Fine Grained Type Theory: A fine-grained take on (im)predicativity and (non)propositionality

Thiago Felicissimo

Today we can separate most type systems in two main categories.

- **Impredicative and propositional:** Separates regular types from propositional types, which are treated impredicatively.
- **Predicative and non-propositional:** Mixes regular and propositional types, which are all treated predicatively.

Those two views are *a priori* incompatible. Indeed, a proof that is expressed in an impredicative system might break in a predicative one. Likewise, an object in a non-propositional system that mixes regular and propositional types tightly might also not be expressible in a system that separates them. Therefore, it would seem that any attempt to linking those worlds would be deemed to fail.

Nevertheless, in many cases a proof or mathematical development can be adapted in order to not use such features. Indeed, many non-propositional developments, which use the sum type both to represent the sum of types and the propositional disjunction, can be rewritten in a more fine-grained way, in which we make explicit if we are dealing with a sum or a disjunction.

Likewise, some occurrences of impredicativity are also not needed. For instance, when declaring the induction principle of the natural numbers, we usually state

$$(\Pi P : Nat \to Prop.P\ 0 \to (\Pi n : Nat.P\ n \to P\ (S\ n)) \to \Pi n : Nat.P\ n) : Prop\,,$$

and thus the induction principle is itself a proposition, making use of impredicativity. However, we can also avoid impredicativity and declare

$$(\Pi P : Nat \to Prop.P\ 0 \to (\Pi n : Nat.P\ n \to P\ (S\ n)) \to \Pi n : Nat.P\ n) : \forall Prop\,,$$

where $\forall Prop$ is a "higher" sort then $Prop$. Even thought we do not have the full power of the impredicativity, we are still able to prove many interesting properties of the natural numbers, such as Euclidian division, and we conjecture that most interesting properties fall into this category.

The goal of this document is presenting a system in which (im)predicativity and (non-)propositionality can be decomposed, giving rise to a theory whose proofs are valid in both predicative and impredicative, propositional and non-propositional systems.

1 Introduction

Definition 1.1 (Impredicativity) *Let $<_{\mathcal{A}}$ be the transitive closure of $\mathcal{A}$. We say that a PTS is impredicative if there is $(s_1, s_2, s_3) \in \mathcal{R}$ with $s_3 < s_1$ or $s_3 < s_2$.*

Definition 1.2 (Propositionality) *A PTS is propositional if it is endowed with a special subset of sorts $\mathcal{L} \subseteq \mathcal{S}$, which should be interpreted as the sorts of propositions. Similarly, a propositional sort morphism is one which maps propositional sorts to propositional ones, and non-propositional sorts to non-propositional ones.*

We first start by defining two PTSs to represent the two categories of type systems defined on the introduction.

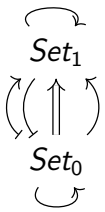
Definition 1.3 (Impredicative Propositional PTS) *We define the impredicative propositional PTS $\mathcal{I}$ by*

- $\mathcal{S}_{\mathcal{I}} = \{\mathit{Prop}, \mathit{Type}_0, \mathit{Type}_1\}$
- $\mathcal{L}_{\mathcal{I}} = \{\mathit{Prop}\}$
- $\mathcal{A}_{\mathcal{I}} = \{(\mathit{Prop}, \mathit{Type}_0), (\mathit{Type}_0, \mathit{Type}_1)\}$
- $\mathcal{R}_{\mathcal{I}} = \{(\mathit{Prop}, \mathit{Type}_i, \mathit{Type}_i), (\mathit{Type}_i, \mathit{Prop}, \mathit{Prop}) \mid i = 0, 1\} \cup \{(\mathit{Type}_i, \mathit{Type}_j, \mathit{Type}_{max(i,j)}) \mid i, j = 0, 1\}$

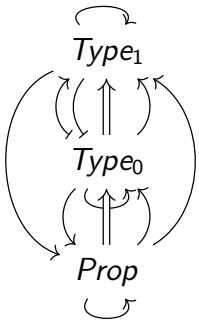
Definition 1.4 (Predicative Non-Propositional PTS) *We define the predicative non-propositional PTS $\mathcal{P}$ by*

- $\mathcal{S}_{\mathcal{P}} = \{\mathit{Set}_0, \mathit{Set}_1\}$
- $\mathcal{A}_{\mathcal{P}} = \{(\mathit{Set}_0, \mathit{Set}_1)\}$
- $\mathcal{R}_{\mathcal{P}} = \{(\mathit{Set}_i, \mathit{Set}_j, \mathit{Set}_{max(i,j)}) \mid i, j = 0, 1\}$

We represent such PTSs through the following diagrams.



PTS $\mathcal{P}$



PTS $\mathcal{I}$

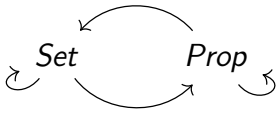
As expected, the PTS $\mathcal{I}$ has a dedicated sort of propositions, which are treated impredicatively, whereas normal types are treated predicatively. On the other hand, the PTS $\mathcal{P}$ is non-propositional, as there is no dedicated sort for propositions, and all types are treated predicatively.

We should interpret these PTSs as approximations of the type systems used by proof assistants. Most of them, such as Coq, HOL, Lean and Matita, use an extension of $\mathcal{I}$, which nevertheless preserves the impredicative and propositional main aspect. On the other hand, the proof assistant Agda extends the PTS $\mathcal{P}$ with an infinite number of universes.

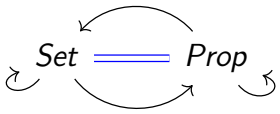
In the following, we explain step by step how these systems can be decomposed through a more fine-grained one. We will define a PTS satisfying the property that each proof or mathematical construction can be propositionaly embedded into both $\mathcal{I}$ and $\mathcal{P}$. More precisely, there should be two sort morphisms from our system to both $\mathcal{I}, \mathcal{P}$ — which should be propositional in the case of $\mathcal{I}$ — which we will also craft step by step. In the following, we can these morphisms $\pi_{\mathcal{I}}$ and $\pi_{\mathcal{P}}$ respectively.

The first problem we tackle is non-propositionality itself. Indeed, in systems like Agda, MLTT and the PTS $\mathcal{P}$, we have $\mathbb{N} : \mathit{Set}_0$ and $\bot : \mathit{Set}_0$, meaning that the sort of regular types and propositional types are identified. However, in order to be able to translate our constructions into propositional

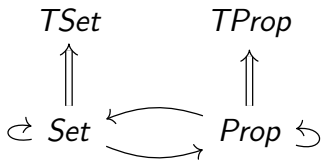
systems as well, it is clear that we need to separate these types. Therefore, we start with a system with two sorts *Set* and *Prop*.



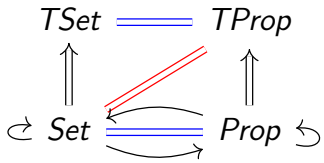
Any construction made in this system can be translated into $\mathcal{I}$ and $\mathcal{P}$. Indeed, with $\mathcal{I}$ we do $\pi_{\mathcal{I}}(\mathit{Set}) = \mathit{Type}_0$ and $\pi_{\mathcal{I}}(\mathit{Prop}) = \mathit{Prop}$, while with $\mathcal{P}$ we identity both sorts by setting $\pi_{\mathcal{P}}(\mathit{Set}) = \pi_{\mathcal{P}}(\mathit{Prop}) = \mathit{Set}_0$. We can then verify easily that $\pi_{\mathcal{P}}$ and $\pi_{\mathcal{I}}$ are sort morphisms. We represent the fact that the *Set* and *Prop* are identified in $\mathcal{P}$ by the blue equal sign, on the following diagram.



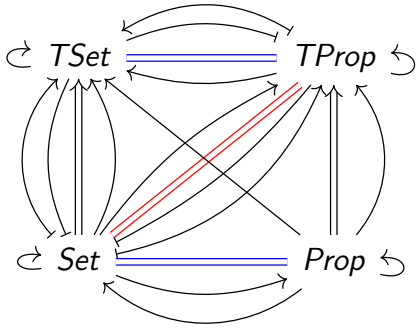
However, this system is still to weak. Indeed, remember that in order to declare a type in a given sort s, this sort needs to be well-sorted, meaning that there is some s' with $(s, s') \in \mathcal{A}$. Therefore, in order to be able to declare types in both *Set* and *Prop*, we add two sorts *TProp* and *TSet*, which play the role of top sorts on *Set* and *Prop*.



We can then recover $\mathcal{P}$ by identifying the sorts horizontally, whereas $\mathcal{I}$ is obtained by identifying sorts of the main diagonal, as shown in the diagram. More precisely, we set $\pi_{\mathcal{P}}(\mathit{TSet}) = \pi_{\mathcal{P}}(\mathit{TProp}) = \mathit{Set}_1$, whereas $\pi_{\mathcal{I}}(\mathit{TProp}) = \mathit{Type}_0$ and $\pi_{\mathcal{I}}(\mathit{TSet}) = \mathit{Type}_1$. We represent this through the following diagram, in which the red equal sign represent that the sorts are identified by $\pi_{\mathcal{I}}$.



In the current system we can now for instance declare $\mathit{Nat} : \mathit{Set}$, $0 : \mathit{Nat}$ and $S : \mathit{Nat} \to \mathit{Nat}$, but also $\bot : \mathit{Prop}$. However, our system is still weak, as we do not have the products in order to declare for instance $\vee : \mathit{Prop} \to \mathit{Prop} \to \mathit{Prop}$, $=: \Pi A : \mathit{Set}.A \to A \to \mathit{Prop}$, or even $\mathit{List} : \mathit{Set} \to \mathit{Set}$. Fortunately, we can add new products to this system, while still preserving the fact that $\pi_{\mathcal{I}}$ and $\pi_{\mathcal{I}}$ are sort morphisms. By adding as many products we can, while satisfying this property, we get the following PTS.



We can now declare for instance $\vee : \mathit{Prop} \to \mathit{Prop} \to \mathit{Prop}$ and $=: \Pi A : \mathit{Set}.A \to A \to \mathit{Prop}$, which allow us to express the propositions

$$\Pi n\ m : \mathit{Nat}.n =_{\mathit{Nat}} m \to S\ n =_{\mathit{Nat}} S\ m\,,$$

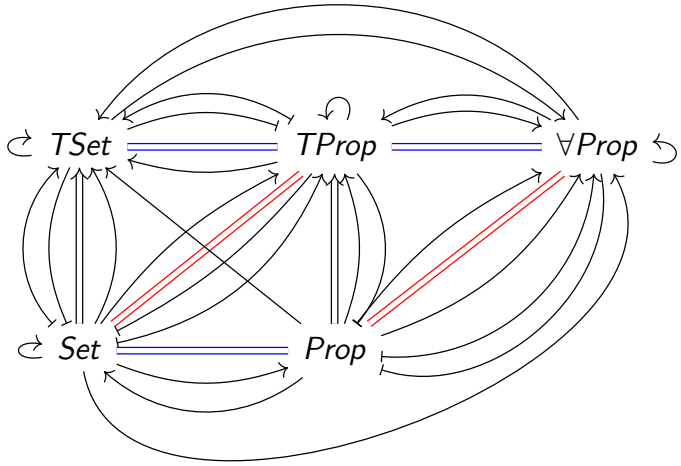
and

$$\Pi n : \mathit{Nat}.n =_{\mathit{Nat}} 0 \vee (n =_{\mathit{Nat}} 0 \to \bot)\,.$$

However, suppose that we wanted to declare the induction principle of Nat, which is essential to prove most interesting properties about the natural numbers. Unfortunately, as our system does not have a product of the type $(\mathit{TProp}, \mathit{Prop}, ?)$ we are completely unable to write any term that quantifies over a proposition. Worst, in the current setting, no product of this kind can be added to the system.

Indeed, if we added $(\mathit{TProp}, \mathit{Prop}, \mathit{Prop})$ or $(\mathit{TProp}, \mathit{Prop}, \mathit{Set})$, then for $\pi_{\mathcal{P}}$ to be a sort morphism we would have to have $(\mathit{Set}_1, \mathit{Set}_0, \mathit{Set}_0)$, which does not hold. Likewise, if we added $(\mathit{TProp}, \mathit{Prop}, \mathit{TProp})$ or $(\mathit{TProp}, \mathit{Prop}, \mathit{TSet})$, then for $\pi_{\mathcal{I}}$ to be a sort morphism we would have to have $(\mathit{Type}_0, \mathit{Prop}, \mathit{Type}_0)$ or $(\mathit{Type}_0, \mathit{Prop}, \mathit{Type}_1)$, which also do not hold.

Fortunately, this can be solved by adding a sort of higher-order propositions $\forall \mathit{Prop}$ to the system, which then allows us to have $(\mathit{TProp}, \mathit{Prop}, \forall \mathit{Prop})$. Then, in order to have the right products in the images of $\pi_{\mathcal{I}}$ and $\pi_{\mathcal{P}}$ we set $\pi_{\mathcal{I}}(\forall \mathit{Prop}) = \mathit{Prop}$ and $\pi_{\mathcal{P}}(\forall \mathit{Prop}) = \mathit{Set}_1$. Actually, we can add all missing products in order for the PTS to be functional, which gives the following system — in which the product $(\mathit{TSet}, \mathit{Prop}, \forall \mathit{Prop})$ is omitted, for readability purposes.



Now we can declare the induction principle of the natural numbers as

$$\Pi P : Nat \to Prop.P\ 0 \to (\Pi n : Nat.P\ n \to P\ (S\ n)) \to \Pi n : Nat.P\ n\,,$$

which has type $\forall Prop$. Moreover, note that when translating this term using $\pi_{\mathcal{P}}$ and $\pi_{\mathcal{I}}$ we get exactly what we want in both systems : in $\mathcal{I}$

$$\Pi P : Nat \to Prop.P\ 0 \to (\Pi n : Nat.P\ n \to P\ (S\ n)) \to \Pi n : Nat.P\ n$$

is of type *Prop*, making use of impredicativity, while in $\mathcal{P}$

$$\Pi P : Nat \to Set_0.P\ 0 \to (\Pi n : Nat.P\ n \to P\ (S\ n)) \to \Pi n : Nat.P\ n$$

is of type Set_1, which lives in a higher sort as required by predicativity.

2 Preliminary

2.1 PTS

Terms

$$A, B, M, N ::= x \in \mathcal{X} \mid c \in \mathcal{C} \mid s \in \mathcal{S} \mid MN \mid \lambda x : A.M \mid \Pi x : A.B$$

Conversion Context closure of the following

$$(\lambda x : A.M)N \hookrightarrow_\beta M(N/x)$$

Typing rules

Context forming rules

$$\frac{}{-;- \text{ well-formed}}\ \text{Empty} \qquad \frac{\Sigma;- \vdash A : s \qquad c \notin \Sigma}{\Sigma, c : A;- \text{ well-formed}}\ \text{Decl-sig} \qquad \frac{\Sigma;\Gamma \vdash A : s \qquad x \notin \Gamma}{\Sigma;\Gamma, x : A \text{ well-formed}}\ \text{Decl-ctx}$$

Conversion rule

$$\frac{\Sigma;\Gamma \vdash M : A \qquad \Sigma;\Gamma \vdash B : s \qquad A \equiv B}{\Sigma;\Gamma \vdash M : B}\ \text{Conv}$$

Regular type/term forming rules

$$\frac{\Sigma;\Gamma \text{ well-formed} \qquad (s_1, s_2) \in \mathcal{A}}{\Sigma;\Gamma \vdash s_1 : s_2}\ \text{Sort}$$

$$\frac{\Sigma;\Gamma \vdash A : s_1 \qquad \Sigma;\Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Sigma;\Gamma \vdash \Pi x : A.B : s_3}\ \text{Prod}$$

$$\frac{\Sigma;\Gamma \text{ well-formed} \qquad x : A \in \Gamma}{\Sigma;\Gamma \vdash x : A}\ \text{Var}$$

$$\frac{\Sigma;\Gamma \text{ well-formed} \qquad c : A \in \Sigma}{\Sigma;\Gamma \vdash c : A}\ \text{Const}$$

$$\frac{\Sigma;\Gamma \vdash A : s_1 \qquad \Sigma;\Gamma, x : A \vdash B : s_2 \qquad \Sigma;\Gamma, x : A \vdash M : B \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Sigma;\Gamma \vdash \lambda x : A.M : \Pi x : A.B}\ \text{Abs}$$

$$\frac{\Sigma;\Gamma \vdash M : \Pi x : A.B \qquad \Sigma;\Gamma \vdash N : A}{\Sigma;\Gamma \vdash MN : B(N/x)}\ \text{App}$$

2.2 Sort morphisms

Definition 2.1 (Sort morphism) *A sort morphism between two PTSs is a function between their underlying set of sorts compatible with the relations $\mathcal{A}$ and $\mathcal{R}$. More explicitly, a sort morphism between $P_1 = (\mathcal{S}_1, \mathcal{A}_1, \mathcal{R}_1)$ and $P_2 = (\mathcal{S}_2, \mathcal{A}_2, \mathcal{R}_2)$ is a function $f : \mathcal{S}_1 \to \mathcal{S}_2$ such that*

$$(s_1, s_2) \in \mathcal{A}_1 \Rightarrow (f(s_1), f(s_2)) \in \mathcal{A}_2$$
$$(s_1, s_2, s_3) \in \mathcal{R}_1 \Rightarrow (f(s_1), f(s_2), f(s_3)) \in \mathcal{R}_2\,.$$

PTSs and sort morphisms actually assemble into a category, which we call **PTS**. Explicitly,

- $Obj(\mathbf{PTS}) = \{(S, A, R) \mid S \text{ a set}, A \subseteq S \times S, R \subseteq S \times S \times S\}$
- $\mathbf{PTS}(P, Q) =$ sort morphisms from P to Q

Theorem 2.1 *The category **PTS** has products. Moreover, given the PTSs $P = (\mathcal{S}_P, \mathcal{A}_P, \mathcal{R}_P)$ and $Q = (\mathcal{S}_Q, \mathcal{A}_Q, \mathcal{R}_Q)$, the product $P \times Q$ is explicitly described by*

- $\mathcal{S}_{P\times Q} = \mathcal{S}_P \times \mathcal{S}_Q$
- $\mathcal{A}_{P\times Q} = \{((s_1, s_1'), (s_2, s_2')) \mid (s_1, s_2) \in \mathcal{A}_P, (s_1', s_2') \in \mathcal{A}_Q'\}$
- $\mathcal{R}_{P\times Q} = \{((s_1, s_1'), (s_2, s_2'), (s_3, s_3')) \mid (s_1, s_2, s_3) \in \mathcal{R}_P, (s_1', s_2', s_3') \in \mathcal{R}_Q\}$

Moreover, given $X \in Obj(\mathbf{PTS})$, $f_P \in \mathbf{PTS}(X, P)$ and $f_Q \in \mathbf{PTS}(X, Q)$, then the unique arrow $f_P \times f_Q : X \to P \times Q$ making the diagram commute is described explicitly by

$$f_P \times f_Q = s \mapsto (f_P(s), f_Q(s))\,.$$

Proof. The projections given by $\pi_P : (s, s') \mapsto s$ and $\pi_Q : (s, s') \mapsto s'$ are clearly sort morphisms. The function $s \mapsto (f_P(s), f_Q(s))$ defines a sort morphism, as

$$\begin{aligned}(s_1, s_2) \in \mathcal{A}_X &\Rightarrow (f_P(s_1), f_P(s_2)) \in \mathcal{A}_P \wedge (f_Q(s_1), f_Q(s_2)) \in \mathcal{A}_Q \\ &\Rightarrow ((f_P(s_1), f_Q(s_1)), (f_P(s_2), f_Q(s_2))) \in \mathcal{A}_{P\times Q}\end{aligned}$$

and

$$\begin{aligned}(s_1, s_2, s_3) \in \mathcal{R}_X &\Rightarrow (f_P(s_1), f_P(s_2), f_P(s_3)) \in \mathcal{R}_P \wedge (f_Q(s_1), f_Q(s_2), f_Q(s_3)) \in \mathcal{R}_Q \\ &\Rightarrow ((f_P(s_1), f_Q(s_1)), (f_P(s_2), f_Q(s_2)), (f_P(s_3), f_Q(s_3))) \in \mathcal{R}_{P\times Q}\end{aligned}$$

Finally, we clearly see that $f_P \times f_Q$ is the unique morphism satisfying $\pi_P \circ (f_P \times f_Q) = f_P$ and $\pi_Q \circ (f_P \times f_Q) = f_Q$. ∎

Corollary 2.1 *The product of two functional PTSs is a functional PTS.*

Proof. Follows directly from the characterization of the product. ∎

Definition 2.2 *Given a PTS P and a morphism f with domain P, we define the fiber contraction of f as the PTS obtained by identifying all the sorts in the same fiber. More formally, we have*

- $\mathcal{S} =$ *the fibers of f*
- $\mathcal{A} = \{([s_1], [s_2]) \mid \exists s_1' \in [s_1], s_2' \in [s_2] \; st \; (s_1', s_2') \in \mathcal{A}_P\}$

- $\mathcal{R} = \{([s_1],[s_2],[s_3]) \mid \exists s_1' \in [s_1], s_2' \in [s_2], s_3' \in [s_3]\ st\ (s_1', s_2', s_3') \in \mathcal{R}_P\}$

Note that the contraction is not really a quotient, as we do not ask for the relations $\mathcal{A}, \mathcal{R}$ be compatible with this relation. We write $fc(f)$ for the fiber contraction of f.

Lemma 2.1 *Let P, Q be PTSs and π_P, π_Q be the projections of $P \times Q$. We have $fc(\pi_P) = P$ iff*

$$(\mathcal{A}_P \neq \emptyset \Rightarrow \mathcal{A}_Q \neq \emptyset) \wedge (\mathcal{R}_P \neq \emptyset \Rightarrow \mathcal{R}_Q \neq \emptyset)\,.$$

Proof. $\Leftarrow$: We have of course $\mathcal{S}_{fc(\pi_P)} = \mathcal{S}_P$. Now suppose $(s_1, s_2) \in \mathcal{A}_P$. As $\mathcal{A}_P$ is nonempty then $\mathcal{A}_Q$ is also, and thus take any $(s_1', s_2') \in \mathcal{A}_Q$. We have $((s_1, s_1'), (s_2, s_2')) \in \mathcal{A}_{P\times Q}$ and thus $(s_1, s_2) \in \mathcal{A}_{fc(\pi_P)}$. However, we also obviously have $\mathcal{A}_{fc(\pi_P)} \subseteq \mathcal{A}_P$, and thus $\mathcal{A}_{fc(\pi_P)} = \mathcal{A}_P$. The same argument holds for $\mathcal{R}$.

$\Rightarrow$: If $\mathcal{A}_P \neq \emptyset$, then $\mathcal{A}_{fc(\pi_P)} \neq \emptyset$. But if we had $\mathcal{A}_Q = \emptyset$, then $\mathcal{A}_{P\times Q}$ would be empty, and so as $\mathcal{A}_{fc(\pi_P)}$. The same argument holds for $\mathcal{R}$. ■

@Gilles: The notion of compatibility of PTSs you proposed us to define turns of to be very weak. According to the previous lemma, two PTSs P, Q are compatible iff

$$(\mathcal{A}_P \neq \emptyset \iff \mathcal{A}_Q \neq \emptyset) \wedge (\mathcal{R}_P \neq \emptyset \iff \mathcal{R}_Q \neq \emptyset)\,.$$

Given a sort morphism $\mathbf{PTS}(P, Q)$ we extend it to terms and contexts in the most natural way.

Theorem 2.2 *Given two PTSs P, Q, a morphism $f \in \boldsymbol{PTS}(P, Q)$ and a judgment $\Sigma; \Gamma \vdash_P M : A$, then we have $[\![\Sigma]\!]_f; [\![\Gamma]\!]_f \vdash_Q [\![M]\!]_f : [\![A]\!]_f$ in Q.*

Proof. The intuition is the following: as the sort morphism is compatible with $\mathcal{A}$ and $\mathcal{R}$ then any occurrence of the rules Sort and Prod will still be valid. The proof is done by induction on the judgment tree. ■

3 A fine-grained take on $\mathcal{U}$

x proposes a universal theory $\mathcal{U}$ for expressing logics in which many systems can be expressed as fragments, such as predicate logic, CoC, HOL, etc. However, this theory does allow for the expression of predicative logic theories in an adequate way. Therefore, in this section we explore how the ideas seen until now can be used to extend the theory $\mathcal{U}$, in order to accommodate predicative logic systems.

As in $\mathcal{U}$, we start declaring

Appendix A References

References

[1] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, Jan. 1993.

Appendix B End-notes

You can click on the number to get back to where the end-note was made.