Outline

- Verificationist and pragmatist meaning theory
- Canonical proofs and atomic subtyping
- Identity and substitution
- Defining higher-order subtyping
- Sound and complete rules for higher-order subtyping
- Intersections
- Higher-order subtyping extended
- Monads: the bridge to functional programming
- The value restriction and distributivity
What is Logic About?

• Not truth, but *consequence*
• Not particular propositions, but *arbitrary* propositions

\[
\begin{align*}
&\text{socrates is a man} & P(c) \text{ true} \\
&\text{all men are mortal} & \forall x. P(x) \supset Q(x) \text{ true} \\
\hline
&\text{socrates is mortal} & Q(c) \text{ true}
\end{align*}
\]

• Proofs are *parametric* in atomic propositions
• If \( A \text{ true} \) then \([B/P]A \text{ true}\) for any \( B\)
The Meaning of Propositions

• Atomic propositions are parameters
• Meaning of compound propositions should be determined by their constituents
• *Verificationist*: meaning of a proposition is determined by its verifications

\[
\frac{A \text{ true} \quad B \text{ true}}{A \& B \text{ true}} \quad \&I
\]

• Introduction rules define: read bottom-up
• Elimination rules are justified from introduction rules
The Meaning of Propositions

- **Pragmatist**: meaning of a proposition is determined by its uses

\[
\frac{A \land B \text{ true}}{A \text{ true}} \quad \&E_1 \quad \frac{A \land B \text{ true}}{B \text{ true}} \quad \&E_2
\]

- Elimination rules define: read top-down
- Introduction rules are justified from elimination rules
An Aside

- Verificationist = ML programmer (= call-by-value)
- Pragmatist = Haskell programmer (= call-by-name)
- See [Zeilberger POPL’08]
- Origins in [Dummett’76] [Martin-Löf’83]
Hypothetical Reasoning

- Hypothetical reasoning from a notion of consequence

\[
\begin{align*}
A \land B & \equiv B \land A \\
A \land B & \equiv B \land A
\end{align*}
\]

- Written as hypothetical judgment

\[A \land B \equiv B \land A\]
Canonical Proofs

- Verificationist to establish conclusion
- Pragmatist to use hypotheses
- Meet at an atomic proposition $P$

\[
\frac{A \text{ verif} \quad B \text{ verif}}{A \& B \text{ verif} \quad \& I} \quad \frac{P \text{ use} \quad P = Q}{Q \text{ verif}}
\]

\[
\frac{A \& B \text{ use} \quad \& E_1}{A \text{ use} \quad A \& B \text{ use} \quad \& E_2}{B \text{ use}}
\]

- Bidirectional subformula property
Implication

• Implication requires hypothetical reasoning

\[
\begin{align*}
A & \text{ use} \\
\vdots \\
B & \text{ verif} \\
\hline
A \supset B & \text{ verif} \quad \supset I^x \\
\hline
A \supset B & \text{ use} \quad A & \text{ verif} \\
\vdash B & \text{ use} \\
\end{align*}
\]

• Maintains bidirectional subformula property
Harmony

- **Identity**
  
  \[ A \text{ use} \vdash A \text{ verif} \] for any proposition \( A \)

  - Assume for atomic propositions (= parameters)
  - Ensure for compound propositions

- **(Hereditary) Proof Substitution**
  
  \[ \text{If } \Gamma, A \text{ use} \vdash C \text{ verif} \]
  \[ \text{and } \Gamma \vdash A \text{ verif} \]
  \[ \text{then } \Gamma \vdash C \text{ verif} \]

  - Hereditary substitution to obtain canonical proofs
Relating Atomic Propositions

- Relate (otherwise parametric) atomic propositions
- Based on entailment

\[
\text{man} \leq \text{mortal} \quad \quad P \leq Q \quad (P \vdash Q)
\]

- \( P \leq Q \) is an assumption on parameters \( P \) and \( Q \)
- Use when logical connectives have been eliminated

\[
\frac{P \text{ use } P = Q}{Q \text{ verif}} \quad \rightarrow \quad \frac{P \text{ use } P \leq Q}{Q \text{ verif}}
\]
Rules of Atomic Subtyping

- Reflexivity, by identity: \( P \text{ use } \vdash P \text{ verif} \)

\[
P \leq P
\]

- Transitivity, by substitution from: \( P \text{ use } \vdash Q \text{ verif} \text{ and } Q \text{ use } \vdash R \text{ verif} \)

\[
P \leq Q \quad Q \leq R
\]

\[
P \leq R
\]
Curry-Howard-DeBruijn

- Propositions as types; proofs as terms
- Annotate judgments with proof terms
  - $A \text{ verif}$ becomes $M \iff A$ (check $M$ against $A$)
  - $A \text{ use}$ becomes $R \Rightarrow A$ ($R$ synthesizes $A$)
- Canonical proofs as canonical terms ($\beta$-normal, $\eta$-long)

\[
\begin{align*}
\Gamma, x \Rightarrow A &\vdash M \iff B \\
\Gamma &\vdash \lambda x. M \iff A \rightarrow B \\ \\
\Gamma &\vdash R \Rightarrow P \quad P \leq Q \\
\Gamma &\vdash R \iff Q \\
\Gamma, x \Rightarrow A &\vdash x \Rightarrow A \\
\Gamma &\vdash R \Rightarrow A \rightarrow B \\
\Gamma &\vdash M \iff A \\
\Gamma &\vdash R M \Rightarrow B \\
\end{align*}
\]
Identity Revisited

• Canonical term grammar

  Canonical terms  \( M ::= \lambda x. M \mid R \)

  Atomic terms   \( R ::= x \mid R M \)

• Identity with terms

  \( x \Rightarrow A \vdash \eta_A(x) \leftrightarrow A \) for any proposition \( A \)

• \( \eta_A(R) \) defined by induction on \( A \)

  \[
  \eta_P(R) = R \\
  \eta_{A\rightarrow B}(R) = \lambda x. \eta_B(R \eta_A(x))
  \]

• Definition extends modularly to new connectives
Substitution Revisited

• Hereditary substitution with terms

If $\Gamma \vdash M \leftrightarrow A$ and $\Gamma, x \Rightarrow A \vdash N \leftrightarrow C$
then $\Gamma \vdash [M/x]_A N \leftrightarrow C$

• Define $[M/x]_A N$ by nested induction on $A$ and $N$

$[M/x]_A (\lambda y. N) = \lambda y. [M/x]_A N$

$[M/x]_A (R) = [[M/x]_A R \quad \text{if} \; \text{hd}(R) = x$

$[M/x]_A (R) = [M/x]_A R \quad \text{if} \; \text{hd}(R) \neq x$

$[M/x]_A (R N) = ([M/x]_A R) ([M/x]_A N)$

$[M/x]_A (y) = y$
Hereditary Substitution

- Call $\llbracket M/x \rrbracket_A(R)$ when $\text{hd}(R) = x$
- Returns canonical term and its type $M' : A'$ with $A'$ a constituent of $A$

$$\llbracket M/x \rrbracket_A(x) = M : A$$
$$\llbracket M/x \rrbracket_A(R N) = \llbracket N'/y \rrbracket_B M' : C$$

where $\llbracket M/x \rrbracket_A N = N'$

and $\llbracket M/x \rrbracket_A R = \lambda y. M' : B \rightarrow C$

- Refers back to ordinary substitution with smaller type
Application: Logical Frameworks

- LF Logical Framework
- Based on dependent types $\lambda^\Pi$
  - Higher-order abstract syntax
  - Judgments as types
- Object language and rules are encoded as signatures with constant declarations
- Theory of subtyping presented here extends to dependent types [Lovas & Pf’07]
  - Allow declarations $a \leq b$ for type families $a$ and $b$.
  - A little more later in this talk
Example: Encoding CBV

- **Signature (with exp : type and val : type)**
  
  \[
  \text{lam} : (\text{val} \rightarrow \text{exp}) \rightarrow \text{val} \\
  \text{app} : \text{exp} \rightarrow \text{exp} \rightarrow \text{exp} \\
  \text{val} \leq \text{exp}
  \]

- **Sample derivation**

  \[
  \begin{align*}
  x & \Rightarrow \text{val} \vdash x \Rightarrow \text{val} & \text{val} \leq \text{exp} \\
  x & \Rightarrow \text{val} \vdash x \leftarrow \text{exp} \\
  \text{lam} & \Rightarrow (\text{val} \rightarrow \text{exp}) \rightarrow \text{val} \\
  \lambda x. x & \leftarrow \text{val} \rightarrow \text{exp} \\
  \text{lam} (\lambda x. x) & \leftarrow \text{val}
  \end{align*}
  \]
Example: Encoding Polytypes

• Predicative polymorphism
  
  \text{arrow} : \text{simp}_\text{tp} \rightarrow \text{simp}_\text{tp} \rightarrow \text{simp}_\text{tp}
  
  \text{forall} : (\text{simp}_\text{tp} \rightarrow \text{poly}_\text{tp}) \rightarrow \text{poly}_\text{tp}
  
  \text{simp}_\text{tp} \leq \text{poly}_\text{tp}

• Impredicative polymorphism

  \text{arrow} : \text{simp}_\text{tp} \rightarrow \text{simp}_\text{tp} \rightarrow \text{simp}_\text{tp}
  
  \text{forall} : (\text{poly}_\text{tp} \rightarrow \text{poly}_\text{tp}) \rightarrow \text{poly}_\text{tp}
  
  \text{simp}_\text{tp} \leq \text{poly}_\text{tp}
Summary So Far

- Meaning of connectives given by canonical proofs
- Analyze the structure of terms in two directions
- Bi-directional type-checking under CHdB isomorphism
- Identity and substitution principles
- Subtyping as entailment on atomic propositions
- Applicable to dependent types
Subtyping at Higher Types

• What about the usual co-/contra-variant rule of subtyping?

\[
B_1 \leq A_1 \quad A_2 \leq B_2 \quad \Rightarrow \quad A_1 \rightarrow A_2 \leq B_1 \rightarrow B_2
\]

• No need for such a rule
  • The logical meaning of $\rightarrow$ is already given
  • The logical meaning of $\leq$ is already given

• Canonical forms (specifically: $\eta$-long forms) are crucial
We can define a notion of subtyping several ways:

1. $A \leq_1 B$ if for all $\Gamma \vdash M \leftarrow A$ we have $\Gamma \vdash M \leftarrow B$

2. $A \leq_2 B$ if for all $\Gamma, x \Rightarrow B \vdash N \leftarrow C$
   we have $\Gamma, x \Rightarrow A \vdash N \leftarrow C$

3. $A \leq_3 B$ if $x \Rightarrow A \vdash \eta_A(x) \leftarrow B$

4. $A \leq_4 B$ if for all $\Gamma, x \Rightarrow B \vdash N \leftarrow C$ and $\Gamma \vdash M \leftarrow A$
   we have $\Gamma \vdash [M/x]_B N \leftarrow C$

These are all equivalent!

Only compare types of the same shape, since type shape determines term shape (cf. “refinement restriction”)

Rules for Higher-Order Subtyping

• Rules so far for atomic types

\[
\begin{align*}
P & \leq P \\
P \leq Q & \quad Q \leq R \\
then \\
P & \leq R
\end{align*}
\]

• Also equivalent to (1)–(4) is adding the following rule

\[
\begin{align*}
B_1 & \leq A_1 \\
A_2 & \leq B_2 \\
A_1 \rightarrow A_2 & \leq B_1 \rightarrow B_2
\end{align*}
\]

• General reflexivity and transitivity principles hold
• Mirror identity and substitution for entailment
Subtyping Completeness

• The rules are complete in a strong sense:

\[ A \leq B \text{ iff for all } \Gamma \vdash M \iff A \text{ we have } \Gamma \vdash M \iff B! \]

• Possible due to the *open-ended* interpretation of subtyping
  • Quantification over \( \Gamma \)
  • Subtypings are stable under all extensions
  • Consistent with open-ended nature of LF

• Contrast with functional programming
  • Interested in *closed* terms for evaluation
  • Datatypes are inductive or recursive, not open-ended
  • Ironically, function spaces are open-ended
  • Cannot expect completeness
Subtyping Alone is Insufficient

• ... in practice
• Consider types \texttt{even} and \texttt{odd}
• What is the type of successor?
• Need \textit{intersection types}
  \begin{align*}
  z & : \texttt{even} \\
  s & : (\texttt{even} \rightarrow \texttt{odd}) \land (\texttt{odd} \rightarrow \texttt{even})
  \end{align*}
• Expresses \textit{multiple} properties of \textit{one} term in a single type
Meaning Explanations

• Resume verificationist ($\land I$) and pragmatist ($\land E_i$) programs

$$\Gamma \vdash M \leftarrow A \quad \Gamma \vdash M \leftarrow B \quad \Gamma \vdash M \leftarrow A \land B \quad \land I$$

$$\Gamma \vdash R \Rightarrow A \land B \quad \land E_1 \quad \Gamma \vdash R \Rightarrow A \land B \quad \land E_2$$

• Type-theoretic, but not purely “logical”
• Again, no new subtyping rules, nothing else needed!
• Only rule concerned with subtyping remains the same

$$\Gamma \vdash R \Rightarrow P \quad P \leq Q \quad \Gamma \vdash R \leftarrow Q$$
Example: 2 is even

- Recall
  \[ z : \text{even} \]
  \[ s : (\text{even} \rightarrow \text{odd}) \land (\text{odd} \rightarrow \text{even}) \]
- To check \( s(sz) \iff \text{even} \) we use
  - \( s \Rightarrow \text{odd} \rightarrow \text{even} \) for the outer occurrence
  - \( s \Rightarrow \text{even} \rightarrow \text{odd} \) for the inner occurrence
  - \( z \Rightarrow \text{even} \) for the occurrence of \( z \)
Example: Open-Endedness

- Add type `empty` with no constructor
- We do not know that `empty ≤ even`:
  - `x ⇒ empty ⊢ x ⇐ empty` but `x ⇒ empty ⊬ x ⇐ even` unless we specify `empty ≤ even`.
- We do not know that `even ∧ odd ≤ empty` for a similar reason.
- Currently there is no way to specify this [future work]
Characterizing HO Subtyping

• The characterizations of subtyping from before are still equivalent. For example

\( A \leq_1 B \) if for all \( \Gamma \vdash M \Leftarrow A \) we have \( \Gamma \vdash M \Leftarrow B \)

\( A \leq_3 B \) if \( x \Rightarrow A \vdash \eta_A(x) \Leftarrow B \)

• Several sound and complete set of rules for subtyping
  • Axiomatic
  • Sequent calculus
Axiomatic Formulation I

- Reflexivity and transitivity

\[ A \leq A \quad A \leq B \quad B \leq C \quad \Rightarrow \quad A \leq C \]

- Functions

\[ B_1 \leq A_1 \quad A_2 \leq B_2 \quad \Rightarrow \quad A_1 \rightarrow A_2 \leq B_1 \rightarrow B_2 \]
Axiomatic Formulation II

• Intersections

\[
\begin{align*}
A \leq B & \quad A \leq C \\
A \leq B \wedge C \\
\hline
A \wedge B \leq C & \quad B \leq C \\
A \wedge B \leq C
\end{align*}
\]

• Distributivity

\[
(A \to B) \wedge (A \to C') \leq A \to (B \wedge C')
\]
Sequent Calculus for Subtyping

- Use sequent calculus to show decidability
- Not necessary in an implementation!
  - Only to understand subtyping
  - Only atomic subtyping is required for type-checking
- Judgment $\Delta \leq C$

\[
\frac{\Delta \leq A \quad \Delta \leq B}{\Delta \leq A \land B} \quad \Delta, A \land B \leq C
\]

\[
\frac{\Delta, P \leq Q}{\Delta, [A_i \rightarrow B_i]_i \leq A \rightarrow B}
\]

\[
\frac{\Delta, A, B \leq C}{\Delta, A \land B \leq C} \quad \Delta, P \leq Q
\]

\[
\frac{[A \leq A_i]_i \quad [B_i]_i \leq B}{\Delta, [A_i \rightarrow B_i]_i \leq A \rightarrow B}
\]
Algorithmic Typing

• Typing rules are non-deterministic

\[
\frac{\Gamma \vdash R \Rightarrow A \land B}{\Gamma \vdash R \Rightarrow A}
\]

\[
\frac{\Gamma \vdash R \Rightarrow A \land B}{\Gamma \vdash R \Rightarrow B}
\]

• A “more efficient” system
  • \(\Gamma \vdash R \Rightarrow \Delta\) \((R\ has\ all\ types\ in\ \Delta)\)
  • \(\Gamma \vdash M \Leftarrow A\) \((M\ checks\ against\ A)\)

• Rules should maximally break down intersection
• Elide this refinement
Algorithmic Typing Rules

\[ \Gamma, x \Rightarrow A \vdash M \Leftarrow B \quad \Gamma \vdash R \Rightarrow \Delta, P \quad P \leq Q \]

\[ \Gamma \vdash \lambda x. M \Leftarrow A \rightarrow B \quad \Gamma \vdash R \Leftarrow Q \]

\[ \Gamma \vdash M \Leftarrow A \quad \Gamma \vdash M \Leftarrow B \]

\[ \Gamma \vdash M \Leftarrow A \land B \]

\[ \Gamma, x \Rightarrow A \vdash x \Rightarrow A \quad \Gamma \vdash R \Rightarrow \Delta, [A_i \rightarrow B_i]_i \quad [\Gamma \vdash M \Leftarrow A_i]_i \]

\[ \Gamma \vdash R M \Rightarrow [B_i]_i \]

\[ \Gamma \vdash R \Rightarrow \Delta, A, B \]

\[ \Gamma \vdash R \Rightarrow \Delta, A \land B \]
Summary

- Logical meaning explanations via canonical proofs
- Simple and uniform system of subtyping and intersections
- Subtyping defined on atomic types only
- Intersection defined for synthesis and checking only
- Clear derived notions of subtyping for higher-order types
- Sound and complete set of rules
- Compare and intersect only types with compatible shape
  - Since type shape determines shape of canonical terms
- Applies to dependent types with morally identical rules
Other Encoding Examples

• Barendregt’s $\lambda$-cube
  • Uniform presentation of typing at all levels
  • Different corners of cube with different intersections

• Syntactic inclusions and properties
  • Values, expressions, and evaluation in functional languages
  • Hereditary Harrop formulas and logic programming

• Dependent properties of proofs
  • (Weak) (head) normal forms
  • Cut-free sequent proofs
  • Uniform sequent and natural deduction proofs
Can we use this approach to design type systems for functional languages?

Logical Framework (LF)
- Complete rules for subtyping and intersection
- *Open* interpretation of atomic types
- *Closed* interpretation of function spaces

Functional programming
- *Closed* interpretation of atomic types
- *Open* interpretation of function spaces
- Canonical forms no longer fully characterize meaning
- Operational semantics, non-termination, effects
A Bridge

- Criterion (3) offers a bridge between logical frameworks and functional programming

\[(3) \quad A \leq_3 B \text{ if } x \Rightarrow A \vdash \eta_A(x) \iff B\]

- Does not quantify over arbitrary terms or contexts
- $\eta$-expansions are available in functional language
- $\eta$-expansions are always canonical identity maps
- $\eta$-expansions depend only on type constructs in $A$ and $B$
- Stability under language extensions
A Problem

- Some rules are *unsound* because they rely on the closed interpretation of function spaces (canonical forms, pure)
  - Intersection introduction requires a value restriction
  - Drop distributivity of $\land$ over $\rightarrow$
  - For counterexamples see [Davies & Pf. ICFP’00]
- Idea: use *monads* to isolate effects!
- Understand the effect of effects on subtyping and intersections
Monads in Judgmental Form

- Maintain verificationist and pragmatist approach
- Meaning explanation by introduction and elimination forms
- Need new logical judgment
  - $\Gamma \vdash A \text{ lax}$ (logically: lax truth of $A$)
- Type-theoretic version (with proof terms)
  - $\Gamma \vdash E \leftarrow A$
    - $E$ is a potentially effectful computation of type $A$
  - Do not consider specific effects
Judgmental Rules

- Relating lax truth to truth
- A pure term $M$ is a computation of type $A$

$$\Gamma \vdash M \iff A$$

$$\Gamma \vdash M \iff A$$

- Substitution: we can compose computations $F$ before $E$

If $\Gamma \vdash F \iff A$

and $\Gamma, x \Rightarrow A \vdash E \iff C$

then $\Gamma \vdash \langle F/x \rangle_A E \iff C$
The Monad Type Constructor

• Type \{A\} for pure terms denoting computations of type \(A\)

• Verificationist definition

\[
\begin{align*}
\Gamma \vdash E \leftarrow A \\
\Gamma \vdash \{E\} \leftarrow \{A\} \\
\{\}\Gamma I
\end{align*}
\]

• Pragmatist definition

\[
\begin{align*}
\Gamma \vdash R \Rightarrow \{A\} \\
\Gamma, x \Rightarrow A \vdash E \leftarrow C \\
\Gamma \vdash \text{let} \{x\} = R \text{ in } E \leftarrow C \\
\{\}\Gamma E
\end{align*}
\]
Identity and Substitution Revisited

- **Leftist** hereditary substitution $\langle E/x \rangle_A F$
- Defined by nested induction on $A$, $E$, and $F$
  \[
  \langle \text{let } \{ y \} = R \text{ in } E/x \rangle_A F = \text{let } \{ y \} = R \text{ in } \langle E/x \rangle_A F
  \]
  \[
  \langle M/x \rangle_A E = [M/x]_A E
  \]
- Identity principle via $\eta$-expansion
  \[
  \eta_{\{A\}}(R) = \{ \text{let } \{ x \} = R \text{ in } \eta_A(x) \}
  \]
Subtyping and Intersections

• Rules for subtyping and intersection are not affected!
  • They were concerned with truth
  • Lax truth is a derived notion
• Orthogonality of language constructs pays off!
• It is *not* the case that

\[
\Gamma \vdash E \leftarrow A \quad \Gamma \vdash E \leftarrow B \\
\Gamma \vdash E \leftarrow A \land B \quad \text{wrong!}
\]
Higher-Order Subtyping

- Derived notions of subtyping remain unchanged
- New rule, axiomatically

\[ A \leq B \]
\[ \{ A \} \leq \{ B \} \]
No Distributivity

- No distributivity: \( \{A\} \land \{B\} \nless \{A \land B\} \)
- Quick check via \(\eta\)-expansion

\[
\begin{align*}
  x \Rightarrow \{P\} \land \{Q\} & \vdash x \Rightarrow \{P\} \land \{Q\} \\
  x \Rightarrow \{P\} \land \{Q\} & \vdash x \Rightarrow \{P\} \\
  x \Rightarrow \{P\} \land \{Q\} & \vdash \text{let } \{y\} = x \text{ in } y \Leftarrow \{P \land Q\}
\end{align*}
\quad \text{fails}
\]

\[
\begin{align*}
  y \Rightarrow P & \vdash y \Leftarrow P \land Q \\
  y \Rightarrow P & \vdash y \Leftarrow P \land Q \\
  y \Rightarrow P & \vdash \text{let } \{y\} = x \text{ in } y \Leftarrow \{P \land Q\}
\end{align*}
\]
Subtyping in Sequent Form

• In sequent form: commit

\[ \frac{A \leq B}{\Delta, \{A\} \leq \{B\}} \]

• Contrast with functions

\[ \frac{[A \leq A_i]_i \quad [B_i]_i \leq B}{\Delta, [A_i \rightarrow B_i]_i \leq A \rightarrow B} \]

• Take all \( i \) such that \( A \leq A_i \)
The Value Restriction

• Call-by-value with effects requires some modifications
  • Value restriction on intersection introduction

\[ \frac{\Gamma \vdash v : \sigma \quad \Gamma \vdash v : \tau}{\Gamma \vdash v : \sigma \land \tau} \]

• No distributivity
• Subtyping at higher order (terms are not \( \eta \)-long)
  • Easy: we did all the work already!
• Typing annotations (terms are not \( \beta \)-normal)
  • Not entirely easy [Dunfield & Pf’03] [Dunfield’07]
Deriving Restricted Rules

- Derive rules from standard encoding into monadic form

  Types \( \tau ::= b \mid \tau_1 \to \tau_2 \)

  Computations \( e ::= e_1 e_2 \mid v \)

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

- Type translation

  \( |b| = b \)

  \( |\tau_1 \to \tau_2| = |\tau_1| \to \{ |\tau_2| \} \)

  \( |\tau_1 \land \tau_2| = |\tau_1| \land |\tau_2| \)

- Form of term translation

  \( |e| = E \) computations as monadic expressions

  \( |v| = M \) values as (pure) terms
Value Restriction

- Properties of term translation (elided)
  - If $e : \tau$ then $|e| \leftarrow |\tau|$
  - If $v : \tau$ then $|v| \leftarrow |\tau|$
- Value restriction since $\wedge$ does not distribute over $\{ \}$.

\[
\Gamma \vdash \{ E \} \iff \{ A \} \quad \Gamma \vdash \{ E \} \iff \{ B \}
\]

\[
\Gamma \vdash \{ E \} \iff \{ A \wedge B \}
\]

wrong!

- Also recall: no rule to split $E \leftarrow A \wedge B$
Distributivity

- Distributivity cannot be derived because

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

but

\[(A \rightarrow \{B\}) \land (A \rightarrow \{C\}) \not\leq A \rightarrow \{B \land C\}\]

- So

\[(\tau \rightarrow \sigma) \land (\tau \rightarrow \rho) \not\leq \tau \rightarrow (\sigma \land \rho)\]
Other Considerations

- System has empty conjunction \( \top \) (elided for this talk)
- Interpretation into type theory with proof irrelevance [mostly done]
  - Roughly: \( M \leftarrow A \) becomes \( \langle M, N \rangle \leftarrow \Sigma x:\tilde{A}. \hat{A}(x) \) where
    - \( \tilde{A} \) is a type representing the shape of \( A \)
    - \( \hat{A}(x) \) is a type family on elements of type \( \tilde{A} \)
    - \( N \leftarrow \tilde{A}(M) \) is a proof that \( M \) has property \( \tilde{A} \)
    - \( \langle M_1, N_1 \rangle = \langle M_2, N_2 \rangle \) iff \( M_1 = M_2 \) (by definition)
- Complexity due to definitional equality
- Coercion interpretation? [future work]
And More

- Union types are complex, but fit the story [Zeilberger’07] [Dunfield’07] [Dunfield & Pf’03]
  - Co-value restriction on union types
- Parametric polymorphism is harder [Davies’05] [Dunfield]
- Deeper analysis of call-by-name and call-by-value
  - Judgmental method and focusing [Zeilberger POPL’08]
  - Positive and negative intersections, unions [Zeilberger’07]
  - Addings products, sums, and negations
Summary: Origins

- Atomic propositions as parameters
- Atomic subtyping as assumptions
- Verificationist and pragmatist definitions of
  - logical connectives
  - canonical proofs
  - intersections
  - monads
Summary: Destinations

- Exceedingly simple rules with subtyping and intersections
- Derived notion of higher-order subtyping
  - from substitution or $\eta$-expansion
  - with sound and complete sets of structural rules
- Richly expressive logical framework $\lambda^{\Pi \leq ^{\Lambda}}$
- Explanation of value restriction on intersections in call-by-value via monadic embedding
All (Non-Derived) Rules

\[
\begin{align*}
\Gamma, x \Rightarrow A & \vdash M \Leftarrow B \\
\Gamma & \vdash \lambda x. M \Leftarrow A \rightarrow B \quad \rightarrow I \\
\Gamma & \vdash R \Rightarrow P \quad P \leq Q \\
\Gamma & \vdash R \Leftarrow Q \\
\Gamma & \vdash M \Leftarrow A \\
\Gamma & \vdash M \Leftarrow B \\
\Gamma & \vdash M \Leftarrow A \land B \quad \land I \\
\Gamma, x \Rightarrow A & \vdash x \Rightarrow A \\
\Gamma & \vdash R \Rightarrow A \rightarrow B \\
\Gamma & \vdash R M \Rightarrow B \quad \rightarrow E \\
\Gamma & \vdash R \Rightarrow A \land B \\
\Gamma & \vdash R \Rightarrow A \quad \land E_1 \\
\Gamma & \vdash R \Rightarrow B \\
\Gamma & \vdash R \Rightarrow B \quad \land E_2 \\
\Gamma & \vdash R \Rightarrow A \land B \\
\Gamma & \vdash R \Rightarrow A \land B \quad \land F \\
P \leq P \\
P \leq Q \\
Q \leq S \\
P \leq S
\end{align*}
\]