A History of Subtyping

Benjamin C. Pierce
University of Pennsylvania

PLMW, August 2023, Seattle
What Does Subtyping Mean?

Benjamin C. Pierce
University of Pennsylvania

PLMW, August 2023, Seattle
• Grew up in Redlands, CA
  • About halfway between Los Angeles and Palm Springs

• PhD from CMU
  • Advised by Bob Harper and John Reynolds

• Postdocs at Edinburgh, INRIA, Cambridge
  • With Robin Milner, Didier Rémy

• Taught at Indiana University for two years

• At Penn since 1998
I like...

writing

music

contact improv

Kids :-)
What about you?

Ask the people on both sides of you...

• Name?
• Hometown?
• Favorite kind of music?
• Favorite language with subtyping?
What this talk is about

• Some basic stuff about typing and subtyping
  • (that may be familiar)

• Some other basic stuff
  • (that may be less familiar)

• Some history

• Some people
Please interrupt me!
What Does Subtyping Mean?
What Does Subtyping Mean?
What Does Typing Mean?
What Does Typing Mean?

Let’s return to the source...
The Lambda Calculus
Syntax

\[ e ::= c \mid x \mid \lambda x.e_1 \mid e_1 e_2 \]

expressions

constants

variables

function abstractions

function applications
Operational Semantics

\[(\lambda x. \ e_1) \ e_2 \rightarrow [e_2/x]e_1 \quad (\text{R-BETA})\]

\[\frac{e_1 \rightarrow e'_1}{e_1 \ e_2 \rightarrow e'_1 \ e_2} \quad (\text{R-APP1})\]

\[\frac{e_2 \rightarrow e'_2}{e_1 \ e_2 \rightarrow e_1 \ e'_2} \quad (\text{R-APP2})\]

\[\frac{e_1 \rightarrow e'_1}{\lambda x. \ e_1 \rightarrow \lambda x. \ e'_1} \quad (\text{R-ABS})\]
Example

\[(\lambda x. \lambda y. x) \ s \ t \ \rightarrow \ (\lambda y. \ s) \ t \ \rightarrow \ s\]

Formally...

\[
\frac{(\lambda x. \lambda y. x) \ s \ \rightarrow \ \lambda y. s \ \ (R-\text{BETA})}{(\lambda x. \lambda y. x) \ s \ t \ \rightarrow \ (\lambda y. s) \ t \ \ (R-\text{APP1})}
\]
Aside: Reduction Strategies

Most programming languages restrict this “full beta-reduction” to a deterministic function.

• Call by name
• Call by value
• Lazy
• Etc.

These distinctions are not needed for this talk.
Known for …

- the lambda calculus
- the Church–Turing thesis
  - … that every effectively calculable function is a computable function
- the undecidability of first-order logic
- (and much more!)

“With his doctoral student Alan Turing, Church is considered one of the founders of computer science.” [Wikipedia]
From the Mathematics Genealogy project...

<table>
<thead>
<tr>
<th>Name</th>
<th>School</th>
<th>Year</th>
<th>Descendants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hensel, Gustav</td>
<td>Princeton University</td>
<td>1963</td>
<td></td>
</tr>
<tr>
<td>Kemeny, John</td>
<td>Princeton University</td>
<td>1949</td>
<td>84</td>
</tr>
<tr>
<td>Kleene, Stephen</td>
<td>Princeton University</td>
<td>1934</td>
<td>1465</td>
</tr>
<tr>
<td>Kochen, Simon</td>
<td>Princeton University</td>
<td>1959</td>
<td>38</td>
</tr>
<tr>
<td>L'Abbé, Maurice</td>
<td>Princeton University</td>
<td>1951</td>
<td></td>
</tr>
<tr>
<td>Malitz, Isaac (Richard)</td>
<td>University of California, Los Angeles</td>
<td>1976</td>
<td></td>
</tr>
<tr>
<td>Mar, Gary</td>
<td>University of California, Los Angeles</td>
<td>1985</td>
<td>1</td>
</tr>
<tr>
<td>Massey, Gerald</td>
<td>Princeton University</td>
<td>1964</td>
<td>230</td>
</tr>
<tr>
<td>Rabin, Michael</td>
<td>Princeton University</td>
<td>1957</td>
<td>459</td>
</tr>
<tr>
<td>Rescher, Nicholas</td>
<td>Princeton University</td>
<td>1951</td>
<td>70</td>
</tr>
<tr>
<td>Richter, Wayne</td>
<td>Princeton University</td>
<td>1963</td>
<td>203</td>
</tr>
<tr>
<td>Ritchie, Robert</td>
<td>Princeton University</td>
<td>1960</td>
<td>51</td>
</tr>
<tr>
<td>Robbin, Joel</td>
<td>Princeton University</td>
<td>1965</td>
<td>11</td>
</tr>
<tr>
<td>Rogers, Jr., Hartley</td>
<td>Princeton University</td>
<td>1952</td>
<td>843</td>
</tr>
<tr>
<td>Rosser, J. Barkley</td>
<td>Princeton University</td>
<td>1934</td>
<td>1168</td>
</tr>
<tr>
<td>Routley, Richard</td>
<td>Princeton University</td>
<td>1981</td>
<td>1</td>
</tr>
<tr>
<td>Scott, Dana</td>
<td>Princeton University</td>
<td>1958</td>
<td>623</td>
</tr>
<tr>
<td>Shapiro, Norman</td>
<td>Princeton University</td>
<td>1955</td>
<td></td>
</tr>
<tr>
<td>Smullyan, Raymond</td>
<td>Princeton University</td>
<td>1959</td>
<td>12</td>
</tr>
<tr>
<td>Turing, Alan</td>
<td>Princeton University</td>
<td>1938</td>
<td>230</td>
</tr>
<tr>
<td>Winder, Robert</td>
<td>Princeton University</td>
<td>1962</td>
<td></td>
</tr>
</tbody>
</table>

According to our current on-line database, Alonzo Church has 36 students and 6307 descendants.

We welcome any additional information.
<table>
<thead>
<tr>
<th>Name</th>
<th>School</th>
<th>Year</th>
<th>Degree</th>
<th>Advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alonzo Church</td>
<td>Princeton University</td>
<td>1927</td>
<td>Ph.D.</td>
<td>Oswald Veblen</td>
</tr>
</tbody>
</table>

From the Mathematics Genealogy project...

<table>
<thead>
<tr>
<th>Name</th>
<th>School</th>
<th>Year</th>
<th>Degree</th>
<th>Advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hensel, Gustav</td>
<td>Princeton University</td>
<td>1963</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kemény, John</td>
<td>Princeton University</td>
<td>1949</td>
<td>84</td>
<td></td>
</tr>
</tbody>
</table>

Click [here](#) to see the students listed in chronological order.

Rabin, Michael

Routley, Richard

Scott, Dana

Shapiro, Norman

Smullyan, Raymond

Turing, Alan

Winder, Robert

According to our current on-line database, Alonzo Church has 36 [students](#) and 6307 [descendants](#). We welcome any additional information.
Adding Records

\[ e ::= \ldots | \{ l_1 = e_1, \ldots, l_n = e_n \} | e.l \]

**Construction**

\[ e_i \rightarrow e_i' \]

\[ \{ \ldots, l_i = e_i, \ldots \} \rightarrow \{ \ldots, l_i = e_i', \ldots \} \quad (R-RCD) \]

**Projection**

\[ \{ \ldots, l_i = e_i, \ldots \}.l_i \rightarrow e_i \quad (R-RCD\text{PROJ}) \]
Examples

\[ (\lambda x: \text{String.} \ x) \{n='Bob'\} \]
\[ (\lambda x: \text{Student.} \ . \ x.n) \{n='Alice', \ g=3.8\} \]
\[ (\lambda y: \text{Person.} \ . \ {n='Alice', \ g=3.8}) \{n='Bob'\} \]
\[ (\lambda y: \text{Person.} \ . \ {n='Alice', \ g=3.8}) \{n='Bob'\} \]

\[ (\lambda x: \mathbb{R}. \ x)(\lambda x: \mathbb{R}. \ x)(\lambda x: \mathbb{R}. \ x) \]
**Examples**

- **Constants**
  - $\lambda x: \text{String. } x$
  - $\lambda x: \mathbb{R}. x$
  - $\lambda x. x x$
  - $(\lambda x. x x) (\lambda x. x x)$
  - $3.8$
  - $0.0$

- **Records**
  - $(\lambda x: \text{Student. } x.n) \{n='\text{Alice}', g=3.8\}$
  - $(\lambda x: \text{Student. } \lambda y: \text{Person. } x) \{n='\text{Bob}'\} \{n='\text{Alice}', g=3.8\}$
  - $(\lambda y: \text{Person. } \{n='\text{Alice}', g=3.8\}) \{n='\text{Bob}'\}$
  - $\{n='\text{Alice}', g=3.8\}$

- **Functions**
  - 'Alice'
  - 'Bob'
  - 'Alice' 3.8

**Compound Expressions**
The Simply Typed Lambda-Calculus
Syntax

$$T ::= B \mid T_1 \to T_2 \mid \{l_1:T_1, \ldots , l_n:T_n\}$$

$$e ::= c \mid x \mid \lambda x:T. e_1 \mid e_1 \; e_2 \mid \{l_1=e_1, \ldots , l_n=e_n\} \mid e.l$$

$$\Gamma ::= \cdot \mid \Gamma_1, \; x:T$$

**base types**  **function types**  **record types**

**empty context**  **extended context**
Example

Person := \{n: String\}
Student := \{n: String, g: Real\}

bob := \{n='Bob'\} : Person
alice := \{n='Alice', g=3.8\} : Student
Typing

\[
\begin{align*}
\Gamma \vdash x : T & \quad \text{(T-VAR)} \\
\Gamma \vdash x \in T & \\
\Gamma, \, x : T_1 \vdash e_1 \in T_2 & \quad \text{(T-ABS)} \\
\Gamma \vdash \lambda x : T_1. \, e_1 \in T_1 \rightarrow T_2 & \\
\Gamma \vdash e_1 \in T_1 \rightarrow T_2 & \quad \Gamma \vdash e_2 \in T_1 & \quad \text{(T-APP)} \\
\Gamma \vdash e_1 \, e_2 \in T_2 & \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash e_1 \in T_1 & \quad \ldots & \quad \Gamma \vdash e_n \in T_n & \quad \text{(T-RCD)} \\
\Gamma \vdash \{l_1 = e_1, \ldots, l_n = e_n\} \in \{l_1 : T_1, \ldots, l_n : T_n\} & \\
\Gamma \vdash e \in \{l_1 : T_1, \ldots, l_n : T_n\} & \quad \text{(T-PROJ)} \\
\Gamma \vdash e.\,l_i \in T_i & \\
\text{typeofConst}(c) = C & \quad \text{(T-CONST)} \\
\Gamma \vdash c \in C &
\end{align*}
\]
Typing Derivations

\[
\begin{align*}
\frac{x: S \in (x: S, y: T)}{(T-VAR)} & \quad \text{subderivation for } S \\
\frac{x: S, y: T \vdash x \in S} {(T-ABS)} & \quad \text{subderivation for } T \\
\frac{x: S \vdash \lambda y: T. x \in T \rightarrow S} {(T-ABS)} & \quad \Delta_1 \\
\Delta_2 & \quad \vdash s \in S \\
\cdot \vdash \lambda x: S. \lambda y: T. x \in S \rightarrow T \rightarrow S & \quad (T-App) \\
\cdot \vdash (\lambda x: S. \lambda y: T. x) s \in T \rightarrow S & \quad (T-App) \\
\cdot \vdash (\lambda x: S. \lambda y: T. x) s \ t \in S & \quad (T-App)
\end{align*}
\]
Correctness

Theorem (Preservation):

If $\Gamma \vdash e \in U$ and $e \longrightarrow e'$, then $\Gamma \vdash e' \in U$.

In particular:

If $\Gamma \vdash (\lambda x: T. e_1) e_2 \in U$, then $\Gamma \vdash [e_2/x]e_1 \in U$. 
What does typing “mean”? 
Haskell B. Curry

Known (in PL) especially for:

• the **Curry-Howard Correspondence** between the fundamental structures found in logic and in computation

• And, of course, the “currying” operation on multi-argument functions

\[ S \times T \to U \sim S \to T \to U \]

Also: How many people have three PLs named after them??
“Church style” vs. “Curry style”

“There are two versions of type assignment in the λ-calculus:

• **Church-style**, in which the type of each variable is fixed, and
• **Curry-style** (also called “domain free”), in which it is not.

As an example, in Church-style typing, \( \lambda x : A. x \) is the identity function on type A, and it has type \( A \rightarrow A \) but not \( B \rightarrow B \) for a type B different from A.

In Curry-style typing, \( \lambda x. x \) is a general identity function with type \( C \rightarrow C \) for every type C.”

Bridging Curry and Church’s typing style, Kamareddin et al, 2016
But the distinction goes deeper...

I.e., this is not “just a matter of type inference”
“Extrinsic” View
“Extrinsic” View

Terms come first

\[
\lambda x: \text{String. } x \\
\lambda x: \mathbb{R}. x \\
(\lambda x: \text{Student. } x. n) \{n='\text{Alice}', \ g=3.8\} \quad '\text{Alice}' \quad '\text{Bob}' \\
(\lambda x: \text{Student. } \lambda y: \text{Person. } x) \{n='\text{Bob'}\} \{n='\text{Alice}', \ g=3.8\} \\
(\lambda y: \text{Person. } \{n='\text{Alice}', \ g=3.8\}) \{n='\text{Bob'}\} \\
\{n='\text{Alice}', \ g=3.8\}
\]
"Extrinsic" View

Terms come first

Then reduction

\[ \lambda x: \text{String}. \; x \]

3.8  0.0

\{n='Bob'\}

\( (\lambda x: \text{Student.} \; x. \; n) \{n='Alice', \; g=3.8\} \) "Alice"  'Bob'

\( (\lambda x: \text{Student.} \lambda y: \text{Person.} \; x) \{n='Bob'\} \) \{n='Alice', \; g=3.8\}

\( (\lambda y: \text{Person.} \; \{n='Alice', \; g=3.8\}) \{n='Bob'\} \) \{n='Alice', \; g=3.8\}
“Extrinsic” View

Terms come first
Then reduction
Then types

\[(\lambda x: \text{Student}. \ x.n) \{n='\text{Alice}', \ g=3.8\}\]

\[\{n='\text{Bob}', \ g=3.8\}\]

\[(\lambda y: \text{Person}. \ {n='\text{Alice}', \ g=3.8}) \{n='\text{Bob}'\}\]

\[\{n='\text{Alice}', \ g=3.8\}\]
“Intrinsic” View
“Intrinsic” View

Types come first
“Intrinsic” View

Types come first
Then (typed) terms

- \( \vdash 3.8 \in \mathbb{R} \)
- \( \vdash 0.0 \in \mathbb{R} \)
- \( \lambda x : \text{String}. \ x \)
- \( \lambda x : \mathbb{R}. \ x \)
- \( \vdash \{n = 'Bob'\} \in \text{Person} \)
- \( \vdash \{n = 'Alice', g = 3.8\} \in \text{String} \)
- \( \vdash \{n = 'Alice', g = 3.8\} \in \text{Student} \)

\( \vdash \{n = 'Bob', \{n = 'Alice', g = 3.8\}\} \in \text{Student} \)

\( \vdash \{n = 'Bob'\} \in \text{Student} \)

\( \vdash \{n = 'Bob', \{n = 'Alice', g = 3.8\}\} \in \text{Student} \)
“Intrinsic” View

Types come first
Then (typed) terms
Then reduction

\[ \vdash 3.8 \in \mathbb{R} \]
\[ \vdash 0.0 \in \mathbb{R} \]
\[ \lambda x : \text{String}. \ x \]
\[ \lambda x : \mathbb{R}. \ x \]
\[ \vdash \{n='Bob'\} \in \text{Person} \]
\[ \vdash \{n='Alice', g=3.8\} \in \text{String} \]
\[ \vdash \{n='Alice', g=3.8\} \in \text{Student} \]
\[ \vdash \{n='Bob'\} \in \text{Person} \]
\[ \vdash \{n='Bob'\} \in \text{Student} \]
\[ \vdash \{n='Alice', g=3.8\} \in \text{Student} \]
“Intrinsic” View

Types come first
Then (typed) terms
Then reduction

\[
\begin{align*}
  x &: \text{Student} \in (x: \text{Student}, y: \text{Person}) \quad (\text{T-VAR}) \\
  x &: \text{Student}, y &: \text{Person} \vdash x \in \text{Student} \quad (\text{T-Abs}) \\
  x &: \text{Student} \vdash \lambda y: \text{Person}. x \in \text{Person} \to \text{Student} \quad (\text{T-Abs}) \\
  \vdash \lambda x: \text{Student}. \lambda y: \text{Person}. x \in \text{Student} \to \text{Person} \to \text{Student} \quad (\text{T-Abs}) \\
  \vdash \{n='\text{Alice}', g=3.8\} \in \text{Student} \quad (\text{T-App}) \\
  \vdash \{n='\text{Bob}'\} \in \text{Person} \quad (\text{T-App}) \\
  \vdash (\lambda x: \text{Student}. \lambda y: \text{Person}. x) \{n='\text{Bob}'\} \{n='\text{Alice}', g=3.8\} \in \text{Student} \\
  \vdash (\lambda y: \text{Person}. \{n='\text{Alice}', g=3.8\}) \{n='\text{Bob}'\} \in \text{Student} \\
  \vdash \{n='\text{Alice}', g=3.8\} \in \text{Student}
\end{align*}
\]
Types come first
Then (typed) terms
Then reduction
Types come first
Then **typing derivations**
Then reduction
Reduction on typing derivations??
Sure!
Reduction on Derivations

\[ \Delta_1 \]
\[ \Gamma, x: T \vdash e_1 \in U \quad \text{(T-Abs)} \]
\[ \Gamma \vdash \lambda x: T. e_1 \in T \rightarrow U \]
\[ \Gamma \vdash (\lambda x: T. e_1) e_2 \in U \]
\[ \Delta_2 \]
\[ \Gamma \vdash e_2 \in T \quad \text{(T-App)} \]
\[ \Gamma \vdash e_2 \in T \]
Reduction on Derivations

\[
\begin{align*}
\Delta_1 &\quad \Gamma, x: T \vdash e_1 \in U & \Delta_2 &\quad \Gamma \vdash e_2 \in T \\
\Gamma \vdash \lambda x: T. e_1 \in T \rightarrow U &\quad \Gamma \vdash [\Delta_2/x]\Delta_1 & \Gamma \vdash e_2 \in T &\quad \Gamma \vdash [e_2/x]e_1 \in U \\
&\quad \Gamma \vdash (\lambda x: T. e_1) e_2 \in U & &\quad \text{(T-App)} & \text{(T-Abs)}
\end{align*}
\]

i.e., in \( \Delta_1 \) replace every leaf where the T-Var rule is used to look up \( x \) with a copy of \( \Delta_2 \).
For example...

\[
\begin{align*}
\frac{x : S \in (x : S, y : T)} {x : S, y : T \vdash x \in S} & \quad \text{(T-Var)} \\
\frac{x : S \vdash \lambda y : T. x \in T \to S} {x : S \vdash \lambda y : T. x \in T \to S} & \quad \text{(T-Abs)} \\
\Delta_1 & \quad \Delta_1 \\
\frac{\vdash \lambda x : S. \lambda y : T. x \in S \to T \to S} {\vdash \lambda x : S. \lambda y : T. x \in S \to T \to S} & \quad \Delta_2 \\
\frac{\vdash s \in S} {\vdash s \in S} & \quad \Delta_2 \\
\frac{\vdash (\lambda x : S. \lambda y : T. x) s \in T \to S} {\vdash (\lambda x : S. \lambda y : T. x) s \in T \to S} & \quad \Delta_2 \\
\frac{\vdash \lambda y : T. s \in T \to S} {\vdash \lambda y : T. s \in T \to S} & \quad \Delta_2 \\
\frac{\vdash t \in T} {\vdash t \in T} & \quad \Delta_2 \\
\frac{\vdash (\lambda y : T. s) t \in S} {\vdash (\lambda y : T. s) t \in S} & \quad \Delta_2 \\
\end{align*}
\]
Subtyping
Motivation

A perfectly reasonable program that is not typeable in the STLC...

\[(\lambda x:\text{Person}. \ x.\text{n}) \ alice\]

:-(
Inventors of the Simula and Simula-67 languages
Simula-67 was the first language to incorporate subtyping
• (The underlying idea was inspired by Tony Hoare)
In some research teams a new idea is treated with loving care: "How interesting!", "Beautiful!". This was not the case in the SIMULA development. When one of us announced that he had a new idea, the other would brighten up and do his best to kill it off. Assuming that the person who got the idea is willing to fight, this is a far better mode of work than the mode of mutual admiration. We think it was useful for us, and we succeeded in discarding a very large number of proposals.
Long career in famous research labs (Bell Labs, DEC SRC, Microsoft Cambridge); currently at Oxford

Many contributions to PL (and systems biology!)
• “Typeful programming”
• Bounded quantification (System $F_{\omega}$)
• Record calculi
• Mobile Ambients
• A Theory of Objects (with Abadi)

Winner of the Dahl-Nygaard prize in 2007 (among many other awards)
à J.-Y. GIRAUD
l'inventeur du Système F

rules

ok

non non

fumer

Musée des Systèmes Templiers
Musée des Hermans
Luca’s Dijkstra font

<table>
<thead>
<tr>
<th>Dijkstra</th>
<th>32</th>
<th>33</th>
<th>34</th>
<th>35</th>
<th>36</th>
<th>37</th>
<th>38</th>
<th>39</th>
<th>40</th>
<th>41</th>
<th>42</th>
<th>43</th>
<th>44</th>
<th>45</th>
<th>46</th>
<th>47</th>
<th>/</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABC</td>
<td>DEF</td>
<td>GHI</td>
<td>J</td>
<td>K</td>
<td>LM</td>
<td>NO</td>
<td>P</td>
<td>QR</td>
<td>ST</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>]</td>
</tr>
<tr>
<td>PQRT</td>
<td>SU</td>
<td>TV</td>
<td>W</td>
<td>X</td>
<td>YZ</td>
<td>[ \</td>
<td>] ^</td>
<td>_</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
</tr>
<tr>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
</tr>
<tr>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td>}</td>
<td>~</td>
<td>^</td>
<td>_</td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>113</td>
<td>114</td>
<td>115</td>
<td>116</td>
<td>117</td>
<td>118</td>
<td>119</td>
<td>120</td>
<td>121</td>
<td>122</td>
<td>z</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
<td>127</td>
<td></td>
</tr>
</tbody>
</table>
Trained first as a professional musician, then did a PhD in theoretical physics

A few of his contributions:
- Definitional ("metacircular") interpreters
- Continuations
- Polymorphic lambda-calculus
- Forsythe, a language with intersection types
- Syntactic control of interference -> Separation logic
- Intrinsic semantics of subtyping
We add just one new rule to the typing relation – the so-called “Rule of Subsumption”:

\[
\frac{\Gamma \vdash e \in S \quad S <: T}{\Gamma \vdash e \in T} \quad \text{(T-SUB)}
\]
Subtyping

\[
\begin{align*}
B & <: B \quad (S-BASE) \\
S_1 & \rightarrow S_2 \quad T_1 & <: S_1 \quad T_2 & <: S_2 \quad (S-ARROW) \\
S_1 & <: T_1 \quad \ldots \quad S_n & <: T_n \\
\{l_1:S_1, \ldots, l_n:S_n, k_1:U_1, \ldots, k_m:U_m\} & <: \{l_1:T_1, \ldots, l_n:T_n\} \quad (S-RCD)
\end{align*}
\]
Now our term typechecks! :-)

\[
\begin{align*}
\text{\(x\) : Person} & \vdash x \in \text{Person} \quad \text{(T-VAR)} \\
\text{\(x\) : Person} & \vdash x.n \in \text{String} \quad \text{(T-Abs)} \\
\vdash \lambda x: \text{Person.} \ x.n \in \text{Person} \rightarrow \text{String} \quad \text{(T-Sub)} \\
\vdash \lambda x: \text{Person.} \ x.n \in \text{Student} \rightarrow \text{String} \quad \text{(T-App)} \\
\vdash (\lambda x: \text{Person.} \ x.n) \ alice \in \text{String}
\end{align*}
\]
Professor at MIT. PhD (1970) with John McCarthy on chess endgames(!).

Some big contributions to PL:
• Data abstraction
  • CLU language
• Semantics of subtyping
  • Liskov substitution principle (with Jeanette Wing)

(Also major contributions in distributed systems.)

Turing award, 1998

One of the first women to earn a PhD in Computer Science.

Who was the very first?

Barbara Liskov 1939 -
PhD in CS, 1965 (Wisconsin)

Missed being the very first CS PhD by a few hours!
But we’re not quite done...

This derivation...

\[
\begin{align*}
\Gamma, x: T &\vdash e_1 \in U & (T-\text{Abs}) \\
\Gamma &\vdash \lambda x: T. e_1 \in T \rightarrow U & (T-\text{Abs}) \\
\Gamma &\vdash e_2 \in T & (T-\text{Abs}) \\
\Gamma &\vdash (\lambda x: T. e_1) e_2 \in U & (T-\text{App}) \\
\end{align*}
\]

...doesn’t match the LHS of the beta-reduction rule:
New reduction rule!

\[
\begin{aligned}
\Delta_1 & \quad \Delta_2 \\
\Gamma \vdash e_1 \in S_1 \rightarrow S_2 & \quad T_1 \ll S_1 \\
\hline
\Gamma \vdash e_1 \in T_1 \rightarrow T_2 & \quad S_1 \rightarrow S_2 \ll T_1 \rightarrow T_2 \\
\end{aligned}
\]

\[
\begin{aligned}
\Delta_3 & \quad \Delta_4 \\
\hline
\Gamma \vdash e_1 \in S_1 \rightarrow S_2 & \quad \Delta_2 \\
\Gamma \vdash e_2 \in T_1 & \quad \Delta_4 \\
S_1 \rightarrow S_2 \ll T_1 \rightarrow T_2 & \quad \Gamma \vdash e_2 \in T_1 \\
\end{aligned}
\]

\[
\begin{aligned}
\Gamma \vdash e_1 e_2 \in T_2 & \quad \Gamma \vdash e_1 e_2 \in T_2 \\
\end{aligned}
\]

(Plus a similar rule for when T-Sub appears between T-Rcd and T-Proj.)
So... which is better?
The **extrinsic** approach is appropriate when types truly are ‘after-the-fact descriptions’ of underlying untyped behavior
• e.g., gradual type systems for untyped languages

The **intrinsic** approach is needed when types “matter for meaning”...
• coercions between numeric types, strings, etc
• Haskell typeclasses, etc.
• record calculi
Must we choose?

No!

E.g., Liquid Haskell
• Intrinsic core (Haskell)
• Extrinsic refinement types
What I hope you got out of this talk

• The distinction between intrinsic (Church-style) and extrinsic (Curry-style) typing
  • and why it matters
• How it extends to languages with subtyping
• A sense of a few important people
• Fun?
Thank you!!

Any more questions, discussion, ...?