Pure Type Systems: Extensions and Restrictions

Fairouz Kamareddine
Heriot-Watt University
Edinburgh, UK

Brasilia, Brasil, 24 May 2017
Types and sets existed since antiquity

- Euclid’s *Elements* (circa 325 B.C.) begins with:
  1. A *point* is that which has no part;
  2. A *line* is breadthless length.
  ...
  15. A *circle* is a plane figure contained by one line such that all the straight lines falling upon it from one point among those lying within the figure are equal to one another.

- 1..15 define points, lines, and circles which Euclid distinguished.

- Euclid always mentioned to which *class* (points, lines, etc.) an object belonged.

- In Euclid’s Elements (Book IX, Proposition 20):
  Consider any finite list of prime numbers $p_1, p_2, ..., p_n$. At least one additional prime number not in this list exists.
Paradoxes in the 20th century led to the creation of explicit theories of sets and types (and this was in parallel).

Researchers in set theory and type theory don’t collaborate often despite the huge overlap and complementarity of their work.

However, an impressive body of work exists to explain the strengths and weaknesses of theories and this work is a promising avenue in both areas.
Zeno of Elea, C. 590-525 B.C.E., devised some paradoxical arguments against the possibility of motion.

Since the calculus was developed partly to deal with motion, these paradoxical arguments are important for the foundations of analysis.

Three of the most important of these are in Aristotle in his Physics.

- **Dichotomy.** *There is no motion, because what moves must arrive at the middle of its course before it reaches the end.* In other words, to leave the room, you first have to get halfway to the door, then you have to get halfway from that point to the door, etc. No matter how close you are to the door, you have to go half the remaining distance before proceeding.

- **Arrow.** *The flying arrow is at rest,* because a thing is at rest when occupying its own space at a given time, as the arrow does at every instant of its alleged flight.
A short history of numbers

- *natural numbers* like 0, 1, 2, which were used to count (sheep for example);
- *integers* like 0, 1, -1, 2, -2, etc. which were also used to count;
- *rational numbers* which are the quotients or fractions of integers like 2/3 and which were used to measure (the ancients used *anthyphairesis*/Alternated substitution to evaluate Ratios);
  - $2/3 = 0.6666666...$ where 6 repeats over and over
  - $41/333 = 0.123123123123...$ where 123 repeats over and over.
- *irrational numbers* like $\sqrt{2}$, $\sqrt{3}$, $\pi$;
  - $\pi = 3.14159265358979323846264338...$
  - $\sqrt{2} = 1.41421356237309504880168872420969807856967187537694...$
It started with incommensurability

- According to Webster’s dictionary, *commensurability* is *divisibility without remainder by a common unit*.
- Hence 6 and 9 are commensurable (since they are both divisible by 3).
- Attempts to find the unit which measures exactly the side and diagonal of a square led to the proof of the *incommensurability* of the side and diagonal of a square.
- This result on incommensurability implies that $\sqrt{2}$ *is not a rational number*. That is, it cannot be represented as the quotient of two integers.
With incommensurability, number was no longer everything

- The discovery of the incommensurability of the side and diagonal of a square showed that the Pythagorean idea that *number is everything would not work*.

- The Pythagoreans needed to treat quantities which are not numbers.

- For them, numbers are rationals and quantities are the incommensurable (which we call *real numbers*).

- The Greeks constructed geometric figures (recall Euclid), but took numbers as given. They separated numbers, which are discrete, from continuous magnitudes (quantities/real numbers).

- They did not use fractions to approximate continuous magnitudes.

- They did not construct the reals, nor multiply them nor divide them, etc.
Euclid used anthyphairesis to find the greatest common divisor of two numbers.
Ratio of 12 to 5 = $[2,2,2]=2 + \frac{1}{2 + \frac{1}{2}}$

Figure 2: Ratio of 12 to 5
Ratio of 22 to 6 = \([3,1,2]\) = 3 + \(\frac{1}{1 + \frac{1}{2}}\)

Figure 3: Ratio of 22 to 6
Ratio of $\sqrt{2}$ to 1 is $[1, 2, 2, 2, \ldots]$ 

$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{\cdots}}}}$ etc.

\[
\begin{array}{|c|c|}
\hline
1 & \sqrt{2} - 1 \\
\hline
1 - \sqrt{2} & \sqrt{2} - 1 \\
\hline
\cdots & 3 - 2\sqrt{2} \\
\hline
\end{array}
\]
The ratio of 15 to 4 is \([3, 1, 3] = 3 + \frac{1}{1 + \frac{1}{3}}\).
The ratio of 20 to 7 is $[2, 1, 6] = 2 + \frac{1}{1 + \frac{1}{6}}$.
The ratio of 15 to 10 is $[1, 2] = 1 + \frac{1}{2}$
The ratio of $\sqrt{3}$ to 1 is characterised by
$[1, 1, 2, 1, 2, 1, 2, \ldots] = [1, \bar{1}, 2]$
Real numbers need to be constructed (using approximations like Dedekind cuts, Cauchy sequences, etc.)

The idea of using fractions to approximate continuous magnitudes developed first in the Arab world during the middle ages, and only came to Europe in the 16th and 17th centuries.

This idea would have been assumed by both Newton and Leibniz.

Although the Greeks did not construct magnitudes (real numbers), they still studied them after discovery of incommensurability.
In 18th and 19th century, irrational numbers were divided into two categories: *Algebraic* and *transcendental*.

- A number is algebraic if it is the root of a non-zero polynomial with rational coefficients. For example, $\sqrt{2}$ is algebraic since it is the solution to $x^2 - 2 = 0$.
- An irrational number that is not algebraic is called *transcendental* (i.e. cannot be made of algebraic equations). For example, $\pi$ is transcendental.
- *Transcendental was coined by Leibniz in 17th century who showed that $\sin(x)$ is not an algebraic function of $x$.*
- Until the discovery of irrationals like $\sqrt{2}$, the pythagorean expected all numbers to be rational.
The discovery of transcendental numbers

- Until the 17th century it was expected that numbers *should fit the algebraic mould.*
- In the *18th century* it was shown that $\pi$ is irrational and *conjectured* that $\pi$ is transcendental.
- In the 19th century, *proofs were given* of the existence of transcendental numbers and that $\pi$ *is transcendental.*
- Once $\pi$ was shown transcendental meant that the old problem of *squaring the circle became impossible.*
Researchers in the 19th century continued to go deeper into numbers

- 1821: Many controversies in analysis were solved by Cauchy. E.g., he gave a precise definition of convergence in his Cours d’Analyse.
- 1872: Due to the more exact definition of real numbers given by Dedekind, the rules for reasoning with real numbers became even more precise.
- 1895-1897: Cantor began formalizing set theory and made contributions to number theory.
- 1889: Peano formalized arithmetic, but did not seriously treat logic or quantification.
- Cantor’s diagnolisation argument and the size of the natural numbers versus the size of the real numbers will impact the size of what can be computable versus what cannot.
Cantor proved that *algebraic numbers are countable*. Hence there are only *as many algebraic numbers as there are natural numbers*. Cantor proved that *the transcendental numbers are uncountable*. Cantor proved that the size of the algebraic numbers is infinite, but is the *smallest infinite that exists*. The size of the transcendental numbers is a much much larger infinite.
Later on it was shown that:

- The size of the computable functions is the size of the algebraic numbers, the smallest infinite $\aleph_0$.
- The size of the non-computable functions is the size of the transcendental numbers (the monster infinite), which according to Cantor’s Continuum hypothesis is the infinite $\aleph_1$ which is the next one up after $\aleph_0$.
- This means that there are a lot more functions that are impossible to compute than there are computable functions.
From Cantor to Frege, Russell and Type Theory

- **General definition of function 1879** [8] is key to Frege’s formalisation of logic.
- 1892-1903 Frege’s *Grundgesetze der Arithmetik*, could handle elementary arithmetic, set theory, logic, and quantification.
- **Self-application of functions** was at the heart of Russell’s paradox 1902 [29].
- To **avoid paradox** Russell controled function application via **type theory**.
- Russell [30] **1903** gives the first type theory: the **Ramified Type Theory (RTT)**.
- **RTT** is used in Russell and Whitehead’s *Principia Mathematica* [32] 1910–1912.

Following from Leibniz and Frege, researchers started calling for logical methods that could decisively answer questions at hand.

Hilbert believed that every mathematical problem should either have a solution or we should definitely know that no such solution exists.

We must Know. We will know.

Back in 1928, Hilbert posed the Entscheidung/Decision Problem.

This problem dates back to Leibniz and asks for an algorithm that takes as input a statement of first order logic and returns as output one of two possible answers: yes when the statement is always valid, or no otherwise.
Hilbert advocated the idea (which became known as *Hilbert’s program*) that there should be an axiomatization of all of mathematics that is

- *a complete* (i.e., every true formula can be derived) and
- *consistent* (i.e., does not contain a contradiction)

such that *every mathematical problem should either have a solution or we should definitely know that no such solution exist.*
• The results of the 1930s would establish the limitations of computers even before computers were built.
• No matter how fast and advanced computers get (and they are advancing at an amazing speed, considering that they did not exist in 1930s).
• Before we knew what computers could do, we had results telling us what computers could never do.
• These results of the limitations of the computer, will never change.
• They are set in stone just like the impossibility of squaring a circle.
Can we solve/compute everything?

- Turing answered the question via a *machine for running/computing programs.*
  
  *A function $f$ is computable iff $f$ can be computed on a Turing machine.*

- Church invented the $\lambda$-calculus, a *language for describing programs.*
  
  *A function $f$ is computable iff $f$ can be described in the $\lambda$-calculus.*

- Note that Church’s $\lambda$-calculus was initially intended as a language of programs and logic, but it turned out to be inconsistent (Kleene and Rosser) and Church restricted the $\lambda$-calculus to programs.

- Goedel’s result meant that *no absolute guarantee can be given that many significant branches of mathematics are entirely free of contradictions.*

- This means: we can compute a very small ($\sim$ly countable, size of $\mathbb{N}$) amount compared to what we will never be able to compute (uncountable, size of $\mathbb{R}$).
Church’s *simply typed $\lambda$-calculus* $\lambda \rightarrow [4] 1940 = \lambda$-calculus + STT.

- The hierarchies of types/orders in RTT and STT are *unsatisfactory*.
- Hence, birth of *different systems of functions and types*, each with *different functional power*. 
Simply typed \( \lambda \)-calculus was adopted in theorem provers like HOL and was used to make sense of other programming languages (e.g., pascal).

Then, simple types were independently extended to polymorphic logic and programming languages.

Dependent types (necessary for reasoning about proofs inside the system) were also introduced in Automath by de Bruijn.

Types continue to play an influential role in the design and implementation of programming languages and theorem provers.
Syntax of $\lambda$-calculus

Type Free

- $A ::= x \mid AB \mid \lambda x. B$
- $(\lambda x. B)C \rightarrow_{\beta} B[x := C].$

With simple types:

- $\sigma ::= T \mid \sigma \rightarrow \tau$
- $A ::= x \mid AB \mid \lambda x:\sigma. B$
- $(\lambda x : \sigma.B)C \rightarrow_{\beta} B[x := C].$

With dependent types:

- $A ::= x \mid \ast \mid \square \mid AB \mid \lambda x:A.B \mid \Pi x:A.B$
- $(\lambda x : A.B)C \rightarrow_{\beta} B[x := C].$
- Sometimes also with $(\Pi x : A.B)C \rightarrow_{\Pi} B[x := C].$
Church’s Simply Typed \( \lambda \)-calculus in modern notation

- Terms: \( A ::= x \mid AB \mid \lambda x:\sigma. B \)
- Types: \( ::= T \mid \sigma \rightarrow \tau \)
- \( \Gamma \) is an environment (set of declaration).
- Rules:

\[
\begin{align*}
\text{(start)} & \quad \frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma} \\
\text{(\( \lambda \))} & \quad \frac{\Gamma, x:\sigma \vdash A : \tau}{\Gamma \vdash \lambda x:\sigma.A : \sigma \rightarrow \tau} \\
\text{(app\( \Pi \))} & \quad \frac{\Gamma \vdash A : \sigma \rightarrow \tau \quad \Gamma \vdash B : \sigma}{\Gamma \vdash AB : \tau}
\end{align*}
\]
Common features of modern types and functions

- We can **construct** a type by abstraction. (Write $A : \ast$ for $A$ is a type)
  - $\lambda_{y:A}.y$, the identity over $A$ has type $A \rightarrow A$
  - $\lambda_{A:*}.\lambda_{y:A}.y$, the polymorphic identity has type $\Pi_{A:*}.A \rightarrow A$
- We can **instantiate** types. E.g., if $A = \mathbb{N}$, then the identity over $\mathbb{N}$
  - $(\lambda_{y:A}.y)[A := \mathbb{N}]$ has type $(A \rightarrow A)[A := \mathbb{N}]$ or $\mathbb{N} \rightarrow \mathbb{N}$.
  - $(\lambda_{A:*}.\lambda_{y:A}.y)\mathbb{N}$ has type $(\Pi_{A:*}.A \rightarrow A)\mathbb{N} = (A \rightarrow A)[A := \mathbb{N}]$ or $\mathbb{N} \rightarrow \mathbb{N}$.
- $(\lambda_{x:\alpha}.A)B \rightarrow_{\beta} A[x := B]$ 
  $(\Pi_{x:\alpha}.A)B \rightarrow_{\Pi} A[x := B]$
- Write $A \rightarrow A$ as $\Pi_{y:A}.A$ when $y$ not free in $A$. 


Special Pure Type Systems

- **Syntax**: \( s ::= * | \Box_i \) where \( i \geq 1 \).
  
  We assume \( S \) set of sorts \( s \).

- **\( A ::= x^s | s | AB | \lambda x^s : A.B | \Pi x^s : A.B \)**

- \( A_{F_\omega} = A_{CC} = \{(*, \Box_1), (\Box_i, \Box_{i+1})\} \)
  
  \( A_Z = \{(*, \Box_1), (\Box_1, \Box_2), (\Box_2, \Box_3)\} \)

- \( R_{CC} = \{(*, *, *), (\Box_i, *, *), (*, \Box_i, \Box_j), (\Box_i, \Box_j, \Box_{\max\{i,j\}})\} \)
  
  \( R_{F_\omega} = \{(*, *, *), (\Box_i, *, *), (\Box_i, \Box_j, \Box_{\max\{i,j\}})\} \)
  
  \( R_Z = \{(*, *, *)\} \cup \{(\Box_i, *, *)|1 \leq i \leq 3\} \cup \{(\Box_i, \Box_j, \Box_{\max\{i,j\}})|1 \leq i, j \leq 2\} \)

- \( A_Z \subset A_{F_\omega} = A_{CC} \) and \( R_Z \subset R_{F_\omega} \subset R_{CC} \).

- **Formation rule:**
  
  \[
  \Gamma \vdash A : s_1 \quad \Gamma, x^s : A \vdash B : s_2 \quad \text{if } (s_1, s_2, s_3) \in R
  \]
  
  \[
  \Gamma \vdash \Pi x^s : A.B : s_3
  \]

  \[
  \]
The PTS rules

(axiom) \[ \langle \rangle \vdash s_1 : s_2 \text{ if } (s_1, s_2) \in A \]

(start) \[ \Gamma \vdash A : s \quad x^s \notin \text{DOM}(\Gamma) \quad \frac{}{\Gamma, x^s : A \vdash x^s : A} \]

(weak) \[ \Gamma \vdash A : B \quad \Gamma \vdash C : s \quad x^s \notin \text{DOM}(\Gamma) \quad \frac{}{\Gamma, x^s : C \vdash A : B} \]

(\Pi) \[ \Gamma \vdash A : s_1 \quad \Gamma, x^s : A \vdash B : s_2 \quad (s_1, s_2, s_3) \in R \quad \frac{}{\Gamma \vdash \Pi_{x^s : A}.B : s_2} \]

(\lambda) \[ \Gamma, x^s : A \vdash b : B \quad \Gamma \vdash \Pi_{x^s : A}.B : s' \quad \frac{}{\Gamma \vdash \lambda_{x^s : A}.b : \Pi_{x^s : A}.B} \]

(conv) \[ \Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_\beta B' \quad \frac{}{\Gamma \vdash A : B'} \]

(app\(\Pi\)) \[ \Gamma \vdash F : \Pi_{x^s : A}.B \quad \Gamma \vdash a : A \quad \frac{}{\Gamma \vdash Fa : B[x^s := a]} \]
Desirable properties of a type system with reduction $r$

- **$r$ is Church Rosser (CR)**
  
  If $A \to \to_r A'$ and $A \to \to_r A''$ then there is a $B$ such that $A' \to \to_r B$ and $A'' \to \to_r B$.

- **Typing is preserved under reduction**
  
  If $\Gamma \vdash A : B$ and ($A \to \to_r A'$ or $B \to \to_r B'$ or $\Gamma \to \to_r \Gamma'$) then $\Gamma' \vdash A' : B'$.

- **Strong Normalisation (SN)**
  
  If $\Gamma \vdash A : B$ then SN$_{\to_r}(A)$ and SN$_{\to_r}(B)$.
  
  SN properties for CC and $F_\omega$ have the same proof-theoretic strength as higher-order arithmetic ($HA_\omega$).
  
  CC and $F_\omega$ can be proven consistent within Heyting arithmetic.
Do we always need CR to hold

For example, in Krivine’s $\lambda_c$ (the language of realisers in classical realisability which is $\lambda$-calculus plus the control operator $cc$), CR does not hold. Instead we have determinism.

Subject Expansion

If $\Gamma \vdash A : B$ and $A' \rightarrow^r A$ then $\Gamma \vdash A' : B$.

The Type System Characterizes Strong Normalisation

If $M$ is a type free term such that $SN \rightarrow^r(M)$ then there are $\Gamma, A, B$, such that $\Gamma \vdash A : B$ and $TE(A) = M$.

Decidability of Type checking and typability

Given $A, B$, and $\Gamma$ do we have $\Gamma \vdash A : B$?

Given $A$ are there $\Gamma, B$ such that $\Gamma \vdash A : B$?

Given $A$ and $\Gamma$ is there $B$ such that $\Gamma \vdash A : B$?
Assume only $\ast, \square$. Let $\mathcal{A} = \{ (\ast, \square) \}$. Write $(s_1, s_2, s_2)$ as $(s_1, s_2)$ and let $R \subseteq \{ (\ast, \ast), (\ast, \square), (\square, \ast), (\square, \square) \}$.

<table>
<thead>
<tr>
<th></th>
<th>Simple Poly-morphic</th>
<th>Dependent Constructors</th>
<th>Related system</th>
<th>Refs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda \rightarrow$</td>
<td>$\ast, \ast$</td>
<td>$\square, \ast$</td>
<td>$\lambda^T, F$</td>
<td>[4, 2, 13]</td>
</tr>
<tr>
<td>$\lambda_2$</td>
<td>$\ast, \ast$</td>
<td>$\square, \ast$</td>
<td>$\lambda^T, F$</td>
<td>[10, 28]</td>
</tr>
<tr>
<td>$\lambda P$</td>
<td>$\ast, \ast$</td>
<td>$(\ast, \square)$</td>
<td>AUT-QE, LF</td>
<td>[6, 11]</td>
</tr>
<tr>
<td>$\lambda \omega$</td>
<td>$\ast, \ast$</td>
<td>$\square, \ast$</td>
<td>POLYREC</td>
<td>[27]</td>
</tr>
<tr>
<td>$\lambda P_2$</td>
<td>$\ast, \ast$</td>
<td>$\square, \ast$</td>
<td>F$\omega$</td>
<td>[22]</td>
</tr>
<tr>
<td>$\lambda \omega$</td>
<td>$\ast, \ast$</td>
<td>$(\ast, \square)$</td>
<td>F$\omega$</td>
<td>[10]</td>
</tr>
<tr>
<td>$\lambda P \omega$</td>
<td>$\ast, \ast$</td>
<td>$(\ast, \square)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lambda C$</td>
<td>$\ast, \ast$</td>
<td>$(\ast, \square)$</td>
<td>CC</td>
<td>[5]</td>
</tr>
</tbody>
</table>
The 8 Systems of the Barendregt Cube

The Barendregt Cube

$\lambda 2$

$\lambda \omega$

$\lambda P$

$\lambda P2$

$\lambda C$

$\lambda P \omega$

$(\Box, \ast) \in R$

$(\Box, \Box) \in R$

$(\ast, \Box) \in R$
Typing Polymorphic identity needs (□, *)

\[
\frac{y : * \vdash y : *}{y : * \vdash \Pi x : y. y : *}
\] by (\Pi) (\ast, \ast)

\[
\frac{y : *, x : y \vdash x : y}{y : * \vdash \lambda x : y. x : \Pi x : y. y}
\] by (\lambda)

\[
\vdash * : \square \quad \frac{y : * \vdash \Pi x : y.x : \Pi x : y. y}{\vdash \Pi y : *. \Pi x : y.y : *}
\] by (\Pi) by (\square, \ast)

\[
\frac{y : * \vdash \lambda x : y.x : \Pi x : y.y}{\vdash \lambda y : *. \lambda x : y.x : \Pi y : *. \Pi x : y.y}
\] by (\lambda)
The Cube with parametric constants

- Let \((\ast, \ast) \subseteq \mathbb{R}, P \subseteq \{(\ast, \ast), (\ast, \square), (\square, \ast), (\square, \square)\}\).

- \(\lambda R P = \lambda R\) and the two rules \((\textbf{C-weak})\) and \((\textbf{C-app})\):

\[
\begin{array}{c}
\Gamma \vdash b : B \\
\Gamma, \Delta_i \vdash B_i : s_i \\
\Gamma, \Delta \vdash A : s \\
\hline
\Gamma, c(\Delta) : A \vdash b : B
\end{array}
\quad (s_i, s) \in P, c \text{ is } \Gamma\text{-fresh}
\]

\[
\begin{array}{c}
\Gamma_1, c(\Delta) : A, \Gamma_2 \vdash b_i : B_i[x_j := b_j]_{j=1}^{i-1} \quad (i = 1, \ldots, n) \\
\Gamma_1, c(\Delta) : A, \Gamma_2 \vdash A : s \quad (\text{if } n = 0) \\
\hline
\Gamma_1, c(\Delta) : A, \Gamma_2 \vdash c(b_1, \ldots, b_n) : A[x_j := b_j]_{j=1}^{n}
\end{array}
\]

\(\Delta \equiv x_1 : B_1, \ldots, x_n : B_n.\)

\(\Delta_i \equiv x_1 : B_1, \ldots, x_{i-1} : B_{i-1}\)
The refined Barendregt Cube

\[ (\Box, \ast) \in R \]
\[ (\Box, \ast) \in P \]
\[ (\ast, \Box) \in P \]
\[ (\ast, \Box) \in R \]
The $\pi$-cube: $R_\pi = R_\beta \setminus (\text{conv}_\beta) \cup (\text{conv}_\beta \Pi)$, $\beta \Pi$

- $(\lambda x: \alpha. A)B \rightarrow_\beta A[x := B]$  
- $(\Pi x: \alpha. A)B \rightarrow_\Pi A[x := B]$

\[
\begin{array}{c}
\text{(axiom)} & \text{(start)} & \text{(weak)} & \text{(\Pi)} & \text{(\lambda)} & \text{(app\Pi)} \\
\hline
(\text{conv}_\beta \Pi) & \Gamma \vdash A : B & \Gamma \vdash B' : s & B =_\beta \Pi B' & \Gamma \vdash A : B'
\end{array}
\]

\textit{Lemma:} $\Gamma \vdash_\beta A : B$ iff $\Gamma \vdash_\pi A : B$

\textit{Lemma:} The $\beta$-cube and the $\pi$-cube satisfy the desirable for type systems.
The $\pi_i$-cube: $R_{\pi_i} = R_\pi \setminus (\text{app}_\Pi) \cup (i\text{-app}_\Pi)$, $\rightarrow_{\beta\Pi}$

$\begin{align*}
\text{(app}_\Pi) & \quad \frac{\Gamma \vdash F : \Pi_x:A.B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x:=a]} \\
\text{(axiom)} & \quad \text{(start)} \quad \text{(weak)} \quad (\Pi) \quad (\lambda) \\
\text{(conv}_{\beta\Pi}) & \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_{\beta\Pi} B'}{\Gamma \vdash A : B'} \\
\text{(i-app}_\Pi) & \quad \frac{\Gamma \vdash F : \Pi_x:A.B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\Pi_x:A.B)a}
\end{align*}$

Lemma:

- If $\Gamma \vdash_{\beta} A : B$ then $\Gamma \vdash_{\pi_i} A : B$.
- If $\Gamma \vdash_{\pi_i} A : B$ then $\Gamma \vdash_{\beta} A : [B]_\Pi$ where $[B]_\Pi$ is the $\Pi$-normal form of $B$. 
The $\pi_i$-cube loses TC and SR properties
Let $\Gamma = z : \ast, x : z$. We have that $\Gamma \vdash_{\pi_i} (\lambda y : z. y) x : (\Pi y : z. z) x$.
- We do not have TC $(\Pi y : z. z) x \not\equiv \Box$ and $\Gamma \not\vdash_{\pi_i} (\Pi y : z. z) x : s$.
- We do not have SR $(\lambda y : z. y) x \rightarrow^\beta \Pi x$ but $\Gamma \not\vdash_{\pi_i} x : (\Pi y : z. z) x$.

But we have:
- We have STT
- We have PT
- We have SN
- We have a weak form of TC If $\Gamma \vdash_{\pi_i} A : B$ and $B$ does not have a $\Pi$-redex then either $B \equiv \Box$ or $\Gamma \vdash_{\pi_i} B : s$.
- We have a weak form of SR If $\Gamma \vdash_{\pi_i} A : B$, $B$ is not a $\Pi$-redex and $A \rightarrow^\beta \Pi A'$ then $\Gamma \vdash_{\pi_i} A' : B$. 
The problem can be solved by re-incorporating Frege and Russell’s notions of low level functions (which was lost in Church’s notion of function)

\[
\begin{align*}
\text{(start-a)} & \quad \frac{\Gamma \vdash A : s \quad \Gamma \vdash B : A}{\Gamma, x = B : A \vdash x : A} \quad x \notin \text{DOM} (\Gamma) \\
\text{(weak-a)} & \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s \quad \Gamma \vdash D : C}{\Gamma, x = D : C \vdash A : B} \quad x \notin \text{DOM} (\Gamma)
\end{align*}
\]

**Figure 4:** Basic abbreviation rules BA

\[
\Gamma, x = B : A \vdash C : D \\
\Gamma \vdash (\lambda x : A.C)B : D[x := B]
\]

**Figure 5:** (let\(\backslash\)) where \(\backslash = \lambda\) or \(\backslash = \Pi\)
The $\beta_a$-cube: $R_{\beta_a} = R_{\beta} + BA + \text{let}_\beta$, $\rightarrow_{\beta}$

(axiom)  (start) (weak) ($\Pi$) ($\lambda$) (app$\Pi$) (conv$\beta$)

(start-a) \[
\begin{array}{ll}
\Gamma \vdash A : s & \Gamma \vdash B : A \\
\hline
\Gamma, x = B : A \vdash x : A \\
\end{array}
\]

(weak-a) \[
\begin{array}{llll}
\Gamma \vdash A : B & \Gamma \vdash C : s & \Gamma \vdash D : C \\
\hline
\Gamma, x = D : C \vdash A : B \\
\end{array}
\]

(let$\beta$) \[
\begin{array}{ll}
\Gamma, x = B : A \vdash C : D \\
\hline
\Gamma \vdash (\lambda x : A. C)B : D[x := B] \\
\end{array}
\]

Lemma: The $\beta_a$-cube satisfies the desirable properties except for typability of subterms.
If $A$ is $\vdash$-legal and $B$ is a subterm of $A$ such that every bachelor $\lambda x : D$ in $B$ is also bachelor in $A$, then $B$ is $\vdash$-legal.
The $\pi_a$-cube: $R_{\pi_a} = R_\pi + BA + \text{let}_\beta + \text{let}_\Pi$, $\rightarrow_\beta_\Pi$

(axiom) (start) (weak) (ÎΠ) (Îλ) (appÎΠ) (convÎβÎΠ)

(start-a)

\[
\frac{\Gamma \vdash A : s \quad \Gamma \vdash B : A}{\Gamma, x = B : A \vdash x : A} \quad x \notin \text{DOM} (\Gamma)
\]

(weak-a)

\[
\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s \quad \Gamma \vdash D : C}{\Gamma, x = D : C \vdash A : B} \quad x \notin \text{DOM} (\Gamma)
\]

(let$_\beta$)

\[
\frac{\Gamma, x = B : A \vdash C : D}{\Gamma \vdash (\lambda x : A. C)B : D[x := B]}
\]

(let$_\Pi$)

\[
\frac{\Gamma, x = B : A \vdash C : D}{\Gamma \vdash (\Pi x : A. C)B : D[x := B]}
\]

Lemma: The $\pi_a$-cube satisfies the same properties as the $\beta_a$. 
The $\pi_{ai}$-cube: $R_{\pi_{ai}} = R_{\pi_a} \setminus \text{app}_\Pi + \text{i-app}_\Pi$, $\rightarrow_\beta \Pi$

Let $\Gamma = z : *, x : z$. We have that $\Gamma \vdash_{\pi_{ai}} (\lambda y : z.y)x : (\Pi y : z.z)x$.

- **We NOW have TC** although $\Gamma \not\vdash_{\pi} (\Pi y : z.z)x : s$, we have $\Gamma \vdash_{\pi_{ai}} (\Pi y : z.z)x : s$
  - By (weak-a) $z : *, x : z, y = x : z \vdash_{\pi_{ai}} z : *$.
  - Hence by (let$\Pi$) $z : *, x : z \vdash_{\pi_{ai}} (\Pi y : z.z)x : *[y := x] \equiv *$.

- **We NOW have SR** $(\lambda y : z.y)x \rightarrow_\beta \Pi x$.
  - Although $\Gamma \not\vdash_{\pi} x : (\Pi y : z.z)x$, we have $\Gamma \vdash_{\pi_{ai}} x : (\Pi y : z.z)x$
  - Since $z : *, x : z \vdash_{\pi_{ai}} x : z$, and $z : *, x : z \vdash_{\pi_{ai}} (\Pi y : z.z)x : *$ and $z : *, x : z \vdash z =_\beta \Pi (\Pi y : z.z)x$, we use (conv$_{\beta \Pi}$) to get:
  - $z : *, x : z \vdash_{\pi_{ai}} x : (\Pi y : z.z)x$.  

Kamareddine Pure Type Systems: Extensions and Restrictic Brasilia, Brasil, 24 May 2017 47 / 71
Degrees of terms

- We define $\# : \Lambda \to \{0, 1, 2, 3\}$ by $\#(\square) = 3$, $\#(*) = 2$, $\#(x^\varsigma) = \#(\varsigma) - 2$, $\#(\pi\delta.A) = \#(A)B = \#(A)$.
- For $A \in \Lambda$, $\#(A)$ is called the degree of $A$.
- $A : B$ is OK iff $\#(A) = \#(B) - 1$.
- If $\Gamma \vdash A : B$ then for any $C : D$ in either $\Gamma$ or $A$ or $B$ we have $C : D$ is OK.
- $A$ is *kind* iff $\#(A) = 2$.
  - $A$ is *constructor* or $A$ is *type* iff $\#(A) = 1$.
  - $A$ is *object* iff $\#(A) = 0$.
- $\#(A) = 3$ iff $A = \square$. 
Guy Steele’s discussion of most popular programming language in computer science

Computer Science Metanotation (CSM)

- **Data Types:**
  - Built-in: numbers, arrays, lists, etc.
  - User-defined: Records, Abstract Data Types or Symbolic Expressions (written in BNF).
- **Code:** Inference rules (written in Gentzen notation)
- **Conditionals:** rule dispatch via nondeterministic pattern-matching
- **Repetition:** overlines and/or ellipsis notations, and sometimes iterators
- **Primitive expressions:** logic and mathematics
- **Special operation:** capture-free substitution within a symbolic expression
According to Steele

Early contributors include

- Gentzen
- Bakus
- Naur
- Church

Steel focuses around difficulties of use of BNF notation:

- Substitution
- Overline and Ellipsis
- Formalisation and Mechanisation of CSM.
Gerhard Gentzen with his rule metanotation for natural deduction:

“3.1. Eine Schlußfigur läßt sich in der Form Schreiben:

$$
\begin{array}{c}
\vdash N_1 \ldots N_\nu \\
\end{array}
\Rightarrow
\begin{array}{c}
B \\
\end{array}
(\nu \geq 1),

wobei \( N_1, \ldots, N_\nu \) Formeln sind. \( N_1, \ldots, N_\nu \) heißen dann die Oberformeln, \( B \) heißt die Unterformel der Schlußfigur.”

(Gerhard Gentzen, 1934 [9])
• John Backus influenced by Emil Post’s productions gives a syntax to write production rules \textit{with multiple alternatives} for a context-free grammar for the International Algorithmic Language:

\begin{verbatim}
"< digit >::= 0 \text{ or } 1 \text{ or } 2 \text{ or } 3 \text{ or } 4 \text{ or } 5 \text{ or } 6 \text{ or } 7 \text{ or } 8 \text{ or } 9

<integer> ::= < digit > \text{ or } < integer > < digit >"
\end{verbatim}

(John Backus, 1959)

• Peter Naur uses Backus notation where
  \begin{itemize}
  \item $\equiv \implies ::=$ and
  \item $\lor \implies |$ and \textit{gave nonterminals the same names used in the text.}
  \end{itemize}

\begin{verbatim}
"< unsigned integer > ::= < digit > | < unsigned integer > < digit >
<integer> ::= < unsigned integer > | + < unsigned integer > | - < unsigned integer >"
\end{verbatim}

(Naur, report on Algol 60, CACM)
Data Declaration à la Alonzo Church

Type Free

- $A ::= x \mid AB \mid \lambda x.B$
- $(\lambda x.B)C \rightarrow_\beta B[x := C]$. 

With simple types:

- $\sigma ::= T \mid \sigma \rightarrow \tau$
- $A ::= x \mid AB \mid \lambda \sigma : \sigma.B$
- $(\lambda x : \sigma.B)C \rightarrow_\beta B[x := C]$. 

With dependent types:

- $A ::= x \mid * \mid \Box \mid AB \mid \lambda x : A.B \mid \Pi x : A.B$
- $(\lambda x : A.B)C \rightarrow_\beta B[x := C]$. 
- Sometimes also with $(\Pi x : A.B)C \rightarrow_\Pi B[x := C]$. 

Kamareddine Pure Type Systems: Extensions and Restrictic Brasilia, Brasil, 24 May 2017 53 / 71
With dependent/polymorphic types

\[
\begin{align*}
\frac{\Gamma, x:A \vdash b : B \quad \Gamma \vdash \Pi_{x:A}B : s}{\Gamma \vdash \lambda x:A.b : \Pi_{x:A}B} & \\
\frac{\Gamma \vdash F : \Pi_{x:A}B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x:=a]} & \\
\frac{\Gamma \vdash F : \Pi_{x:A}B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : (\Pi_{x:A}B)a}
\end{align*}
\]

With simple types:

\[
\begin{align*}
\frac{\Gamma, x:\sigma \vdash b : \tau}{\Gamma \vdash \lambda x:A.b : \sigma \rightarrow \tau} & \\
\frac{\Gamma \vdash F : \tau \rightarrow \sigma \quad \Gamma \vdash a : \tau}{\Gamma \vdash Fa : \sigma}
\end{align*}
\]
In this talk we concentrate on the development of calculi rather than the development of the notation.

- The challenge is to develop expressive calculi that have clear syntax, semantics, and the desirable properties (Church-Rosser, correctness, termination).
- Nonetheless, notation is important.
- For example, simply changing the order of functions and arguments, and restructuring parenthesis, enable us to:
  - Express things that would be hard to do in the old notation.
  - Reduce proofs of strong normalisation to proofs of weak normalisation.
  - Make computations more efficient.
  - Avoid unnecessary/redundant computations and allow for free lazy, local, or global reductions.
Lambda Calculus à la de Bruijn

- $A ::= x \mid AB \mid \lambda x. B$
- $A ::= x \mid < B > A \mid [x]B$
- $I(x) = x$, $I(\lambda x. B) = [x]I(B)$, $I(AB) = \langle I(B) \rangle I(A)$
- $(\lambda x.\lambda y.xy)z$ translates to $\langle z \rangle [x][y]\langle y \rangle x$.
- The *applicator wagon* $\langle z \rangle$ and *abstractor wagon* $[x]$ occur NEXT to each other.
- $(\lambda x. A)B \rightarrow_\beta A[x := B]$ becomes $\langle B \rangle [x] A \rightarrow_\beta [x := B]A$
- The “bracketing structure” of $((\lambda_x.(\lambda_y.\lambda_z.x)c)b)a$ is ‘$[1 \ 2 \ [3 \ ]2 \ ]1 \ ]3$’, where ‘[’ and ‘]’ match.
- The bracketing structure of $\langle a \rangle \langle b \rangle [x] \langle c \rangle [y] [z] \langle d \rangle$ is simpler: $[ [ [ [ ] ] ] ]$.
- $\langle b \rangle [x]$ and $\langle c \rangle [y]$ are AT-pairs whereas $\langle a \rangle [z]$ is an AT-couple.
Redexes in de Bruijn’s notation

Classical Notation

\[
((\lambda_x.(\lambda_y.\lambda_z.zd)c)b)a \\
\downarrow\beta \\
((\lambda_y.\lambda_z.zd)c)a \\
\downarrow\beta \\
(\lambda_z.zd)a \\
\downarrow\beta \\
ad
\]

de Bruijn’s Notation

\[
\langle a \rangle \langle b \rangle [x] \langle c \rangle [y] [z] \langle d \rangle z \\
\downarrow\beta \\
\langle a \rangle \langle c \rangle [y] [z] \langle d \rangle z \\
\downarrow\beta \\
\langle a \rangle [z] \langle d \rangle z \\
\downarrow\beta \\
\langle d \rangle a
\]

This makes it easy to study local/global mini reductions into the \(\lambda\)-calculus, Kamareddine et al [16, 17]
Some notions of reduction studied in the literature

<table>
<thead>
<tr>
<th>Name</th>
<th>In Classical Notation</th>
<th>In de Bruijn's notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(θ)</td>
<td>((λ_x.N)P)Q</td>
<td>⟨Q⟩⟨P⟩[x]N</td>
</tr>
<tr>
<td></td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td></td>
<td>(λ_x.NQ)P</td>
<td>⟨P⟩[x]⟨Q⟩N</td>
</tr>
<tr>
<td>(γ)</td>
<td>(λ_x.(λ_y.N)P)</td>
<td>⟨P⟩[x][y]N</td>
</tr>
<tr>
<td></td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td></td>
<td>λ_y.(λ_x.N)P</td>
<td>[y]⟨P⟩[x]N</td>
</tr>
<tr>
<td>(γC)</td>
<td>((λ_x.(λ_y.N)P)Q)</td>
<td>⟨Q⟩⟨P⟩[x][y]N</td>
</tr>
<tr>
<td></td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td></td>
<td>(λ_y.(λ_x.N)P)Q</td>
<td>⟨Q⟩[y]⟨P⟩[x]N</td>
</tr>
<tr>
<td>(g)</td>
<td>((λ_x.(λ_y.N)P)Q)</td>
<td>⟨Q⟩⟨P⟩[x][y]N</td>
</tr>
<tr>
<td></td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td></td>
<td>(λ_x.N[y := Q])P</td>
<td>⟨P⟩[x][y := Q]N</td>
</tr>
<tr>
<td>(β_e)</td>
<td>?</td>
<td>⟨Q⟩ś[y]N</td>
</tr>
<tr>
<td></td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td></td>
<td>?</td>
<td>ś[y := Q]N</td>
</tr>
</tbody>
</table>
A Few Uses of these reductions/term reshuffling

- Regnier [26] uses $\theta$ and $\gamma$ in analyzing perpetual reduction strategies.
- Term reshuffling is used by Kfoury, Tiuryn, Urzyczyn, Wells in [21, 19] in analyzing typability problems.
- Nederpelt [23], de Groote [7], Kfoury+ Wells [20], and Kamareddine [15] use generalised reduction and/or term reshuffling in relating SN to WN.
- Ariola etal [1] uses a form of term-reshuffling in obtaining a calculus that corresponds to lazy functional evaluation.
- Kamareddine etal [16, 14, 18, 3] show that they could reduce space/time needs in computation.
Even more: de Bruijn’s generalised reduction has better properties

\[(\beta)\quad (\lambda_x.M)N \rightarrow M[x := N] \]
\[(\beta_I)\quad (\lambda_x.M)N \rightarrow M[x := N] \quad \text{if } x \in \text{FV}(M) \]
\[(\beta_K)\quad (\lambda_x.M)N \rightarrow M \quad \text{if } x \notin \text{FV}(M) \]
\[(\theta)\quad (\lambda_x.N)PQ \rightarrow (\lambda_x.NQ)P \]
\[(\beta_e)\quad (M)\bar{s}[x]N \rightarrow \bar{s}\{N[x := M] \quad \text{for } \bar{s} \text{ well-balanced.} \]

- Kamareddine [15] shows that $\beta_e$ satisfies *Church Rosser, PSN*, postponement of $K$-contraction and conservation (latter 2 properties fail for $\beta$-reduction).

- **Conservation of $\beta_e$:** If $A$ is $\beta_e$I-normalisable then $A$ is $\beta_e$-strongly normalisable.

- **Postponement of $K$-contraction:** Hence, discard arguments of $K$-redexes after I-reduction. This gives flexibility in implementation: *unnecessary work can be delayed, or even completely avoided.*
Attempts have been made at establishing some reduction relations for which postponement of \( K \)-contractions and conservation hold.

The picture is as follows (-N stands for normalising and \( r \in \{\beta_I, \theta_K\} \)).

\[ (\beta_K\text{-postponement for } r) \text{ If } M \rightarrow_{\beta_K} N \rightarrow_r O \text{ then } \exists P \text{ such that } M \rightarrow^+_{\beta_I \theta_K} P \rightarrow_{\beta_K} O \]

\[ (\text{Conservation for } \beta_I) \text{ If } M \text{ is } \beta_I\text{-N then } M \text{ is } \beta_I\text{-SN} \]

\[ (\text{Conservation for } \beta + \theta) \text{ If } M \text{ is } \beta_I \theta_K\text{-N then } M \text{ is } \beta\text{-SN} \quad [7] \]

De Groote does not produce these results for a single reduction relation, but for \( \beta + \theta \) (this is more restrictive than \( \beta_e \)).

\( \beta_e \) is the first single relation to satisfy \( \beta_K \)-postponement and conservation.

Kamareddine [15] shows that:

\[ (\beta_e K\text{-postponement for } \beta_e) \text{ If } M \rightarrow_{\beta_e K} N \rightarrow_{\beta_e l} O \text{ then } \exists P \text{ such that } M \rightarrow_{\beta_e l} P \rightarrow^+_{\beta_e K} O \]

\[ (\text{Conservation for } \beta_e) \text{ If } M \text{ is } \beta_{el}\text{-N then } M \text{ is } \beta_e\text{-SN} \]
Canonical typing

There are reasons why separating the questions “what is the type of a term” (via $\tau$) and “is the term typable” (via $\vdash$), is advantageous:

- The canonical type of $A$ is easy to calculate.
- $\tau(A)$ plays the role of a preference type for $A$. If $A \equiv \lambda x:*.(\lambda y:*y)x$ then $\tau(\langle\rangle, A) \equiv \Pi x:.*.(\Pi y:.*.*)x \rightarrow^* \Pi y:.*$, the type of $A$.
- The conversion rule is no longer needed as a separate rule in the definition of $\vdash$. It is accommodated in our application rule:

\[
\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash AB} \quad \text{if} \quad \tau(\Gamma, A) = \beta \Pi \Pi x:C . D \quad \text{and} \quad \tau(\Gamma, B) = \beta \Pi C
\]
Higher degrees: If we use \( \lambda^1 \) for \( \Pi \) and \( \lambda^2 \) for \( \lambda \) then we can aim for a possible generalization. In fact, we can extend our system by incorporating more different \( \lambda \)'s. For example, with an infinity of \( \lambda \)'s, viz. \( \lambda^0, \lambda^1, \lambda^2, \lambda^3 \ldots \), we replace \( \tau(\Gamma, \lambda_x:A.B) \equiv \Pi_x:A.\tau(\Gamma.\lambda_x:A, B) \) and \( \tau(\Gamma, \Pi_x:A.B) \equiv \tau(\Gamma.\lambda_x:A, B) \) by the following:

\[
\tau(\Gamma, \lambda^{i+1}_x:A.B) \equiv \lambda^i_x:A.\tau(\Gamma.\lambda_x:A, B), \text{ for } i = 0, 1, 2, \ldots \text{ where } \lambda^0_x:A.B \equiv B
\]

There may be circumstances in which one desires to have more “layers” of \( \lambda \)'s (see de Bruijn 1974).
This notion enables one to separate the judgement \( \Gamma \vdash A : B \) in two: \( \Gamma \vdash A \) and \( \tau(\Gamma, A) = B \).

\[
\begin{align*}
\tau(\Gamma, *) & \equiv \Box \\
\tau(\Gamma, x) & \equiv A \text{ if } (A\lambda_x) \in \Gamma \\
\tau(\Gamma, (a\delta)F) & \equiv (a\delta)\tau(\Gamma, F) \\
\tau(\Gamma, (A\lambda_x)B) & \equiv (A\Pi_x)\tau(\Gamma(A\lambda_x), B) \quad \text{if } x \notin \text{dom}(\Gamma) \\
\tau(\Gamma, (A\Pi_x)B) & \equiv \tau(\Gamma(A\lambda_x), B) \quad \text{if } x \notin \text{dom}(\Gamma)
\end{align*}
\]
In usual type theory:

- the type of \((\ast \lambda_x)(x\lambda_y)y\) is \((\ast \Pi_x)(x\Pi_y)x\) and
- the type of \((\ast \Pi_x)(x\Pi_y)x\) is \(\ast\).

With our \(\tau\), we get the same result:

- \(\tau(\langle\rangle, (\ast \lambda_x)(x\lambda_y)y) \equiv (\ast \Pi_x)\tau((\ast \lambda_x), (x\lambda_y)y) \equiv (\ast \Pi_x)(x\Pi_y)x\) and
- \(\tau(\langle\rangle, (\ast \Pi_x)(x\Pi_y)x) \equiv \tau((\ast \lambda_x), (x\Pi_y)x) \equiv \tau((\ast \lambda_x)(x\lambda_y), x) \equiv \ast\)
Let $\Gamma_0 \equiv<>$, $\Gamma_1 \equiv (\lambda z)$, $\Gamma_2 \equiv (\lambda z)(\lambda y)$, $\Gamma_3 \equiv \Gamma_2(\lambda x)$. We want to find the canonical type of $(\lambda z)(B\delta)(\lambda y)(y\delta)(\lambda x)x$ in $\Gamma_0$.

$$(\Gamma_0 \tau)\; (\lambda z)\; (B\delta)\; (\lambda y)\; (y\delta)\; (\lambda x) \times$$

$$(\Gamma_1 \tau)\; (B\delta)\; (\lambda y)\; (y\delta)\; (\lambda x) \times$$

$$(B\delta)\; (\Gamma_1 \tau)\; (\lambda y)\; (y\delta)\; (\lambda x) \times$$

$$(B\delta)\; (\lambda y)\; (y\delta)\; (\Gamma_2 \tau)\; (\lambda x) \times$$

$$(B\delta)\; (\lambda y)\; (y\delta)\; (\Gamma_2 \tau)\; (\lambda x) \times$$

$$(B\delta)\; (\lambda y)\; (y\delta)\; (\lambda x)\; (\Gamma_3 \tau) \times$$

$$(B\delta)\; (\lambda y)\; (y\delta)\; (\lambda x)\; (\Gamma_3 \tau) \times$$
New Typability

(\vdash\text{-axiom}) \quad \langle\rangle \vdash \ast

(\vdash\text{-start rule}) \quad \frac{\Gamma \vdash A}{\Gamma (A \lambda_x) \vdash x} \text{ if vc}

(\vdash\text{-weakening rule}) \quad \frac{\Gamma \vdash A \quad \Gamma \vdash D}{\Gamma (A \lambda_x) \vdash D} \text{ if vc}

(\vdash\text{-application rule}) \quad \frac{\Gamma \vdash F \quad \Gamma \vdash a}{\Gamma \vdash (a \delta) F} \text{ if ap}

(\vdash\text{-abstraction rule}) \quad \frac{\Gamma (A \lambda_x) \vdash b \quad \Gamma \vdash (A \Pi_x) B}{\Gamma \vdash (A \lambda_x) b} \text{ if ab}

(\vdash\text{-formation}) \quad \frac{\Gamma \vdash A \quad \Gamma (A \lambda_x) \vdash B}{\Gamma \vdash (A \Pi_x) B} \text{ if fc}
vc (variable condition): \( x \notin \Gamma \) and \( \tau(\Gamma, A) \to_{\beta\Pi} S \) for some \( S \)

ap (application condition): \( \tau(\Gamma, F) \equiv_{\beta\Pi} (A\Pi x)B \) and \( \tau(\Gamma, a) \equiv_{\beta\Pi} A \) for some \( A, B \).

ab (abstraction condition): \( \tau(\Gamma(A\lambda x), b) \equiv_{\beta\Pi} B \) and \( \tau(\Gamma, (A\Pi x)B) \to_{\beta\Pi} S \) for some \( S \).

fc (formation condition): \( \tau(\Gamma, A) \to_{\beta\Pi} S_1 \) and \( \tau(\Gamma(A\lambda x), B) \to_{\beta\Pi} S_2 \) for some rule \((S_1, S_2)\).
Properties of ⊢

Define $\overline{A}$ to be the $\beta\Pi$-normal form of $A$.

**Lemma**

If $\Gamma \vdash A$ then $\downarrow \tau(\Gamma, A)$ and $\Gamma \vdash_\beta A : \tau(\Gamma, A)$

**Lemma**

*(Subject Reduction for $\vdash$ and $\tau$)*

$\Gamma \vdash A \land A \rightarrow_\beta\Pi A' \Rightarrow [\Gamma \vdash A' \land \tau(\Gamma, A) =_\beta\Pi \tau(\Gamma, A')]$

**Theorem**

*(Strong Normalisation for $\vdash$)*

If $A$ is $\Gamma^\vdash$-legal, then $SN_\rightarrow_\beta (A)$.

**Lemma**

$\Gamma \vdash_\beta A : B \iff \Gamma \vdash A$ and $\tau(\Gamma, A) =_\beta\Pi B$ and $B$ is $\vdash_\beta$-legal type.
From Frege’s low level functions to PTSs that capture strong normalisation

- Kamareddine and Wells 2017, has incorporated Frege’s low level of functions to create PTSs with intersection types which contain all the ordinary PTSs (including the $\beta$-cube given above and its extensions with parameters/Frege’s functions.
- The $f$-cube is the $\beta$-cube extended with finite set declarations in the form of ordinary mathematical notion of function.
- **Theorem:** If $\Gamma \vdash_f A : B$ then $A$ and $B$ are strongly normalising.
- **Theorem:** If a type free term of the $\lambda$-calculus $M$ is strongly Normalising then $M$ is typable in the $f$-cube.
- Urzyczyn proved $U = (\lambda r. h(r(\lambda f \lambda s. f s))(r(\lambda q. \lambda g. g q)))(\lambda o. o o o o)$ is untypable in $F_\omega$. Hence $U$ is untypable in any system of the cube.
- But $U$ is strongly normalising.
- Kamareddine and Wells 2017 prove that $U$ is typable in the $f$-cube: There are $\Gamma, A$ such that $\Gamma \vdash_f U : A$. 
A hierarchy of systems that classify important properties of CR, SN, SR.

Not only types are used to derive important properties and avoid paradoxes and non termination, but also types classify non termination.


