Chapter 7

Σ-types

From the previously established *family of types* $\{B(x)\}_{x\in A}$, there frequently arises a necessity for a type that the *union* of each $B(x)$, namely $\bigcup_{x\in A} B(x)$. The Σ-type, formally introduced in this chapter, serves precisely this purpose, functioning as the *join* of such a type family.

While the definiton constitutes an extension of the type system detailed in the previous chapter, the methodology employed here is deliberately structured to provide a template for analogous extensions in subsequent chapters.

7.1 Σ-types (or Dependent product types)

The Σ type, also referred to as the *dependent product type*, constitutes an extension of the simple product type found in STLC. While STLC's product type takes the form $A \times B$, the Σ-type generalizes this by replacing the second component B with a type family $\{B(x)\}_{x\in A}$. Consequently, elements of a Σ-type are not arbitrary paris from $A \times \{B(x)\}_{x\in A}$; rather, they are constrained to be pairs (M, N) such that $M \in A$ and $N \in B(M)$.

In out standard notation, where A denotes the type of the bound variable x, and B represents a type that may or may not contain x as a free variable, the family of types is implicitly captured, and the Σ-type is rendered as $(x : A) \times B$.[*1]The following the preceding two chapters, we now adduce illustrative examples of the Σ-type within the context of List and Calender structures.

List

Consider the Σ-type $(n : Nat) \times \mathbf{List}_{Nat}(n)$. Its elements are ordered pairs consisting of a natural numbers n and lists of natural numbers of length n. Exemplars include $(0, [])$, $(1, [2])$, and $(1, [3])$. Crucially, this singule type comprehensively encompass lists of any arbitrary length (where elements are drawn from Nat).

The advantage of $(n : Nat) \times \mathbf{List}_{Nat}(n)$ lies in its capacity to encode list length information at the *type level*. Contrast this with the term P o the STLC product type $Nat \times \mathbf{List}_{Nat}$. While P is indeed a list, its length remains opaque without an explicit computation, such as $length(P)$. Consequently, the safety of operations like $\mathbf{head}(P)$ or $\mathbf{tail}(P)$ cannot be ascertained prior to the program execution.

[*1] The standard notational conventions for the Σ-type include $(\Sigma x : A)B(x)$ or $(\Sigma A)(\lambda x.B)$.

In stark contradistinction, for a term P of type $(n : Nat) \times \mathbf{List}_{Nat}(n)$, P is a pair and $\pi_2(P)$ is its list component. Crucially, we ascertain that this list itlsef is of type $\mathbf{List}_{Nat}(\pi_1(P))$. This means the length of the list, $\pi_1(P)$, is known at the type level. Such type-level knowledge enables static verification of the safty of $\mathbf{head}(P)$ and $\mathbf{tail}(P)$ through mere *type checking*, thereby obviating the need for runtime validation.

Calendar

Consider athe formal representation of a Gregorian date. This can precicely captured as a pair consisting of a term m of type **Month** and a term of type $\mathbf{Day}(m)$. Such a construction is a term of the Σ-type:

$$(m : \mathbf{Month}) \times \mathbf{Day}(m)$$

The utility of the Σ-type extends to scenarios requiring a disjoint union or a generalized union. Analogous to the **List** example discussed previously, the Σ-type $(m : \mathbf{Month}) \times \mathbf{Day}(m)$ can be intuitively understood as the union of all possible $\mathbf{Day}(m)$ types, for each specific month m. More abstractly, it serves to formalize constructions akin to $\bigcup_{m \in \mathbf{Month}} \mathbf{Day}(m)$, where the type of elements in the union depends on the index.

7.2 Syntax

Analogous to the treatment of Π-types, we proceed to formally introduce new syntactic constructs. This requires a systematic extension of our definition of DTT, that comprise of definitions of preterms, free variables, substitution, typing rules consisting of formation rule, introduction rule(s), and elimination rule(s), and reduction rules.

7.2.1 Preterms

To generalize the product types, one must replace the product type syntax $\Lambda \times \Lambda$ in the preterm definition with the *Sigma*-type syntax $(x : \Lambda) \times \Lambda$.

Definition 165 (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 $c \in \mathcal{C}on$):

$$\begin{aligned} \Lambda \quad ::= \quad & x \mid c \mid \mathsf{type} \mid \mathsf{kind} \\ & \mid (x : \Lambda) \to \Lambda \mid \lambda x.\Lambda \mid \Lambda\Lambda \\ & \mid (x : \Lambda) \times \Lambda \mid (\Lambda, \Lambda) \mid \pi_1(\Lambda) \mid \pi_2(\Lambda) \end{aligned}$$

The constructs $(x : \Lambda) \times \Lambda$ denotes a Σ-type. Its associated proof terms are as follows: the ordered pair (Λ, Λ) serves as the introduction proof term, and $\pi_1(\Lambda), \pi_2(\Lambda)$ function as the first and second *projections*, respectively, for its elimination rules.

7.2.2 Free variables

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

Definition 166 (Free variables in Σ-types)**.** In addition to Definition 130,

$$\begin{aligned} fv((x:A)\times B) &\overset{def}{\equiv} fv(A)\cup(fv(B)-\{x\}) \\ fv((M,N)) &\overset{def}{\equiv} fv(M)\cup fv(N) \\ fv(\pi_1(M)) &\overset{def}{\equiv} fv(M) \\ fv(\pi_2(M)) &\overset{def}{\equiv} fv(M) \end{aligned}$$

7.2.3 Substitution

Substitution is recursively defined as follows.

Definition 167 (Substitution for Σ-types)**.** In addition to Definition 132,

$$\begin{aligned} ((x:A)\times B)[L/x] &\overset{def}{\equiv} (x:A[L/x])\times B \\ ((y:A)\times B)[L/x] &\overset{def}{\equiv} (y:A[L/x])\times (B[L/x]) \quad \textit{where } x\notin fv(B) \textit{ or } y\notin fv(L). \\ ((M,N))[L/x] &\overset{def}{\equiv} (M[L/x],N[L/x]) \\ (\pi_1(M))[L/x] &\overset{def}{\equiv} \pi_1(M[L/x]) \\ (\pi_2(M))[L/x] &\overset{def}{\equiv} \pi_2(M[L/x]) \end{aligned}$$

Definition 168 (α-conversion rules)**.** In addition to Definition 133,

$$(x:A)\times B \equiv (y:A)\times B[y/x] \quad \textit{where } y\notin fv(A)\cup fv(B).$$

7.3 Typing Rules

7.3.1 Σ-types

The definitions of signatures, contexts and judgments are adopted verbatim from Section 6.3. To the axioms, structural rules, and inference rules established in Section 6.4, the following are appended.

Definition 169 (Σ-Formation Rule)**.**

$$\frac{\begin{array}{c}\overline{x:A}^{\,i}\\ \vdots \\ A:\mathsf{type} \quad B:\mathsf{type}\end{array}}{(x:A)\times B:\mathsf{type}}\,(\Sigma F),i$$

The (ΣF) rule, mirroring the (ΠF) rule presented in Definition 146, establishes the necessary condition for a preterm of Σ-type to be well-formed as a type. Specifically, for $(x:A)\times B$ to constitute a valid type, A must itself be a type, and B must also be

a type under the assumption that x is a variable of type A. Recalling the discussion in Subsection 6.7.1, this rule can also be viewed as a new introduction rule for the sort **type**, furnishing a method for constructing terms of **type**.

Definition 170 (Σ-Introduction Rule)**.**

$$\frac{M : A \quad N : B[M/x]}{(M, N) : (x : A) \times B}\ (\Sigma I)$$

The introduction rule for Σ-type, denoted (ΣI), simultaneously establishes the well-formedness condition for pairs and provides the verification condition for Σ-type. A canonical proof of $(x : A) \times B$ takes the form of (M, N), where M must be a term of type A and N must be a term of type $B[M/x]$. The latter condition merits further elucidation: When B contains x as a free variable, B represents a type family indexed by x. Consequently, the term N can only form a valid pair with M if N inhabits the type B with M substituted for x.

Definition 171 (Σ-Elimination Rule)**.**

$$\frac{M : (x : A) \times B}{\pi_1(M) : A}\ (\Sigma E) \qquad \frac{M : (x : A) \times B}{\pi_2(M) : B[\pi_1(M)/x]}\ (\Sigma E)$$

The elimination rule for Σ-type, denoted (ΣE), articulates their pragmatic or "use" conditions, concurrently stipulating the well-formedness of their projections. This rule enables the construction of proof terms for type A and type $B[\pi_1(M)/x]$, corresponding to the first projection operation π_1, and the second projection operation π_2, respectively.

Remark 172. As mentioned in Remark 104, Weakening emerges as a derived rule within DTT extended with Σ-types. Consequently, its explicit definition as an independent rule is rendered superfluous.

7.3.2 Conjunction

Conjunctions in Propositional logic and FoL can be represented by the following types.

Definition 173 (Conjunction)**.** $A \times B \stackrel{def}{\equiv} (x : A) \times B \quad where\ x \notin fv(B).$

Exercise 174. Show that the following rules are derivable when $x \notin fv(B)$ (Hint: Use Theorem 134).

$$\frac{M : A \quad N : B}{(M, N) : A \times B}\ (\times I) \qquad \frac{M : A \times B}{\pi_1(M) : A}\ (\times E) \qquad \frac{M : A \times B}{\pi_2(M) : B}\ (\times E)$$

Exercise 175. Prove the validity of the following inferences under the signature

$A : \mathsf{type},\ B : \mathsf{type},\ C : \mathsf{type}.$

$$A \times (B \times C) \dashv\vdash (A \times B) \times C$$
$$A \times A \dashv\vdash A$$
$$A \times B \ \vdash\ B \times A$$

Exercise 176 (Currying/Uncurrying)**.** Suppose that A, B, C are preterms, x, y, z are variables, and $\vdash A : \mathsf{type}$, $x : A \vdash B : \mathsf{type}$, and $x : A, y : B \vdash C : \mathsf{type}$ holds. Prove the following inferences.

$$(x : A) \to (y : B) \to C \vdash (z : (x : A) \times B) \to C[\pi_1 z/x][\pi_2 z/y]$$
$$(z : (x : A) \times B) \to C \vdash (x : A) \to (y : B) \to C[(x, y)/z]$$

7.4 Reduction

Following the methodology established in the preceding chapter, we commence by formally defining the one-step β-reduction as follows.

Definition 177 (One-step β-reduction for $\Lambda_{\Pi\Sigma}$)**.** In addition to the rules in Definition 151:

Redex rule

$$(\lambda x.M)N \to_\beta M[N/x]$$
$$\pi_1(M, N) \to_\beta M$$
$$\pi_2(M, N) \to_\beta N$$

Structure rules

$$\frac{A \to_\beta A'}{(x : A) \to B \to_\beta (x : A') \to B} \qquad \frac{B \to_\beta B'}{(x : A) \to B \to_\beta (x : A) \to B'}$$

$$\frac{M \to_\beta M'}{\lambda x.M \to_\beta \lambda x.M'} \qquad \frac{M \to_\beta M'}{MN \to_\beta M'N} \qquad \frac{N \to_\beta N'}{MN \to_\beta MN'}$$

$$\frac{A \to_\beta A'}{(x : A) \times B \to_\beta (x : A') \times B} \qquad \frac{B \to_\beta B'}{(x : A) \times B \to_\beta (x : A) \times B'}$$

$$\frac{M \to_\beta M'}{(M, N) \to_\beta (M', N)} \qquad \frac{N \to_\beta N'}{(M, N) \to_\beta (M, N')}$$

$$\frac{M \to_\beta M'}{\pi_1(M) \to_\beta \pi_1(M')} \qquad \frac{M \to_\beta M'}{\pi_2(M) \to_\beta \pi_2(M')}$$

Multi-step β-reduction, denoted as $\twoheadrightarrow_\beta$, is defined analogously to Definition 57. It share the fundamental properties with its counterpaert in the previous chapter, and

exhibits both reflexivity and transitivity. Furthermore, β-equivalence, denoted as $=_\beta$, when defined following Definition 61, constitutes an equivalence relation.

Exercise 178. Prove that Substitution Lemma I-IV (Lemma 153-Lemma 159) holds in this system.

7.5 Subject Reduction Theorem

As shown below, Subject reduction is also valid for this extention.

Theorem 179 (Subject Reduction for DTT)**.** For any context Γ, preterms M, M', A, if $\Gamma \vdash M : A$ and $M \to_\beta M'$, then $\Gamma \vdash M' : A$.

Proof. Case of $\underline{d = 0}$: M is a β-redex. The cases are divided below according to the form of $M \to_\beta M'$.
⊞ $\underline{\pi_1(M, N) \to_\beta M}$: Suppose that $\Gamma \vdash \pi_1(M, N) : A$ is proven by the following diagram.

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}_M & \mathcal{D}_N \\ M : A & N : B[M/x]\end{matrix}}{(M, N) : (x : A) \times B}\,(\Sigma I)}{\pi_1(M, N) : A}\,(\Sigma E)$$

Thus we have $\vdash M : A$ by $\mathcal{D}_M$.
Case of $\underline{d > 0}$: Suppose that Theorem 161 holds when $M \to_\beta M'$ is proven in less than d-steps (IH). Let us prove that Theorem 161 holds when $M \to_\beta M'$ is proved in d-steps.
⊞ $\underline{\text{Cases of } (x : A) \times B \to_\beta (x : A') \times B}$: Suppose that $(x : A) \times B$ has a type type, which is proved by the following proof diagram.

$$\dfrac{\begin{matrix}\mathcal{D}_A & \overline{x : A}^{\,i} \\ A : \mathsf{type} & \mathcal{D}_B \\ & B : \mathsf{type}\end{matrix}}{(x : A) \times B : \mathsf{type}}\,(\Pi F), i$$

and $(x : A) \to B \to_\beta (x : A') \to B$ is proved as follows.

$$\dfrac{\begin{matrix}\mathcal{D}_\beta \\ A \to_\beta A'\end{matrix}}{(x : A) \times B \to_\beta (x : A') \times B}$$

Thus, we obtain the following proof.

$$\dfrac{\begin{matrix}\mathcal{D}_A & \mathcal{D}_\beta \\ A : \mathsf{type} & A \to_\beta A'\end{matrix}}{A' : \mathsf{type}}\,(IH) \qquad \dfrac{\dfrac{\overline{x : A'}^{\,1}}{x : A}\,(CONV) \qquad \begin{matrix}\mathcal{D}_B \\ B : \mathsf{type}\end{matrix}}{(x : A') \times B : \mathsf{type}}\,(\Sigma F), 1$$

The cases of $\pi_2(M, N) \to_\beta N$, $(x : A) \times B \to_\beta (x : A) \times B'$, $(M, N) \to_\beta (M, N')$, and $(M, N) \to_\beta (M, N')$ are left to the readers as an exercise. □

Exercise 180. Complete the proof of Theorem 179.

7.6 Discussion

7.6.1 Π-types as Generalized Products

The nomenclature of Π-types and Σ-types, while suggestive, often engenders initial confusion among beginners. Conventionally, the symbol Π evokes the notion of a product, a natural analogue to the Cartesian product in set theory. Conversely, Σ typically signifies a sum or union. However, within the realm of type theory, the relationship to their STLC counterparts presents a seemingly counterintuitive duality: the Σ-type generalizes the STLC product type, while the Π-type generalizes the STLC function type. This apparent inversion warrants careful elucidation.

The Π-type, $(i : I) \to B(i)$, is precisely a generalized product type. To apprehend this, consider its set-theoretic antecedent: for an index set I (which may be finite or infinite, not merely binary), and a family of sets $\{B(i)\}_{i\in I}$ such that each $B(i)$ is a set whose elements are indexed by i, the generalized product is the set of all functions $f : I \to \bigcup_{i\in I} B(i)$ such that $f(i) \in B(i)$ for all $i \in I$. In this vein, the Π-type $(i : I) \to B(i)$ represents the product of the family of types $B(i)$, indexed by the type I.

To illustrate, consider a type $M(2)$ (as defined in Subsection 11.6.2) with precisely two elements, say 1 and 2. Then, the Π-type $(x : M(2)) \to B(x)$ is isomorphic to the familiar binary product $B(1) \times B(2)$.

Exercise 181. Prove the above.

In contrast, the Σ-type, $(i : I) \times B(i)$, is understood as a generalized disjoint union or sum. In set theory, for an index set I and a family of sets $B(i)_{i\in I}$, the disjoint union is the set of all pairs (i, y) where $i \in I$ and $y \in B(i)$. Similarly, the Σ-type $(i : I) \times B(i)$ can be regarded as the disjoint union of the family of types $B(x)$, indexed by the type I.

In essence, when the initial type serves as an index, Π-types and Σ-types function as generalized products and generalized sums, respectively. However, when the initial type is not conceived as an index, these types take on alternative interpretations: the Π-type signifies a fibred function type, representing functions whose codomain depends on their argument, while the Σ-type denotes a fibred product type, representing pairs where the type of the second component depends on the value of the first. This dual interpretation is fundamental to their versatility in dependent type theory.