Chapter 9

Enumeration Types

Enumeration types are types for any finite sets: they are types each of which is inhabited by a finite number of terms. A particular enumeration type is defined from a given enumeration of n-number of constructors.

9.1 Syntax

As in the previous chapters, preterms, free variables, substitution, typing rules, and reduction rules are defined in this order.

9.1.1 Preterms

We extend the syntax of DTT given in Definition 182 as follows.

Definition 196 (Preterms)**.** A collection of *preterms* (notation Λ) under $(\mathcal{V}ar, \mathcal{C}on)$ is recursively defined by the following BNF grammar (where $x \in \mathcal{V}ar$ and $c \in \mathcal{C}on$):

$$\begin{array}{rcl} \Lambda & ::= & x \mid c \mid \mathsf{type} \mid \mathsf{kind} \\ & & \mid (x:\Lambda) \to \Lambda \mid \lambda x.\Lambda \mid \Lambda\Lambda \\ & & \mid (x:\Lambda) \times \Lambda \mid (\Lambda,\Lambda) \mid \pi_1(\Lambda) \mid \pi_2(\Lambda) \\ & & \mid \Lambda + \Lambda \mid \iota_1(\Lambda) \mid \iota_2(\Lambda) \mid \mathsf{unpack}^{\Lambda}_{\Lambda}(\Lambda,\Lambda) \\ & & \mid \{a_1,\ldots,a_n\} \mid a_1 \mid \ldots \mid a_n \mid \mathsf{case}^{\Lambda}_{\Lambda}(\Lambda,\ldots,\Lambda) \end{array}$$

$\{a_1,\ldots,a_n\}$ is the enumeration type and $a_1,\ldots,a_n$ are its constructors. $\mathsf{case}^P_M(N_1,\ldots,N_n)$ is the constructor for the elimination rule of the enumeration type. Three caveats should be made about this definition.

1. Note that, although $\{a_1,\ldots,a_n\}$ is the DTT counterpart of the extensional sets in set theory, the enumeration type is NOT a set. First, there are no operations such as union and intersection among the enumeration types, as defined in set theory.[*1] Second, there are not such things as subsets of an enumeration type. Third, the elements of the enumeration type do not allow permutations. In other words, $\{\}$ is not a constructor that constructs a new enumeration type using n-terms $M_1,\ldots,M_n$.

[*1] If these were possible, it would be also possible for a single preterm to belong to multiple enumeration types, but this is not desirable in view of type checking.

2. The symbols $a_1, \ldots, a_n$ in the above definition are meta-symbols. In each instance of DTT, an enumeration type is introduced by providing constructors corresponding to $a_1, \ldots, a_n$ concretely in each DTT system. More precisely, the enumeration type defined above is a class of types.
3. We may implement multiple enumeration types in one DTT system. In Section 9.4 and thereafter, we will see that the above classes enumeration types define prima-facie different types such as absurdity, truth, and finite entities. In that case, different enumeration types are assumed to have mutually disjoint sets of constructors. That is, the same constructor cannot belong to more than one enumeration type.

The free variable, substitution, and typing rules follow the rules described below.

9.1.2 Free variables

A set of free variables of a preterm is defined by the following set of rules.

Definition 197 (Free variables in enumeration types)**.** In addition to Definition 183,

$$\begin{aligned} fv(\{a_1, \ldots, a_n\}) &\stackrel{def}{\equiv} \varnothing \\ fv(a_i) &\stackrel{def}{\equiv} \varnothing \quad \textit{where } 1 \leq i \leq n. \\ fv(\mathsf{case}_M^P\,(N_1, \ldots, N_n)) &\stackrel{def}{\equiv} fv(M) \cup fv(P) \cup fv(N_1) \cup \cdots \cup fv(N_n) \end{aligned}$$

9.1.3 Substitution

Substitution is recursively defined as follows.

Definition 198 (Substitution)**.** In addition to Definition 184,

$$\begin{aligned} \{a_1, \ldots, a_n\}\,[L/x] &\stackrel{def}{\equiv} \{a_1, \ldots, a_n\} \\ a_i[L/x] &\stackrel{def}{\equiv} a_i \quad \textit{where } 1 \leq i \leq n. \\ \mathsf{case}_M^P\,(N_1, \ldots, N_n)\,[L/x] &\stackrel{def}{\equiv} \mathsf{case}_{M[L/x]}^{P[L/x]}\,(N_1[L/x], \ldots, N_n[L/x]) \end{aligned}$$

9.2 Typing Rules

The definition of signatures, contexts and judgments follow those of Section 6.3. The following axioms, structural rules and inference rules are added to those introduced in Section 8.2.

Definition 199 ({}-Formation Rule)**.**

$$\frac{}{\{a_1, \ldots, a_n\} : \mathsf{type}}\ (\{\}F)$$

The $(\{\}\, F)$ rule states that the enumeration type consisting of $a_1, \ldots, a_n$ is a type.

Definition 200 ($\{\}$-Introduction Rule)**.** For each i such that $1 \leq i \leq n$:

$$\frac{}{a_i : \{a_1, \ldots, a_n\}}\ (\{\}I)$$

The $(\{\})$ rule states that each a_i has type $\{a_1, \ldots, a_n\}$ and that there are n-constructors for the type $\{a_1, \ldots, a_n\}$.

Definition 201 ($\{\}$-Elimination Rule)**.**

$$\frac{M : \{a_1, \ldots, a_n\} \quad P : \{a_1, \ldots, a_n\} \to \mathsf{type} \quad N_1 : P(a_1) \quad \ldots \quad N_n : P(a_n)}{\mathrm{case}^P_M\,(N_1, \ldots, N_n) : P(M)}\ (\{\}E)$$

The $(\{\}\, E)$ rule states that if P is a proposition about preterms of the enumeration type and $P(a_i)$ holds for each a_i, then $P(M)$ holds for any term M of type $\{a_1, \ldots, a_n\}$. This means that terms of type $\{a_1, \ldots, a_n\}$ are exhausted by $a_1, \ldots, a_n$.

Exercise 202. Prove that Substitution Lemma I (Lemma 153) holds in this system.

9.3 Reduction

The β-reduction of this system obtains by adding the following rules to the system in the previous chapter.

Definition 203 (One-step β-reduction)**.**

Redex rule

$$\mathrm{case}^P_{a_i}\,(N_1, \ldots, N_n) \to_\beta N_i \quad \textit{where } 1 \leq i \leq n.$$

Structure rule

$$\frac{M \to_\beta M'}{\mathrm{case}^P_M\,(N_1, \ldots, N_n) \to_\beta \mathrm{case}^P_{M'}\,(N_1, \ldots, N_n)}$$

$$\frac{P \to_\beta P'}{\mathrm{case}^P_M\,(N_1, \ldots, N_n) \to_\beta \mathrm{case}^{P'}_M\,(N_1, \ldots, N_n)}$$

$$\frac{N_1 \to_\beta N_1'}{\mathrm{case}^P_M\,(N_1, \ldots, N_n) \to_\beta \mathrm{case}^P_M\,(N_1', \ldots, N_n)}$$

$$\vdots$$

$$\frac{N_n \to_\beta N_n'}{\mathrm{case}^P_M\,(N_1, \ldots, N_n) \to_\beta \mathrm{case}^P_M\,(N_1, \ldots, N_n')}$$

The redex rule is the case where M in the elimination rule is given as a_i. In this

case, since N_i is the proof of P for a_i, N_i can be used as it is.

There is no structural β-rule corresponding to $(\{\}\, F)$. The reason for this is that $\{a_1, \ldots, a_n\}$ has no subformula, namely, each a_i is not a subformula of $\{a_1, \ldots, a_n\}$. There is also no β-rule corresponding to $(\{\}\, I)$, which is also because there is no subformula for each a_i.

Exercise 204. Prove that Substitution Lemmas I–IV (Lemma 153–Lemma 159) hold in this system.

Exercise 205. Prove that Subject reduction theorem (Theorem **??**) holds in this system.

9.4 Bottom Type

The first example of enumeration types is the *bottom type*, the type representing absurdity. Absurdity is understood as a proposition which has no proof, which is straightforwardly represented as an enumeration type inhabited by no term, i.e. the enumeration type with $n = 0$.

Definition 206 ($\bot$ type)**.** $\quad \bot \stackrel{def}{\equiv} \{\}$

By instantiating Definition 199 and Definition 201 with $n = 0$, the following formation and elimination rules for $\bot$ are obtained.

Definition 207 ($\bot$-Formation Rule)**.**

$$\frac{}{\bot : \mathsf{type}}\ (\bot F)$$

$\bot$ has no introduction rule, since an enumeration type with n elements has n introduction rules, according to Definition 200.

Definition 208 ($\bot$-Elimination Rule)**.**

$$\frac{M : \bot \quad P : \bot \to \mathsf{type}}{\mathsf{case}^P_M\,() : P(M)}\ (\bot E)$$

Interestingly, the $(\bot E)$ rule is a dependent type version of (EFQ), the "Ex Falso Quodlibet," in the sense that (EFQ) is straightforwardly derived from $(\bot E)$.

Theorem 209 (Ex Falso Quodlibet)**.**

$$\frac{A : \mathsf{type} \quad M : \bot}{\mathsf{case}_M() : A}\ (EFQ)$$

Proof.

$$\dfrac{\dfrac{M : \bot \quad \dfrac{A : \mathsf{type} \quad \dfrac{}{x : \bot}\,1}{A : \mathsf{type}}\,(WK)}{\lambda x.A : \bot \to \mathsf{type}}\,(\to I),_1}{\dfrac{\mathsf{case}_M^{\lambda x.A}\,() : (\lambda x.A)M}{\mathsf{case}_M^{\lambda x.A}\,() : A}\,(CONV)}\,(\bot E)$$

□

This means that dependent type theory with enumeration types of arity 0 is an intuitionistic type theory. We may define the following alias (supressing A and M when they are obvious in the context) and use it for the subsequent sections.

Definition 210. $\mathsf{efq} \stackrel{def}{\equiv} \mathsf{case}_M^{\lambda x.A}\,()$

Exercise 211 (Easy)**.** Prove that $\bot$ is a unit element of (simple) disjoint union types, namely, for any A s.t. $\vdash A : \mathsf{type}$, the following inference holds.

$$\bot + A \dashv\vdash A \dashv\vdash A + \bot$$

By means of $\bot$, the negation of A (notation: $\neg A$) is defined as follows, in the same way as Definition 25.

Definition 212 (Negation)**.** $\neg A \stackrel{def}{\equiv} A \to \bot$

Exercise 213. Confirm that the judgments in Exercise 28 and Exercise 29 are also derivable in dependent type theory.

Example 214. An example that requires (EFQ) and negation is inferences known as *modus tollendo ponens*, namely, $A \vee B,\ \neg A \Rightarrow B$.

(32) Susan or Mary killed John, but (it turned out that) Mary didn't kill John. $\Longrightarrow$ Susan killed John.

This inference is schematized as $\mathbf{Ksj} + \mathbf{Kmj}, \neg\mathbf{Kmj} \vdash \mathbf{Ksj}$ *true*, the proof diagram of which is given below.

$$\dfrac{u : \mathbf{Ksj} + \mathbf{Kmj} \quad \dfrac{\dfrac{}{x : \mathbf{Ksj}}\,1 \quad \dfrac{\mathbf{Ksj} : \mathsf{type} \quad \vdots \quad \dfrac{\dfrac{}{x : \mathbf{Kmj}}\,1 \quad v : \neg\mathbf{Kmj}}{vx : \bot}\,(\to E)}{\mathsf{efq} : \mathbf{Ksj}}\,(EFQ)}{\mathsf{unpack}_u\,(\lambda x.x, \lambda x.\mathsf{efq}) : \mathbf{Ksj}}\,(+E),_1$$

Exercise 215. Prove the following, assuming $\vdash A : \mathsf{type}$ and $x : A \vdash B : \mathsf{type}$.

$$u : (x : A) \to B \vdash \neg\,(x : A) \times \neg B\ \mathit{true}$$
$$u : \neg((x : A) \times B) \vdash (x : A) \to \neg B\ \mathit{true}$$

9.5 Top Type

The second example of enumeration types is the *top type*, the type representing necessary truth. Necessary truth is understood as a proposition that there is always a proof

for it, regardless of contexts. This conception of truth is similated as an enumeration type with one element. In other words, the top type is a type which is defined to have a certain proof.

The particular constructor is prepared for this proof, (), called a *unit*, an empty pair, which is a proof term for the top type $\top$.

Definition 216 ($\top$ type)**.** $\top \stackrel{def}{\equiv} \{()\}$

According to Definition 199, $\top$-Formation Rule, Introduction Rules, and Elimination Rule derive.

Definition 217 ($\top$-Formation Rule)**.**

$$\frac{}{\top : \mathsf{type}} \, (\top F)$$

Definition 218 ($\top$-Introduction Rules)**.**

$$\frac{}{() : \top} \, (\top I)$$

Definition 219 ($\top$-Elimination Rule)**.**

$$\frac{M : \top \quad P : \top \to \mathsf{type} \quad N : P()}{\mathsf{case}^P_M(N) : P(M)} \, (\top E)$$

Exercise 220 (Easy)**.** Prove that $\top$ is a unit element of (simple) product types, namely, for any A s.t. $\vdash A : \mathsf{type}$, the following inference holds.

$$\top \times A \dashv\vdash A \dashv\vdash A \times \top$$

9.6 Entity Type

The third example of a enumeration type is the entity type, a type that represents the domain of finite entities, or *what exists*.

By *defining* the entity type as an enumeration type as in (33), it sets up a a simple and small situation containing only of John, Bill, Mary, and Susan as its elements, under which we may discuss meaning of sentences and inferences between them.

(33) $\mathsf{entity} \stackrel{def}{\equiv} \{john, bill, mary, susan\}$

As this simple use case shows, the main purpose of introducing entity type is not to *claim* what exists, but to provide a way to *assume* what exist. The question *what exist* is not a question of logic nor linguistics, but a question of ontology, one of the long stranding philosophical problems. The definition of entities via enumeration types allows us to discuss inferences regarding entities while leaving this ontological question open. Another advantage of this method is that the definition of entity

as an enumeration type provides information that there is no other entity, via its elimination rule, which is called *circumscription* in the field of artificial intelligence.

By assuming the entity type as (33), the formation, introduction and elimination rules of the entity type are instantiated from Definition 199 as follows.

Definition 221 (entity-Formation Rule)**.**

$$\frac{}{\mathsf{entity} : \mathsf{type}}\ (\mathbf{e}F)$$

Definition 222 (entity-Introduction Rules)**.**

$$\frac{}{\mathit{john} : \mathsf{entity}}\ (\mathbf{e}I)\quad \frac{}{\mathit{bill} : \mathsf{entity}}\ (\mathbf{e}I)\quad \frac{}{\mathit{mary} : \mathsf{entity}}\ (\mathbf{e}I)\quad \frac{}{\mathit{susan} : \mathsf{entity}}\ (\mathbf{e}I)$$

From the perspective of Curry=Howard correspondence, $\vdash \mathit{john} : \mathsf{entity}$, to take an instance, plays a double role: First, it declares that *john* is a term of type entity (or, John exists). This is a proof-theoretic counterpart of $[\![\mathit{john}]\!]_{M,g} \in D_M$ in model-theoretic semantics, namely, the denotation of *John* is an element of the domain of entities[*2]. Second, $\vdash \mathit{john} : \mathsf{entity}$ means that *john* is a *proof* of the proposition that there exists an entity.

Definition 223 (entity-Elimination Rule)**.**

$$\frac{M : \mathsf{entity} \quad P : \mathsf{entity} \to \mathsf{type} \quad N_j : P(\mathit{john}) \quad N_b : P(\mathit{bill}) \quad N_m : P(\mathit{mary}) \quad N_s : P(\mathit{susan})}{\mathsf{case}^P_M\,(N_j, N_b, N_m, N_s) : P(M)}\ (\mathbf{e}E)$$

The elimination rules states that, given a one-place predicate P so holds for every one of John, Bill, Mary and Susan, P holds for any entity.

Remark 224. One may consider the entity type consisting only of the above four elements is too simple as an assumption on what exists, thus unrealistic. Two remarks shall be made on this.

1. We may define an enumeration type with any number of constructors, if only they are finite.
2. Even with a small number of constructors, a toy setting of this kind is useful to check the validity of a given inference in a minimal setting. This corresponds to the situation in model-theoretic semantics that one may construct a model M whose domain D_M is defined as the set $\{\mathit{john}, \mathit{bill}, \mathit{mary}, \mathit{susan}\}$, which is unrealistic of the same degree, but provides a mean to verify or falsify a certain inference.

Exercise 225. What happens if we define entity as $\varnothing$ (the enumeration type with no element)? It is well known that first-order logic brings about problematic predictions if we define D_M, the domain of entity, as an empty set.

[*2] Names of the constructors, *john*, *bill*, *mary*, and *susan*, do not have to match the names that denote them: alternatively, for example, entity can be defined, in a more anonymous way, as the enumeration type $\{a_1, a_2, a_3, a_4\}$. In either cases, the name of a constructor and the sound/sign of a proper name are associated via lexical item. See Subsection **??**.

9.7 Discussions

9.7.1 Quantification over Finite Entities

For systems with entity type, universal quantification formulae in FoL can be expressed with the following type.

Definition 226 (Universal Quantification)**.** $\forall xA \stackrel{def}{\equiv} (x : \mathsf{entity}) \to A$

Exercise 227. Show that the following rules are derivable where $z \notin fv(\forall xA)$.

$$\frac{\forall xA : \mathsf{type} \quad \begin{array}{c}\overline{x : \mathsf{entity}}^{\,i}\\ \vdots \\ A\ true\end{array}}{A\ true}\ (\forall I),i \qquad \frac{\forall xA\ true \quad M : \mathsf{entity}}{A[M/x]\ true}\ (\forall E)$$

where x does not appear free in any undischarged assumptions in the diagram of A, and in $\forall xA$.

Therefore, the Π-types represent propositions that generalize the implication and universal quantification. In addition, the existential quantification formula of FoL can be expressed by the following type.

Definition 228 (Existential Quantification)**.** $\exists xA \stackrel{def}{\equiv} (x : \mathsf{entity}) \times A$

Exercise 229. Show that the following rules are derivable.

$$\frac{A[M/x]\ true \quad M : \mathsf{entity}}{\exists xA\ true}\ (\exists I) \qquad \frac{\exists xA\ true \quad \begin{array}{c}\overline{A\ true}^{\,i}\\ \vdots \\ B\ true\end{array}}{B\ true}\ (\exists E),i$$

where x does not appear free in any undischarged assumptions in the diagram of B except A, nor in B.

Therefore, the Σ-types represent a proposition that generalizes conjunctions and existential quantifications.

9.7.2 Verification of ∀ and ¬∃ propositions

Generally, ∃ and ¬∀ propositions can be verified by a single witness, while ∀ and ¬∃ propositions can be verified only *analytically*, if they quantify over possibly infinite domain.

When we assume that entity is a finite collection of entities, universal quantification over entity, which is generally not provable, becomes provable. Consider the truth of a universal sentence (34) whose semantic representation in DTS is (35).

(34) Every girl is smart.

(35) $\forall x(\mathbf{girl}(x) \to \mathbf{smart}(x))$

Under the entity type (36) and the signature (37).

(36) $\text{entity} \stackrel{def}{\equiv} \{mary, susan, john\}$

(37) $\sigma \stackrel{def}{\equiv} \begin{array}{rcl} \mathbf{girl} & : & \text{entity} \to \text{type}, \\ \mathbf{smart} & : & \text{entity} \to \text{type}, \\ m_g & : & \mathbf{girl}(mary), \\ s_g & : & \mathbf{girl}(susan), \\ j_g & : & \neg\mathbf{girl}(john), \\ m_s & : & \mathbf{smart}(mary), \\ s_s & : & \mathbf{smart}(susan), \\ j_s & : & \neg\mathbf{smart}(john) \end{array}$

Then, the following judgment holds,

$$\Gamma \vdash_\sigma \forall x(\mathbf{girl}(x) \to \mathbf{smart}(x))\ true$$

which is proved as follows.

$$\mathcal{D}_{mary} \equiv \dfrac{\dfrac{\overline{m_s : \mathbf{smart}(mary)}^{(VAR)} \quad \overline{y : \mathbf{girl}(mary)}^{2}}{m_s : \mathbf{smart}(mary)}^{(WK)}}{\lambda y.m_s : \mathbf{girl}(mary) \to \mathbf{smart}(mary)}^{(\to I),2}$$

$$\mathcal{D}_{susan} \equiv \dfrac{\dfrac{\overline{s_s : \mathbf{smart}(susan)}^{(VAR)} \quad \overline{y : \mathbf{girl}(susan)}^{2}}{s_s : \mathbf{smart}(susan)}^{(WK)}}{\lambda y.s_s : \mathbf{girl}(susan) \to \mathbf{smart}(susan)}^{(\to I),2}$$

$$\mathcal{D}_{john} \equiv \dfrac{\dfrac{\overline{x : \mathbf{girl}(john)}^{3} \quad \overline{g_s : \neg\mathbf{girl}(john)}^{(VAR)}}{g_s(x) : \bot}^{(\to I)} \quad \begin{array}{c}\vdots \\ \mathbf{smart}(john) : \text{type}\end{array}}{\text{case}_{g_s(x)}() : \mathbf{smart}(john)}^{(\bot E)}}{\lambda x.(\text{case}_{g_s(x)}()) : \mathbf{girl}(john) \to \mathbf{smart}(john)}^{(\to I),3}$$

$$\dfrac{\dfrac{\overline{x : \{mary, susan, john\}}^{1} \quad \mathcal{D}_{mary} \quad \mathcal{D}_{susan} \quad \mathcal{D}_{john}}{\text{case}_x\left(\lambda y.m_s, \lambda y.s_s, \lambda x.(\text{case}_{g_s(x)}())\right) : \mathbf{girl}(x) \to \mathbf{smart}(x)}^{(\{\}E),1}}{\lambda x.\text{case}_x\left(\lambda y.m_s, \lambda y.s_s, \lambda x.(\text{case}_{g_s(x)}())\right) : (x : \text{entity}) \to \mathbf{girl}(x) \to \mathbf{smart}(x)}^{(\Pi I),1}$$

Exercise 230. Confirm that (37) satisfies c.o.v.