SUBJECT REDUCTION FOR PURE TYPE SYSTEMS

Charlotte Marchal | 261031516 Dashiell Rich | 261002837
Milton Rosenbaum | 260972050 Zhaoshen Zhai | 261003108

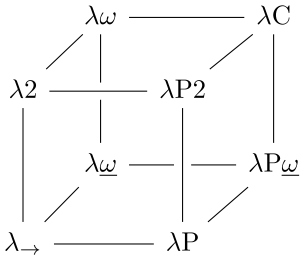
Abstract. Following [GN91] and [Bar91], we study the basics of *pure type systems*, which abstract many of the constructs found in the eight systems of the λ-*cube*. We start with a brief introduction to the systems of the λ-cube, discuss their expressive power, and introduce pure type systems as a unifying framework in which they can be studied. We then give a detailed proof of subject reduction for arbitrary pure type systems.

Subject reduction is a crucial property of a type system that guarantees its 'computational consistency' by ensuring that reductions of a well-typed expression remains well-typed, and also supports the slogan that 'well-typed programs do not go wrong'. It is thus desirable that it can be stated and proven uniformly across many different type systems, and this is the goal of the present note.

To this end, we follow [Bar91] and work within the framework of *pure type systems*, which is an abstraction of type systems based on the idea of 'dependencies' between terms and types. In the simply-typed λ-calculus $\lambda_{\to}$, terms depend on terms: $F\,x$ is a term depending on the term x, and we can abstract this dependency to a function $\lambda x{:}\alpha_1.(F\,x)$ of type $\alpha_1 \to \alpha_2$, where $F\,x{:}\alpha_2$. The other three ways that terms and types can be mutually dependent are present in other type systems, all extending $\lambda_{\to}$.

- In the polymorphic λ-calculus $\lambda 2$, the term $I_\alpha \coloneqq \lambda x{:}\alpha.x$ depends on the type α, and this abstracts to $\lambda\alpha{:}{*}.I_\alpha$ of type $\forall\alpha{:}{*}.(\alpha \to \alpha)$. Here, '$\alpha{:}{*}$' formalizes '$\alpha$ is a type' within $\lambda 2$.
- In the (weak) higher-order λ-calculus $\lambda\underline{\omega}$, the type $\alpha \to \alpha$ depends on the type α, and this abstracts to a *constructor* $\lambda\alpha{:}{*}.(\alpha \to \alpha)$ of *kind* $* \to *$. Similarly, in the λ-calculus $\lambda\mathrm{P}$ of dependent types, the type $\alpha_1^n \to \alpha_2$ depends on the term n, and abstracts to a constructor $\lambda n{:}\mathbb{N}.(\alpha_1^n \to \alpha_2)$ of kind $\mathbb{N} \to *$.

Imposing that the type systems that we care about all extend $\lambda_{\to}$, and hence terms can depend on terms, we obtain a total of $8 = 2^3$ type systems with all possible combinations of dependencies, called the λ-*cube*:



Type system	Dependencies			
$\lambda_{\to}$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\square,*)$		
$\lambda\underline{\omega}$	$(*,*)$		$(\square,\square)$	
$\lambda\omega = \lambda 2\underline{\omega}$	$(*,*)$	$(\square,*)$	$(\square,\square)$	
$\lambda\mathrm{P}$	$(*,*)$			$(*,\square)$
$\lambda\mathrm{P2}$	$(*,*)$	$(\square,*)$		$(*,\square)$
$\lambda\mathrm{P}\underline{\omega}$	$(*,*)$		$(\square,\square)$	$(*,\square)$
$\lambda\mathrm{C} = \lambda\mathrm{P}\omega$	$(*,*)$	$(\square,*)$	$(\square,\square)$	$(*,\square)$

The systems $\lambda\mathrm{P}\underline{\omega}$, $\lambda\mathrm{P2}$, and $\lambda\omega \coloneqq \lambda 2\underline{\omega}$ have three kinds of dependencies, while the strongest system of them all, the *calculus of constructions* $\lambda\mathrm{C} \coloneqq \lambda\mathrm{P}\omega$, have all four kinds of dependencies.

To explain the table on the right, we observe due to the mutual dependencies between terms and types, it is no longer natural to separate their definitions. Instead, we propose a uniform object, called a *pseudoterm*, and consider a judgement $M{:}N$ between pseudoterms M and N. These pseudoterms include terms, types, and kinds, so this begs the question: what is $*$? If $*{:}*$, then one might encounter a Russell-like paradox of the 'type of all types', so instead, we introduce a new symbol $\square$, the 'sort of all kinds', and assert that $*{:}\square$. Similarly, $(* \to *){:}\square$ and $(\mathbb{N} \to *){:}\square$, so sorts also capture the identity of constructors.

In the table, the notation (s_1, s_2) means that inhabitants of s_2 can depend on those of s_1, and moreover, that we can abstract over inhabitants of s_1 and output those of s_2. For instance, the '$(\square,*)$' in $\lambda 2$ indicate that terms (inhabitants of $*$) depend on types (inhabitants of $\square$), and that we can abstract over types and output terms (say, in $\lambda\alpha{:}{*}.I_\alpha$). Note that $(* \to *){:}\square$, so this allows for higher-order polymorphism as well.

Date: April 22, 2025.
Extended abstracted for the final project for Comp527: Logic and Computation taught by Professor Brigitte Pientka.
Link to presentation: **TODO**

Pure type systems. To delineate the hierarchy of terms, types, and kinds, one naturally starts abstractly and axiomatizes the notion of a type system.

Definition 1. A *pure type system* is a tuple $\sigma \coloneqq (\mathcal{C}, \mathcal{V}, \mathcal{S}, \mathcal{A}, \mathcal{R})$ consisting of a set $\mathcal{C}$ of *constants*, a set $\mathcal{V}$ of *variables*, a set $\mathcal{S} \subseteq \mathcal{C}$ of *sorts*, a set $\mathcal{A} \subseteq \mathcal{C}^2$ of *axioms*, and a set $\mathcal{R} \subseteq \mathcal{S}^3$ of *rules*.

Intuitively, *sorts* are universes imposing some sort (pun intended) of classification, *constants* are symbols living in a sort/constant as dictated by *axioms* (for instance, $0{:}\mathbb{N}$, $\mathbb{N}{:}*$, and $*{:}\Box$), and the *rules* restrict the formation of abstractions as motivated above. For the systems in the λ-cube, we write $(s_1, s_2) \coloneqq (s_1, s_2, s_2)$.

Notation 2. Throughout, let σ be denote an arbitrary pure type system.

TODO: link this with the λ-cube by interpreting $\lambda_{\rightarrow}$ as a PTS.

Definition 3. The collection of σ-*pseudoterms* is defined by $T \coloneqq \mathcal{V} \mid \mathcal{C} \mid (T\,T) \mid (\lambda\mathcal{V}{:}T.T) \mid (\Pi\mathcal{V}{:}T.T)$. Pairs $(A, B) \in T^2$ are called σ-*assignments*, written $A{:}B$, and a finite sequence thereof is called a σ-*pseudocontext*.

Definition 4. The β-*reduction* relation is the least relation on σ-terms satisfying the following for all σ-terms A, A', A'': the *principal reduction rule* $(\lambda x{:}A.A')A'' \to_\beta A'[A''/x]$, and the *congruence rules* $A\,A' \succ A\,A''$, $A'\,A \succ A''\,A$, $\lambda x{:}A.A' \succ \lambda x{:}A.A''$, $\lambda x{:}A'.A \succ \lambda x{:}A''.A$, $\Pi x{:}A.A' \succ \Pi x{:}A.A''$, and $\Pi x{:}A'.A \succ \Pi x{:}A''.A$.

Notation 5. We write $\twoheadrightarrow_\beta$ for the reflexive and transitive closure of $\to_\beta$, and $=_\beta$ for the equivalence relation generated by $\twoheadrightarrow_\beta$. A σ-term of the form $(\lambda x{:}A.A')A''$ is called a β-*redex*.

Definition 6. Let Γ be a σ-pseudocontext and let M, N be σ-pseudoterms. We say that Γ *proves* $M{:}N$, and write $\Gamma \vdash M{:}N$, if there is a finite well-founded tree $\mathcal{D}$, called a *derivation*, such that the following hold.

1. Vertices of $\mathcal{D}$ are of the form $\Delta \vdash A{:}B$, where A and B are σ-pseudoterms and Δ is a σ-pseudocontext.
2. The root of $\mathcal{D}$ is $\Gamma \vdash M{:}N$ and the leaves of $\mathcal{D}$ are instances of $\vdash c{:}c'$, where $(c, c') \in \mathcal{A}$.
3. Each interior vertex of $\mathcal{D}$ is a conclusion of an *inference rule*, whose successors are exactly the premises.

The inference rules of σ are as follows. Below, $s \in \mathcal{S}$, $x \in \mathcal{V} \setminus \operatorname{dom}\Gamma$, $(s_1, s_2, s_3) \in \mathcal{R}$, and $C =_\beta C'$.

$$\frac{\Gamma \vdash A{:}s}{\Gamma, x{:}A \vdash x{:}A}\ \textsc{Init} \qquad \frac{\Gamma \vdash A{:}s \quad \Gamma \vdash B{:}C}{\Gamma, x{:}A \vdash B{:}C}\ \textsc{Weak} \qquad \frac{\Gamma \vdash B{:}C \quad \Gamma \vdash C'{:}s}{\Gamma \vdash B{:}C'}\ \textsc{Conv} \qquad \frac{\Gamma \vdash B_1{:}s_1 \quad \Gamma, x{:}B_1 \vdash B_2{:}s_2}{\Gamma \vdash (\Pi x{:}B_1.B_2){:}s_3}\ \Pi\text{-}\textsc{rule}$$

$$\frac{\Gamma \vdash B_1{:}s_1 \quad \Gamma, x{:}B_1 \vdash B_2{:}s_2 \quad \Gamma, x{:}B_1 \vdash C{:}B_2}{\Gamma \vdash (\lambda x{:}B_1.C){:}(\Pi x{:}B_1.B_2)}\ \lambda\text{-}\textsc{rule} \qquad \frac{\Gamma \vdash B_1{:}(\Pi x{:}C_1.C_2) \quad \Gamma \vdash B_2{:}C_1}{\Gamma \vdash B_1\,B_2{:}C_2[B_2/x]}\ \textsc{App}$$

Definition 7. If $\Gamma \vdash A{:}B$, then Γ is a σ-*context* and A, B are σ-*terms*.

Lemma 8 (Substitution Lemma; [GN91, Lemma 17])**.** *Let Γ and $\Gamma_1, y{:}A, \Gamma_2$ be σ-contexts and let A, M, N, P be σ-terms. If $\Gamma_1, y{:}A, \Gamma_2 \vdash M{:}N$ and $\Gamma \vdash P{:}A$, then $(\Gamma_1, \Gamma_2)[P/y] \vdash M[P/y]{:}N[P/y]$.*

Lemma 9 (Stripping Lemma; [GN91, Lemma 19])**.** *Let Γ be a σ-context and let M, N, P be σ-terms.*

1. *If $\Gamma \vdash c{:}P$ where $c \in \mathcal{C}$, then $P =_\beta c'$ and $(c, c') \in \mathcal{A}$ for some $c' \in \mathcal{C}$.*
2. *If $\Gamma \vdash x{:}P$ where $x \in \mathcal{V}$, then $P =_\beta Q$ for some σ-term Q such that $(x{:}Q) \in \Gamma$.*
3. *If $\Gamma \vdash (\Pi x{:}M.N){:}P$, then $\Gamma \vdash M{:}s_1$, $\Gamma, x{:}M \vdash N{:}s_2$, and $P =_\beta s_3$ for some $(s_1, s_2, s_3) \in \mathcal{R}$.*
4. *If $\Gamma \vdash (\lambda x{:}M.N){:}P$, then $\Gamma \vdash M{:}s_1$, $\Gamma, x{:}M \vdash Q{:}s_2$, $\Gamma, x{:}M \vdash N{:}Q$, $\Gamma \vdash P{:}s_3$, and $P =_\beta \Pi x{:}M.Q$ for some $(s_1, s_2, s_3) \in \mathcal{R}$ and σ-term Q.*
5. *If $\Gamma \vdash M\,N{:}P$, then $\Gamma \vdash M{:}(\Pi x{:}A.B)$, $\Gamma \vdash N{:}A$, and $P =_\beta B[N/x]$ for some σ-terms A and B.*

Theorem 10 (Subject Reduction; [GN91, Lemma 22])**.** *Let Γ, Γ' be σ-contexts and let M, M', N be σ-terms.*

1. *If $\Gamma \vdash M{:}N$ and $M \twoheadrightarrow_\beta M'$, then $\Gamma \vdash M'{:}N$.*
2. *If $\Gamma \vdash M{:}N$ and $\Gamma \twoheadrightarrow_\beta \Gamma'$, then $\Gamma' \vdash M{:}N$.*

Proof. We proceed by simultaneous induction on the derivation $\mathcal{D} : \Gamma \vdash M : N$ when $M \to_\beta M'$ and $\Gamma \to_\beta \Gamma'$; the general case follows by iteration. We first prove (1), and split into cases with similar proofs.

- If $\mathcal{D}$ ends with Init, then there is no redex in M. If $\mathcal{D}$ ends with Conv, then there are derivations $\mathcal{D}_1 : \Gamma \vdash M : N'$ and $\mathcal{D}_2 : \Gamma \vdash N' : s$ for some $s \in \mathcal{S}$ and some σ-term N' such that $N' =_\beta N$. By IH_1, we have $\Gamma \vdash M' : N'$, on which Conv with $\mathcal{D}_2$ gives $\Gamma \vdash M' : N$. The case when $\mathcal{D}$ ends with Weak is similar.
- If $\mathcal{D}$ ends with Π-rule, say with $M = \Pi x : B_1 . B_2$, then the Stripping Lemma furnish some $(s_1, s_2, s_3) \in \mathcal{R}$ and derivations $\mathcal{D}_1 : \Gamma \vdash B_1 : s_1$ and $\mathcal{D}_2 : \Gamma, x : B_1 \vdash B_1 : s_2$ such that $N =_\beta s_3$. By definition of $\to_\beta$, two cases occur: if there is a σ-term B_1' such that $B_1 \to_\beta B_1'$, then by IH_1 on $\mathcal{D}_1$, we have $\mathcal{D}_1' : \Gamma \vdash B_1' : s_1$. Moreover, IH_2 on $\mathcal{D}_2$ gives $\mathcal{D}_2' : \Gamma, x : B_1' \vdash B_2 : s_2$, so applying Π-rule on $\mathcal{D}_1'$ and $\mathcal{D}_2'$ gives $\Gamma \vdash (\Pi x : B_1' . B_2) : s_3$, on which Conv gives $\Gamma \vdash (\Pi x : B_1' . B_2) : N$. The second case when $B_2 \to_\beta B_2'$ for some σ-term B_2' is the same (in fact, easier). The case when $\mathcal{D}$ ends with λ-rule is similar (and again has two subcases).
- If $\mathcal{D}$ ends with App, say with $M = B_1\, B_2$, then reductions within either B_1 or B_2 are trivial. Thus, we can take $x \in \mathcal{V} \backslash \operatorname{dom} \Gamma$ such that $B_1 = \lambda x : A_1 . A_2$, and assume $M = (\lambda x : A_1 . A_2) B_2 \to_\beta A_2[B_2/x]$. The Stripping Lemma then furnish σ-terms C_1 and C_2 such that $N =_\beta C_2[B_2/x]$ and derivations $\mathcal{D}_1 : \Gamma \vdash (\lambda x : A_1 . A_2) : (\Pi x : C_1 . C_2)$ and $\mathcal{D}_2 : \Gamma \vdash B_2 : C_1$. Again, the Stripping Lemma applied to $\mathcal{D}_1$ then furnish $(s_1, s_2, s_3) \in \mathcal{R}$, a σ-term C_2' such that $\Pi x : C_1 . C_2 =_\beta \Pi x : A_1 . C_2'$, and derivations $\mathcal{E}_1 : \Gamma \vdash A_1 : s_1$, $\mathcal{E}_2 : \Gamma, x : A_1 \vdash C_2' : s_2$, and $\mathcal{E}_3 : \Gamma, x : A_1 \vdash A_2 : C_2'$. Observe that $A_1 =_\beta C_1$, so Conv on $\mathcal{D}_2$ and $\mathcal{E}_1$ gives $\mathcal{D}_0 : \Gamma \vdash B_2 : A_1$, and using the Substitution Lemma with $(\mathcal{D}_0, \mathcal{E}_2)$ and $(\mathcal{D}_0, \mathcal{E}_3)$ give $\mathcal{E}_2' : \Gamma \vdash C_2'[B_2/x] : s_2$ and $\mathcal{E}_3' : \Gamma \vdash A_2[B_2/x] : C_2'[B_2/x]$; note that $\Gamma[B_2/x] = \Gamma$ since $x \notin \operatorname{dom} \Gamma$. Finally, since $C_2 =_\beta C_2'$ and $N =_\beta C_2[B_2/x]$, applying Conv on $\mathcal{E}_2'$ and $\mathcal{E}_3'$ gives $\Gamma \vdash A_2[B_2/x] : N$.

For (2), if the last rule of $\mathcal{D}$ is either App, Conv, Π-rule, or λ-rule, then we are done by IH_2; indeed, Γ is unchanged for App and Conv, and in Π-rule and λ-rule, reductions take place within Γ. Suppose that the last rule of $\mathcal{D}$ is Init or Weak, so with the notation of Definition 6, $\Gamma = \Gamma_0, x : A$ for some σ-term A and $x \in \mathcal{V}$. If the reduction occurs within Γ_0, then we are done by IH_2. Otherwise, $A \to_\beta A'$ for some σ-term A.

- If $\mathcal{D}$ ends with Init, then $\Gamma_0 \vdash A : s$. Applying IH_1, we have $\Gamma_0 \vdash A' : s$, and hence $\Gamma_0, x : A' \vdash x : A'$ from Init. Since $A \to_\beta A'$, we see that $A =_\beta A'$, so $\Gamma_0, x : A' \vdash x : A$ by Conv, as desired.
- If $\mathcal{D}$ ends with Weak, then there are derivations $\mathcal{D}_1 : \Gamma_0 \vdash A : s$ and $\mathcal{D}_2 : \Gamma_0 \vdash B : C$. Applying IH_1, we obtain a derivation $\mathcal{D}_1' : \Gamma_0 \vdash A' : s$, and applying Weak on $\mathcal{D}_1'$ and $\mathcal{D}_2$ gives $\Gamma_0, x : A' \vdash B : C$. ■

References

[GN91] H. Geuvers and M. Nederhof, *Modular proof of strong normalization for the calculus of constructions*, Journal of Functional Programming **1** (1991), no. 2, 155-189.

[Bar91] H. Barendregt, *Introduction to Generalized Type Systems*, Journal of Functional Programming **1** (1991), no. 2, 125-154.