Appendices

Formal Semantics in Modern Type Theories,
First Edition. Stergios Chatzikyriakidis and Zhaohui Luo.

Appendix 1

Simple Type Theory $\mathcal{C}$

A1.1. Inference rules of $\mathcal{C}$

The following are inference rules of Church's simple type theory $\mathcal{C}$, the extensional core of Montague's IL, introduced and explained in section 1.3.1. The below inference rules were given in Luo and Soloviev (2017).[1]

Rules for contexts, base types and λ-calculus:

$$\frac{}{\langle\rangle\ valid} \qquad \frac{\Gamma \vdash A\ type \quad x \notin FV(\Gamma)}{\Gamma,\ x{:}A\ valid} \qquad \frac{\Gamma \vdash P : \mathbf{t}}{\Gamma,\ P\ true\ valid}$$

$$\frac{\Gamma\ valid}{\Gamma \vdash \mathbf{e}\ type} \qquad \frac{\Gamma\ valid}{\Gamma \vdash \mathbf{t}\ type} \qquad \frac{\Gamma,\ x{:}A,\ \Gamma'\ valid}{\Gamma,\ x{:}A,\ \Gamma' \vdash x : A} \qquad \frac{\Gamma,\ P\ true,\ \Gamma'\ valid}{\Gamma,\ P\ true,\ \Gamma' \vdash P\ true}$$

$$\frac{\Gamma \vdash A\ type \quad \Gamma \vdash B\ type}{\Gamma \vdash A \to B\ type} \qquad \frac{\Gamma, x{:}A \vdash b : B}{\Gamma \vdash \lambda x{:}A.b : A \to B} \qquad \frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B}$$

Rules for truth of logical formulae:

$$\frac{\Gamma \vdash P : \mathbf{t} \quad \Gamma \vdash Q : \mathbf{t}}{\Gamma \vdash P \Rightarrow Q : \mathbf{t}} \qquad \frac{\Gamma,\ P\ true \vdash Q\ true}{\Gamma \vdash P \Rightarrow Q\ true} \qquad \frac{\Gamma \vdash P \Rightarrow Q\ true \quad \Gamma \vdash P\ true}{\Gamma \vdash Q\ true}$$

$$\frac{\Gamma \vdash A\ type \quad \Gamma, x{:}A \vdash P : \mathbf{t}}{\Gamma \vdash \forall x{:}A.P : \mathbf{t}} \qquad \frac{\Gamma, x{:}A \vdash P\ true}{\Gamma \vdash \forall x{:}A.P\ true} \qquad \frac{\Gamma \vdash \forall x{:}A.P(x)\ true \quad \Gamma \vdash a : A}{\Gamma \vdash P(a)\ true}$$

1 The presentation of $\mathcal{C}$ in (Luo and Soloviev 2017) was based on (Luo 1990b) where, however, Church's simple type theory was called HOL and presented by means of a formulae-as-types formulation.

Conversion rules for formulae's truth:[2]

$$\frac{\Gamma \vdash P\ true \quad \Gamma \vdash Q : \mathbf{t}}{\Gamma \vdash Q\ true} \quad (P \simeq Q)$$

A1.2. Logical operators in $\mathcal{C}$

In Church's simple type theory $\mathcal{C}$, logical operators can be defined by means of $\Rightarrow$ and $\forall$ as follows, where in the definition of equality (the last line), a and b are of the same type A.

$$\begin{aligned}
true &= \forall X : \mathbf{t}.\ X \Rightarrow X \\
false &= \forall X : \mathbf{t}.\ X \\
P \wedge Q &= \forall X : \mathbf{t}.\ (P \Rightarrow Q \Rightarrow X) \Rightarrow X \\
P \vee Q &= \forall X : \mathbf{t}.\ (P \Rightarrow X) \Rightarrow (Q \Rightarrow X) \Rightarrow X \\
\neg P &= P \Rightarrow false \\
\exists x : A.P(x) &= \forall X : \mathbf{t}.\ (\forall x : A.(P(x) \Rightarrow X)) \Rightarrow X \\
(a = b) &= \forall P : A \to \mathbf{t}.\ P(a) \Rightarrow P(b)
\end{aligned}$$

2 See footnote 10 on p.8 for the explanation of the β-conversion relation $\simeq$.

Appendix 2

Type Constructors

A2.1. Π-types

$$(\Pi) \qquad \frac{\Gamma \vdash_\Delta A\ type \quad \Gamma,\ x{:}A \vdash_\Delta B\ type}{\Gamma \vdash_\Delta \Pi x{:}A.B\ type}$$

$$(Abs) \qquad \frac{\Gamma,\ x{:}A \vdash_\Delta b : B}{\Gamma \vdash_\Delta \lambda x{:}A.b : \Pi x{:}A.B}$$

$$(App) \qquad \frac{\Gamma \vdash_\Delta f : \Pi x{:}A.B \quad \Gamma \vdash_\Delta a : A}{\Gamma \vdash_\Delta f(a) : [a/x]B}$$

$$(\beta) \qquad \frac{\Gamma,\ x{:}A \vdash_\Delta b : B \quad \Gamma \vdash_\Delta a : A}{\Gamma \vdash_\Delta (\lambda x{:}A.b)(a) = [a/x]b : [a/x]B}$$

A2.2. Σ-types

$$(\Sigma) \qquad \frac{\Gamma \vdash_\Delta A\ type \quad \Gamma,\ x{:}A \vdash_\Delta B\ type}{\Gamma \vdash_\Delta \Sigma x{:}A.B\ type}$$

$$(Pair) \qquad \frac{\Gamma \vdash_\Delta a : A \quad \Gamma \vdash_\Delta b : [a/x]B \quad \Gamma, x{:}A \vdash_\Delta B\ type}{\Gamma \vdash_\Delta (a, b) : \Sigma x{:}A.B}$$

$$(Proj_1) \qquad \frac{\Gamma \vdash_\Delta p : \Sigma x{:}A.B}{\Gamma \vdash_\Delta \pi_1(p) : A}$$

Formal Semantics in Modern Type Theories,
First Edition. Stergios Chatzikyriakidis and Zhaohui Luo.

$$(Proj_2)\qquad \frac{\Gamma \vdash_\Delta p : \Sigma x{:}A.B}{\Gamma \vdash_\Delta \pi_2(p) : [\pi_1(p)/x]B}$$

$$(Conv_1)\qquad \frac{\Gamma \vdash_\Delta a : A \quad \Gamma \vdash_\Delta b : [a/x]B}{\Gamma \vdash_\Delta \pi_1(a,b) = a : A}$$

$$(Conv_2)\qquad \frac{\Gamma \vdash_\Delta a : A \quad \Gamma \vdash_\Delta b : [a/x]B}{\Gamma \vdash_\Delta \pi_2(a,b) = b : [a/x]B}$$

A2.3. Disjoint union types

$$(+)\qquad \frac{\Gamma \vdash_\Delta A\ type \quad \Gamma \vdash_\Delta B\ type}{\Gamma \vdash_\Delta A + B\ type}$$

$$(Inl)\qquad \frac{\Gamma \vdash_\Delta a : A \quad \Gamma \vdash_\Delta B\ type}{\Gamma \vdash_\Delta inl(a) : A + B}$$

$$(Inr)\qquad \frac{\Gamma \vdash_\Delta b : B \quad \Gamma \vdash_\Delta A\ type}{\Gamma \vdash_\Delta inr(b) : A + B}$$

$$(Case)\frac{\Gamma \vdash_\Delta c : A + B\ \Gamma, x{:}A \vdash_\Delta f(x) : C(inl(x))\ \Gamma, y{:}B \vdash_\Delta g(y) : C(inr(y))}{\Gamma \vdash_\Delta case(x.f(x),\ y.g(y),\ c) : C(c)}$$

$$(Case1)\frac{\Gamma \vdash_\Delta a : A\ \Gamma, x{:}A \vdash_\Delta f(x) : C(inl(x))\ \Gamma, y{:}B \vdash_\Delta g(y) : C(inr(y))}{\Gamma \vdash_\Delta case(x.f(x),\ y.g(y),\ inl(a)) = f(a) : C(inl(a))}$$

$$(Case2)\frac{\Gamma \vdash_\Delta b : B\ \Gamma, x{:}A \vdash_\Delta f(x) : C(inl(x))\ \Gamma, y{:}B \vdash_\Delta g(y) : C(inr(y))}{\Gamma \vdash_\Delta case(x.f(x),\ y.g(y),\ inr(b)) = g(b) : C(inr(b))}$$

A2.4. The unit type and finite types

The unit types $\mathbf{1}_w$ are governed by the following rules:

$$\frac{\vdash_\Delta \Gamma}{\Gamma \vdash_\Delta \mathbf{1}_w\ type}$$

$$\frac{\vdash_\Delta \Gamma}{\Gamma \vdash_\Delta w : \mathbf{1}_w}$$

$$\frac{\Gamma, z{:}\mathbf{1}_w \vdash_\Delta C(z)\ type \quad \Gamma \vdash_\Delta c : C(w) \quad \Gamma \vdash_\Delta z : \mathbf{1}_w}{\Gamma \vdash_\Delta \mathcal{E}_w(C,c,z) : C(z)}$$

$$\frac{\Gamma, z{:}\mathbf{1}_w \vdash_\Delta C(z)\ type \quad \Gamma \vdash_\Delta c : C(w)}{\Gamma \vdash_\Delta \mathcal{E}_w(C, c, w) = c : C(w)}$$

In the Coq code for homonymy in Appendix A7.2, `Onerun` is the unit type $\mathbf{1}_{run}$.

In general, one can inductively define finite types $Fin(n)$ indexed by $n : Nat$, where Nat is the inductive type of natural numbers. $Fin(n)$ consists of n objects with the following introduction rules (we omit their elimination and computation rules):

$$\frac{\Gamma \vdash_\Delta n : Nat}{\Gamma \vdash_\Delta zero(n) : Fin(n+1)} \qquad \frac{\Gamma \vdash_\Delta n : Nat \quad \Gamma \vdash_\Delta i : Fin(n)}{\Gamma \vdash_\Delta succ(n, i) : Fin(n+1)}$$

Abusing the language, one may adopt his/her own choice to denote the objects of a finite type. For instance, in example 2.2 in section 2.4, names such as $John$ are used to denote the objects of a_D that is $Fin(6)$, which may also be written as $\{John, Paul, George, Ringo, Brian, Bob\}$ (this is not a set-theoretical notation).

Appendix 3

Prop and Logical Operators in Impredicative MTTs

In impredicative MTTs like UTT (Luo 1994), there is the internal totality $Prop$ of all logical propositions and we can define all of the other logical operators by means of $\forall$.

A3.1. Prop

$$(Prop1) \qquad \frac{\vdash_\Delta \Gamma}{\Gamma \vdash_\Delta Prop\ type}$$

$$(Prop2) \qquad \frac{\Gamma \vdash_\Delta P : Prop}{\Gamma \vdash_\Delta P\ type}$$

$$(\forall) \qquad \frac{\Gamma \vdash_\Delta A\ type \quad \Gamma,\ x{:}A \vdash_\Delta P : Prop}{\Gamma \vdash_\Delta \forall x{:}A.P : Prop}$$

$$(Abs_P) \qquad \frac{\Gamma,\ x{:}A \vdash_\Delta b : P \quad \Gamma,\ x{:}A \vdash_\Delta P : Prop}{\Gamma \vdash_\Delta \lambda x{:}A.b : \forall x{:}A.P}$$

$$(App_P) \qquad \frac{\Gamma \vdash_\Delta f : \forall x{:}A.P \quad \Gamma \vdash_\Delta a : A}{\Gamma \vdash_\Delta f(a) : [a/x]P}$$

$$(\beta_P) \qquad \frac{\Gamma,\ x{:}A \vdash_\Delta b : P \quad \Gamma \vdash_\Delta a : A}{\Gamma \vdash_\Delta (\lambda x{:}A.b)(a) = [a/x]b : [a/x]P}$$

Formal Semantics in Modern Type Theories,
First Edition. Stergios Chatzikyriakidis and Zhaohui Luo.

A3.2. Logical operators

Here are the definitions of the logical operators by means of $\forall$ (a special case of Π) in an impredicative type theory (see, for example, section 5.1 of Luo (1994)).

$$
\begin{aligned}
P \Rightarrow Q &= \forall x : P.\, Q \\
\textbf{true} &= \forall X : Prop.\, X \Rightarrow X \\
\textbf{false} &= \forall X : Prop.\, X \\
P \wedge Q &= \forall X : Prop.\, (P \Rightarrow Q \Rightarrow X) \Rightarrow X \\
P \vee Q &= \forall X : Prop.\, (P \Rightarrow X) \Rightarrow (Q \Rightarrow X) \Rightarrow X \\
\neg P &= P \Rightarrow \textbf{false} \\
\exists x : A.P(x) &= \forall X : Prop.\, (\forall x : A.(P(x) \Rightarrow X)) \Rightarrow X \\
(a =_A b) &= \forall P : A \rightarrow Prop.\, P(a) \Rightarrow P(b)
\end{aligned}
$$

Appendix 4

And for Coordination

In Chatzikyriakidis and Luo (2012), we have only studied how to give generic typing to coordinating connectives such as "and", but not how to characterize their semantic behaviors. For example, as described in section 2.3.2, we have introduced the universe LTYPE with which "and" can be interpreted as $And : \Pi A{:}\text{LTYPE}.\ A \to A \to A$, but we have not described ways to define the semantic operator And.

This appendix is based on the second author's notes (Luo 2018b). It uses And as an example to describe how to define behaviors of coordinating connectives for each type in LTYPE. Formally, the introduction rules of LTYPE are given in Figure A4.1 – it contains the following three kinds of types as its objects:

1) the predicate types of the form $\Pi x_1{:}A_1 ... \Pi x_n{:}A_n.Prop$;

2) the types representing CNs (i.e. the types in CN);

3) the universe CN itself.

$$\frac{}{PType : Type} \qquad \frac{}{Prop : PType} \qquad \frac{A : \text{LTYPE} \quad P(x) : PType\ [x{:}A]}{\Pi x{:}A.P(x) : PType}$$

$$\frac{}{\text{LTYPE} : Type} \qquad \frac{}{\text{CN} : \text{LTYPE}} \qquad \frac{A : \text{CN}}{A : \text{LTYPE}} \qquad \frac{A : PType}{A : \text{LTYPE}}$$

Figure A4.1. *Introduction rules for* LTYPE

For each A : LTYPE, the semantic behavior of $And(A) : A \to A \to A$ is characterized by case analysis on A as follows:

1) If $A = \Pi x_1{:}A_1 ... \Pi x_n{:}A_n.Prop$ then, for any $f, g : A$,

$$And(A, f, g)(x_1, ..., x_n) =_{\text{df}} f(x_1, ..., x_n) \ \wedge\ g(x_1, ..., x_n) \ :\ Prop.$$

When $n = 0$, the above reduces to (2.36) in section 2.3.2.

Formal Semantics in Modern Type Theories,
First Edition. Stergios Chatzikyriakidis and Zhaohui Luo.

2) If $A : \mathsf{CN}$ then, for any distributive predicate $P : A \to Prop$,[1]

$$P(And(A, a_1, a_2)) =_{\text{df}} P(a_1) \ \wedge \ P(a_2).$$

3) If $A = \mathsf{CN}$, then for $A_1, A_2 : \mathsf{CN}$ and for $C : \mathsf{CN}$ such that $A_i \leq C$ $(i = 1, 2)$,

$$And(\mathsf{CN}, A_1, A_2) =_{\text{df}} \Sigma x{:}C.\ \text{IS}_C(A_1, x) \ \wedge \ \text{IS}_C(A_2, x),$$

where IS is the operator for propositional forms studied in section 3.2.3 and section 7.1.

Then, the examples (2.25-2.32) in section 2.3.2 are given semantics as follows:

(A4.1) (sentences)
⟦John walks and Mary talks⟧
$= And(Prop, walk(j), talk(m)) = walk(j) \wedge talk(m).$

(A4.2) (verbs)
⟦John walks and talks⟧
$= And(Human \to Prop, walk, talk)(j) = walk(j) \wedge talk(j).$

(A4.3) (adjectives)
⟦Mary is pretty and smart⟧
$= And(Human \to Prop, pretty, smart)(m) = pretty(m) \wedge smart(m).$

(A4.4) (adverbs)
⟦The plant died slowly and agonizingly⟧
$= And(T, slowly, agonizingly)(Plant, die, the\_plant)$
$= slowly(Plant, die, the\_plant) \wedge agonizingly(Plant, die, the\_plant),$
where $T = \Pi A{:}\mathsf{CN}.(A \to Prop) \to (A \to Prop)$.

(A4.5) (quantified NPs)
⟦Every student and some professors came⟧
$= And((Human \to Prop) \to Prop, \forall(Student), \exists(Professor))(come)$
$= \forall(Student, come) \wedge \exists(Professor, come).$

(A4.6) (quantifiers)
⟦Some but not all students got an A⟧
$= And(T, some, notAll)(Student, getA)$
$= \exists(Student, getA) \wedge \neg\forall(Student, getA),$
where $some$ and $notAll$ are of type $T = \Pi A{:}\mathsf{CN}.(A \to Prop) \to Prop$ and defined as $some(A, P) = \exists(A, P)$ and $notAll(A, P) = \neg\forall(A, P)$.

(A4.7) (proper names)
⟦John and Mary went to Italy⟧
$= goItaly(And(Human, j, m)) = goItaly(j) \wedge goItaly(m).$

1 Note that there is no such reductive equation for a verb whose semantics is a collective predicate, as illustrated by a sentence such as "Whitehead and Russell wrote *Principia Mathematica*".

(A4.8) (CNs)

$[\![$A friend and colleague came$]\!]$
$= \exists(And(\mathsf{CN}, Friend, Colleague), come)$
$= \exists(\Sigma x{:}Human.\textsc{is}(Friend, x) \wedge \textsc{is}(Colleague, x), come)$
$\Leftrightarrow \exists x{:}Human.\ \textsc{is}(Friend, x) \wedge \textsc{is}(Colleague, c) \wedge come(x).$

Appendix 5

Formal System LF_Δ

LF_Δ (Luo 2014, 2019a) extends the logical framework LF, in Chapter 9 of Luo (1994), by adding signatures to judgment forms and inference rules.

A5.1. LF_Δ

Judgements forms. LF_Δ has the following six forms of judgments:

– $\Delta\ valid$, which asserts that Δ is a valid signature.

– $\vdash_\Delta \Gamma$, which asserts that Γ is a valid context under Δ;

– $\Gamma \vdash_\Delta K\ kind$, which asserts that K is a kind in Γ under Δ;

– $\Gamma \vdash_\Delta k : K$, which asserts that k is an object of kind K in Γ under Δ;

– $\Gamma \vdash_\Delta K_1 = K_2$, which asserts that K_1 and K_2 are equal kinds in Γ under Δ;

– $\Gamma \vdash_\Delta k_1 = k_2 : K$, which asserts that k_1 and k_2 are equal objects of kind K in Γ under Δ.

Inference rules. The inference rules of LF_Δ are given below, where $\langle\rangle$ is the empty sequence and $dom(p_1 : K_1,\ ...,\ p_n : K_n) = \{p_1, ..., p_n\}$.[1]

1 For those who are familiar with LF, it may be easier to note that, besides the rules for signatures, all of the inference rules of LF i.e., those in Figures 9.1 and 9.2 of Chapter 9 of Luo (1994), become inference rules of LF_Δ after replacing $\vdash$ by $\vdash_\Delta$ (and changing the judgment form "$\Gamma\ valid$" to "$\vdash_\Delta \Gamma$").

Formal Semantics in Modern Type Theories,
First Edition. Stergios Chatzikyriakidis and Zhaohui Luo.

Signatures and contexts:[2]

$$\frac{}{\langle\rangle\ valid} \qquad \frac{\langle\rangle \vdash_\Delta K\ kind \quad c \notin dom(\Delta)}{\Delta,\ c : K\ valid} \qquad (*)\ \frac{\vdash_{\Delta, c:\ K, \Delta'} \Gamma}{\Gamma \vdash_{\Delta, c:\ K, \Delta'} c : K}$$

$$\frac{\Delta\ valid}{\vdash_\Delta \langle\rangle} \qquad \frac{\Gamma \vdash_\Delta K\ kind \quad x \notin dom(\Gamma)}{\vdash_\Delta \Gamma,\ x : K} \qquad (*)\ \frac{\vdash_\Delta \Gamma, x : K, \Gamma'}{\Gamma, x : K, \Gamma' \vdash_\Delta x : K}$$

Equality rules:

$$\frac{\Gamma \vdash_\Delta K\ \ kind}{\Gamma \vdash_\Delta K = K} \quad \frac{\Gamma \vdash_\Delta K = K'}{\Gamma \vdash_\Delta K' = K} \quad \frac{\Gamma \vdash_\Delta K = K' \ \Gamma \vdash_\Delta K' = K''}{\Gamma \vdash_\Delta K = K''}$$

$$\frac{\Gamma \vdash_\Delta k : K}{\Gamma \vdash_\Delta k = k : K} \quad \frac{\Gamma \vdash_\Delta k = k' : K}{\Gamma \vdash_\Delta k' = k : K} \quad \frac{\Gamma \vdash_\Delta k = k' : K \ \Gamma \vdash_\Delta k' = k'' : K}{\Gamma \vdash_\Delta k = k'' : K}$$

$$\frac{\Gamma \vdash_\Delta k : K \ \Gamma \vdash_\Delta K = K'}{\Gamma \vdash_\Delta k : K'} \quad \frac{\Gamma \vdash_\Delta k = k' : K \ \Gamma \vdash_\Delta K = K'}{\Gamma \vdash_\Delta k = k' : K'}$$

The kind type:

$$\frac{\Gamma\ valid}{\Gamma \vdash_\Delta Type\ \ kind} \quad \frac{\Gamma \vdash_\Delta A : Type}{\Gamma \vdash_\Delta El(A)\ \ kind} \quad \frac{\Gamma \vdash_\Delta A = B : Type}{\Gamma \vdash_\Delta El(A) = El(B)}$$

Dependent product kinds:

$$\frac{\Gamma \vdash_\Delta K\ \ kind \ \Gamma, x{:}K \vdash_\Delta K'\ \ kind}{\Gamma \vdash_\Delta (x{:}K)K'\ \ kind} \quad \frac{\Gamma \vdash_\Delta K_1 = K_2 \ \Gamma, x{:}K_1 \vdash_\Delta K_1' = K_2'}{\Gamma \vdash_\Delta (x{:}K_1)K_1' = (x{:}K_2)K_2'}$$

$$\frac{\Gamma, x{:}K \vdash_\Delta k : K'}{\Gamma \vdash_\Delta [x{:}K]k : (x{:}K)K'} \quad \frac{\Gamma \vdash_\Delta K_1 = K_2 \ \ \Gamma, x{:}K_1 \vdash_\Delta k_1 = k_2 : K}{\Gamma \vdash_\Delta [x{:}K_1]k_1 = [x{:}K_2]k_2 : (x{:}K_1)K}$$

$$\frac{\Gamma \vdash_\Delta f : (x{:}K)K' \ \Gamma \vdash_\Delta k : K}{\Gamma \vdash_\Delta f(k) : [k/x]K'} \quad \frac{\Gamma \vdash_\Delta f = f' : (x{:}K)K' \ \Gamma \vdash_\Delta k_1 = k_2 : K}{\Gamma \vdash_\Delta f(k_1) = f'(k_2) : [k_1/x]K'}$$

$$\frac{\Gamma, x{:}K \vdash_\Delta k' : K' \ \Gamma \vdash_\Delta k : K}{\Gamma \vdash_\Delta ([x{:}K]k')(k) = [k/x]k' : [k/x]K'} \quad \frac{\Gamma \vdash_\Delta f : (x{:}K)K' \ \ x \notin FV(f)}{\Gamma \vdash_\Delta [x{:}K]f(x) = f : (x{:}K)K'}$$

2 Note that the assumptions in a signature or in a context can be derived – this is characterized by the rules marked by $(*)$.

A5.2. Σ-types in $\mathbf{LF}_{\Delta}$

In LF_{Δ}, various type constructors can be specified by declaring constants as done in LF (Luo 1994). For instance, Σ-types can be specified in LF_{Δ} by declaring the following constants:

$$\begin{aligned}
\Sigma &: (A{:}Type)(B{:}(A)Type)Type \\
pair &: (A{:}Type)(B{:}(A)Type)(x{:}A)(y{:}B(x)\Sigma(A,B) \\
\pi_1 &: (A{:}Type)(B{:}(A)Type)(p{:}\Sigma(A,B))A \\
\pi_2 &: (A{:}Type)(B{:}(A)Type)(p{:}\Sigma(A,B))B(\pi_1(p))
\end{aligned}$$

with $\pi_1(A,B,pair(A,B,a,b)) = a$ and $\pi_2(A,B,pair(A,B,a,b)) = b$. Then, the rules in section 2.2.2 (or those in Appendix A2.2) are all derivable (note that, for pairs (a,b), we use $pair(a,b)$ in LF_{Δ}-notation). Detailed explanations are omitted.

Appendix 6

Rules for Dot-Types

In this appendix, we list the rules for dot-types as given in Luo (2009c, 2012b), except that we have added signatures in judgments. Note that dot-types are not ordinary inductive types and they are also governed by special rules that involve coercive subtyping.

Formation rule:

$$\frac{\vdash_\Delta \Gamma \quad \langle\rangle \vdash_\Delta A : Type \quad \langle\rangle \vdash_\Delta B : Type \quad \mathcal{C}(A) \cap \mathcal{C}(B) = \emptyset}{\Gamma \vdash_\Delta A \bullet B : Type}$$

Introduction rule:

$$\frac{\Gamma \vdash_\Delta a : A \quad \Gamma \vdash_\Delta b : B \quad \Gamma \vdash_\Delta A \bullet B : Type}{\Gamma \vdash_\Delta \langle a, b\rangle : A \bullet B}$$

Elimination rules:

$$\frac{\Gamma \vdash_\Delta c : A \bullet B}{\Gamma \vdash_\Delta p_1(c) : A}$$

$$\frac{\Gamma \vdash_\Delta c : A \bullet B}{\Gamma \vdash_\Delta p_2(c) : B}$$

Computation rules:

$$\frac{\Gamma \vdash_\Delta a : A \quad \Gamma \vdash_\Delta b : B \quad \Gamma \vdash_\Delta A \bullet B : Type}{\Gamma \vdash_\Delta p_1(\langle a, b\rangle) = a : A}$$

Formal Semantics in Modern Type Theories,
First Edition. Stergios Chatzikyriakidis and Zhaohui Luo.

$$\frac{\Gamma \vdash_\Delta a : A \quad \Gamma \vdash_\Delta b : B \quad \Gamma \vdash_\Delta A \bullet B : Type}{\Gamma \vdash_\Delta p_2(\langle a, b\rangle) = b : B}$$

Projections as coercions:

$$\frac{\Gamma \vdash_\Delta A \bullet B : Type}{\Gamma \vdash_\Delta A \bullet B \leq_{p_1} A : Type}$$

$$\frac{\Gamma \vdash_\Delta A \bullet B : Type}{\Gamma \vdash_\Delta A \bullet B \leq_{p_2} B : Type}$$

Coercion propagation:

$$\frac{\Gamma \vdash_\Delta A \bullet B : Type \quad \Gamma \vdash_\Delta A' \bullet B' : Type \quad \Gamma \vdash_\Delta A \leq_{c_1} A' : Type \quad \Gamma \vdash_\Delta B = B' : Type}{\Gamma \vdash_\Delta A \bullet B \leq_{d_1[c_1]} A' \bullet B' : Type}$$

$$\frac{\Gamma \vdash A \bullet B : Type \quad \Gamma \vdash_\Delta A' \bullet B' : Type \quad \Gamma \vdash_\Delta A = A' : Type \quad \Gamma \vdash_\Delta B \leq_{c_2} B' : Type}{\Gamma \vdash_\Delta A \bullet B \leq_{d_2[c_2]} A' \bullet B' : Type}$$

$$\frac{\Gamma \vdash_\Delta A \bullet B : Type \quad \Gamma \vdash_\Delta A' \bullet B' : Type \quad \Gamma \vdash_\Delta A \leq_{c_1} A' : Type \quad \Gamma \vdash_\Delta B \leq_{c_2} B' : Type}{\Gamma \vdash_\Delta A \bullet B \leq_{d[c_1,c_2]} A' \bullet B' : Type}$$

where the coercions d_1, d_2 and d are defined as: $d_1[c_1](x) = \langle c_1(p_1(x)), p_2(x)\rangle$, $d_2[c_2](x) = \langle p_1(x), c_2(p_2(x))\rangle$ and $d[c_1, c_2](x) = \langle c_1(p_1(x)), c_2(p_2(x))\rangle$.

Appendix 7

Coq Codes

A7.1. Some basic ontology and subtyping declarations

```
Definition CN:= Set.
Parameters Mary Helen: Woman.
Parameters John George: Man.

(** Common Nouns as Types *)
Parameters Phy Info Institution  Woman Delegate : CN.
Parameters Man Object Surgeon Animal Human: CN.

(** Subtyping relations **)
Axiom sw: Surgeon -> Human. Coercion sw: Surgeon >-> Human.
Axiom dh: Delegate -> Human. Coercion dh: Delegate >-> Human.
Axiom mh: Man -> Human. Coercion mh: Man >-> Human.
Axiom wh: Woman -> Human. Coercion wh: Woman >-> Human.
Axiom ha: Human -> Animal. Coercion ha: Human>-> Animal.
Axiom ao: Animal -> Object. Coercion ao: Animal>-> Object.
Axiom bi: Bank -> Institution. Coercion bi: Bank >-> Institution.

(** Some quantifiers *)
Definition some:=
    fun A: CN=> fun P :A -> Prop=> exists x: A, P(x).
Definition all:=
    fun A: CN=> fun P: A -> Prop=> forall x: A, P(x).
Definition no:=
    fun A: CN=> fun P: A->Prop=> forall x: A, not(P(x)).
```

Formal Semantics in Modern Type Theories,
First Edition. Stergios Chatzikyriakidis and Zhaohui Luo.

A7.2. Simple homonymy by overloading in coercive subtyping

```
(* Simple Homonymy by Overloading in Coercive Subtyping *)
Set Implicit Arguments.

(* Unit type for "run" -- Onerun *)
Inductive Onerun: Set:= run.
Definition T1:= Human -> Prop.
Definition T2:= Human -> Institution -> Prop.
Parameter run1: T1.
Parameter run2: T2.
Definition r1 (r: Onerun): T1:= run1. Coercion r1: Onerun >-> T1.
Definition r2 (r: Onerun): T2:= run2. Coercion r2: Onerun >-> T2.

(*John runs quickly*)
Parameter quickly: forall (A: CN), (A -> Prop) -> (A -> Prop).
Definition john_runs_quickly:= quickly (run:T1) John.

(*John runs a bank*)
Definition john_runs_a_bank:= exists b: Bank, (run: T2) John b.
```

A7.3. Intersective and subsective adjectives

```
(* Coq's record types are Sigma-types *)
Parameter Irish: Human -> Prop.
Record Irishhuman: CN:= mkirishhuman
    {h:> Human; I: Irish h}.
Record Irishdelegate: CN:= mkirishdelegate
    {d:> Delegate; I2: Irish d}.

(*Skilful as a polymorphic type*)
Parameter skilful: forall A: CN, A -> Prop.
Record Skilfulman: CN:= mkskilfulman
{m:> Man; S: skilful Man m}.
Record Skilfulhuman : CN:= mkskilfulhuman
    {h2:> Human; S2: skilful Human h2}.
Parameter walk: Human -> Prop.

Theorem delegate1:
    (some Irishdelegate) walk -> (some Delegate) walk.
     cbv. firstorder. Qed.
```

```
Theorem delegate2:
    (some Man) Irish -> (some Human) Irish.
     cbv.  firstorder. Qed.

Theorem skill1:
    (some Skilfulman) Irish -> (some Man) Irish.
     cbv. firstorder. Qed.

Theorem skill2:
    (some Surgeon) (skilful Surgeon) -> (some Man)( skilful Man).
     cbv. firstorder. Abort all.
```

A7.4. Privative adjectives

```
(*Dealing with fake*)
Definition CN:= Set.
Parameter G_R: CN.
Parameter G_F: CN.
Definition G:= (sum G_R G_F).
Definition fake_g (x: G):=
match x with| inl _ => False | inr _ => True end.
Fixpoint real_g  (x: G):=
match x with| inl _ => True | inr _ => False  end.
Parameter fake real: forall A: CN, A -> Prop.
Section FAKE.
Variable FAKE: fake G = fake_g.
Variable REAL: real G = real_g.
Record fakegun: CN:= mkfakegun
{g:> G; g1: fake G g}.
Record realgun: CN:= mkrealgun
{r:> G; r1: real G r}.

(*That gun is either real or fake*)
Variable tg: G.
Theorem   either:
    real G  tg \/ fake G  tg.
    replace (real(G)) with real_g. replace (fake G) with fake_g.
    cbv. destruct tg. left.  trivial. right. trivial.   Qed.
```

A7.5. Multidimensional adjectives

```
(*Dealing with multidimensional adjectives Health as an inductive
type where  the dimensions are enumerated. This is just an
enumerated type*)
Definition Degree:= Set.
Inductive Health: Degree:= Heart|Blood|Cholesterol.
Parameter Healthy: Health -> Human -> Prop.
Definition sick:=fun y: Human => ~ (forall x : Health,
          Healthy x y).
Definition healthy:= fun y: Human => forall x: Health,
          Healthy x y.

Theorem HEALTHY:
    healthy John -> Healthy Heart John /\ Healthy Blood John
    /\ Healthy Cholesterol John.
    cbv. intros. split. apply H.
    split. apply H. apply H. Qed.

Theorem HEALTHY2:
    healthy John -> not (sick John).
    cbv. firstorder. Qed.

Theorem HEALTHY3:
    (exists x: Health, Healthy x John) -> healthy John.
     cbv. firstorder. Abort.

Theorem HEALTHY4:
    (exists x: Health, not (Healthy x John)) -> healthy John.
     cbv. firstorder. Abort.

Theorem HEALTHY5:
    (exists x: Health, not (Healthy x John))  -> sick John.
     cbv. firstorder.  Qed.

(*Multidimensional noun Artist*)
Inductive Health: Degree:= a1|a2|a3.
Parameter  DIM_CN : forall  D: Degree,   Human ->  D  -> Prop.
Record Artist: Set:= mkartist
    {h:> Human; EI:  forall a: art,
     (DIM_CN h a)}.
```

A7.6. Gradable adjectives

```
(*Degree is type of names of degrees
-- d: Degree corresponds to type D(d)*)
(*So, Degree is a Tarski universe!*)
(*Here is an example with three degrees*)
Require Import Omega.
Inductive Degree: Set:= HEIGHT | AGE | IDIOCY.
Definition D (d: Degree):= nat.
Definition Height:= D(HEIGHT).
Definition Age:= D(AGE).
Definition Idiocy:= D(IDIOCY).

(*Universe CN_G of indexed CNs*)
Definition CN_G (_:Degree):= Set.
Parameter Human: CN_G(HEIGHT).
Parameter John Mary Kim: Human.
Parameter height: Human -> Height.

(*Type of physical objects indexed with a degree*)
Parameter PHY : forall d: Degree, CN_G(d).

(*ADJ(D,A) of syntax of adjectives whose domain is A: CN_G(d)*)
Parameter ADJ: forall d: Degree, CN_G(d)->Set.
Parameter TALL SHORT: ADJ HEIGHT Human.
Parameter IDIOTIC: ADJ IDIOCY Human.
Parameter ENORMOUS: forall d: Degree,  ADJ d  (PHY(d)).

(*Standard function*)
Parameter STND: forall d: Degree, forall A:CN_G(d),
       ADJ d A -> D(d).

(*semantics of tall, taller_than*)
Definition tall (h:Human):= ge (height h) (STND HEIGHT
       Human TALL).
Definition taller_than (h1:Human) (h2:Human) :=
    gt (height h2) (height h1).

(*Some simple theorems*)
Theorem TALLER:
    taller_than Mary John /\ height Mary =
    170 -> gt (height John) 170.
    cbv. intro. omega. Qed.
```

```
Theorem trans:
    taller_than Mary John /\ taller_than Kim Mary ->
    taller_than Kim John.
    cbv. intro. omega. Qed.

(*Definition for idiot, enormous and enormous idiot*)
Parameter IHuman: Idiocy -> CN_G(IDIOCY).
Definition Idiot:=  sigT(fun x: Idiocy=> prod (IHuman x)
    (ge x (STND IDIOCY Human IDIOTIC))).
Definition enormous  (d: Degree)(A: CN_G(d))(d1: D d):=
    fun P: A =>  ge (d1)  (STND  d (PHY(d))(ENORMOUS d)).
Record enormousidiot: Set:= mkeidiot
    {h:> Idiot; EI: enormous IDIOCY
    (IHuman(projT1(h)))(projT1(h))(projT1(projT2(h)))
    /\  ge (STND  IDIOCY (PHY(IDIOCY))(ENORMOUS IDIOCY))
    (STND IDIOCY Human IDIOTIC)}.

(*From enormous idiot it follows that there exists an idiot
such that their standard of idiocy is higher or equal than the
standard for enormous idiots*)
Theorem  ENORMOUS1:
    enormousidiot -> exists H: Idiot,
    projT1(H) >= STND IDIOCY (PHY IDIOCY) (ENORMOUS IDIOCY).
    cbv. firstorder. Qed.

(*From enormous idiot it follows that there exists an idiot such
that their standard of idiocy is higher or equal to both
the standard for enormous idiots and the standard for idiotic
humans*)
Theorem ENORMOUS2:
    enormousidiot -> exists H: Idiot,
    projT1(H) >= STND IDIOCY (PHY IDIOCY) (ENORMOUS IDIOCY)
    /\ projT1(H) >=  (STND IDIOCY Human IDIOTIC).
    cbv. firstorder. unfold Idiot in h0. exists h0. firstorder.
    unfold enormous in H. firstorder. elim  h0. intros. destruct
    p. omega. Qed.

(*From enormous idiot it follows that there exists an idiot such
that their standard of idiocy is higher or equal to the standard
for enormous idiots and  idiotic humans and also the standard for
enormous idiots is higher than that for idiotic humans*)
Theorem ENORMOUS3:
    enormousidiot -> exists H: Idiot,
```

```
    projT1(H) >= STND IDIOCY (PHY IDIOCY) (ENORMOUS IDIOCY)
    /\ projT1(H) >= (STND IDIOCY Human IDIOTIC)
    /\  STND IDIOCY (PHY IDIOCY) (ENORMOUS IDIOCY)
    >= (STND IDIOCY Human IDIOTIC).
    cbv. firstorder. unfold Idiot in h0. exists h0. firstorder.
    unfold enormous in H. firstorder. elim  h0. intros.
    destruct p.
    omega. Qed.
```

A7.7. Veridical adverbs

```
Definition  VER_PROP (Q: Prop):= forall P: Prop, P -> Q.
Parameter walk: Human -> Prop.
Definition fortunately:= VER_PROP.

Theorem VER1:
    fortunately (walk John) -> walk John.
    cbv. firstorder. apply H with (fortunately (walk John)).
    cbv. firstorder. Qed.

Set Implicit Arguments.
Definition  VER_VP (A: CN)  (Q: A -> Prop)(x: A):=
    forall P: A -> Prop,  P x -> Q x.
Definition quickly:= VER_VP.

Theorem VER2:
    (quickly walk) John -> walk John.
    cbv. intro. apply H with (quickly walk).
    cbv. firstorder. Qed.

Theorem VER3:
    forall Q: Prop, VER_PROP Q -> Q.
cbv. firstorder. apply H with (VER_PRO Q).
cbv. assumption. Qed.

Theorem VER4 (P: Prop):
    forall Q: Prop, (P -> Q) -> (P ->  VER_PROP Q).
    cbv. firstorder. Qed.
```

A7.8. Manner adverbs

```
Parameter Manner: Set.
```

```
Parameter Agent: CN.
Inductive Evt_m: Manner -> Type:=  EVT_m : forall m: Manner,
   Evt_m m.
Inductive Evt_ma: Manner -> Agent -> Type:=  EVT_ma
    : forall m: Manner, forall a: Agent,  Evt_ma m a.
Parameter illegible_m: forall m: Manner, EVT_m m -> Prop.
Parameter illegible_ma:
    forall m: Manner, forall a: Agent, EVT_ma m a -> Prop.

Unset Implicit Arguments.
Parameter write: forall m: Manner,  Human -> Evt_m m -> Prop.
Set Implicit Arguments.
Definition illegibly_m:= fun (m: Manner)(A: CN)
    (P: A -> Evt_m m -> Prop) (x: A) (E: Evt_m m) =>
    P x E /\ illegible_m m.
Parameter m: Manner.
Parameter u: Evt_m m.

Theorem MANNER:
    exists m: Manner,  exists e: Evt_m m,
    illegibly_m (write m) John e -> illegible_m (m).
    exists m. exists u. cbv. firstorder. Qed.

Definition illegibly_a:= fun (m: Manner)(a: Agent)(A: CN)
    (P: A -> Evt_ma m a -> Prop)  (x: A) (E: Evt_ma m a)
    => P x E /\ illegible_ma m a.
```

A7.9. Individuation

```
Require Import Setoid.
(*DomCN is the old CN, the universe of common nouns*)
Definition DomCN:=Set.

(*Equiv is an equivalence relation on elements of DomCN*)
Record Equiv (A:DomCN): Type:= mkEquiv
    {eq1:> A -> A -> Prop; _eq2: reflexive A(eq1) /\
    symmetric A eq1 /\ transitive A eq1}.
Record Equiv_K (A:DomCN): Type:= mkEquiv_K
    {eq2:> A -> A -> Prop; _eq3: reflexive A(eq2) /\
    symmetric A eq2}.

(*CN is a setoid, here expressed as a record,
second projection is the equiv relation on the type*)
```

```
Record CN:= mkCN
    {D:> DomCN; E:Equiv D}.
Record CN_K:= mkCN_K
    {D2:> DomCN; E2:Equiv_K D2}.

(*Types in DomCN*)
Parameter Human  Man: DomCN.
Parameter John: Human.
Axiom mh: Man->Human. Coercion mh: Man>->Human.

(*IC for Human*)
Parameter IC_Human: Equiv(Human).

(*This is the CN type in Capitals as a setoid*)
Definition HUMAN:= mkCN Human IC_Human.

(*Given the IC for man as being inherited from human*)
Definition AIC_Man:= fun x: Man=>fun y: Man=> IC_Human(x)(y).

(*We prove the equivalence of AIC_Man*)
Theorem EQ:
    reflexive  Man  AIC_Man
    /\  symmetric Man AIC_Man
    /\ transitive Man AIC_Man.
    cbv. destruct  IC_Human. destruct _eq4. destruct H0.
    split. intro. unfold reflexive in H.  apply H. split.
    intro. unfold symmetric in H0. intuition. intro.
    unfold transitive in H1. intro. intro. intros.
    apply H1 with y. assumption. assumption. Qed.

(*A first definition of three, prety standard but takes into
account the IC, B and B2 are the two respective projections*)
Definition three_0:= fun A: CN => fun P: A.(D)->Prop=>
    exists x: A.(D),P(x) /\ exists y: A.(D), P(y)
    /\  exists z: A.(D),  P(z) /\ not((A.(E))(x)(y))
    /\ not((A.(E))(y)(z)) /\ not((A.(E))(x)(z)).

(*Similarly for some*)
Definition some:=fun A: CN=>
    fun P: A.(D)->Prop=> exists x: A.(D), P(x).

(*Predicates are functions from the first CN projection to Prop*)
Parameter walk: HUMAN.(D) -> Prop.
```

```
(*Defining the IC for man and then MAN*)
Definition IC_Man:=  mkEquiv Man AIC_Man EQ.
Definition MAN:= mkCN Man IC_Man.

(*A proof that if three men walk, three humans walk*)
Theorem MANWALK:
    (three_0 MAN) walk-> (three_0 HUMAN) walk.
    cbv. intros. destruct H. destruct H. destruct H0.
    destruct H0. destruct H1. destruct H1. destruct H2.
    destruct H3. exists x. split. intuition. exists x0.
    split. intuition. exists x1. split. intuition. intuition.
    Qed.

(*Defining the dot type  PhyInfo and declaring the coercions*)
Parameters Phy: DomCN.
Parameter Info: DomCN.
Set Implicit Arguments.
Record PhyInfo: DomCN:= mkPhyInfo
    {p :> Phy; i :> Info}.
Parameter Book: DomCN.
Axiom bf: Book-> PhyInfo. Coercion bf: Book >-> PhyInfo.

(*IC for Phy, Info and PhyInfo*)
Parameter IC_Phy: Equiv(Phy).
Parameter IC_Info: Equiv(Info).
Definition PHY:= mkCN Phy IC_Phy.
Definition INFO:= mkCN Info IC_Info.
Definition AIC_PhyInfo := fun a1: (PhyInfo)=>fun b1: ( PhyInfo)=>
    IC_Phy(a1.(p))( b1.(p)) \/ IC_Info(a1.(i))( b1.(i)).

(*PhyInfo equiv is only reflexive and symmetric*)
Theorem EQ1:
    reflexive  PhyInfo  AIC_PhyInfo /\
    symmetric PhyInfo  AIC_PhyInfo.
    cbv. destruct IC_Phy. firstorder. destruct IC_Info.
    firstorder. Qed.

(*The _K partial equivalence is used for PhyInfo*)
Definition IC_PhyInfo:=  mkEquiv_K PhyInfo  AIC_PhyInfo EQ1.
Definition PHYINFO:= mkCN_K PhyInfo IC_PhyInfo.
```

```
(*Defining the IC criteria for book*)
Definition AIC_Book1:= fun x: Book=>fun y: Book=> IC_Phy(x)(y).
Definition AIC_Book2:= fun x: Book=>fun y: Book=> IC_Info(x)(y).

Theorem EQ2:
    reflexive  Book  AIC_Book1
    /\  symmetric Book AIC_Book1
    /\  transitive Book AIC_Book1.
    cbv. destruct  IC_Phy. destruct _eq4. destruct H0. split.
    intro. unfold reflexive in H.  apply H. split. intro.
    unfold symmetric in H0. intuition. intro. unfold transitive
    in H1. intro. intro. intros. apply H1 with y.
    assumption. assumption. Qed.

Theorem EQ3:
    reflexive  Book  AIC_Book2
    /\  symmetric Book AIC_Book2
    /\ transitive Book AIC_Book2.
    cbv. destruct  IC_Info. destruct _eq4. destruct H0.
    split. intro. unfold reflexive in H.  apply H. split. intro.
    unfold symmetric in H0. intuition. intro. unfold transitive
    in H1. intro. intro. intros. apply H1 with y. assumption.
    assumption. Qed.

Definition IC_Book1:=  mkEquiv Book  AIC_Book1 EQ2.
Definition IC_Book2:=  mkEquiv  Book  AIC_Book2 EQ3.
Definition BOOK1:= mkCN Book IC_Book1.
Definition BOOK2:= mkCN Book IC_Book2.

Parameter picked_up: Human -> Phy -> Prop.
Parameter mastered: Human -> Info -> Prop.

Definition three:= fun A: DomCN => fun P: (PhyInfo) -> Prop=>
    exists x y z: Book, not(IC_PhyInfo((x))((y)))
    /\ not((IC_PhyInfo)((y))((z)))
    /\ not((IC_PhyInfo)((x))((z)))
    /\ P x /\ P y /\ P z.
Unset Implicit Arguments.
Definition and:= fun A: DomCN=> fun P: A->Prop=>fun Q: A->Prop=>
    fun x: A=>  P(x) /\ Q(x).

(*Double distinctness case*)
Theorem copred_dd1:
```

```
    (three Book )(and PhyInfo(picked_up John)(mastered John)) ->
    (three_0 PHY) (picked_up John)
    /\ (three_0 INFO) (mastered John).
    cbv. intros. destruct IC_Phy. destruct IC_Info.
    firstorder. Qed.

(*Double distinctness with conclusion expanded from the start*)
Theorem copred_dd2:
        (three Book )(and PhyInfo(picked_up John )(mastered John))
        -> exists x y z: Book, not((PHY.(E))((x))((y)))
        /\ not((PHY.(E))((y))((z)))
        /\ not((PHY.(E))((x))((z)))
        /\ picked_up John (x)
        /\ picked_up John y
        /\ picked_up John z
        /\ not((INFO.(E))((x))((y)))
        /\ not((INFO.(E))((y))((z)))
        /\not((INFO.(E))((x))((z)))
        /\ mastered  John (x)
        /\ mastered John y
        /\ mastered John z.
        cbv. firstorder. Qed.

Parameter Heavy: Phy -> Prop.
Record Heavybook: DomCN:= mkheavybook
    {b1:> Book; b2: Heavy b1}.

Definition AIC_HBook1 := fun x: Heavybook=>fun y: Heavybook=>
    IC_Book1(x)(y).

Theorem EQ4:
    reflexive  Heavybook  AIC_HBook1
    /\ symmetric Heavybook AIC_HBook1
    /\ transitive Heavybook AIC_HBook1.
    cbv. destruct  IC_Phy. destruct _eq4. destruct H0. split.
    intro. unfold reflexive in H.  apply H. split. intro. unfold
    symmetric in H0.  intuition. intro. unfold transitive in H1.
    intro. intro. intros. apply H1 with y. assumption. assumption.
    Qed.

Definition IC_HBook1:=  mkEquiv Heavybook  AIC_HBook1 EQ4.
Definition HBOOK1:= mkCN Heavybook IC_HBook1.
```

```
Definition AIC_HBook2:=fun x: Heavybook=>fun y:Heavybook=>
    IC_Book2(x)(y).

Theorem EQ5:
    reflexive  Heavybook  AIC_HBook2
    /\ symmetric Heavybook AIC_HBook2
    /\ transitive Heavybook AIC_HBook2.
    cbv. destruct  IC_Info. destruct _eq4. destruct H0. split.
    intro. unfold reflexive in H.  apply H. split. intro.
    unfold symmetric in H0. intuition. intro. unfold transitive
    in H1. intro. intro. intros. apply H1 with y. assumption.
    assumption. Qed.

Definition IC_HBook2:=  mkEquiv Heavybook AIC_HBook2 EQ5.
Definition HBOOK2:= mkCN Heavybook IC_HBook2.

Theorem HEAVYBOOK:
    (some HBOOK2) ( mastered John)-> (some INFO)( mastered John).
    firstorder. Qed.

Theorem HEAVYBOOK1:
    (some HBOOK1) (picked_up John)-> (some PHY)( picked_up John).
    firstorder. Qed.

Theorem HEAVYBOOK2:
    (three_0 HBOOK1) (picked_up John)->
    (three_0 PHY)( picked_up John).
     cbv.  firstorder. Qed.

Theorem HEAVYBOOK3:
    (three_0 HBOOK2) ( mastered John)->
    (three_0 INFO)( mastered John).
    cbv. intros. firstorder. Qed.

(*More than one adjectives and individuation*)
Parameter Informative: Info->Prop.
Record Heavyinfobook: DomCN:= mkheavyinfobook
    {l4 :>Heavybook; _ : Informative l4}.
Definition AIC_HIBook1:= fun x:
    Heavyinfobook=> fun y:  Heavyinfobook=> IC_HBook1(x)(y).

Theorem EQ6:
    reflexive  Heavyinfobook  AIC_HIBook1 /\
```

```
    symmetric Heavyinfobook AIC_HIBook1 /\
    transitive Heavyinfobook AIC_HIBook1.
    cbv. destruct  IC_Phy. destruct _eq4. destruct H0. split.
    intro. unfold reflexive in H.  apply H. split. intro.
    unfold symmetric in H0. intuition. intro. unfold transitive
    in H1. intro. intro. intros. apply H1 with y. assumption.
    assumption. Qed.

Definition IC_HIBook1:=  mkEquiv Heavyinfobook  AIC_HIBook1 EQ6.
Definition HIBOOK1:= mkCN Heavyinfobook IC_HIBook1.

Definition AIC_HIBook2:=
    fun x: Heavyinfobook=>fun y: Heavyinfobook=> IC_HBook2(x)(y).

Theorem EQ7:
    reflexive  Heavyinfobook  AIC_HIBook2
    /\  symmetric Heavyinfobook AIC_HIBook2
    /\ transitive Heavyinfobook AIC_HIBook2.
    cbv. destruct  IC_Info. destruct _eq4. destruct H0. split.
    intro. unfold reflexive in H.  apply H. split. intro. unfold
    symmetric in H0. intuition. intro. unfold transitive in H1.
    intro. intro. intros. apply H1 with y. assumption. assumption.
    Qed.

Definition IC_HIBook2:=  mkEquiv Heavyinfobook  AIC_HIBook2 EQ7.
Definition HIBOOK2:= mkCN Heavyinfobook IC_HIBook2.

Theorem HEAVYIBOOK2:
    (three_0 HBOOK1) (picked_up John)->
    (three_0 PHY)( picked_up John).
    cbv. firstorder. Qed.

Theorem HEAVYIBOOK3:
    (three_0 HIBOOK2) ( mastered John)->
    (three_0 INFO)( mastered John).
    cbv. intros. firstorder. Qed.

Theorem HEAVYIBOOK4:
    (three_0 HIBOOK1) (picked_up John)->
    (three_0 HBOOK1)( picked_up John).
    cbv. firstorder. Qed.
```

```
Theorem HEAVYIBOOK5:
    (three_0 HIBOOK2) ( mastered John) ->
    (three_0 HBOOK2)( mastered John).
    cbv. intros. firstorder. Qed.
```