text{fix } u. \lambda x. M \)
Data Structure Invariants and Subtyping

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

\begin{align*}
    \text{Natural Numbers} & \quad \text{nat} ::= \epsilon | \text{pos} \\
    \text{Positive Numbers} & \quad \text{pos} ::= \text{pos}0 | \text{nat}1
\end{align*}

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

• New rules (ignore redundancy):

\[
\begin{align*}
\Gamma \vdash \epsilon \downarrow \text{n} & \quad \text{(no } \epsilon \downarrow \text{pos)} \\
\Gamma \vdash M \downarrow \text{pos} & \quad \Gamma \vdash M \downarrow \text{pos} \\
\Gamma \vdash M0 \downarrow \text{n} & \quad \Gamma \vdash M0 \downarrow \text{pos} \\
\Gamma \vdash M \downarrow \text{n} & \quad \Gamma \vdash M \downarrow \text{n} \\
\Gamma \vdash M1 \downarrow \text{n} & \quad \Gamma \vdash M1 \downarrow \text{pos} \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash M \uparrow \text{n} & \quad \Gamma \vdash N_\epsilon \downarrow C & \quad \Gamma, x: \text{pos} \vdash N_0 \downarrow C & \quad \Gamma, y: \text{nat} \vdash N_1 \downarrow C \\
\Gamma \vdash \text{case } M \text{ of } \epsilon & \Rightarrow N_\epsilon | x0 \Rightarrow N_0 | y1 \Rightarrow N_1 \downarrow C \\
\Gamma \vdash M \uparrow \text{pos} & \quad \text{(no } N_\epsilon \downarrow C) & \quad \Gamma, x: \text{pos} \vdash N_0 \downarrow C & \quad \Gamma, y: \text{nat} \vdash N_1 \downarrow C \\
\Gamma \vdash \text{case } M \text{ of } \epsilon & \Rightarrow N_\epsilon | x0 \Rightarrow N_0 | y1 \Rightarrow N_1 \downarrow C \\
\end{align*}
\]
Subtyping Judgment

- New judgment $A \leq B$  \quad $A$ 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)
  \[
  \text{If } A \leq B \text{ and } B \leq C \text{ then } A \leq C.
  \]
- Subsumption rule, replaces $\uparrow \downarrow$
  \[
  \frac{\Gamma \vdash M \uparrow A \quad \Gamma \vdash A \leq C}{\Gamma \vdash M \downarrow C}
  \]
Subtyping of Data Types

- From the typing rules:
  \[ \text{pos} \leq \text{nat} \quad \text{nat} \leq \text{bits} \quad \text{pos} \leq \text{bits} \]

- In general, a lattice

- Example of need for subsumption rule
  \[ x : \text{pos} \vdash x \downarrow \text{nat} \]
  since \[ x : \text{pos} \vdash x \uparrow \text{pos} \]
  and \[ \text{pos} \leq \text{nat} \]

- Subtyping of functions
  \[
  \frac{B_1 \leq A_1 \quad A_2 \leq B_2}{A_1 \to A_2 \leq B_1 \to B_2}
  \]
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 $E[\cdot]$, reduction $M \rightarrow M'$
- 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 $\text{shiftl} = \lambda x. x \mathbf{0}$.

\[
\begin{align*}
\vdash \lambda x. x \mathbf{0} & \downarrow \text{bits} \to \text{bits} \\
\vdash \lambda x. x \mathbf{0} & \downarrow \text{nat} \to \text{bits} \\
\vdash \lambda x. x \mathbf{0} & \downarrow \text{pos} \to \text{pos}
\end{align*}
\]

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

• Especially troublesome for recursive functions

\[
\neg \quad \text{fix inc. } \lambda n. \quad \text{case } n \\
\quad \quad \text{of } \epsilon \Rightarrow \epsilon \mathbf{1} \\
\quad \quad \quad | \ x \mathbf{0} \Rightarrow x \mathbf{1} \\
\quad \quad \quad | \ x \mathbf{1} \Rightarrow (\text{inc } x) \mathbf{0} \quad \% \text{ inc } x : \text{pos?} \\
\downarrow \quad \text{nat} \to \text{nat}
\]
Intersection Types $A \land B$

- Introduction and elimination forms

\[
\frac{\Gamma \vdash V \downarrow A \quad \Gamma \vdash V \downarrow B}{\Gamma \vdash V \downarrow A \land B} \ \land I
\]

\[
\frac{\Gamma \vdash M \uparrow A \land B}{\Gamma \vdash M \uparrow A} \ \land E_1 \quad \frac{\Gamma \vdash M \uparrow A \land B}{\Gamma \vdash M \uparrow B} \ \land E_2
\]

- Subject of judgment identical in premises and conclusion

- $A \land B$ 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 (∼ sequent calculus)

\[
\frac{A \leq B \quad A \leq C}{A \leq B \land C} ^\land R
\]

\[
\frac{A \leq C}{A \land B \leq C} ^\land L_1 \quad \frac{B \leq C}{A \land B \leq C} ^\land L_2
\]

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

\[
\left[(A \rightarrow B) \land (A \rightarrow C) \leq A \rightarrow (B \land C)\right]
\]

would disturb orthogonality and is unsound with effects [Davies & Pf, ICFP’00]
Example: External vs Internal Invariants

- Reconsider example

\[
\text{inc} \equiv \text{fix inc}. \lambda n. \text{case } n \\
\text{of } \epsilon \Rightarrow \epsilon 1 \\
| x \text{0} \Rightarrow x 1 \\
| x 1 \Rightarrow (\text{inc } x) 0 \quad \% \text{ inc } x : \text{pos(!)}
\]

- Then

\[
\vdash \text{inc} \downarrow (\text{bits} \rightarrow \text{bits}) \land (\text{nat} \rightarrow \text{pos}) \\
(\text{bits} \rightarrow \text{bits}) \land (\text{nat} \rightarrow \text{pos}) \leq \text{nat} \rightarrow \text{nat} \\
(\text{bits} \rightarrow \text{bits}) \land (\text{nat} \rightarrow \text{pos}) \leq \text{pos} \rightarrow \text{pos}
\]

- But

\[
\not\vdash \text{inc} \downarrow \text{nat} \rightarrow \text{nat}
\]

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
  \[ \Gamma \vdash V \downarrow \top \top I \]  
  (no $\top E$ rule)

- Subtyping rules
  \[ A \leq \top \top R \]  
  (no $\top L$ rule)

- Value restriction necessary for progress theorem, otherwise, e.g.
  \[ [ \vdash (\epsilon \epsilon) \downarrow \top ] \]

- Confirms value restriction

- Useful for unreachable code, e.g.
  \[
  \text{casenat}_C : \quad (\text{nat} \rightarrow C \rightarrow (\text{pos} \rightarrow C') \rightarrow (\text{nat} \rightarrow C')) \\
  \wedge (\text{pos} \rightarrow \top \rightarrow (\text{pos} \rightarrow C') \rightarrow (\text{nat} \rightarrow C'))
  \]
Outline

• Introduction
• Guiding Principles
• Atomic Subtyping
• Intersection Types
• Union Types
• [Dependent Types]
• Conclusion
Union Types $A \lor B$

- Another property type, not constructor type

- Introduction rules

\[
\frac{\Gamma \vdash M \downarrow A}{\Gamma \vdash M \downarrow A \lor B} \lor I_1 \quad \frac{\Gamma \vdash M \downarrow B}{\Gamma \vdash M \downarrow A \lor B} \lor I_2
\]

- Problem: how do we write the elimination rule?

- A fundamental problem in natural deduction! [Prawitz’65] [Girard]

- Subtyping is straightforward ($\sim$ sequent calculus!)

\[
\frac{A \leq B}{A \leq B \lor C} \lor R_1 \quad \frac{A \leq C}{A \leq B \lor C} \lor R_2
\]

\[
\frac{A \leq C \quad B \leq C}{A \lor B \leq C} \lor L
\]
The Substitution Approach

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

- Union elimination

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

- 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 \( x \),

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

- Still not sound with effects(?)
Further Towards a Solution

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

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

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

- Use

\[ \Gamma_0 = f : (B_1 \to C_1) \land (B_2 \to C_2), \]
\[ g : A \to (B_1 \lor B_2), \]
\[ x : A \]

- Show \( \Gamma_0 \vdash f (gx) \downarrow C_1 \lor C_2 \)

- Using evaluation context \( f [\] \)

\[
\begin{align*}
\Gamma_0 \vdash g x \uparrow B_1 \lor B_2 & \quad \Gamma_0, y : B_1 \vdash f y \downarrow C_1 & \quad \Gamma_0, y : B_2 \vdash f y \downarrow C_2 \\
\Gamma_0, y : B_1 \vdash f y \downarrow C_1 \lor C_2 & \quad \Gamma_0, y : B_2 \vdash f y \downarrow C_2 \lor C_2 \\
\Gamma_0 \vdash f (gx) \downarrow C_1 \lor C_2
\end{align*}
\]
Empty Type \( \bot \)

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

\[
\begin{align*}
\Gamma &\vdash M \uparrow \bot & \Gamma &\vdash E[M] \downarrow C \quad \bot^E
\end{align*}
\]

(no \( \bot I \) rule)

- Restriction to evaluation contexts critical
- Counterexample: for \( \text{abort} : \text{nat} \rightarrow \bot \),

\[
((\varepsilon \varepsilon) (\text{abort} \varepsilon)) \downarrow C
\]

for any \( C \), but violates progress.

- Note: \((\varepsilon \varepsilon) []\) is not an evaluation context!
- Subtyping

\[
\begin{align*}
\text{(no } \bot R \text{ rule)} &\quad \bot \leq C \quad \bot^L
\end{align*}
\]
Another Problem

- System is not yet general enough
- Example: use

\[ \Gamma_1 = f : \text{nat} \to (B_1 \to C_1) \land (B_2 \to C_2), \]
\[ h : \text{nat} \to \text{nat} \]
\[ g : A \to (B_1 \lor B_2), \]
\[ x : A \]

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

• Add “unary disjunction” rule

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

• Realizes a substitution principle that is normally admissible

• Also form of analytic cut

• Now

\[
\frac{\Gamma_1, n:\text{nat} \vdash f \ n \uparrow D \quad \Gamma_1, n:\text{nat}, k:D \vdash k \ (g \ x) \downarrow C_1 \lor C_2}{\Gamma_1 \vdash h \ \epsilon \uparrow \text{nat}} \quad \frac{\Gamma_1, n:\text{nat} \vdash f \ n \ (g \ x) \downarrow C_1 \lor C_2}{\Gamma_1 \vdash f \ (h \ \epsilon) \ (g \ x) \downarrow C_1 \lor C_2}
\]

for

\[
\begin{align*}
\Gamma_1 &= f : \text{nat} \to (B_1 \to C_1) \land (B_2 \to C_2), \\
h : \text{nat} \to \text{nat}, \\
g : A \to (B_1 \lor B_2), \\
x : A \\
D &= (B_1 \to C_1) \land (B_2 \to C_2)
\end{align*}
\]
Summary of Tri-Directional Rules

- **Binary case (union elimination)**
  \[ \Gamma \vdash M \uparrow A \vee B \quad \Gamma, x : A \vdash E[x] \downarrow C \quad \Gamma, x : B \vdash E[x] \downarrow C \]
  \[ \Gamma \vdash E[M] \downarrow C \quad \sqrt{E} \]

- **Unary case (substitution)**
  \[ \Gamma \vdash M \uparrow A \quad \Gamma, x : A \vdash E[x] \downarrow C \]
  \[ \Gamma \vdash E[M] \downarrow C \]

- **Zeroary case (contradiction)**
  \[ \Gamma \vdash M \uparrow \bot \]
  \[ \Gamma \vdash E[M] \downarrow C \quad \bot E \]

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

- Progress Theorem

\[ \text{If } \vdash M : A \text{ then either } \vdash M \text{ val or } M \rightarrow M'. \]

- Preservation Theorem

\[ \text{If } \vdash M : A \text{ and } M \rightarrow M' \text{ then } \vdash M' : A \]

- Tri-directional type-checking is decidable.

- Critical lemmas are substitution and various inversion properties

- Example: Determinacy

\[ \text{If } \vdash V : A \lor B \text{ then } \vdash V : A \text{ or } \vdash V : B \]

- 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,

\[
\begin{align*}
\text{let } n &= h \epsilon \text{ in } \\
\text{let } k &= fn \text{ in } \\
\text{let } y &= gx \text{ in } \\
\text{let } z &= ky \text{ in } \\
& z \text{ end end end end }
\end{align*}
\]
Left Rules for Type-Checking

- Also considered by Barbanera et al. (there: admissible)

- The following left rules are sound, but not admissible

\[
\frac{\Gamma, x:A, \Gamma' \vdash M \downarrow C}{\Gamma, x:A \land B, \Gamma' \vdash M \downarrow C} \quad \land L_1
\]

\[
\frac{\Gamma, x:B, \Gamma' \vdash M \downarrow C}{\Gamma, x:A \land B, \Gamma' \vdash M \downarrow C} \quad \land L_2
\]

\[
\frac{\Gamma, x:A, \Gamma' \vdash M \downarrow C}{\Gamma, x:A \lor B, \Gamma' \vdash M \downarrow C} \quad \lor L
\]

\[
\frac{\Gamma, x:B, \Gamma' \vdash M \downarrow C}{\Gamma, x:A \lor B, \Gamma' \vdash M \downarrow C} \quad \lor L
\]

- **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,
  
  - \( \land \leftrightarrow \times \) (product type)
  
  - \( \top \leftrightarrow 1 \) (unit type)
  
  - \( \lor \leftrightarrow + \) (disjoint sum type)
  
  - \( \perp \leftrightarrow 0 \) (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]

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

  determines where $\forall L$ 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 $\Gamma$, $M$, generates all $A$)
  - Analysis ($\Gamma \vdash M \downarrow A$, given $\Gamma$, $M$, $A$, 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, $\top$, 0) and property types ($\wedge$, $\top$, $\lor$, $\bot$))
Practicality for Intersection Types

- Bi-directional checking is practical for $\wedge$, $T$ 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 $A \lor B$
  - 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 ($\text{list}(i)$), universal dependent types ($\Pi a. A$), and existential dependent types ($\Sigma a. A$) \cite{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 \( \delta, \delta' \)

\[
\frac{\Gamma \vdash \delta \preceq \delta' \quad \Gamma \vdash i \doteq j}{\Gamma \vdash \delta(i) \leq \delta'(j)}
\]
Example: Lists

• Introduction

\[ \Gamma \vdash \text{nil} \downarrow \text{list}(0) \quad \Gamma \vdash M \downarrow \text{bits} \quad \Gamma \vdash L \downarrow \text{list}(n) \quad \Gamma \vdash \text{cons}(M, L) \downarrow \text{list}(n + 1) \]

• Elimination

\[ \Gamma \vdash L \uparrow \text{list}(n) \]

\[ \Gamma, n \doteq 0 \vdash N_1 \downarrow C \]

\[ \Gamma, x:\text{bits}, a:\text{nat}, n \doteq a + 1, l:\text{list}(a) \vdash N_1 \downarrow C \]

\[ \Gamma \vdash \text{case } L \text{ of } \text{nil} \Rightarrow N_1 \mid \text{cons}(x, l) \Rightarrow N_2 \downarrow C \]
Example Types

- **Definite**

  $$append : \prod n: \mathbb{nat}. \prod k: \mathbb{nat}. \text{list}(n) \to \text{list}(k) \to \text{list}(n + k)$$

- **Indefinite**

  $$hd : (\text{list}(0) \to \bot) \land (\prod n: \mathbb{nat}. \text{list}(n + 1) \to \text{list}(n))$$

  $$tl : \prod n: \mathbb{nat}. \text{list}(n) \to (\text{list}(n - 1) \lor \text{list}(0))$$

  $$filter0 : \prod n: \mathbb{nat}. \text{list}(n) \to \Sigma k: \mathbb{nat}. \text{list}(k)$$

- **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 \prod a : \gamma. A} \quad \text{\(\Pi I\)}
\]

- Universal elimination

\[
\frac{\Gamma \vdash M \uparrow \prod a : \gamma. A \quad \Gamma \vdash i : \gamma}{\Gamma \vdash M \uparrow [i/a] A} \quad \text{\(\Pi E\)}
\]

- Subtyping

\[
\frac{\Gamma, b : \gamma \vdash A \leq B}{\Gamma \vdash A \leq \forall b : \gamma. B} \quad \text{\(\forall R\)}
\]

\[
\frac{\Gamma \vdash [i/a] A \leq B \quad \Gamma \vdash i : \gamma}{\Gamma \vdash \forall a : \gamma. A \leq B} \quad \text{\(\forall L\)}
\]
Existential Dependent Types

- Existential dependent types as property type
- Existential introduction

\[
\Gamma \vdash M \downarrow [i/a]A \quad \Gamma \vdash i : a \\
\Gamma \vdash M \downarrow \Sigma a:\gamma. A
\]

\[\Sigma I\]

- Existential elimination (requires contextual form)

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

\[\Sigma E\]

- Subtyping

\[
\Gamma \vdash A \leq [i/b]B \quad \Gamma \vdash i : \gamma \\
\Gamma \vdash A \leq \Sigma b:\gamma. B
\]

\[\Sigma R\]

\[
\Gamma, a:\gamma \vdash A \leq B \\
\Gamma \vdash \Sigma a:\gamma. A \leq B
\]

\[\Sigma L\]
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., \textit{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

  ```
  let x = ref (ε 1) : nat ref ∧ pos ref in
  x := ε; % use x : nat ref
  ! x % use x : pos ref
  end : pos
  ```

  evaluates to $\epsilon$ which does not have type pos.

- Analogous counterexample with parametric polymorphism:

  ```
  let x = ref (λ y. y) : ∀α. (α → α) ref in
  x := (λ y. ε); % use x : (nat → nat) ref
  (! x) (ε 1) % use x : (pos → pos) ref
  end : pos
  ```
Distributivity is Unsound with Effects

- Recall distributivity

\[
(A \rightarrow B) \land (A \rightarrow C) \leq A \rightarrow (B \land C)
\]

- Counterexample:

\[\vdash \lambda u. \text{ref} (\epsilon \ 1) : (\text{unit} \rightarrow \text{nat ref}) \land (\text{unit} \rightarrow \text{pos ref})\]

by distributivity and subsumption:

\[\vdash \lambda u. \text{ref} (\epsilon \ 1) : \text{unit} \rightarrow (\text{nat ref} \land \text{pos ref})\]

\[\vdash (\lambda u. \text{ref} (\epsilon \ 1)) \langle \rangle : \text{nat ref} \land \text{pos ref}\]

- In a program:

\[
\text{let } x = (\lambda u. \text{ref} (\epsilon \ 1)) \langle \rangle : \text{nat ref} \land \text{pos ref}
\]

\[
\text{in } \ldots \text{ end } \quad \% \text{as on previous slide}
\]