TECHNICAL REPORT 94-5e

On Extensions of $F_{\leq}$: Second-Order Lambda-Calculus with Subtyping

Daisuke Suzuki

March 1, 1994

Department of Information Science
Faculty of Science, University of Tokyo

7–3–1 Hongo, Bunkyo-Ku Tokyo, 113 Japan

TITLE

On Extensions of $F_{\leq}$: Second-Order Lambda-Calculus with Subtyping

AUTHORS

Daisuke Suzuki

KEY WORDS AND PHRASES

$F_{\leq}$, parametric polymorphism, ad-hoc polymorphism, integration of two kinds of polymorphism

ABSTRACT

$F_{\leq}$ was an extension of a second-order λ-calculus F which has parametric polymorphism with subtyping and bounded quantification, introduced by Ghelli to apply second-order λ-calculi to the framework of object-oriented languages. However, it is impossible to know the amount of information of a type-checked term before evaluation since the subsumption rules are included in $F_{\leq}$. To overcome this problem, we provided new calculi $cF^{+}_{\leq}$ and $cF^{++}_{\leq}$ as extensions of $F_{\leq}$, and investigated their proof-theoretic properties. $cF^{+}_{\leq}$ includes a new type $\mathrm{Range}(X)$, and $cF^{++}_{\leq}$ includes two new types $\mathrm{Range}(X)$ and $\mathrm{Inst}(X,T)$. For example, the type $\mathrm{Range}(X)$ is provided for a type variable X whose upper bound is a function type, as a codomain type of X . When a function type $S \to T$ is substituted for X , $\mathrm{Range}(S \to T)$ is reduced to T . For a type variable X whose upper bound is a universal type, the type $\mathrm{Inst}(X,U)$ is provided. When a universal type $\forall Y \leq S.T$ is substituted for X , $\mathrm{Inst}(\forall Y \leq S.T, U)$ is reduced to $T[Y := U]$. We showed that types are strongly normalizing under these reductions, and this is why, in $cF^{+}_{\leq}$ and $cF^{++}_{\leq}$, a type-checked term can only have the minimal possible type in $F_{\leq}$ after reductions occurred. We also applied the techniques of these calculi to the other calculus $cF^{\&}_{\leq+}$ which integrates parametric polymorphism and ad-hoc polymorphism and investigated its proof-theoretical properties and its applications.

REPORT DATE

March 1, 1994

WRITTEN LANGUAGE

English

TOTAL NO. OF PAGES

76

NO. OF REFERENCES

64

ANY OTHER IDENTIFYING INFORMATION OF THIS REPORT

DISTRIBUTION STATEMENT

This technical report is available ONLY via anonymous FTP from `ftp.is.s.u-tokyo.ac.jp` (directory `/pub/tech-reports`).

SUPPLEMENTARY NOTES

DEPARTMENT OF INFORMATION SCIENCE
Faculty of Science, University of Tokyo
7-3-1 Hongo, Bunkyo-ku Tokyo, 113 Japan

On Extensions of $F_{\leq}$: Second-Order λ-Calculus with Subtyping

by
Daisuke Suzuki

Master Thesis

Submitted to
the Graduate School of the University of Tokyo
in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Information Science

on Feb., 1994

Abstract

$F_{\leq}$ was an extension of a second-order λ-calculus F which has parametric polymorphism with subtyping and bounded quantification, introduced by Ghelli to apply second-order λ-calculi to the framework of object-oriented languages. However, it is impossible to know the amount of information of a type-checked term before evaluation since the subsumption rules are included in $F_{\leq}$. To overcome this problem, we provided new calculi $cF^{+}_{\leq}$ and $cF^{++}_{\leq}$ as extensions of $F_{\leq}$, and investigated their proof-theoretic properties. $cF^{+}_{\leq}$ includes a new type $\mathrm{Range}(X)$, and $cF^{++}_{\leq}$ includes two new types $\mathrm{Range}(X)$ and $\mathrm{Inst}(X,T)$. For example, the type $\mathrm{Range}(X)$ is provided for a type variable X whose upper bound is a function type, as a codomain type of X. When a function type $S \to T$ is substituted for X, $\mathrm{Range}(S \to T)$ is reduced to T. For a type variable X whose upper bound is a universal type, the type $\mathrm{Inst}(X,U)$ is provided. When a universal type $\forall Y \leq S.T$ is substituted for X, $\mathrm{Inst}(\forall Y \leq S.T, U)$ is reduced to $T[Y := U]$. We showed that types are strongly normalizing under these reductions, and this is why, in $cF^{+}_{\leq}$ and $cF^{++}_{\leq}$, a type-checked term can only have the minimal possible type in $F_{\leq}$ after reductions occurred. We also applied the techniques of these calculi to the other calculus $cF^{\&+}_{\leq}$ which integrates parametric polymorphism and ad-hoc polymorphism and investigated its proof-theoretical properties and its applications.

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Motivation . . . 1
- 1.3 Contribution . . . 2
- 1.4 Outline . . . 3

2 Survey of Polymorphism **5**
- 2.1 Polymorphism . . . 5
- 2.2 Parametric Polymorphism . . . 5
  - 2.2.1 Second-order Lambda Calculus . . . 6
  - 2.2.2 Application to Object-Oriented Languages . . . 6
- 2.3 Ad-hoc Polymorphism . . . 6
- 2.4 Symbiosis of Parametric and Ad-hoc Polymorphism . . . 9
  - 2.4.1 Integrating Intersection Types . . . 9
  - 2.4.2 Integrating Overloaded Function . . . 9

3 $cF^{+}_{\leq}$ —A Type System without Type instantiation **11**
- 3.1 Introduction . . . 11
- 3.2 Type . . . 11
  - 3.2.1 Type Judgements . . . 14
  - 3.2.2 Subtype Relation . . . 15
- 3.3 Term . . . 21
  - 3.3.1 Term . . . 22
  - 3.3.2 Reduction system . . . 23
- 3.4 Strong Normalization . . . 31
  - 3.4.1 Outline of the Proof . . . 31
  - 3.4.2 Untyped Translation . . . 32
  - 3.4.3 Termination of Type-erasure of $cF^{+}_{\leq}$ Terms . . . 32
  - 3.4.4 Termination of $cF^{+}_{\leq}$ Terms . . . 39
- 3.5 Example . . . 39
- 3.6 Conclusion . . . 40

4 $cF^{++}_{\leq}$ —A Type System with Type instantiation **41**
- 4.1 Introduction . . . 41
- 4.2 Kind . . . 41
- 4.3 Type . . . 42
  - 4.3.1 Type Judgements . . . 43
  - 4.3.2 Subtype Relation . . . 44
- 4.4 Term . . . 48
  - 4.4.1 Term . . . 49

4.4.2 Reduction 50
4.5 Conclusion 53

5 Symbiosis of Parametric and Ad-hoc Polymorphisms 55
5.1 Introduction 55
5.2 Type 56
5.3 Term 61
5.4 Reduction 62
5.5 Example of Applications to Object-Oriented Languages 65
5.6 Conclusion 66

6 Conclusion 67
6.1 Conclusion 67
6.2 Future Work 67

List of Figures

2.1 Program in CLOS 7
2.2 Program in C++ 8

5.1 Example of the Structure of $\cap$-closed Set 59

Chapter 1

Introduction

1.1 Background

Polymorphism is one of the important tools for increasing programming efficiency and re-usability of programs. Polymorphism has two classifications: "parametric vs ad hoc" (see [Str67]) and "implicit vs explicit". A *parametric polymorphic function* is defined uniformly over types. That is, though a parametric polymorphic function behaves as a collection of monomorphic functions, they have the *shared* procedure written as a single lambda expression. System F [Gir72] enables us to treat parametric polymorphic functions as first-order objects. From the point of view of object-oriented languages which support the notion of subtyping, $F_{\leq}$ [Ghe90, CG92, CG93] is presented by Ghelli. $F_{\leq}$ is a conservative extension of F, which has the notion of subtyping and allows us to specify bounds on $\forall$-types. In addition, there are many studies about characterizing a parametric polymorphic function in model theory and category theory.

1.2 Motivation

However, in $F_{\leq}$, there is a problem because the system $F_{\leq}$ has the subsumption rule as follows:

$$\frac{\Gamma \vdash e{:}S \quad \Gamma \vdash S \leq T}{\Gamma \vdash e{:}T} \ (\text{Sub}) \ .$$

To tell the problem easier, we now consider an extension of $F_{\leq}$ with records, where a record types is expressed as:

$$[l_1{:}T_1, \cdots, l_n{:}T_n],$$

the subtyping rule of record types is expressed as:

$$\frac{\text{For all } \ i = 1, \cdots, n \ , \ \Gamma \vdash S_i \leq T_i}{\left\{ \begin{array}{l} \Gamma \vdash [l_1{:}S_1, \cdots, l_n{:}S_n, l_{n+1}{:}S_{n+1}, \cdots, l_{n+m}{:}S_{n+m}] \\ \qquad\qquad \leq [l_1{:}T_1, \cdots, l_n{:}T_n] \end{array} \right.} \ (\leq\text{-record}) \ ,$$

and record terms and their functions are described as:

$\{l_1{=}m_1, \cdots, l_n{=}m_n\}$	record term
$m \cdot l$	record field selection
$\{l_1{=}m_1, \cdots, l_n{=}m_n\} \cdot l_i \rightarrow m_i$	reduction of field selection $(i = 1, \ldots, n)$

.

In the system, a raw term $e = \Lambda X{\leq}[l{:}T].\lambda x^X.x \cdot l$ can have the type $\forall X{\leq}[l{:}T].X \to T$ because

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{X \leq [l{:}T] \vdash X \ \mathsf{type}}{X \leq [l{:}T] \vdash x^X : X} \quad X \leq [l{:}T] \vdash X \leq [l{:}T]}{X \leq [l{:}T] \vdash x^X : [l{:}T]}\ (\mathrm{Sub})}{X \leq [l{:}T] \vdash x^X \cdot T}}{X \leq [l{:}T] \vdash \lambda x^X.x \cdot l : X \to T}}{\vdash \Lambda X{\leq}[l{:}T].\lambda x^X.x \cdot l : \forall X{\leq}[l{:}T].X \to T}\ .$$

However, for any T's subtype S and any term n of the type S, when e is applied to a type $[l{:}S]$ and a term $\{l{=}n\}$ of the type $[l{:}S]$, $e\{[l{:}S]\}(\{l{=}n\})$ has the type T, not S, because

$$\dfrac{\dfrac{\vdots}{\vdash e : \forall X{\leq}[l{:}T].X \to T} \qquad \dfrac{\dfrac{\vdots}{\vdash S \leq T}}{\vdash [l{:}S] \leq [l{:}T]}}{\vdash e[l{:}S] : [l{:}S] \to T} \qquad \dfrac{\vdots}{\vdash n : S}}{\vdash e[l{:}S]\{l{=}n\} : T}\ .$$

while n, where

$$e\{[l{:}S]\}(\{l{=}n\}) \quad \to \quad (\lambda x^{[l:S]}.x \cdot l)(\{l{=}n\}) \quad \to \quad \{l{=}n\} \cdot l \quad \to \quad n,$$

can have the type S. That is, before reductions occur, we do not know whether we can fully obtain the information the term n of type S has.

1.3 Contribution

To overcome this problem, we provide new calculi $cF^+_{\leq}$ and $cF^{++}_{\leq}$ as extensions of the system $cF_{\leq}$ ($F_{\leq}$ where the subsumption rule is replaced with the coercion rule) enriched with two new types $\mathrm{Range}(X)$ and $\mathrm{Inst}(X,T)$ where X is a type variable. In the case of $\mathrm{Range}(X)$, function types may be substituted for X and when a function type $S \to U$ is substituted for X, $\mathrm{Range}(X)$ is reduced to the type U. In the case of $\mathrm{Inst}(X,T)$, (bounded) universal types may be substituted for X and a universal type $\forall X{\leq}S.U$ is substituted for X, $\mathrm{Inst}(X,T)$ is "reduced" to the type $U[X := T]$.

In the system $F_{\leq}$, every term expression can have two or more types and therefore much information may not be obtained. On the contrary, in $cF^+_{\leq}$, for each term expression, its type is uniquely determined and the amount of information loss be smaller.

$cF^+_{\leq}$ has other virtues: In $cF_{\leq}$, which is a variety of $F_{\leq}$ where the subsumption rule is replaced with the rule of coercion operator, we can write few terms which can type-check without a coercion operator. For example, in $cF_{\leq}$, $e = \Lambda X{\leq}(S \to T).\lambda x^X.\lambda y^S.xy$ cannot have its type, while, in $F_{\leq}$, it has its type $\forall X{\leq}(S \to T).X \to (S \to T)$. In $cF^+_{\leq}$, e has a type $\forall X{\leq}(S \to T).X \to (S \to \mathrm{Range}(X))$. When a type $U \to V$ which is a subtype of $S \to T$ is applied, $e\{U \to V\}$ has a type $(U \to V) \to (S \to V)$ because $\mathrm{Range}(U \to V)$ is reduced to V.

We think that $cF^+_{\leq}$ is more applicable to object-oriented languages than $F_{\leq}$ because for each term, its type is uniquely determined in $cF^+_{\leq}$, and because $cF^+_{\leq}$ satisfies Subject Reduction, that is, for each term, its type does not change when any reduction occurs.

1.4 Outline

We describe the contents of the rest of this thesis.

2. In **Chapter 2**, we give more details of the background of this thesis. We give a survey of parametric polymorphism, ad-hoc polymorphism, and calculi which integrate two kinds of polymorphisms.
3. **Chapter 3** is devoted to the study of a calculus $cF^{+}_{\leq}$. $cF^{+}_{\leq}$ is a second order λ-calculus with coercion operator and bounded quantification. To let the unicity of type hold in $cF^{+}_{\leq}$, the rules of coercion operator are used, instead of the subsumption rule in $F_{\leq}$. To avoid the problem about the termination of the "reduction" of types, the type instantiation rule $\mathrm{Inst}(\forall X{\leq}.S.T,U) \Rightarrow T[X:=U]$ is not used. Instead, we enrich this calculus with records and a new type $\mathrm{Sel}(X,l)$. In $\mathrm{Sel}(X,l)$, record types which have a field of label l will be substituted for X, and when $[\ldots,l{:}T,\ldots]$ *is* substituted for X, $\mathrm{Sel}(X,l)$ is reduced to T which is the type of the field l. Moreover, we show that $cF^{+}_{\leq}$ has good proof-theoretical properties: Subject Reduction, Church-Rosser, and Strong Normalization. To prove the Strong Normalization, we use a method of saturated sets, a variation of untyped version of reducibility predicates [Gir72, GLT89]. In $cF^{+}_{\leq}$, $e = \Lambda X{\leq}[l{:}T].\lambda x^{X}.x \cdot l$ (which already appeared in Section 1.2) has the type $\forall X{\leq}[l{:}T].X \rightarrow \mathrm{Sel}(X,l)$ and when e is applied to a type $[l{:}S]$ where S is a subtype of T, $e\{[l{:}S]\}$ has the type $[l{:}S] \rightarrow S$ because $\mathrm{Sel}([l{:}S],l)$ is reduced to S. Moreover, when $e\{[l{:}S]\}$ is applied to $\{l{=}n\}$ of the type $[l{:}S]$, $e\{[l{:}S]\}(\{l{=}n\})$ has the type S and it is reduced to n of the type S.
4. In **Chapter 4**, we give a calculus $cF^{++}_{\leq}$ by adding the type instantiation rule to $cF^{+}_{\leq}$. In $cF^{++}_{\leq}$, we show that the type system of $cF^{++}_{\leq}$ is well defined, for example, that the type instantiation rule (like β-reduction in λ-calculi) $\mathrm{Inst}(\forall X{\leq}S.T,U) \Rightarrow T[X:=U]$ always terminates. He proves it by attributing types to "kind". We also shows that $cF^{++}_{\leq}$ satisfies the Subject Reduction and the Church-Rosser.
5. **Chapter 5** tells about $cF^{\&}_{\leq}+$, another variety of $cF^{+}_{\leq}$ which implements symbiosis of parametric polymorphism and (a very restricted version of) ad-hoc polymorphism. In $cF^{\&}_{\leq}+$, the facility of ad-hoc polymorphism is implemented by overloaded functions as in $F^{\&}_{\leq}$. An overloaded function is a set of functions whose behaviors are not related to the others. Every function in it has its eigen type, and when the overloaded function is applied to a type, the component function whose eigen type matches the argument type is driven and returns a value. All eigen types in $cF^{\&}_{\leq}+$ are restricted to base types and record types whose all elements are base types.
   In $cF^{\&}_{\leq}+$, type instantiation rules are adopted but permitted only for overloaded types. We also show that $cF^{\&}_{\leq}+$ satisfies Church-Rosser and Subject Reduction. Finally we give some examples of applying $cF^{\&}_{\leq}+$ to object-oriented languages.
   The contents of Chapter 5 are not related to those of Chapter 4. So, if you skip Chapter 4 or Chapter 5, no confusion will occur.
6. Finally the conclusion of this thesis is written in **Chapter 6**. The future works for our research are also described.

Acknowledgements

I wish to express my gratitude to Masami Hagiya and colleagues.

Chapter 2

Survey of Polymorphism

2.1 Polymorphism

Polymorphism is one of the important tools for increasing programming efficiency and re-usability of programs. Polymorphism has a classification: "parametric vs ad-hoc". In addition, there are many studies about characterizing a parametric polymorphic function in model theory and category theory. However, in F (and $F_{\leq}$), a polymorphic function whose behavior on different types is not related cannot be written.

2.2 Parametric Polymorphism

A *parametric polymorphic function* is a function which is defined uniformly over types. That is, though a parametric polymorphic function behaves as a collection of monomorphic functions, they have the *shared* procedure written as a single lambda expression.

Example 2.2.1.
$PolyId = \Lambda X.\lambda x^X.x$ is a polymorphic identity function. When it is applied to a type S, $(PolyId)\{S\}$ is a function which receives a value of type S and returns it without any modification. In other words, $PolyId$ behaves as $\{\lambda x^S.x \mid S : \texttt{type}\}$, which is a collection of identity functions for all types.

The notion of parametric polymorphism is important and used in many functional programming languages (ML [AM87, AM91], Haskell [HW90], etc.).

Example 2.2.2.
Here are polymorphic functions in ML, `hd` and `tl`. `hd` receives a finite list of some type X and returns its head and `tl` receives a finite list of some type X and returns it after its head eliminated. The two functions behaves regardless of the type of arguments (`int list`, `string list`, `int list list`). They are polymorphic since they are uniformly defined over all list types.

```
- fun hd [] = raise Hd
=   | hd (x::_) = x;
val hd = fn : 'a list -> 'a
- fun tl [] = raise Tl
=   | tl (_::y) = y;
val tl = fn : 'a list -> 'a list
- hd [2,3,4,5];
```

```
val it = 2 : int
- hd ["aa", "bb"];
val it = "aa" : string
- tl [2,3,4,5];
val it = [3,4,5] : int list
- tl ["aa", "bb"];
val it = ["bb"] : string list
- hd [[2,3,4],[3],[]];
val it = [2,3,4] : int list
- tl [[2,3,4],[3],[]];
val it = [[3],[]] : int list list
-
```

2.2.1 Second-order Lambda Calculus

System F[Gir72, GLT89, Rey74] enables us to treat parametric polymorphic functions as first-order objects. A parametric polymorphic function is also characterized in a model as a function which preserves all the relations between all the types. To explain it in more detail, suppose that $F(t) \to G(t)$ is a type of a polymorphic function with a type variable t. When there exists a relation $\mathcal{R}$ between two types S and T, then we can form a relation $\mathcal{F}(\mathcal{R})$ between $F(S)$ and $F(T)$, and $\mathcal{G}(\mathcal{R})$ between $G(S)$ and $G(T)$. A parametric function is a collection of functions $\{f(t) : F(t) \to G(t) \mid t : \mathsf{type}\}$ indexed by types such that for each relation $\mathcal{R}$ between two types S and T, when $x \in F(S)$ and $y \in F(T)$ are related by $\mathcal{F}(\mathcal{R})$, then $f\{S\}(x) \in G(S)$ and $f\{T\}(y) \in G(T)$ are related by $\mathcal{G}(\mathcal{R})$. Reynolds characterized polymorphism using this in the first order case and gave an idea to extend it to the second order case [Rey83]. This property have been later proved to be a natural condition for a model to have natural categorical properties [Has90, Has91, MR91].

2.2.2 Application to Object-Oriented Languages

From the point of view of object-oriented languages which support the notion of subtyping, $F_{\leq}$ [CG92, CG93, Ghe90] is introduced by Ghelli. $F_{\leq}$ is an abstract version of the language *Fun* [CW85]. The language *Fun* was defined as a formal system to collect some different capabilities, as parametric polymorphism, hierarchical polymorphism, recursion and abstract data types, which can be found in different programming language. $F_{\leq}$ is in some way the kernel of the language *Fun*. $F_{\leq}$ is also a conservative extension of F, which allows us to specify bounds on $\forall$-function types. F-bounded was also abstracted from *Fun* by Cook et al. [CCHMO89], where a type variable X has a upper bound which can contain a type variable X. F-bounded is useful for typing object-oriented programs which involve recursive types: it allows quantification over types that are "structurally similar" to the recursive type $\mathsf{Rec}t.\mathrm{F}[t]$. F-bounded is currently included in many proposals for strongly typed object-oriented languages (*e.g.* [Bru93, Mit90, KLM94].

However, its proof-theoretical property and semantics are not widely studied because of its complicatedness [Ghe93].

However in F (and $F_{\leq}$ and F-bounded), a polymorphic function whose behavior on different types is not related to each other cannot be written.

2.3 Ad-hoc Polymorphism

Such a polymorphic function is called an *ad-hoc polymorphic function*. An example of such a function is a function which is written by type case. In other words, it performs a dispatch on a type passed as an argument. Ad-hoc polymorphism can be modeled using generic functions. For example, Common Lisp Object System, or CLOS [DG87, BDGKK88], which is a standard for merging object oriented programming with Common Lisp, implements message passing using generic functions, and in C++[Str88] multiple inheritance is implemented using overloading.

```
(defclass secondvec () ((x :initarg :x :reader x) (y :initarg :y :reader y)))

(defclass thirdvec (secondvec) ((z :initarg :z :reader z)))

(defgeneric size (secondvec))

(defmethod size ((s secondvec)) (let ((x (x s)) (y (y s)))
                                  (sqrt (+ (* x x) (* y y)))))

(defmethod size ((s thirdvec)) (let ((x (x s)) (y (y s)) (z (z s)))
                                 (sqrt (+ (* x x) (* y y) (* z z)))))
```

Figure 2.1: Program in CLOS

Example 2.3.1.

Here is an example in CLOS in Figure 2.1. The generic function `size` is a function which receives a vector value and returns its size. When `size` is applied to $\{x{=}1, y{=}1\}$ of type $[x{:}\mathrm{Int}, y{:}\mathrm{Int}]$, it returns a value $\sqrt{x^2+y^2}$. When `size` is applied to $\{x{=}1, y{=}1, z{=}1\}$ of type $[x{:}\mathrm{Int}, y{:}\mathrm{Int}, z{:}\mathrm{Int}]$, it returns a value $\sqrt{x^2+y^2+z^2}$. `size` is ad-hoc because it is defined dependently on the type of its argument.

The result of execurion is:

```
>(size (make-instance 'secondvec :x 1 :y 1))
1.4142135623730951

>(size (make-instance 'thirdvec :x 1 :y 1 :z 1))
1.7320508075688772

>
```

.

Example 2.3.2.

Here is an example in C++[Str88] in Figure 2.2. In the following program, the class `human` and its subclass `student` are defined, where `human`, `id`, `name`, and `sex`, which are the methods of `human`, are inherited by `student`, while `print` not inherited.

The result of execution is:

```
% ./a.out
Name = Taro ,sex = male
Name = Hanako ,sex = female, school = U_TOKYO ,year = 1
%
```

The notion of ad-hoc polymorphism is also important and used in many programming languages. However, ad-hoc polymorphism has not received so much attention as parametric polymorphism. Wadler tried to preset type classes which permit overloading of arithmetic operators in the framework of functional programming language practically [WB89]. In [CGL92], with the definition of the $\lambda\&$-calculus, Castagna, Ghelli and Longo started a theoretical analysis of ad-hoc polymorphism with the basis of message passing [Ghe91]. In $\lambda\&$, an overloaded function is represented by a list of ordinary functions. When an overloaded function is applied to an argument, it selects only one function from the component functions according to the type of the argument, and returns the result. Therefore, in $\lambda\&$, an ad-hoc polymorphic function whose behavior on different argument types is entirely unrelated is easily written as a first-class object. Tsuiki proposed a calculus λ_m [Tsu92, Tsu93, Tsu94a], which is a simply typed λ-calculus enriched with subtyping, coercion and merge operator. Merge operator is provided by generalizing from record merge operator of the extension of ML type system with record [OB88, Wan89]. In λ_m, the notion of merge to functions is introduced and it is good-natured in the sense that the behavior

```
#include <stream.h>
#include <strings.h>

class human {
public:
        int id;
        char name[30];
        enum {male , female} sex;
        void print();
};

class student:public human
{
public:
        char school[20];
        int year;
        void print();
};

void human::print() {
        cout << "Name = " << name << " ,sex = " << ((sex == male) ?
                                                    "male" : "female")
             << "\n";
};


void student::print() {
        cout << "Name = " << name << " ,sex = " << ((sex == male) ?
                                                    "male" : "female")
             << ", school = " << school << " ,year = " << year
             << "\n";
};

main()
{
        human a;
        student b;
        a.id = 1;
        strcpy(a.name,"Taro");
        a.sex = 0;
        a.print();
        b.id = 2;
        strcpy(b.name,"Hanako");
        b.sex = 1;
        strcpy(b.school,"U_TOKYO");
        b.year = 1;
        b.print();
}
```

Figure 2.2: Program in C++

on a subtype *inherits* those on its supertypes. A merge of functions activates all of appropriate functions and merges their effects. Therefore, component functions work together to calculate a result; and thus the behavior of a function on each type is closely related. We may say that the merge operator will bridge the gap between parametric and ad-hoc polymorphism. In order to select applicable functions, $\lambda\&$ and λ_m use type information at runtime in contrast to the simply typed lambda calculus which ignore type information at runtime. $\lambda\&C^*$ is introduced by Tsuiki [Tsu94] in order to overcome the problem that properties of subject reduction property and strong normalization are satisfied in $\lambda\&$. In [Tsu94] the method of compiling $\lambda\&G$, which is a simplification of $\lambda\&C^*$, to a untyped calculus is provided and therefore type information need not be used at runtime in $\lambda\&G$.

2.4 Symbiosis of Parametric and Ad-hoc Polymorphism

2.4.1 Integrating Intersection Types

The idea of intersection types consists of enriching the collection of types with a new type $\sigma \wedge \tau$ for every pair of types σ and τ . This new type is thought of as containing all the elements of σ that are also elements of τ . The most intriguing and potentially useful property of intersection types is their ability to express an essentially unbounded amount of information about the components of a program. Intersection types can also be viewed as a natural type-theoretic analogy of multiple inheritance. Pierce provided a detailed investigation of a (second order) typed λ-calculus $F_\wedge$ [Pie91] combining intersection types and bounded quantification. In this calculus parametric and finitary (ad-hoc) polymorphisms complement each other. As a joint work with Turner, Pierce recently provided a typed λ-calculus combining intersection types and bounded quantification $F^\omega_\wedge$ [PT93, PT93a] in the framework of higher-order λ-calculus F_ω [Gir72]. However, in this framework, calculi which satisfy the unicity of type are not widely investigated.

2.4.2 Integrating Overloaded Function

Castagna introduced $F^\&_\leq$ [Cas93], which integrates parametric polymorphism and "explicit" ad-hoc polymorphism. In $F^\&_\leq$, a function which performs a dispatch on a type passed as an argument would be written as: (X means type variables)

$$\Lambda X \leq T_1.exp_1 \ \& \ \Lambda X \leq T_2.exp_2 \ \& \cdots \& \ \Lambda X \leq T_n.exp_n$$

When this function is applied to a type V , $\Lambda X \leq T_i.exp_i$ which satisfies $T_i = \min\{T_k \mid k = 1, \ldots, n \ , V \leq T_k\}$ [1] is selected and returns the value $exp_i[X := V]$. But $F^\&_\leq$ includes many mistakes, which will be described in Chapter 5. However, it provides a clue to integrating parametric polymorphism and ad-hoc polymorphism.

[1] min is defined over subtyping relation.

Chapter 3

$\mathrm{cF}^{+}_{\leq}$—A Type System without Type instantiation

In this thesis, the notations of λ-calculus is based on the Barendregt's standard textbook [Bar84].

3.1 Introduction

As in Section 1.2, Because $F_{\leq}$ (with records) has the subsumption rule as follows:

$$\frac{\Gamma \vdash e{:}S \quad \Gamma \vdash S \leq T}{\Gamma \vdash e{:}T}\ (\mathrm{Sub}),$$

a raw term $e = \Lambda X \leq [l{:}T]\lambda x^{X}.x \cdot l$ can only have the type $\forall X \leq [l{:}T].X \to T$. However, for any T's subtype S and any term n of the type S, when e is applied to the type $[l{:}S]$ and the term $\{l{=}n\}$ (of the type $[l{:}S]$), $e\{[l{:}S]\}(\{l{=}n\})$ has the type T, not S. However n, where $e\{[l{:}S]\}(\{l{=}n\}) \to^{*} n$, can have the type S. That is, before reductions occur, we do not know whether we can fully obtain the information the term n of type S has.

To overcome this problem, we provide a new calculus $cF^{+}_{\leq}$ as an extension of the system $cF_{\leq}$ ($F_{\leq}$ where the subsumption rule is replaced with the coercion rule) with two new types $\mathrm{Range}(X)$ and $\mathrm{Sel}(X,l)$ where X is a type variable. In $cF^{+}_{\leq}$, e has the type $\forall X{\leq}[l{:}T].X \to \mathrm{Sel}(X,l)$ and when e is applied to a type $[l{:}S]$, $e\{[l{:}S]\}$ has a type $[l{:}S] \to S$. Moreover, when $e\{[l{:}S]\}$ is applied to $\{l{=}n\}$ of the type $[l{:}S]$, $e\{[l{:}S]\}(\{l{=}n\})$ has the type S and it is reduced to n of the type S.

In this chapter, we do not consider the type instantiation such that $\mathrm{Inst}(\forall X{\leq}S.T, U) \Rightarrow T[X := U]$ like β-reduction in λ-calculus because the termination of the instantiation is not trivially proved. The calculus which includes the type instantiation will be discussed in the next chapter. Instead of the type instantiation, we consider the calculus including records.

3.2 Type

In this section, we describe the type system $cF^{+}_{\leq}$. In this thesis we assume that there is no subtype relation between base types (for example, $\mathrm{Int} \leq \mathrm{Real}$).

Notation 3.2.1.
We uses these following notations when no specific notation is declared.

Notation	Metavariables
Type Environments	$\Gamma, \Delta, \ldots$
Type Variables	$X, Y, \ldots$
Type Constants	$A, B, \ldots$
Non-well-working Type	$P, Q, \ldots$
Type	$S, T, U, \ldots$
Term Variables	$x, y, \ldots$
Term Constants	$a, b, \ldots$
Term	$e, f, \ldots$

Notation 3.2.2.
We define a countable set of labels for records L

Definition 3.2.3. (Raw Type)
Raw types are defined as follows.

T	:=	P	non-well-working type
	|	W	working type
P	:=	X	type variable
	|	Range(P)	range type
	|	Sel(P, l)	field type ($l \in L$)
W	:=	B	base type
	|	$T \to T$	function type
	|	$\forall X.T$	universal type
	|	$\forall X \leq T.T$	bounded universal type
	|	$[l_1 : T, \ldots, l_n : T]$	record type ($l_i \in L$ $(i = 1, \ldots, n)$)

The definition of raw types is an extension of that of $F_{\leq}$. Raw types are divided into working types and non-well-working types. *Working types* consist of function types and (bounded) universal types, which are the same as defined in $F_{\leq}$. *Non-well-working types* consists of type variables, range types, and field types. Range types have the feature Range(P), where P has a super type that is a function type, and when a function type $S \to U$ is substituted for P, Range(P) is reduced to U.

Field types have the feature Sel(P, l_i), where P has a super type that is a record type, and when a record type $[l_1{:}U_1, \cdots, l_i{:}U_i, \cdots, l_n{:}U_n]$ which contains the field l_i is substituted for P, Sel(P, l_i) is reduced to U_i.

Definition 3.2.4. (Normal Form)
We call a raw type P in *normal form* if P does not contain Range($S \to T$) or Sel($[\cdots, l{:}T, \cdots]$).

Definition 3.2.5.
For a non-well-working type Range(P) (respectively Sel(P, l_i)), a working type $S \to T$ ($[l_1 : T_1, \cdots, l_i : T_i, \cdots, l_n : T_n]$) is substituted for P. The (raw) type Range($S \to T$) (Sel($[l_1 : T_1, \cdots, l_i : T_i, \cdots, l_n : T_n], l_i$)) need to be reduced to a normal form during these rules:

$$\begin{array}{lcl} \text{Range}(S \to T) & \Rightarrow & T \\ \text{Sel}([l_1 : T_1, \cdots, l_i : T_i, \cdots, l_n : T_n], l_i) & \Rightarrow & T_i \ (1 \leq i \leq n) \end{array}$$

After a finite number of reductions, Range($S \to T$) (respectively Sel($[l_1 : T_1, \cdots, l_i : T_i, \cdots, l_n : T_n], l_i$)) is reduced to V which need not be reduced, we write down Range($S \to T$) $\not\Rightarrow V$ (Sel($[l_1 : T_1, \cdots, l_i : T_i, \cdots, l_n : T_n], l_i$) $\not\Rightarrow$ V).

Definition 3.2.6. (Type Environment)
(Γ env) In $cF^{+}_{\leq}$, a *(type) environment* is defined as a sequence of free type variables and pairs of free type variables and its "upper bound". It is mutually defined to that of type-good formarion (in Definition 3.2.11).

$$\phi \quad \mathsf{env}$$

$$\frac{\Gamma \quad \mathsf{env} \quad \Gamma \vdash T \quad \mathsf{type} \quad X \notin \mathrm{FV}(\Gamma)}{\Gamma, X \leq T \quad \mathsf{env}}$$

$$\frac{\Gamma \quad \mathsf{env} \quad X \notin \mathrm{FV}(\Gamma)}{\Gamma, X \quad \mathsf{env}}$$

Notation 3.2.7.
Under an environment $\Gamma = \Delta, X \leq T, \Delta'$, for the type variable X, we call T the *least upper bound* (*l.u.b.*) of X. We represent the least upper bound of X as $\Gamma(X)$.

We want to extend the notion of l.u.b. to non-well-working types.

Definition 3.2.8.
For each type T, we want to define l.u.b. of T, $\Gamma(T)$. Of course, when T is a working type or $T = X$ where $\Gamma = \Delta, X, \Delta'$, we define $\Gamma(T) \equiv T$. Otherwise, that is, when P is a non-well-working type, we define $\Gamma(P)$ as follows:

$$\begin{array}{lcll} \Gamma(X) & \equiv & T & (\text{where } \Gamma = \Delta, X \leq T, \Delta') \\ \Gamma(\mathrm{Range}(P)) & \equiv & V & (\text{where } \mathrm{Range}(\Gamma(P), T) \not\ni V) \\ \Gamma(\mathrm{Sel}(P, l)) & \equiv & V & (\text{where } \mathrm{Sel}(\Gamma(P), l) \not\ni V) \end{array}$$

Definition 3.2.9.
To obtain the working upper bound of a non working type P, we recursively define the *working least upper bound (w.l.u.b.)* $\Gamma^*(P)$ as follows:

$$\begin{array}{lcll} \Gamma^*(T) & = & T & (T \text{ is a working type}) \\ & | & X & (T \text{ is a type variable } X \text{ and } \Gamma = \Delta, X, \Delta') \\ & | & \Gamma^*(V) & (T \text{ is a non-well-working type and } \Gamma(T) \not\ni V) \end{array}$$

Proposition 3.2.10.
The maximum of the size of the recursive definition Γ^* is limited by the size of Γ because of the definition of the type environment Γ.

(**Proof**): Because of the definition of the type environment Γ. ∎

3.2.1 Type Judgements

We have three kinds of type judgements: for the type good-formation ($\Gamma \vdash S \ \ \mathsf{type}$), for the subtyping relation ($\Gamma \vdash S \leq T$) and for the type assignment ($\Gamma \vdash e : S$). We also call the first two kinds of judgements *type judgements*. All kinds of *type judgements* are mutually defined.

Definition 3.2.11. (Type good-formation ($\Gamma \vdash S \ \ \mathsf{type}$))
We define the judgement of the type "good-formation".

$$\frac{X \in FV(\Gamma)}{\Gamma \vdash X \ \ \mathsf{type}} \ (\text{Var})$$

$$\frac{\Gamma \vdash S \ \ \mathsf{type} \quad \Gamma \vdash T \ \ \mathsf{type}}{\Gamma \vdash S \to T \ \ \mathsf{type}} \ (\to)$$

$$\frac{\Gamma, X \vdash T \ \ \mathsf{type}}{\Gamma \vdash \forall X.T \ \ \mathsf{type}} \ (\forall)$$

$$\frac{\Gamma \vdash S \ \ \mathsf{type} \quad \Gamma, X \leq S \vdash T \ \ \mathsf{type}}{\Gamma \vdash \forall X{\leq}S.T \ \ \mathsf{type}} \ (\text{bound-}\forall)$$

$$\frac{\text{For all } i = 1, \cdots, n, \ \Gamma \vdash T_i \ \ \mathsf{type}}{\Gamma \vdash [l_1 : T_1, \cdots, l_n : T_n] \ \ \mathsf{type}} \ (\text{record})$$

$$\frac{\Gamma \vdash P \ \ \mathsf{type} \quad \Gamma^*(P) = S \to T}{\Gamma \vdash \text{Range}(P) \ \ \mathsf{type}} \ (\text{Range})$$

$$\frac{\Gamma \vdash P \ \ \mathsf{type} \quad \Gamma^*(P) = [l_1 : T_1 \cdots, l_i : T_i, \cdots, l_n : T_n]}{\Gamma \vdash \text{Sel}(P, l_i) \ \ \mathsf{type}} \ (\text{record-Var})$$

3.2.2 Subtype Relation

Definition 3.2.12. (Subtype Relation ($\Gamma \vdash S \leq T$))
The definitions of the subtype relation are listed as follows:

$$\frac{\Gamma \vdash T \ \ \mathsf{type}}{\Gamma \vdash T \leq T} \ (\leq\text{-Refl})$$

$$\frac{\Gamma \vdash S \leq T \quad \Gamma \vdash T \leq U}{\Gamma \vdash S \leq U} \ (\leq\text{-Trans})$$

$$\frac{\Gamma \vdash P \ \ \mathsf{type} \quad \Gamma(P) \trianglerighteq V \quad \Gamma \vdash V \leq T}{\Gamma \vdash P \leq T} \ (\leq\text{-Taut})$$

$$\frac{\Gamma \vdash S' \leq S \quad \Gamma \vdash T \leq T'}{\Gamma \vdash S \to T \leq S' \to T'} \ (\leq\text{-}\to)$$

$$\frac{\Gamma \vdash \forall X.T \ \ \mathsf{type} \quad \Gamma, X \vdash T \leq T'}{\Gamma \vdash \forall X.T \leq \forall X.T'} \ (\leq\text{-}\forall)$$

$$\frac{\Gamma \vdash \forall X.T \ \ \mathsf{type} \quad \Gamma \vdash S' \ \ \mathsf{type} \quad \Gamma, X \leq S' \vdash T \leq T'}{\Gamma \vdash \forall X.T \leq \forall X \leq S'.T'} \ (\leq\text{-b}\forall)$$

$$\frac{\Gamma \vdash \forall X \leq S.T \ \ \mathsf{type} \quad \Gamma \vdash S' \leq S \quad \Gamma, X \leq S' \vdash T \leq T'}{\Gamma \vdash \forall X \leq S.T \leq \forall X \leq S'.T'} \ (\leq\text{-bb}\forall)$$

$$\frac{\text{For all } i = 1, \cdots, n,\ \Gamma \vdash S_i \leq T_i}{\left\{ \begin{array}{l} \Gamma \vdash [l_1{:}S_1, \cdots, l_n{:}S_n, l_{n+1}{:}S_{n+1}, \cdots, l_{n+m}{:}S_{n+m}] \\ \qquad\qquad \leq [l_1{:}T_1, \cdots, l_n{:}T_n] \end{array} \right.} \ (\leq\text{-record})$$

$$\frac{\Gamma \vdash \mathrm{Range}(P) \ \ \mathsf{type} \quad \Gamma \vdash \mathrm{Range}(Q) \ \ \mathsf{type} \quad \Gamma \vdash P \leq Q}{\Gamma \vdash \mathrm{Range}(P) \leq \mathrm{Range}(Q)} \ (\mathrm{Range})$$

$$\frac{\Gamma \vdash \mathrm{Sel}(P, l) \ \ \mathsf{type} \quad \Gamma \vdash \mathrm{Sel}(Q, l) \ \ \mathsf{type} \quad \Gamma \vdash P \leq Q}{\Gamma \vdash \mathrm{Sel}(P, l) \leq \mathrm{Sel}(Q, l)} \ (\mathrm{Sel})$$

Definition 3.2.13. (Degree of Type)
Under an environment Γ, for each well-formed type T, we define the size of the type T as follows:

$$
\begin{array}{lcl}
\mathrm{Deg}(\ \mathsf{env})_{\{\phi\}} & = & 1 \\
\mathrm{Deg}(\ \mathsf{env})_{\{\Gamma,X\}} & = & \mathrm{Deg}(\ \mathsf{env})_{\{\Gamma\}} + 1 \\
\mathrm{Deg}(\ \mathsf{env})_{\{\Gamma,X\leq S\}} & = & \mathrm{Deg}(\ \mathsf{env})_{\{\Gamma\}} + \mathrm{Deg}(S)_{\{\Gamma\}} + 1 \\
\mathrm{Deg}(B)_{\{\Gamma\}} & = & 1 \\
 & & (B \text{ is a base type}) \\
\mathrm{Deg}(X)_{\{\Gamma\}} & = & 1 \\
 & & (\Gamma = \Delta, X, \Delta') \\
\mathrm{Deg}(X)_{\{\Gamma\}} & = & \mathrm{Deg}(T)_{\{\Gamma\}} + 1 \\
 & & (\Gamma = \Delta, X \leq T, \Delta') \\
\mathrm{Deg}(\mathrm{Range}(P))_{\{\Gamma\}} & = & \mathrm{Deg}(S)_{\{\Gamma\}} + \mathrm{Deg}(V)_{\{\Gamma\}} + 1 \\
 & & (\ \Gamma \vdash \Gamma(P) \Rrightarrow V) \\
\mathrm{Deg}(S \to T)_{\{\Gamma\}} & = & \mathrm{Deg}(S)_{\{\Gamma\}} + \mathrm{Deg}(T)_{\{\Gamma\}} + 1 \\
\mathrm{Deg}(\forall X{\leq}S.T)_{\{\Gamma\}} & = & \mathrm{Deg}(S)_{\{\Gamma\}} + \mathrm{Deg}(T)_{\{\Gamma,X\leq S\}} + 1 \\
\mathrm{Deg}([l_1 : T_1, \cdots, l_n : T_n])_{\{\Gamma\}} & = & \Sigma_{i=1,\cdots n}(\mathrm{Deg}(T_i)_{\{\Gamma\}}) + 1 \\
\mathrm{Deg}(S \leq T)_{\{\Gamma\}} & = & \mathrm{Deg}(S)_{\{\Gamma\}} + \mathrm{Deg}(T)_{\{\Gamma\}} + 1
\end{array}
$$

From these definitions, the following properties are satisfied.

Proposition 3.2.14.
If $\Gamma \vdash S \leq T$ holds, then $\Gamma \vdash S$ type and $\Gamma \vdash T$ type hold.

(**Proof**): Obvious from definitions. ∎

Proposition 3.2.15.

1. The rule (Range) and (Sel) can be removed from the subtype relation rules.
2. The rule ($\leq$-Trans) can be removed from the subtype relation rules.

(**Proof**):

1. To prove $\Gamma \vdash P \leq P$, we can apply the rule (Refl). To prove $\Gamma \vdash P \leq Q$, we must apply the rule (Taut) after several (Range) or (Sel) rules. Therefore we can first apply the rule (Taut) without (Range) or (Sel). For example,

$$
\dfrac{\dfrac{\Gamma \vdash P' \ \ \mathsf{type} \quad \Gamma(P') \Rrightarrow V' \quad \Gamma \vdash V' \leq Q'}{\Gamma \vdash P' \leq Q'}\ (\text{Taut})}{\begin{array}{c}\vdots \\ \Gamma \vdash P \leq Q\end{array}\ \text{Applying several (} \leq \text{- Range) or (} \leq \text{- Sel)}}
$$

can be replaced with

$$
\dfrac{\Gamma \vdash P \ \ \mathsf{type} \quad \Gamma(P) \Rrightarrow V \qquad \begin{array}{c}\Gamma \vdash V' \leq Q' \\ \vdots \\ \Gamma \vdash V \leq Q\end{array}}{\Gamma \vdash P \leq Q}\ (\text{Taut})
$$

2. We prove that

$$\frac{\Gamma \vdash S \leq T \quad \Gamma \vdash T \leq U}{\Gamma \vdash S \leq U}\ (\text{Tran})$$

can be eliminated by induction on the depth of the proof of $\Gamma \vdash S \leq U$.

(1) (S , T , and U are $\rightarrow$-types.) In this case

$$\frac{\dfrac{\vdots \quad \vdots}{\Gamma \vdash T \leq S \quad \Gamma \vdash S' \leq T'}}{\dfrac{\Gamma \vdash S \rightarrow S' \leq T \rightarrow T'}{\Gamma \vdash S \rightarrow S' \leq U \rightarrow U'}} \qquad \frac{\dfrac{\vdots \quad \vdots}{\Gamma \vdash U \leq T \quad \Gamma \vdash T' \leq U'}}{\Gamma \vdash T \rightarrow T' \leq U \rightarrow U'}$$

Then we can modify the proof:

$$\frac{\dfrac{\dfrac{\vdots \quad \vdots}{\Gamma \vdash U \leq T \quad \Gamma \vdash T \leq S}}{\Gamma \vdash U \leq S} \qquad \dfrac{\dfrac{\vdots \quad \vdots}{\Gamma \vdash S' \leq T' \quad \Gamma \vdash T' \leq U'}}{\Gamma \vdash S' \leq U}}{\Gamma \vdash S \rightarrow S' \leq U \rightarrow U'}$$

By induction hypothesis, $\Gamma \vdash U \leq S$ and $\Gamma \vdash S' \leq U$ can be proved without (Trans).

(2) (S , T , and U are $\forall$-types.)
In this case

$$\frac{\dfrac{\vdots \quad \vdots}{\Gamma \vdash T \leq S \quad \Gamma, X \leq T \vdash S' \leq T'}}{\dfrac{\Gamma \vdash \forall X \leq S.S' \leq \forall X \leq .T'}{\Gamma \vdash \forall X \leq S.S' \leq \forall X \leq U.U'}} \qquad \frac{\dfrac{\vdots \quad \vdots}{\Gamma \vdash U \leq T \quad \Gamma, X \leq U \vdash T' \leq U'}}{\Gamma \vdash \forall X \leq T.T' \leq \forall X \leq U.U'}$$

Then we can modify the proof:

$$\frac{\dfrac{\dfrac{\vdots \quad \vdots}{\Gamma \vdash U \leq T \quad \Gamma \vdash T \leq S}}{\Gamma \vdash U \leq S} \qquad \dfrac{\dfrac{\vdots \quad \vdots}{\Gamma, X \leq U \vdash S' \leq T' \quad \Gamma, X \leq U \vdash T' \leq U'}}{\Gamma, X \leq U \vdash S' \leq U}}{\Gamma \vdash \forall X \leq S.S' \leq \forall X \leq U.U'}$$

Since $\Gamma, X \leq T \vdash S' \leq T'$ and $\Gamma \vdash U \leq T$ hold, $\Gamma, X \leq U \vdash S' \leq T'$ can be easily proved because

$$\frac{\dfrac{\vdots}{\Gamma \vdash U \leq S}}{\Gamma \vdash X \leq S}\ (\text{Taut})\ .$$

By induction hypothesis, $\Gamma \vdash U \leq S$ and $\Gamma, X \leq U \vdash S' \leq U$ can be proved without (Trans).

(3) (S , T , and U are non-working.) In this case

$$\frac{\begin{array}{l}\Gamma \vdash T \leq T \\ \quad \vdots \ \text{Applying several (Taut)} \\ \Gamma \vdash S \leq T\end{array} \qquad \begin{array}{l}\Gamma \vdash U \leq U \\ \quad \vdots \ \text{Applying several (Taut)} \\ \Gamma \vdash T \leq U\end{array}}{\Gamma \vdash S \leq U}$$

Then we can modify the proof:

$$\begin{array}{l}\Gamma \vdash U \leq U \\ \quad\vdots\ \text{Applying several (Taut)} \\ \Gamma \vdash T \leq U \\ \quad\vdots\ \text{Applying several (Taut)} \\ \Gamma \vdash S \leq U\end{array}$$

(4) (S and T are non-working and U is working.) In this case

$$\dfrac{\begin{array}{ll}\Gamma \vdash T \leq T & \qquad\qquad \vdots \\ \quad\vdots\ \text{Applying several (Taut)} & \Gamma \vdash W \leq U\ (\ W\ \text{is a working type.}) \\ \Gamma \vdash S \leq T & \qquad\qquad \vdots\ \text{Applying several (Taut)} \\ & \qquad\qquad \Gamma \vdash T \leq U\end{array}}{\Gamma \vdash S \leq U}$$

Then we can modify the proof:

$$\begin{array}{l}\quad\vdots \\ \Gamma \vdash W \leq U \\ \quad\vdots\ \text{Applying several (Taut)} \\ \Gamma \vdash T \leq U \\ \quad\vdots\ \text{Applying several (Taut)} \\ \Gamma \vdash S \leq U\end{array}$$

(5) (S is non-working and T and U are working.) In this case

$$\dfrac{\begin{array}{ll}\quad\vdots & \\ \Gamma \vdash W \leq T\ (\ W\ \text{is a working type.}) & \\ \quad\vdots\ \text{Applying several (Taut)} & \qquad\vdots \\ \Gamma \vdash S \leq T & \qquad \Gamma \vdash T \leq U\end{array}}{\Gamma \vdash S \leq U}$$

Then we can modify the proof.

$$\begin{array}{l}\dfrac{\begin{array}{ll}\vdots & \qquad\vdots \\ \Gamma \vdash W \leq T & \qquad \Gamma \vdash T \leq U\end{array}}{\Gamma \vdash W \leq U} \\ \quad\vdots\ \text{Applying several (Taut)} \\ \Gamma \vdash S \leq U\end{array}\ .$$

Since W is also working. $\Gamma \vdash W \leq U$ can be proved without (Trans).

∎

Notation 3.2.16.

Assume $\Delta \equiv X_1 \leq T_1, \ldots, X_n \leq T_n$, $X \notin \{X_1, \ldots, X_n\}$ and for every $Y \in \mathrm{FV}(U)$, $Y \notin \{X_1, \ldots, X_n\}$ hold.

We notate as follows:

$$\Delta[X := U] \quad \equiv \quad X_1 \leq T_1[X := U], \ldots, X_n \leq T_n[X := U]$$

For all $i \in 1, \ldots n$, $T_i[X := U] \twoheadrightarrow T_i'$ holds, we write $\Delta[X := U] \twoheadrightarrow \Delta'$, where

$$\Delta' \equiv X_1 \leq T_1', \ldots, X_n \leq T_n'$$

Proposition 3.2.17. (Substitution Lemma about Types)

1. Assume $\Gamma \vdash U$ type , and $\Delta[X := U] \twoheadrightarrow \Delta'$ hold,
   (1) If $\Gamma, X, \Delta \vdash T$ type and $T[X := U] \twoheadrightarrow T'$ hold, then $\Gamma, \Delta' \vdash T'$ type .
   (2) If $\Gamma, X, \Delta \vdash T \leq V$ type , $T[X := U] \twoheadrightarrow T'$ and $V[X := U] \twoheadrightarrow V'$ hold, then $\Gamma, \Delta' \vdash T' \leq V'$.
2. Assume $\Gamma \vdash U \leq S$ holds,
   (1) If $\Gamma, X \leq S, \Delta \vdash T$ type and $\Delta[X := U] \twoheadrightarrow \Delta'$ hold, then $\Gamma, \Delta' \vdash T'$ type .
   (2) If $\Gamma, X \leq S, \Delta \vdash T \leq V$ type , $T[X := U] \twoheadrightarrow T'$ and $V[X := U] \twoheadrightarrow V'$ hold, then $\Gamma, \Delta' \vdash T' \leq V'$.

(**Proof**): We prove only the second case by induction on $\mathrm{Deg}(T)_{\{\Gamma, X \leq S, \Delta\}}$

1. *Case* ($T = X$.)
   Obvious.

   Case ($T = Y$ $(Y \neq X)$.)
   Obvious.

   Case ($T = V \to W$.)
   By induction hypothesis , we can assume that $V[X := U] \twoheadrightarrow V'$, $W[X := U] \twoheadrightarrow W'$, $\Gamma, \Delta' \vdash V'$ type and $\Gamma, \Delta' \vdash W'$ type . Since $(V \to W)[X := U] \twoheadrightarrow V' \to W'$. From assumptions,

$$\frac{\begin{array}{cc}\vdots & \vdots \\ \Gamma, \Delta' \vdash V' \text{ type} & \Gamma, \Delta' \vdash W' \text{ type}\end{array}}{\Gamma, \Delta' \vdash V' \to W' \text{ type}}$$

   Case ($T = \forall Y \leq V.W$.)
   By induction hypothesis , we can assume that $V[X := U] \twoheadrightarrow V'$, $W[X := U] \twoheadrightarrow W'$, $\Gamma, \Delta \vdash' V'$ type and $\Gamma, \Delta, Y \leq V' \vdash W'$ type . Since $(\forall Y \leq V.W)[X := U] \twoheadrightarrow \forall Y \leq V'.W'$. From assumptions,

$$\frac{\begin{array}{cc}\vdots & \vdots \\ \Gamma, \Delta' \vdash V' \text{ type} & \Gamma, \Delta', Y \leq V' \vdash W' \text{ type}\end{array}}{\Gamma, \Delta' \vdash \forall Y \leq V'.W' \text{ type}}$$

   Case ($T = \mathrm{Range}(P)$.)
   We consider two subcases:
   (1) ($P[X := U] \twoheadrightarrow M \to N$)
   Since by induction hypothesis, $\mathrm{Range}(P)[X := U] \twoheadrightarrow N$ and $\Gamma \vdash N$ type hold, we have $\Gamma \vdash N$ type .

$$\begin{array}{c}\vdots \\ \Gamma, \Delta' \vdash N \text{ type}\end{array}$$

(2) ($P[X := U] \Rrightarrow Q$)
Obvious by induction hypothesis, we have:

$$\frac{\begin{array}{c}\vdots\\ \Gamma, \Delta' \vdash Q \ \ \mathsf{type}\end{array}}{\Gamma, \Delta' \vdash \mathrm{Range}(Q) \ \ \mathsf{type}}$$

2. By Subtype Relation Rules:

Case ($\Gamma, X \leq S, \Delta \vdash X \leq X$.)
Since $\Gamma \vdash U \leq U$ holds, then $\Gamma, \Delta' \vdash U \leq U$ also holds.

Case ($\Gamma, X \leq S, \Delta \vdash Y \leq Y$.)
Obvious.

Case ($\Gamma, X \leq S, \Delta \vdash X \leq T$.)
Since $S = S[X := U]$ holds,

$$\frac{\begin{array}{c}\vdots\\ \Gamma, \Delta' \vdash S \leq T'\end{array}}{\Gamma, \Delta' \vdash U \leq T'}$$

Case ($\Gamma, X \leq S, \Delta \vdash \mathrm{Range}(P) \leq T$.)

(1) ($P[X := U] \Rrightarrow M \to N$)
We have $\Gamma(P)[X := U] \Rrightarrow M' \to N'$. Then, by induction hypothesis, we have:

$$\frac{\Gamma \vdash M' \leq M \quad \Gamma \vdash N \leq N'}{\Gamma \vdash M \to N \leq M' \to N'}$$

Since $\mathrm{Range}(P)[X := U] \Rrightarrow N$ and $\Gamma(\mathrm{Range}(P))[X := U] \Rrightarrow N'$, we have $\Gamma \vdash N \leq N'$ and $\Gamma \vdash N' \leq T$. Then we have:

$$\frac{\begin{array}{c}\vdots\\ \Gamma, \Delta' \vdash N \leq T'\end{array}}{}$$

(2) ($P[X := U] \Rrightarrow Q$) Obvious from induction hypothesis, we have:

$$\frac{\begin{array}{c}\vdots\\ \Gamma, \Delta' \vdash R \leq T'\end{array} \qquad \mathrm{Range}((\Gamma, \Delta')(Q)) \Rrightarrow R}{\Gamma, \Delta' \vdash \mathrm{Range}(Q) \leq T'}$$

Case ($\Gamma, X \leq S, \Delta \vdash V \to T \leq V_0 \to T_0$.)

$$\frac{\begin{array}{c}\vdots\\ \Gamma, \Delta' \vdash V_0' \leq V'\end{array} \qquad \begin{array}{c}\vdots\\ \Gamma, \Delta' \vdash T' \leq T_0'\end{array}}{\Gamma, \Delta' \vdash V' \to T' \leq V_0' \to T_0'}$$

Case ($\Gamma, X \leq S, \Delta \vdash \forall Y \leq V.T \leq \forall Y \leq V_0.T_0$.)

$$\frac{\begin{array}{c}\vdots\\ \Gamma, \Delta' \vdash V_0' \leq V'\end{array} \qquad \begin{array}{c}\vdots\\ \Gamma, \Delta', Y \leq V_0' \vdash T' \leq T_0'\end{array}}{\Gamma, \Delta' \vdash \forall Y \leq V'.T' \leq \forall Y \leq V_0'.T_0'}$$

Case ($\Gamma, X \leq S, \Delta \vdash \mathrm{Range}(P) \leq \mathrm{Range}(Q)$.)

(1) ($P[X := U] \Rrightarrow M \to N$, $Q[X := U] \Rrightarrow M' \to N'$)
Since, by induction hypothesis, $\Gamma, \Delta' \vdash M \to N \leq M' \to N'$ holds, we have $\Gamma \vdash N \leq N'$

(2) ($P[X := U] \Rrightarrow P', Q[X := U] \Rrightarrow M' \to N'$)
From induction hypothesis,

$$\frac{\begin{array}{c}\vdots\\ \Gamma, \Delta' \vdash P' \leq M' \to N'\end{array}}{\Gamma, \Delta' \vdash \mathrm{Range}(P') \leq N'}$$

(3) ($P[X := U] \Rrightarrow P', Q[X := U] \Rrightarrow Q'$

$$\frac{\begin{array}{c}\vdots\\ \Gamma, \Delta' \vdash P' \leq Q'\end{array}}{\Gamma, \Delta' \vdash \mathrm{Range}(P') \leq \mathrm{Range}(Q')}$$

∎

3.3 Term

In this section, we describe the terms of $cF^{+}_{\leq}$. We start by the definition of *raw terms*, among which we distinguish the *terms*, i.e. those raw terms possessing types. Raw terms are divided into several classes: variables, constants, function terms (functions), $\forall$ -function terms, record terms, record selection terms, and coerced terms. Coercion is introduced to hold the properties of unicity of type and subject reduction.

Definition 3.3.1. (Raw Term)

e	:=	x^T	(term) variable
	|	c^T	constant
	|	$\lambda x^T.w$	λ-abs
	|	ee	λ-apl
	|	$\Lambda X.e$	Λ-abs
	|	$\Lambda X \leq T.e$	bound Λ-abs
	|	$e\{T\}$	Λ-apl
	|	$\{l_1 = e, \cdots, l_n = e\}$	record-const ($l_i \in L$)
	|	$e \cdot l$	record-select ($l \in L$)
	|	$e\|_T$	coerced term

Types which stick to variables or constants are sometimes omitted.

We need to define the reduction of types ($\Rrightarrow$) which stick to terms because these sorts of types must be in normal form.

Definition 3.3.2.

x^T	$\Rrightarrow$	$x^{T'}$	where $T \Rrightarrow T'$
$\lambda x^T.e$	$\Rrightarrow$	$\lambda x^{T'}.e'$	where $T \Rrightarrow T'$ and $e \Rrightarrow e'$
ef	$\Rrightarrow$	$e'f'$	where $e \Rrightarrow e'$ and $f \Rrightarrow f'$
$\Lambda X.e$	$\Rrightarrow$	$\Lambda X.e'$	where $e \Rrightarrow e'$
$\Lambda X \leq S.e$	$\Rrightarrow$	$\Lambda X \leq S'.e'$	where $S \Rrightarrow S'$ and $e \Rrightarrow e'$
$e\{T\}$	$\Rrightarrow$	$e'\{T'\}$	where $T \Rrightarrow T'$ and $e \Rrightarrow e'$
$e\|_T$	$\Rrightarrow$	$e'\|_{T'}$	where $T \Rrightarrow T'$ and $e \Rrightarrow e'$
$\{l_i = e_i\}_{i=1,\ldots,n}$	$\Rrightarrow$	$\{l_i = e'_i\}_{i=1,\ldots,n}$	where $e_i \Rrightarrow e'_i$ $(i = 1, \ldots, n)$

3.3.1 Term

Definition 3.3.3. (Type assignment ($\Gamma \vdash e : S$))

Type assignment ($\Gamma \vdash e : S$)

Terms are selected by the rules below; since term variables are indexed by their type, the rules do not needed assumptions of the term $(x : T)$:

$$\frac{\Gamma \vdash T \ \ \mathsf{type}}{\Gamma \vdash x^T : T} \ \text{(var)}$$

$$\frac{\Gamma \vdash e : T}{\Gamma \vdash (\lambda x^S.e) : S \to T} \ (\lambda\text{-ab})$$

$$\frac{\Gamma \vdash e : S \to T \quad \Gamma \vdash f : U \quad \Gamma \vdash U \leq S}{\Gamma \vdash ef : T} \ (\lambda\text{-ap})$$

$$\frac{\Gamma \vdash e : P \quad \Gamma \vdash f : U \quad \Gamma^*(P) = S \to T \quad \Gamma \vdash U \leq S}{\Gamma \vdash ef : \mathrm{Range}(P)} \ (\lambda\text{-Range})$$

$$\frac{\Gamma, X \vdash f : T \quad \text{no free variable whose type is } X \text{ does not occur in } f}{\Gamma \vdash (\Lambda X.f) : \forall X.T} \ (\Lambda\text{-ab})$$

$$\frac{\Gamma, X \leq S \vdash e : T \quad \text{no free variable whose type is } X \text{ does not occur in } f}{\Gamma \vdash (\Lambda X{\leq}S.e) : \forall X{\leq}S.T} \ (\text{bound-}\Lambda\text{-ab})$$

$$\frac{\Gamma \vdash e : \forall X.T \quad \Gamma \vdash U \ \ \mathsf{type} \quad T[X := U] \rrarr T'}{\Gamma \vdash e\{U\} : T'} \ (\Lambda\text{-ap-w})$$

$$\frac{\Gamma \vdash e : \forall X{\leq}S.T \quad \Gamma \vdash U \leq S \quad T[X := U] \rrarr T'}{\Gamma \vdash e\{U\} : T'} \ (\text{bound-}\Lambda\text{-ap-w})$$

$$\frac{\text{For all } i = 1, \cdots, n, \Gamma \vdash e_i : T_i}{\Gamma \vdash \{l_1 = e_1, \cdots, l_n = e_n\} : [l_1 : T_1, \cdots, l_n : T_n]} \ \text{(record-const)}$$

$$\frac{\Gamma \vdash e : [\cdots, l{:}T, \cdots]}{\Gamma \vdash e \cdot l : T} \ \text{(record-select)}$$

$$\frac{\Gamma \vdash e : P \quad \Gamma \vdash \mathrm{Sel}(P, l) \ \ \mathsf{type}}{\Gamma \vdash e \cdot l : \mathrm{Sel}(P, l)} \ \text{(record-Sel)}$$

$$\frac{\Gamma \vdash e : S \quad \Gamma \vdash S \leq T}{\Gamma \vdash e|_T : T} \ \text{(coer)}$$

$\lambda x^S.e$ is a function abstraction and ef is a function application. When e has a function type $S \to T$, ef has the type T of e's "output". When e has a non-well-working type P, ef also has a non-well-working type including P.

$\Lambda X{\leq}S.e$ is a $\forall$-function abstraction and $e\{S\}$ is a $\forall$-function application. When e has a universal type $\forall X(\leq U).T$, $e\{S\}$ has the type $T[X := S]$ of e's "output". Note that, in $\Lambda X{\leq}S.e$, no free variable of type X occurs in e.

$\{l_1{=}e_1, \cdots, l_n{=}e_n\}$ is a record term and $e \cdot l$ is a record field selection term. When e has a record type $[l_1{:}T_1, \cdots, l_i{:}T_i, \cdots, l_n{:}T_n]$, $e \cdot l_i$ has the type T_i of the content of the field l_i. When e is a non-well-working type P, $e \cdot l$ also has a non-well-working type $\mathrm{Sel}(P, l)$.

$e|_T$ is a coerced term, where though e has the type S which is T's subtype, $e|_T$ has the type T.

Because this system does not contain the subsumption rule, the following good properties are satisfied.

Theorem 3.3.4.
If $\Gamma \vdash e : S$ then $\Gamma \vdash S$ type

Proposition 3.3.5. (Unicity of Types)
For each expression e, if $\Gamma \vdash e : S$ and $\Gamma \vdash e : T$ hold, then both of S and T are in normal form and $S = T$ holds.

3.3.2 Reduction system

From these definitions, the following properties are satisfied.

Proposition 3.3.6. (Substitution Lemma)

1. If $\Gamma \vdash e : S$ holds, e includes x^T, and $\Gamma \vdash f : T$ holds, then $\Gamma \vdash e[x^T := f] : S$ holds.
2. If $\Gamma, X \leq S, \Delta \vdash e : T$ (respectively $\Gamma, X, \Delta \vdash e : T$), $\Gamma \vdash U \leq S$ ($\Gamma \vdash U$ type), $\Delta[X := U] \Rrightarrow \Delta'$, $e[X := U] \Rrightarrow e'$ and $T[X := U] \Rrightarrow T'$ hold, then $\Gamma, \Delta' \vdash e' : T'$ holds.

(**Proof**):

1. We can prove it by replacing

$$\frac{\begin{array}{c}\vdots\\ \Gamma \vdash T \text{ type}\end{array}}{\Gamma \vdash x^T : T}$$

with

$$\begin{array}{c}\vdots\\ \Gamma \vdash f : T\end{array}$$

2. Induction on the structure of the proof of $\Gamma \vdash e : S$. We prove in some cases:
   (1) ($\Gamma, X \leq S, \Delta \vdash x^X : X$)

$$\frac{\begin{array}{c}\vdots\\ \Gamma, \Delta' \vdash U \text{ type}\end{array}}{\Gamma \vdash x^U : U}$$

   (2) ($\Gamma, X \leq S, \Delta \vdash \Lambda Y \leq V.e : \forall Y \leq V.T$)

$$\frac{\begin{array}{c}\vdots\\ \Gamma, \Delta' Y \leq V' \vdash e' : T'\end{array}}{\Gamma, \Delta' \vdash \Lambda Y \leq e' : \forall Y \leq V'.T'}$$

   (3) ($\Gamma, X \leq S, \Delta \vdash ef : \mathrm{Range}(P)$)
   We only consider the case that $\Gamma, X \leq S, \Delta \vdash e : P$ and $\Gamma, X \leq S, \Delta \vdash f : M$ hold.

(i) ($P[X := U] \Rrightarrow V \to W$)

$$\frac{\Gamma,\Delta' \vdash e' : V \to W \qquad \Gamma,\Delta' \vdash f' : M' \qquad \Gamma,\Delta' \vdash M' \leq V}{\Gamma,\Delta' \vdash e'f' : W}$$

(ii) ($P[X := U] \Rrightarrow P$)

$$\frac{\Gamma,\Delta' \vdash e' : P' \qquad \Gamma,\Delta' \vdash f' : M'}{\Gamma,\Delta' \vdash e'f' : \mathrm{Range}(P')}$$

(4) ($\Gamma, X \leq S, \Delta \vdash e\{T\} : V'$)

$$\frac{\Gamma,\Delta' \vdash \Lambda Y{\leq}T_0'.e : \forall Y{\leq}T_0'.V' \qquad \Gamma,\Delta' \vdash T' \leq T_0' \qquad e[X := U] \Rrightarrow e' \qquad V'[Y := T'] \Rrightarrow V''}{\Gamma,\Delta' \vdash (e'\{T'\}) : V''}$$

∎

Definition 3.3.7. (Reduction)

The reduction rules in $cF^{+}_{\leq}$ are listed as follows:

$$\frac{}{\Gamma \vdash (\lambda x^S.e)f \ \to\ e[x := f|_S]}\ (\beta)$$

$$\frac{e[X := S] \Rrightarrow e'}{\Gamma \vdash (\Lambda X.e)\{S\} \ \to\ e'}\ (\beta 2)$$

$$\frac{e[X := T] \Rrightarrow e'}{\Gamma \vdash (\Lambda X{\leq}S.e)\{T\} \ \to\ e'}\ (\text{bound } \beta 2)$$

$$\frac{}{\Gamma \vdash \{l_1 = e_1, \cdots, l_i = e_i, \cdots, l_n = e_n\} \cdot l_i \ \to\ e_i}\ (\text{Sel})$$

$$\frac{\Gamma \vdash f : T \quad \mathrm{Range}(S) \Rrightarrow V}{\Gamma \vdash (e|_S)f \ \to\ (ef)|_V}\ (\text{coer})$$

$$\frac{\Gamma \vdash e : \forall X{\leq}V.W \quad T[X := U] \Rrightarrow T'}{\Gamma \vdash (e|_{\forall X \leq S.T})\{U\} \ \to\ (e\{U\})|_{T'}}\ (\text{coer2})$$

$$\frac{\mathrm{Sel}(S, l) \Rrightarrow V}{\Gamma \vdash (e|_S) \cdot l \ \to\ (e \cdot l)|_V}\ (\text{coerSel})$$

$$\frac{\Gamma \vdash e \text{ is in normal form} \quad S \text{ is a base type.}}{\Gamma \vdash e|_S \ \to\ e}\ (\text{coerconst})$$

(β) is the rule of function application. From this rule, the coercion operator occurs after reduction

((bounded-) $\beta 2$) is the β-reduction rule for $\forall$ terms but the slightly complicated substitution about types occurs after reduction.

(coer) means that when the function term e is coerced, after we apply e to the argument f, ef is coerced to a proper type.

(coer2) means that when the $\forall$-function term e is coerced, after we apply e to the argument T, $e\{U\}$ is coerced to a proper type.

(coerconst) meant that, for each term e, when e is coerced to a (constant) base type S, the information that e has is not lost at all.

$cF^{+}_{\leq}$ satisfies some good properties as follows:

Theorem 3.3.8. (Subject Reduction)
For each expression e, if $\Gamma \vdash e : T$ and $\Gamma \vdash e \rightarrow e'$ hold, then $\Gamma \vdash e' : T$ holds.

(**Proof**): From the reduction rules:

Case ((β).)
From substitution lemma (Proposition 3.3.6).

Case (($\beta 2$, bound- $\beta 2$).)
From substitution lemma (Proposition 3.3.6).

Case ((Sel).)

$$\dfrac{\dfrac{\vdots\ \Pi_i}{\cdots \quad \Gamma \vdash e_i : T_i \quad \cdots}}{\dfrac{\Gamma \vdash \{l_1 = e_1, \cdots, l_i = e_i, \cdots, l_n = e_n\} : [l_1{:}T_1, \cdots, l_i{:}T_i, \cdots, l_n{:}T_n]}{\Gamma \vdash \{l_1 = e_1, \cdots, l_i = e_i, \cdots, l_n = e_n\} \cdot l_i : T_i}}$$

$$\Downarrow$$

$$\begin{array}{c}\vdots\ \Pi_i \\ \Gamma \vdash e_i : T_i\end{array}$$

Case ((coer).)

1. ($S = S_1 \rightarrow S_2$, $\Gamma \vdash e : U_1 \rightarrow U_2$)
   Range$(S_1 \rightarrow S_2) \trianglerighteq S_2$ holds, then

$$\dfrac{\dfrac{\vdots\ \Pi_1}{\Gamma \vdash e : U_1 \rightarrow U_2} \quad \dfrac{\vdots\ \Sigma_1 \qquad \vdots\ \Sigma_2}{\Gamma \vdash S_1 \leq U_1 \quad \Gamma \vdash U_2 \leq S_2}}{\Gamma \vdash U_1 \rightarrow U_2 \leq S_1 \rightarrow S_2}}{\dfrac{\Gamma \vdash e|_{S_1 \rightarrow S_2} : S_1 \rightarrow S_2}{\Gamma \vdash (e|_{S_1 \rightarrow S_2})f : S_2}} \quad \dfrac{\vdots\ \Pi_2}{\Gamma \vdash f : T} \quad \dfrac{\vdots\ \Sigma_3}{\Gamma \vdash T \leq S_1}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\vdots\ \Pi_1}{\Gamma \vdash e : U_1 \rightarrow U_2} \quad \dfrac{\vdots\ \Pi_2}{\Gamma \vdash f : T} \quad \dfrac{\dfrac{\vdots\ \Sigma_3}{\Gamma \vdash T \leq S_1} \quad \dfrac{\vdots\ \Sigma_1}{\Gamma \vdash S_1 \leq U_1}}{\Gamma \vdash T \leq U_1}}{\Gamma \vdash ef : U_2} \quad \dfrac{\vdots\ \Sigma_2}{\Gamma \vdash U_2 \leq S_2}}{\Gamma \vdash (ef)|_{S_2} : S_2}$$

2. ($S = S_1 \rightarrow S_2$, $\Gamma \vdash e : P$)
   Range$(S_1 \rightarrow S_2) \trianglerighteq S_2$ holds, then

$$\dfrac{\dfrac{\dfrac{\vdots\ \Pi_1}{\Gamma \vdash e : P} \quad \dfrac{\vdots\ \Sigma_1}{\Gamma \vdash P \leq S_1 \to S_2}}{\Gamma \vdash e|_{S_1 \to S_2} : S_1 \to S_2} \quad \dfrac{\vdots\ \Pi_2}{\Gamma \vdash f : T} \quad \dfrac{\vdots\ \Sigma_2}{\Gamma \vdash T \leq S_1}}{\Gamma \vdash (e|_{S_1 \to S_2})f : S_2}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\vdots\ \Pi_1}{\Gamma \vdash e : P} \quad \dfrac{\vdots\ \Pi_2}{\Gamma \vdash f : T} \quad \dfrac{\vdots\ \Sigma_1}{\Gamma \vdash P \leq S_1 \to S_2} \quad \dfrac{\vdots\ \Sigma_2}{\Gamma \vdash T \leq S_1}}{\Gamma \vdash ef : \mathrm{Range}(P)} \quad \dfrac{\dfrac{\vdots\ \Sigma_1}{\Gamma \vdash P \leq S_1 \to S_2}}{\Gamma \vdash \mathrm{Range}(P) \leq S_2}$$
$$\Gamma \vdash (ef)|_{S_2} : S_2$$

In this case we can replace $\Gamma^*(P) \equiv S_1 \to S_2$ with $\Gamma \vdash P \leq S_1 \to S_2$. In the following cases, for instance, we also replace $\Gamma^*(P) \equiv S \to T$ with $\Gamma \vdash S \to T$.

3. ($S = Q$, $\Gamma \vdash e : P$)

$$\dfrac{\dfrac{\dfrac{\vdots\ \Pi_1}{\Gamma \vdash e : P} \quad \dfrac{\vdots\ \Sigma_1}{\vdash P \leq Q}}{\Gamma \vdash e|_Q : Q} \quad \dfrac{\vdots\ \Pi_2}{\Gamma \vdash f : T} \quad \dfrac{\vdots\ \Sigma_2}{\Gamma \vdash Q \leq U \to V} \quad \dfrac{\vdots\ \Sigma_3}{\Gamma \vdash U \leq T}}{\Gamma \vdash (e|_Q)f : \mathrm{Range}(Q)}$$

$$\Downarrow$$

$$\dfrac{\boxed{\mathbf{I}} \quad \dfrac{\dfrac{\vdots\ \Sigma_1}{Q \leq P}}{\Gamma \vdash \mathrm{Range}(Q) \leq \mathrm{Range}(P)}}{\Gamma \vdash (ef)|_{\mathrm{Range}(Q)} : \mathrm{Range}(Q)}$$

where $\mathbf{I}$ is

$$\dfrac{\dfrac{\vdots\ \Pi_1}{\Gamma \vdash e : P} \quad \dfrac{\vdots\ \Pi_2}{\Gamma \vdash f : T} \quad \dfrac{\vdots\ \text{(From Proposition 3.2.15)}}{\Gamma \vdash P \leq U \to V} \quad \dfrac{\vdots\ \Sigma_3}{\Gamma \vdash U \leq T}}{\Gamma \vdash ef : \mathrm{Range}(P)}$$

Case ((coer2).)

$$\dfrac{\dfrac{\dfrac{\vdots\ \Pi_1}{\Gamma \vdash e : \forall X{\leq}V.W} \quad \dfrac{\dfrac{\vdots\ \Sigma_1}{\Gamma \vdash S \leq V} \quad \dfrac{\vdots\ \Sigma_2}{\Gamma, X \leq S \vdash W \leq T}}{\Gamma \vdash \forall X{\leq}V.W \leq \forall X{\leq}S.T}}{\Gamma \vdash e|_{\forall X \leq S.T} : \forall X{\leq}S.T} \quad \dfrac{\vdots\ \Sigma_3}{\Gamma \vdash U \leq S} \quad T[X := U] \not\equiv T'}{\Gamma \vdash (e|_{\forall X \leq S.T})\{U\} : T'}$$

$$\Downarrow$$

$$\dfrac{\boxed{\mathbf{I}}\quad \dfrac{\vdots\ \Sigma_2 \qquad\qquad \vdots\ \Sigma_3}{\Gamma, X \leq S \vdash W \leq T \qquad \Gamma \vdash U \leq S}\ \text{(From Subst. Lemma)}}{\Gamma \vdash W' \leq T'}}{\Gamma \vdash (e\{U\})|_{T'} : T'}$$

where $\mathbf{I}$ is:

$$\dfrac{\dfrac{\vdots\ \Pi_1}{\Gamma \vdash e : \forall X{\leq}V.W} \qquad \dfrac{\dfrac{\vdots\ \Sigma_3}{\Gamma \vdash U \leq S} \qquad \dfrac{\vdots\ \Sigma_1}{\Gamma \vdash S \leq V}}{\Gamma \vdash U \leq V} \qquad W[X := U] \Rrightarrow W'}{\Gamma \vdash e\{U\} : W'}$$

Case ((coer Sel).)

1. $S = [\cdots, l{:}T, \cdots]\ ,\ \Gamma \vdash e : [\cdots, l{:}U, \cdots]$

$$\dfrac{\dfrac{\dfrac{\vdots\ \Pi_1}{\Gamma \vdash e : [\cdots, l{:}U, \cdots]} \qquad \dfrac{\dfrac{\vdots\ \Sigma_1}{\cdots \quad \Gamma \vdash U \leq T \quad \cdots}}{\Gamma \vdash [\cdots, l{:}U, \cdots] \leq [\cdots, l{:}T, \cdots]}}{\Gamma \vdash (e|_{[\cdots, l:T, \cdots]}) : [\cdots, l{:}T, \cdots]}}{\Gamma \vdash (e|_{[\cdots, l:T, \cdots]}) \cdot l : T}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\vdots\ \Pi_1}{\Gamma \vdash e : [\cdots, l{:}U, \cdots]} \qquad \dfrac{\vdots\ \Sigma_1}{\Gamma \vdash U \leq T}}{\Gamma \vdash (e \cdot l)|_T}$$

2. ($S = [\cdots, l{:}T, \cdots]\ ,\ \Gamma \vdash e : P$)

$$\dfrac{\dfrac{\dfrac{\vdots\ \Pi_1}{\Gamma \vdash e : P} \qquad \dfrac{\dfrac{\vdots\ \Sigma_1}{\cdots \quad \Gamma \vdash S \leq T \quad \cdots}}{\Gamma \vdash [\cdots, l{:}S, \cdots] \leq [\cdots, l{:}T, \cdots]}\ \text{Applying (Taut)}}{\Gamma \vdash P \leq [\cdots, l{:}T, \cdots]}}{\Gamma \vdash (e|_{[\cdots, l:T, \cdots]}) : [\cdots, l{:}T, \cdots]}}{\Gamma \vdash (e|_{[\cdots, l:T, \cdots]}) \cdot l : T}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\vdots\ \Pi_1}{\Gamma \vdash e : P} \qquad \dfrac{\dfrac{\vdots\ \Sigma_1}{\text{Sel}([\cdots, l{:}S, \cdots]) \Rrightarrow S \quad \Gamma \vdash S \leq T}}{\Gamma \vdash \text{Sel}(P, l) \leq T}}{\Gamma \vdash e \cdot l : \text{Sel}(P, l)}}{\Gamma \vdash (e \cdot l)|_T}$$

3. ($S = Q$, $\Gamma \vdash e : P$)

$$\dfrac{\dfrac{\dfrac{\vdots\ \Pi_1}{\Gamma \vdash e : P} \qquad \dfrac{\vdots\ \Sigma_1}{\Gamma \vdash P \leq Q}}{\Gamma \vdash (e|_Q) : Q} \qquad \dfrac{\vdots\ \Sigma_2}{\Gamma \vdash Q \leq [\cdots, l{:}T, \cdots]}}{\Gamma \vdash (e|_Q) \cdot l : \mathrm{Sel}(Q, l)}$$

$$\Downarrow$$

$$\dfrac{\boxed{\mathbf{I}} \quad \dfrac{\dfrac{\vdots\ \Sigma_1}{\Gamma \vdash P \leq Q} \qquad \dfrac{\dfrac{\vdots\ \Sigma_2}{\Gamma \vdash Q \leq [\cdots, l{:}T, \cdots]}}{\Gamma \vdash \mathrm{Sel}(Q, l)\ \ \mathsf{type}}}{\Gamma \vdash \mathrm{Sel}(P, l) \leq \mathrm{Sel}(Q, l)}}{\Gamma \vdash (e \cdot l)|_{\mathrm{Sel}(P,l)} : \mathrm{Sel}(P, l)}$$

where **I** is

$$\dfrac{\dfrac{\vdots\ \Pi_1}{\Gamma \vdash e : P} \qquad \dfrac{\dfrac{\dfrac{\vdots\ \Sigma_1}{\Gamma \vdash P \leq Q} \qquad \dfrac{\vdots\ \Sigma_2}{\Gamma \vdash Q \leq [\cdots, l{:}T, \cdots]}}{\Gamma \vdash P \leq [\cdots, l{:}T, \cdots]}}{\Gamma \vdash \mathrm{Sel}(P, l)\ \ \mathsf{type}}}{\Gamma \vdash e \cdot l : \mathrm{Sel}(P, l)}$$

∎

Theorem 3.3.9. (Chrch-Rosser)
For each well-typed expression e , if $\Gamma \vdash e \ \rightarrow^* \ e'$ and $\Gamma \vdash e \ \rightarrow^* \ e''$, there exists an expression f where $\Gamma \vdash e' \ \rightarrow^* \ f$ and $\Gamma \vdash e'' \ \rightarrow^* \ f$ are satisfied.

(**Proof**): Easily proved from those substitution lemmas using parallel reduction[Tak92].

$$\frac{}{\Gamma \vdash x^T \ \triangleright x^T} \ (\text{PFVar})$$

$$\frac{}{\Gamma \vdash c^T \ \triangleright c^T} \ (\text{PFConst})$$

$$\frac{\Gamma \vdash e \ \triangleright e'}{\Gamma \vdash \lambda x^T.e \ \triangleright \lambda x^T.e'} \ (\text{PF}\lambda\text{ab})$$

$$\frac{\Gamma \vdash e \ \triangleright e' \quad \Gamma \vdash f \ \triangleright f'}{\Gamma \vdash ef \ \triangleright e'f'} \ (\text{PF}\lambda\text{ap})$$

$$\frac{\Gamma, X \leq T \vdash e \ \triangleright e'}{\Gamma \vdash \Lambda X \leq T.e \ \triangleright \Lambda X \leq T.e'} \ (\text{PF}\Lambda\text{ab})$$

$$\frac{\Gamma \vdash e \ \triangleright e'}{\Gamma \vdash e\{T\} \ \triangleright e'\{T\}} \ (\text{PF}\Lambda\text{ap})$$

$$\frac{\Gamma \vdash e_1 \ \triangleright e_1' \quad \cdots \quad \Gamma \vdash e_n \ \triangleright e_n'}{\Gamma \vdash \{l_1{=}e_1, \cdots, l_n{=}e_n\} \ \triangleright \{l_1{=}e_1', \cdots, l_n{=}e_n'\}} \ (\text{PFRecordConst})$$

$$\frac{\Gamma \vdash e \ \triangleright e'}{\Gamma \vdash e \cdot l \ \triangleright e' \cdot l} \ (\text{PFRecordSel})$$

$$\frac{\Gamma \vdash e \ \triangleright e'}{\Gamma \vdash e|_T \ \triangleright e'|_T} \ (\text{PFCoerce})$$

$$\frac{\Gamma \vdash e \ \triangleright e' \quad \Gamma \vdash f \ \triangleright f}{\Gamma \vdash (\lambda x^S.e)f \ \triangleright e'[x = f'|_S]} \ (\text{PR}\beta)$$

$$\frac{\Gamma, X \vdash e \ \triangleright e' \quad e'[X := S] \Rrightarrow e''}{\Gamma \vdash (\Lambda X.e)\{S\} \ \triangleright e''} \ (\text{PR}\beta 2)$$

$$\frac{\Gamma, X \leq S \vdash e \ \triangleright e' \quad e'[X := T] \Rrightarrow e''}{\Gamma \vdash (\Lambda X \leq S.e)\{T\} \ \triangleright e''} \ (\text{PRbound}\beta 2)$$

$$\frac{\Gamma \vdash \{l_1{=}e_1, \cdots, l_i{=}e_i, \cdots, l_n{=}e_n\} \ \triangleright \{l_1{=}e_1', \cdots, l_i{=}e_i', \cdots, l_n{=}e_n'\}}{\Gamma \vdash \{l_1{=}e_1, \cdots, l_i{=}e_i, \cdots, l_n{=}e_n\} \cdot l_i \ \triangleright e_i'} \ (\text{PRSel})$$

$$\frac{\Gamma \vdash f : T \quad \text{Range}(S) \Rrightarrow V \quad \Gamma \vdash e \ \triangleright e' \quad \Gamma \vdash f \ \triangleright f'}{\Gamma \vdash (e|_S)f \ \triangleright (e'f')|_V} \ (\text{PRcoer})$$

$$\frac{\Gamma \vdash e : \forall X \leq V.W \quad \Gamma \vdash e \ \triangleright e' \quad T[X := U] \Rrightarrow T'}{\Gamma \vdash (e|_{\forall X \leq S.T})\{U\} \ \triangleright (e'\{U\})|_{T'}} \ (\text{PRcoer2})$$

$$\frac{\text{Sel}(S, l) \Rrightarrow V \quad \Gamma \vdash e \ \triangleright e'}{\Gamma \vdash (e|_S) \cdot l \ \triangleright (e' \cdot l)|_V} \ (\text{PRcoerSel})$$

$$\frac{\Gamma \vdash e \text{ is in normal form} \quad S \text{ is a base type.}}{\Gamma \vdash e|_S \ \triangleright e} \ (\text{PRcoerconst})$$

Lemma 3.3.10.

1. $\Gamma \vdash e \ \triangleright e$
2. If $\Gamma \vdash e \ \rightarrow \ e'$, then $\Gamma \vdash e \ \triangleright e'$.
3. If $\Gamma \vdash e \ \triangleright e'$, then $\Gamma \vdash e \ \rightarrow^* \ e'$.

4. If $\Gamma \vdash e \rhd e'$, $\Gamma \vdash f \rhd f'$, and $\Gamma \vdash f : S$ then $\Gamma \vdash e[x^S := f] \rhd e'[x^S := f']$.
5. If $\Gamma, X \leq S \vdash e \rhd e'$, $\Gamma \vdash T \leq S$, $e[X := T] \equiv\!\!\!\!\!\!\equiv f$ and $e'[X := T] \equiv\!\!\!\!\!\!\equiv f'$ hold,then $\Gamma \vdash f \rhd f'$.

(**Proof**):

- (1,2,3) Obvious.
- (4,5) From substitution lemmas.

∎

Lemma 3.3.11.
Under an environment Γ , for a term g , we define g^* to be the term constructed by applying the rules of the name (P-R*) when possible, and the rules of the name (P-F*) when no (P-R*) can be applied. g^* is uniquely determined and if $\Gamma \vdash g \rhd g'$, then $\Gamma \vdash g' \rhd g^*$.

(**Proof**): g^* is uniquely determined because none of the (P-R*) rules overlap. The reset of the theorem is proved using structural induction on the form of the term g .

Case (x .)
Obvious

Case ($\lambda x^S.e$.)
Then g has the form $\lambda x^S.e$. Then g' has the form $\lambda x^S.e'$ for some e' to which $\Gamma \vdash e \rhd e'$. By induction hypothesis, $\Gamma \vdash e' \rhd e^*$. Therefore $\Gamma \vdash \lambda x^S.e' \rhd \lambda x^S.e^* = (\lambda x^S.e)^*$

Case ($\Lambda X{\leq}S.e$.)
Then g has the form $\Lambda X{\leq}S.e$. Then g' has the form $\Lambda X{\leq}S.e'$ for some e' to which $\Gamma, X \leq S \vdash e \rhd e'$. By induction hypothesis, $\Gamma, X \leq S \vdash e' \rhd e^*$. Therefore $\Gamma \vdash \Lambda X{\leq}S.e' \rhd \Lambda X{\leq}S.e^* = (\Lambda X{\leq}S.e)^*$

Case ($e_0 f$.)
g has the form $e_0 f$. Then there are two subcases.

1. g has the form $(\lambda x^S.e)f$. Then g^* is $e^*[x^S := f^*|_S]$.
   - g' has the form $(\lambda x^S.e')f'$. Then from induction, $\Gamma \vdash e' \rhd e^*$ and $\Gamma \vdash f' \rhd f^*$. Thus from definition, $(\lambda x^S.e')f' \rhd e^*[x^S := f^*|_S]$.
   - g' has the form $e'[x^S := f'|_S]$. Then from induction, $\Gamma \vdash e' \rhd e^*$ and $\Gamma \vdash f' \rhd f^*$, which implies $\Gamma \vdash f'|_S \rhd f^*|_S$ Thus from definition and Proposition 3.3.6, $e'[x^S := f'|_S] \rhd e^*[x^S := f^*|_S]$.
2. g has the form $(e|_P)f$. Then g^* is $(e^* f^*)|_V$ where $\mathrm{Range}(P) \equiv\!\!\!\!\!\!\equiv V$. The proof is similar to previous cases.

Case ($e_0\{T\}$.)
g has the form $e_0\{T\}$. Then there are two subcases.

1. g has the form $(\Lambda X{\leq}S.e)\{T\}$. Then g^* is f^* where $e \equiv\!\!\!\!\!\!\equiv e^*$ and $e^*[X := T] \equiv\!\!\!\!\!\!\equiv f^*$.
   - g' has the form $(\Lambda X{\leq}S.e')\{T\}$. Then from induction, $\Gamma, X \leq S \vdash e' \rhd e^*$. Thus from definition, $(\Lambda X^S.e')\{T\} \rhd f^*$.
   - g' has the form f' where $e'[X := T] \equiv\!\!\!\!\!\!\equiv f'$. Then from induction, $\Gamma, X \leq S \vdash e' \rhd e^*$. Thus from definition and Proposition 3.3.6, $f' \rhd f^*$.
2. g has the form $(e|_{\forall X \leq S', U'})\{T\}$ where $\Gamma \vdash e : \forall X{\leq}S.U \leq \forall X{\leq}S'.U'$. Then g^* is $(e^*[X := T])|_{U'[X:=T]}$. The proof is similar.

Case ($e_0|_S$.)
Similarly to (*Case* ($e_0 f$))

∎

Then we have that $cF^+_{\leq}$ satisfies the Church-Rosser property. ∎

Theorem 3.3.12. (Strong Normalization)
All type-checked terms in $cF^+_{\leq}$ is strongly normalizing.

The proof of this theorem is too long and discussed in Section 3.4.

In $cF^+_{\leq}$, under an environment Γ when f has type S , functions f and $f|_S$ are expected to be the same function because the coercion function from S to S is identity. However, there is no reduction rule which reduces $f|_S$ to f , or f to $f|_S$. Actually, we do not expect that $cF^+_{\leq}$ is extensional from the beginning bacause we did not even have the η -reduction.

Thouth f and $f|_S$ are not equal, we expect that thery behave equally. For the same reason, we expect that coercion are transitive. That is $f|_G|_U$ and $f|_U$ behave equally. For this purpose, we define a notion of equivaence between terms, which is sometimes called applicable congruence [Blo90, Plo77, Mil77].

Definition 3.3.13. (Applicable Congruence)
Two closed terms e and f of the same type are *applicably congruent* ($e =_{\mathrm{AP}} f$) if for every sequent of applications and slot selections $g_1, \ldots, g_n$ (g_i can be a term or a type) deriving e and f to base types, $eg_1 \ldots g_n$ reduces to a normal form c iff $fg_1 \ldots g_n$ reduces to c .

From this proposition and the Chrch-Rosser theorem, every closed expression of a base type is reduced to a unique constant, or do not have a normal form. This justifies the above definition of equality between terms.

Theorem 3.3.14.

1. When $\Gamma \vdash f : S$ holds and f is closed, then f and $f|_S$ are applicably congruent.
2. When $\Gamma \vdash f : S$, $\Gamma \vdash S \leq T$, and $\Gamma \vdash T \leq U$ hold and f is closed, then $f|_T|_U$ and $f|_U$ are applicably congruent.

(**Proof**): (1,2) Easily proved by induction on the degree of S . ∎

3.4 Strong Normalization

We here show the proof of the strong normalizability of $cF^+_{\leq}$ (see Theorem 3.3.12).

3.4.1 Outline of the Proof

We discuss about the proof of the strong normalizability of $cF^+_{\leq}$. First, we prove the strong normalization of all type-erasure of $cF^+_{\leq}$ terms, using the notion of saturated sets which are given in [Gal90, Tai75]. Saturated sets are like candidates of reducibility which are used in the proof of the termination of F terms, and they also give semantics of $cF^+_{\leq}$. We will provide a modified method using in [Ghe90]. After the strong normalization of all type-erasure of $cF^+_{\leq}$ terms, we prove the strong normalization of $cF^+_{\leq}$ terms.

3.4.2 Untyped Translation

Definition 3.4.1.
We define a target of *untyped* λ*-terms+record* Λ_{rec}.

$$
\begin{array}{rcll}
e & ::= & x & (x \in \mathcal{X},\ \mathcal{X} \text{ is a set of all variables}) \\
 & | & c & (\text{constant}) \\
 & | & \lambda x.e & (\lambda\text{-abstraction}) \\
 & | & ee' & (\lambda\text{-application}) \\
 & | & \{l_1{=}e_1, \cdots, l_n{=}e_n\} & (\text{record: which can be abbreviated} \\
 & & & \quad \{l_i{=}e_i\}_{i=1,\ldots,n}) \\
 & & (l_i \in L \text{ and for all } i \neq j, \text{ then } l_i \neq l_j) & \\
 & | & e \cdot l & (\text{record-field selection})
\end{array}
$$

We define *reduction* rules:

$$
\begin{array}{lrcl}
(\beta) & (\lambda x.e)f & \Rightarrow & e[x := f] \\
(\text{Sel}) & \{l_i{=}e_i\}_{i=1,\ldots,n} \cdot l_k & \Rightarrow & e_k \\
 & & & (\text{where } k = 1, \ldots, n)
\end{array}
$$

Notation 3.4.2.
In this section el where (e is a term and $l\,(\in L)$ a label of record) means $e \cdot l$ because sometimes we do not want to distinguish between record selection and term application.

Definition 3.4.3.
We define a translation Erase(e) from $cF^{+}_{\leq}$ to untyped λ-term Λ_{rec}.

$$
\begin{array}{lcl}
\text{Erase}(x^T) & = & x \\
\text{Erase}(c^T) & = & c \\
\text{Erase}(\lambda x^T.e) & = & \lambda x.\text{Erase}(e) \\
\text{Erase}(\Lambda X \leq T.e) & = & \text{Erase}(e) \\
\text{Erase}(ef) & = & \text{Erase}(e)\text{Erase}(f) \\
\text{Erase}(e\{S\}) & = & \text{Erase}(e) \\
\text{Erase}(\{l_i{=}e_i\}_{i=1,\ldots,n}) & = & \{l_i{=}\text{Erase}(e_i)\}_{i=1,\ldots,n} \\
\text{Erase}(e \cdot l) & = & \text{Erase}(e) \cdot l \\
\text{Erase}(e|_T) & = & \text{Erase}(e)
\end{array}
$$

3.4.3 Termination of Type-erasure of $cF^{+}_{\leq}$ Terms

We prove the strong normalization property of translated terms of Λ_{rec} from $cF^{+}_{\leq}$ terms. We call them *type erasures*.

Definition 3.4.4.
(Space Constructor)

$$[S \rightarrow T] \quad = \quad \{e \in \Lambda_{\text{rec}} \mid \forall f \in S \ \Rightarrow \ ef \in T\}$$

Definition 3.4.5.

- (Application Constructor)

$$AP(P,U) \quad = \quad \{e_0 f | e_0 \in P \text{ and } f \in U\}$$

- (Selection Constructor)

$$SEL(P,l) \quad = \quad \{e_0 \cdot l | e_0 \in P\}$$

Notation 3.4.6.
For every term e of Λ_{rec} which is strongly normalizing, we denote the maximum of length of reduction chain of e as $\nu(e)$. For every label $l \in L$, we consider that $\nu(l) = 0$.

Definition 3.4.7.
Let SN be a set of Λ_{rec}-terms which are strongly normalizing. A subset $C \subseteq SN$ is *saturated* iff the following conditions hold:

(S0) If $e \in C$ and $e \to f$ hold, then $f \in C$.
(S1) For every variable $x \in \mathcal{X}$, for all $n \geq 0$ and all $f_1, \ldots, f_n \in SN \cup L$, $xf_1 \ldots f_n \in C$
(S2) For all $e, f \in SN$, for all $n \geq 0$ and all $f_1, \ldots, f_n \in SN \cup L$, if $e[x := f]f_1 \ldots f_n \in C$, then $(\lambda x.e)ff_1 \ldots f_n \in C$
(S3) For all $e \in SN$, for all $n, m \geq 0$, all $f_1, \ldots, f_n \in SN \cup L$, all $e_1, \ldots, e_m \in SN$, if $ef_1 \ldots f_n \in C$, then $\{l_1{=}e_1, \ldots, l_i{=}e_i, l{=}e, l_{i+1}{=}e_{i=1}, \cdots l_m{=}e_m\} \cdot lff_1 \ldots f_n \in C$

SAT is the set of all saturated sets.

Lemma 3.4.8.
$SN \in SAT$

(**Proof**):

(S0) Obvious.
(S1) By induction on the sum of $\nu(f_i)$. Consider any reduction chain starting with $xf_1 \ldots f_n$: $xf_1 \ldots f_i \ldots f_n \Rightarrow xf_1 \ldots f_i' \ldots f_n$ where $f_i \Rightarrow f_i'$ and so it is strongly normalizing, but its depth is strictly smaller than $\nu(f_i)$. Then we can apply induction to $f_1, \ldots, f_i', \ldots, f_n$ obtaining that $xf_1 \ldots f_i' \ldots f_n$ is strongly normalizing, and so is $xf_1 \ldots f_i \ldots f_n$.
(S2) By induction on the sum of $\nu(e[x := f]f_1 \ldots f_m)$ and $\nu(f)$. Suppose $e[x := f]f_1 \ldots f_m \in SN$ and consider any reduction chain starting with $(\lambda x.e)ff_1 \ldots f_m$; we must prove that this chain is finite. We have the following possibilities for the first step of this chain:

1.	$(\lambda x.e)ff_1 \ldots f_m$	$\Rightarrow$	$e[x := f]f_1 \ldots f_m$	
2.	$(\lambda x.e)ff_1 \ldots f_m$	$\Rightarrow$	$(\lambda x.e')ff_1 \ldots f_m$	$e \Rightarrow e'$
3.	$(\lambda x.e)ff_1 \ldots f_i \ldots f_m$	$\Rightarrow$	$(\lambda x.e)ff_1 \ldots f_i' \ldots f_m$	$f_i \Rightarrow f_i'$ $(i = 1, \ldots, m)$
4.	$(\lambda x.e)ff_1 \ldots f_m$	$\Rightarrow$	$(\lambda x.e)f'f_1 \ldots f_m$	$f \Rightarrow f'$

We show that in all the four cases the reduced term is in SN. In case 1 the reduced term belongs to SN by hypothesis. In case 2, observe $e'[x := f]f_1 \ldots f_m$ is a reduct of $e[x := f]f_1 \ldots f_m$, and then belongs to SN. But $\nu(e'[x := f]f_1 \ldots f_m)$ is smaller than $\nu(e[x := f]f_1 \ldots f_m)$; then we can apply to it the induction hypothesis and obtain that $(\lambda x.e')ff_1 \ldots f_m \in SN$. The same reasoning holds for case 3. In case 4, $e[x := f']f_1 \ldots f_m$ is a reduct of $e[x := f]f_1 \ldots f_m$. The induction measure $\nu(e[x := f]f_1 \ldots f_m) + \nu(f)$ is strictly decreased whether x is free in e or not, and the inductive hypothesis can be applied as in the preceding cases, obtaining $(\lambda x.e)f'f_1 \ldots f_m \in SN$.
(S3) Similar to (S2).

∎

Definition 3.4.9.
The interpretation of environments and types is defined by mutual induction.

$$
\begin{array}{lcl}
[\![\phi]\!] & = & SAT^0(\text{a singleton set}) \\
[\![\Gamma, X]\!] & = & \{\langle \vec{R}, C\rangle \mid \vec{R} \in [\![\Gamma]\!], C \in SN\} \\
[\![\Gamma, X \leq T]\!] & = & \{\langle \vec{R}, C\rangle \mid \vec{R} \in [\![\Gamma]\!], C \in SAT, C \subseteq [\![\Gamma \vdash T]\!]\vec{R}\}
\end{array}
$$

The interpretation of types is defined by induction on their size; we will avoid environments Γ and quantifications $\forall \vec{R} \in [\![\Gamma]\!]$, when not necessary, to make the notation less cumbersome:

$$
\begin{array}{lcl}
[\![\Gamma \vdash X_i]\!]\vec{R} & = & R_i \\
[\![\Gamma \vdash B]\!]\vec{R} & = & \bigcap_{C \in SAT, C \subseteq D} C \\
 & & \text{where } D = \{\mathrm{Erase}(e) \mid \Gamma \vdash e : B \text{ and } e \text{ is terminating.}\} \\
[\![S \to T]\!]\vec{R} & = & [[\![S]\!]\vec{R} \to [\![T]\!]\vec{R}] \\
[\![\Gamma \vdash \forall Y \leq S.T]\!]\vec{R} & = & \bigcap_{C \in SAT\ C \subseteq [\![\Gamma \vdash S]\!]\vec{R}} [\![\Gamma, Y \leq S \vdash T]\!](\vec{R}, Y := C) \\
[\![\ [l_1{:}T_1, \cdots, l_n{:}T_n]\ ]\!]\vec{R} & = & \{e \mid \text{For every } i = 1, \ldots, n,\ e \cdot l_i \in [\![T_i]\!]\vec{R}\} \\
[\![\Gamma \vdash \mathrm{Range}(P)]\!]\vec{R} & = & \bigcap_{C \in SAT,\ C \supseteq AP([\![P]\!]\vec{R}, [\![S]\!]\vec{R})} C \\
 & & \text{where } \Gamma^*(\mathrm{Range}(P)) = S \to T \\
[\![\mathrm{Sel}(P, l)]\!]\vec{R} & = & \bigcap_{C \in SAT,\ C \supseteq SEL([\![P]\!]\vec{R}, l)} C
\end{array}
$$

Type variables are interpreted by the type environment. A functional type is simply the type of all terms which is applied to a term of the domain give a term of the codomain. A bounded quantified type is interpreted by a bounded intersection. The substitution-like notation $(\vec{R}, Y := C)$ means simply $\langle \vec{R}, C \rangle$, but should be more suggestive about the fact that we extend the environment $\vec{R}$ with an interpretation C for the type variable Y. The definition is well formed because $(\vec{R}, Y := C) \in [\![\Gamma, Y \leq S]\!]$ follows immediately by the interpretation of type environments. A Range type is interpreted by a minimum saturated set which contains all terms which are given by applying a term of the contents of Range to its argument.

Firstly we must prove that for every type, its interpretation is saturated.

Lemma 3.4.10.
For any subset $\mathcal{C} \subseteq SAT$, $\bigcap_{C \in \mathcal{C}} C \in SAT$ holds.

(**Proof**): Obvious from Definition 3.4.7. ∎

Lemma 3.4.11.
For every type, its interpretation in Definition 3.4.9 is saturated.

(**Proof**): Induction on the structures of the proof of the type good-formatiom. We prove only the four cases: function type, record type, $\forall$-type, and Range-type.

Case ($T \to U$.)

(S0) Assume that $e \in [\![T \to U]\!]\vec{R}$ and $e \to e'$ hold. For every $f \in [\![T]\!]\vec{R}$, $ef \in [\![U]\!]\vec{R}$ holds. Since $ef \to e'f$, we have $e'f \in [\![U]\!]\vec{R}$. Then we have $e' \in [\![T \to U]\!]\vec{R}$.

(S1) For every variable x and for all $m \geq 0$ and $e \in SN$ and $e_1, \ldots, e_m \in SN \cup L$, we have $xe_1 \cdots e_m e \in [\![U]\!]\vec{R}$. Since this holds for every $e \in [\![S]\!]\vec{R}$, we have $xe_1 \cdots e_m \in [\![T \to U]\!]\vec{R}$

(S2) For every $f, e \in SN$,all $m \geq 0$ and $e_1, \ldots, e_m \in SN \cup L$ Assume that $e[x := f]e_1 \ldots e_m \in [\![T \to U]\!]\vec{R}$. This means that, for every $g \in [\![T]\!]\vec{R}$, $e[x := f]e_1 \ldots e_m g \in [\![U]\!]\vec{R}$ holds. Since $[\![U]\!]\vec{R}$ is saturated, by (S2) $(\lambda x.e)fe_1 \ldots e_m g \in [\![U]\!]\vec{R}$, and this is true for every g, we have $(\lambda x.e)fe_1 \ldots e_m \in [\![T \to U]\!]\vec{R}$.

(S3) For every $e \in SN$,all $m \geq 0$, $e_1, \ldots, e_m \in SN \cup L$, $n \geq 0$, and $e'_1, \ldots, e'_n \in SN$. Assume that $ee_1 \ldots e_m \in [\![T \to U]\!]\vec{R}$. This means for every $g \in [\![T]\!]\vec{R}$, $ee_1 \ldots e_m g \in [\![U]\!]\vec{R}$. Since $[\![U]\!]\vec{R}$ is saturated, by (S3) $(\{l_1{=}e'_1, \ldots, l_i{=}e'_i, l{=}e, l_{i+1}{:}e'_{i+1}, \ldots, l_n{=}e'_n\} \cdot l)e_1 \ldots e_m g \in [\![U]\!]\vec{R}$, and this is true for every $g \in [\![T]\!]\vec{R}$, we have $(\{\cdots, l{=}e, \cdots\} \cdot l)e_1 \ldots e_m \in [\![T \to U]\!]\vec{R}$.

Case ($[l_1{:}T_1, \cdots, l_n{:}T_n]$.)

(S0) Similar to the case of function type.

(S1) For every variable x and for all $m \geq 0$, $e_1, \ldots, e_m \in SN \cup L$, and $i = 1, \ldots, n$, we have $(xe_1 \cdots e_m) \cdot l_i \in [\![T_i]\!]\vec{R}$. Then, from definition of $[\![\ [l_1{:}T_1, \cdots, l_n{:}T_n]\ ]\!]\vec{R}$, we have $xe_1 \cdots e_m \in [\![\ [l_1{:}T_1, \cdots, l_n{:}T_n]\ ]\!]\vec{R}$

(S2) For every $f, e \in SN$, all $m \geq 0$ and $e_1, \ldots, e_m \in SN \cup L$, assume that $e[x := f]e_1 \ldots e_m \in [\![\ [l_1{:}T_1, \cdots, l_n{:}T_n]\ ]\!]\vec{R}$. This means that, for every $i = 1, \ldots, n$, $(e[x := f]e_1 \ldots e_m) \cdot l_i \in [\![T_i]\!]\vec{R}$. Since $[\![T_i]\!]\vec{R}$ is saturated, by (S2) $((\lambda x.e)fe_1 \ldots e_m) \cdot l_i \in [\![T_i]\!]\vec{R}$, and this is true for every i, we have $(\lambda x.e)fe_1 \ldots e_m \in [\![\ [l_1{:}T_1, \cdots, l_n{:}T_n]\ ]\!]\vec{R}$.

(S3) For every $e \in SN$, all $m \geq 0$ and $e_1, \ldots, e_m \in SN \cup L$, and all $n' \geq 0$ and $e'_1, \ldots, e'_{n'} \in SN \cup L$, assume that $ee_1 \ldots e_m \in [\![\ [l_1{:}T_1, \cdots, l_n{:}T_n]\ ]\!]\vec{R}$ holds. This means that, for every $i = 1, \ldots, n$, $(ee_1 \ldots e_m) \cdot l_i \in [\![T_i]\!]\vec{R}$. Since $[\![T_i]\!]\vec{R}$ is saturated, by (S2) $((\{l_1{=}e'_1, \ldots, l_j{=}e'_j, l{=}e, l_{j+1}{=}e'_{j+1}, \ldots, l_n{=}e'_{n'}\} \cdot l)e_1 \ldots e_n) \cdot l_i \in [\![T_i]\!]\vec{R}$, and this is true for every i, we have $(\{l_1{=}e'_1, \ldots, l_j{=}e'_j, l{=}e, l_{j+1}{=}e'_{j+1}, \ldots, l_n{=}e'_{n'}\} \cdot l)e_1 \ldots e_m \in [\![\ [l_1{:}T_1, \cdots, l_n{:}T_n]\ ]\!]\vec{R}$.

Case ($\forall Y{\leq}S.T$.)

(S0) Assume $e \in [\![\Gamma \vdash \forall X{\leq}S.T]\!]\vec{R}$ and $e \to e'$ hold. Then for every $C \subseteq [\![\Gamma \vdash S]\!]\vec{R}$ such that $C \in SAT$ holds, $e \in [\![\Gamma, Y \leq S \vdash T]\!](\vec{R}, Y := C)$. By induction hypothesis, we have $e' \in [\![\Gamma, Y \leq S \vdash T]\!](\vec{R}, Y := C)$. Then $e' \in [\![\Gamma, Y \leq S \vdash T]\!](\vec{R}, Y := C)$ holds.

(S1) Assume that $[\![\Gamma \vdash S]\!]\vec{R} \in SAT$ holds. For every variable x and for all $m \geq 0$ and $e_1, \ldots, e_m \in SN \cup L$, and $C \subseteq [\![\Gamma \vdash S]\!]\vec{R}$ such that $C \in SAT$ holds,
we have $xe_1 \cdots e_m e \in [\![\Gamma, Y \leq S \vdash T]\!](\vec{R}, Y := C)$. From the definition, $xe_1 \cdots e_m e \in \bigcap_V^{C \subseteq [\![\Gamma \vdash S]\!]\vec{R}, C \in SAT} [\![\Gamma, Y \leq S \vdash T]\!](\vec{R}, Y := C) \equiv [\![\Gamma \vdash \forall X{\leq}S.T]\!]\vec{R}$.

(S2) For every $f, e \in SN$, all $m \geq 0$ and $e_1, \ldots, e_m \in SN \cup L$ Assume that $e[x := f]e_1 \ldots e_n \in [\![\Gamma \vdash \forall X{\leq}S.T]\!]\vec{R}$.
This means for every $C \subseteq [\![\Gamma \vdash S]\!]\vec{R}$ such that $C \in SAT$ holds, $e[x := f]e_1 \ldots e_n \in [\![\Gamma, Y \leq S \vdash T]\!](\vec{R}, Y := C)$. Since $[\![\Gamma, Y \leq S \vdash T]\!](\vec{R}, Y := C)$ is saturated, by (S2) $(\lambda x.e)fe_1 \ldots e_n \in [\![\Gamma, Y \leq S \vdash T]\!](\vec{R}, Y := C)$, and this is true for every C, we have $(\lambda x.e)fe_1 \ldots e_n \in [\![\Gamma \vdash \forall X{\leq}S.T)]\!]$.

(S3) Similar to (S2)

Case (Range(P) .)
Assume that $\Gamma^*(P) = S \to T$. Since $\Gamma \vdash P \leq S \to T$, we have $[\![P]\!]\vec{R} \subseteq [\![S \to T]\!]\vec{R}$. Then $AP([\![P]\!]\vec{R}, [\![S]\!]\vec{R}) \subseteq [\![T]\!]\vec{R}$ And (S0) $\sim$ (S3) are trivial from Lemma 3.4.10.

∎

Secondly, we show the soundness of the interpretation via subtype relation.

Lemma 3.4.12.
For every proof of subtype relation $\Gamma \vdash S \leq T$, $[\![\Gamma \vdash S]\!]\vec{R} \subseteq [\![\Gamma \vdash T]\!]\vec{R}$ holds.

(**Proof**): Induction on the structure of the proof of the subtype relation.

Case ($\leq$-Refl.)
Trivial.

Case ($\leq$-Trans.)
Obvious from assumptions.

Case ($\leq$-Taut.)

We only need to prove $[\![P]\!]\vec{R} \leq [\![\Gamma(P)]\!]\vec{R}$ holds.

We can consider three cases.

1. In the case that $P = X$ and $(X \leq T) \in \Gamma$ hold, it is proved from Definition 3.4.9.
2. In the case that $P = \mathrm{Range}(P')$ and $\Gamma(P') = S \to T$ hold, we have $\Gamma(P) = \Gamma^*(P) = T$. Then, by induction hypothesis, we have $[\![P']\!]\vec{R} \subseteq [\![S \to T]\!]\vec{R}$. Since $AP([\![P']\!]\vec{R}, [\![S]\!]\vec{R}) \subseteq [\![T]\!]\vec{R}$ and $[\![T]\!]\vec{R} \in SAT$ hold, we have $[\![\mathrm{Range}(P')]\!]\vec{R} \subseteq [\![T]\!]\vec{R}$.
3. In the case where $P = \mathrm{Range}(P')$ and $\Gamma(P') = Q'$ hold, we have $\Gamma^*(P') = \Gamma^*(Q')$. Then we have easily $[\![\mathrm{Range}(P')]\!]\vec{R} \subseteq [\![\mathrm{Range}(Q')]\!]\vec{R}$.

Case ($\leq$-$\to$.)

$$\frac{\Gamma \vdash S' \leq S \quad \Gamma \vdash T \leq T'}{\Gamma \vdash S \to T \leq S' \to T'} \ (\leq\text{-}\to)$$

From assumptions, $[\![S']\!]\vec{R} \subseteq [\![\Gamma \vdash S]\!]\vec{R}$ and $[\![T]\!]\vec{R} \subseteq [\![T']\!]\vec{R}$ hold. For every $f \in [\![S \to T]\!]\vec{R}$ and $e \in [\![S']\!]\vec{R}$, $e \in [\![S]\!]\vec{R}$ holds, and therefore, $fe \in [\![T]\!]\vec{R} \subseteq [\![T']\!]\vec{R}$ holds. Hence, $f \in [\![S' \to T']\!]\vec{R}$ holds.

Case (($\leq$-(bb) $\forall$).)

$$\frac{\Gamma \vdash \forall Y{\leq}S.T \ \ \mathsf{type} \quad \Gamma \vdash S' \leq S \quad \Gamma, X \leq S' \vdash T \leq T'}{\Gamma \vdash \forall Y{\leq}S.T \leq \forall Y{\leq}S'.T'} \ (\leq\text{-bb}\,\forall)$$

From assumptions, $[\![\Gamma \vdash S']\!]\vec{R} \subseteq [\![\Gamma \vdash S]\!]\vec{R}$ and for every $C \subseteq [\![\Gamma \vdash S']\!]\vec{R}$ such that $C \in SAT$ holds, $[\![\Gamma, Y \leq S' \vdash T]\!](\vec{R}, Y := C) \subseteq [\![\Gamma, Y \leq S' \vdash T']\!](\vec{R}, Y := C)$ holds. For every $e \in [\![\Gamma \vdash \forall Y{\leq}S.T]\!]\vec{R}$, for every $C \subseteq [\![\Gamma \vdash S]\!]\vec{R}$ such that $C \in SAT$ holds, $e \in [\![\Gamma, Y \leq S' \vdash T]\!](\vec{R}, Y := C)$, then since $\Gamma \vdash S' \leq S$, for every $C \subseteq [\![\Gamma \vdash S']\!]\vec{R}$ such that $C \in SAT$ holds, $e \in [\![\Gamma, Y \leq S' \vdash T]\!](\vec{R}, Y := C) \subseteq [\![\Gamma, Y \leq S \vdash T']\!](\vec{R}, Y := C) = [\![\Gamma, Y \leq S' \vdash T']\!](\vec{R}, Y := C)$ holds, and therefore $e \in [\![\Gamma \vdash \forall Y{\leq}S'.T']\!]\vec{R}$ holds.

Case ($\leq$-record.)

$$\frac{\text{For all } i = 1, \cdots, n\,,\ \Gamma \vdash S_i \leq T_i}{\left\{ \begin{array}{l} \Gamma \vdash [l_1{:}S_1, \cdots, l_n{:}S_n, l_{n+1}{:}S_{n+1}, \cdots, l_{n+m}{:}S_{n+m}] \\ \qquad\qquad \leq [l_1{:}T_1, \cdots, l_n{:}T_n] \end{array} \right.} \ (\leq\text{-record})$$

Assume $e \in [\![\ [l_1{:}S_1, \cdots, l_n{:}S_n, l_{n+1}{:}S_{n+1}, \cdots, l_{n+m}{:}S_{n+m}]\ ]\!]\vec{R}$. By definition of $[\![\ ]\!]$, for all $i = 1, \ldots n$, $e \cdot l_i \in [\![S_i]\!]\vec{R}$ holds. By induction hypothesis, we have $e \cdot l_i \in [\![T_i]\!]\vec{R}$. Therefore we have $e \in [\![\ [l_1{:}T_1, \cdots, l_n{:}T_n]\ ]\!]\vec{R}$.

Case (Range .)

$$\frac{\Gamma \vdash \mathrm{Range}(P) \ \ \mathsf{type} \quad \Gamma \vdash \mathrm{Range}(Q) \ \ \mathsf{type} \quad \Gamma \vdash P \leq Q}{\Gamma \vdash \mathrm{Range}(P) \leq \mathrm{Range}(Q)} \ (\mathrm{Range})$$

Because Range can be derived from ($\leq$-Taut) by Proposition 3.2.15, we can prove it similarly.

Case (Sel .)

$$\frac{\Gamma \vdash \mathrm{Sel}(P, l) \ \ \mathsf{type} \quad \Gamma \vdash \mathrm{Sel}(Q, l) \ \ \mathsf{type} \quad \Gamma \vdash P \leq Q}{\Gamma \vdash \mathrm{Sel}(P, l) \leq \mathrm{Sel}(Q, l)} \ (\mathrm{Sel})$$

Immediate from $[\![P]\!]\vec{R} \subseteq [\![Q]\!]\vec{R}$ from definitions.

∎

Lemma 3.4.13.

1. $[\![\mathrm{Range}(S \to T)]\!]\vec{R} \equiv [\![T]\!]\vec{R}$
2. $[\![\mathrm{Sel}([\ldots, l{:}T, \ldots], l)]\!]\vec{R} \equiv [\![T]\!]\vec{R}$

(**Proof**): We prove only the first case. The proof of $\subset$ is easy. We prove $\supset$. For every $e \in [\![T]\!]\vec{R}$, $x \notin \mathrm{FV}(e)$, and every $f \in [\![S]\!]\vec{R}$, we have $e[x := f] = e \in [\![T]\!]\vec{R}$. Therefore $(\lambda x.e)f \in [\![T]\!]\vec{R}$ from (S2) and we have $\lambda x.e \in [\![S \to T]\!]\vec{R}$. We can provide a new variable y such that $x \neq y$ and $y \notin \mathrm{FV}(e)$ hold, and from (S1) $y \in [\![S]\!]\vec{R}$. Then we have $(\lambda x.e)y \in [\![\mathrm{Range}(S \to T)]\!]\vec{R}$, and therefore from (S0), we have also $e \in [\![\mathrm{Range}(S \to T)]\!]\vec{R}$. ∎

Finnally we must show that if $\Gamma \vdash e : T$ holds, then we have for every interpretation $\vec{R} \in [\![\Gamma]\!]$, $\mathrm{Erase}(e) \in [\![\Gamma \vdash T]\!]\vec{R}$. (Therefore $\mathrm{Erase}(e)$ is always strongly normalizing.)

Lemma 3.4.14.
For every proof $\Gamma \vdash e : S$, if for every type substitution $\phi : \mathrm{FV}(e) \to \Lambda$, if $\phi(x) \in [\![\Gamma \vdash T]\!]\vec{R}$ for every $x^T \in \mathrm{FV}(e)$, then $\phi(\mathrm{Erase}(e)) \in [\![\Gamma \vdash S]\!]\vec{R}$.

Notation 3.4.15.
For every type substitution ϕ, we define the type substitution $\phi[x := e]$ as follows:

$$\begin{array}{lcl} \phi[x := e](x) & = & e \\ \phi[x := e](y) & = & \phi(y) \quad (y \neq x) \end{array}$$

(**Proof**):
Induction on the structure of the proof of the type-assignment.

Case (var.)
Obvious from definitions.

Case (λ-ap.)

$$\frac{\Gamma \vdash e : S \to T \quad \Gamma \vdash f : U \quad \Gamma \vdash U \leq S}{\Gamma \vdash ef : T} \ (\lambda\text{-ap})$$

By induction hypothesis, we have $\phi(\mathrm{Erase}(e)) \in [\![S \to T]\!]\vec{R}$ and $\phi(\mathrm{Erase}(f)) \in [\![U]\!]\vec{R} \subseteq [\![S]\!]\vec{R}$. By definition of $[\![S \to T]\!]\vec{R}$, we have $(\mathrm{Erase}(e))(\mathrm{Erase}(f)) \in [\![T]\!]\vec{R}$

Case (λ - Range .)

$$\frac{\Gamma \vdash e : P \quad \Gamma \vdash f : U \quad \Gamma^*(P) = S \to T \quad \Gamma \vdash U \leq S}{\Gamma \vdash ef : \mathrm{Range}(P)} \ (\lambda\text{ - Range})$$

By induction hypothesis, we have $\phi(\mathrm{Erase}(e)) \in [\![P]\!]\vec{R}$ and $\phi(\mathrm{Erase}(f)) \in [\![U]\!]\vec{R} \subseteq [\![S]\!]\vec{R}$. By definition of $[\![\mathrm{Range}(P)]\!]\vec{R}$, since $\mathrm{Erase}(ef) = \mathrm{Erase}(e)\mathrm{Erase}(f) \in AP([\![P]\!]\vec{R}, [\![S]\!]\vec{R})$ holds, we have $\mathrm{Erase}(ef) \in [\![\mathrm{Range}(P)]\!]\vec{R}$.

Case (λ-ab.)

$$\frac{\Gamma \vdash f : T}{\Gamma \vdash (\lambda x^S.f) : S \to T} \ (\lambda\text{-ab})$$

We have $\mathrm{FV}(\lambda x^S.f) = \mathrm{FV}(f) - \{x\}$. Let ϕ be any substitution with domain $\mathrm{FV}(f) - \{x\}$, let e be any term in $[\![S]\!]\vec{R}$. By α-conversion, we always assume that $\mathrm{FV}(e)$ is disjoint from $\mathrm{FV}(f)$. Then, we form the subsumption $\phi[x := e]$ with domain $\mathrm{FV}(f)$. By induction hypothesis, we have $\phi[x := e](\mathrm{Erase}(f)) \in [\![T]\!]\vec{R}$. Then by choosing $x = e$, we have $\mathrm{Erase}(f) \in [\![T]\!]\vec{R}$. Note that $\phi[x := e](\mathrm{Erase}(f)) = \phi(\mathrm{Erase}(f))[x := e]$. Thus $\phi(\mathrm{Erase}(f))[x := e] \in [\![T]\!]\vec{R}$. We can apply (S2), and then we have $(\lambda x.\mathrm{Erase}(f))e \in [\![T]\!]\vec{R}$. By the definition of $[\![S \to T]\!]$, we have $\phi(\mathrm{Erase}(\lambda x^S.f)) = \lambda x.(\mathrm{Erase}(f)) \in [\![S \to T]\!]\vec{R}$

Case ((bound-) Λ-ap-w.)

$$\frac{\Gamma \vdash e : \forall Y{\leq}S.T \quad \Gamma \vdash V \leq S \quad T[X := V] \trianglerighteq T'}{\Gamma \vdash e\{V\} : T'} \ (\text{bound-}\Lambda\text{-ap-w})$$

By induction hypothesis, $\phi(\mathrm{Erase}(e)) \in [\![\forall Y{\leq}S.T]\!]\vec{R}$. Since the definition of $[\![\forall Y{\leq}S.T]\!]\vec{R}$, we have $\mathrm{Erase}(e\{V\}) \in [\![\Gamma, Y \leq S \vdash T]\!](\vec{R}, Y := [\![\Gamma \vdash V]\!]\vec{R})$. Therefore $\phi(e\{V\}) = \phi(e) \in [\![\Gamma \vdash T[Y := V]]\!]\vec{R}$.

By Lemma 3.4.13 , we have $[\![\Gamma \vdash T[Y := V]]\!]\vec{R} = [\![\Gamma \vdash T']\!]$. So we have $\phi(e\{V\}) \in [\![\Gamma \vdash T']\!]\vec{R}$.

Case (bound-Λ-ab.)

$$\frac{\Gamma, Y \leq S \vdash f : T}{\Gamma \vdash (\Lambda Y{\leq}S.f) : \forall Y{\leq}S.T} \ (\text{bound-}\Lambda\text{-ab})$$

In this rule, no free variable of type Y occurs in f.

By induction hypothesis, for every $C \in SAT$ such that $C \subseteq [\![\Gamma \vdash S]\!]\vec{R}$, $\phi(\mathrm{Erase}(f)) \in [\![\Gamma, Y \leq S \vdash T]\!](\vec{R}, Y := C)$, that is, $\mathrm{Erase}(\Lambda Y{\leq}S.f) = \mathrm{Erase}(f) \in \bigcap_{C \in SAT,\ C \subseteq [\![\Gamma \vdash S]\!]\vec{R}} [\![\Gamma, Y \leq S \vdash T]\!](\vec{R}, Y := C) = [\![\Gamma \vdash \forall X{\leq}S.T]\!]\vec{R}$.

Case (record.)

$$\frac{\text{For all } i = 1, \cdots, n\,,\ \Gamma \vdash T_i \ \ \mathsf{type}}{\Gamma \vdash [l_1 : T_1, \cdots, l_n : T_n] \ \ \mathsf{type}} \ (\text{record})$$

For all $i = 1, \cdots, n$, assume $e_i \in [\![T_i]\!]\vec{R}$.

By induction hypothesis, we have $\mathrm{Erase}(e_i) \in [\![T_i]\!]\vec{R}$. Applying (S3), we have $\{l_1{=}e_1, \cdots, l_i{=}e_i, \cdots, l_n{=}e_n\} \cdot l_i \in [\![T_i]\!]\vec{R}$. By definition of $[\![\ [l_1{:}T_1, \cdots, l_n{:}T_n]\ ]\!]\vec{R}$, we have $\mathrm{Erase}(\{l_1{=}e_1, \cdots, l_n{=}e_n\}) \in [\![\ [l_1{:}T_1, \cdots, l_n{:}T_n]\ ]\!]\vec{R}$

Case (recordsel.)

By induction hypothesis, we have $\phi(\mathrm{Erase}(e)) \in [\![\ [l_1 : T_1, \cdots, l_n : T_n]\ ]\!]\vec{R}$. By definition of $[\![\ [l_1 : T_1, \cdots, l_n : T_n]\ ]\!]\vec{R}$, we have $\mathrm{Erase}(e \cdot l_i) = (\mathrm{Erase}(e)) \cdot l_i \in [\![T_i]\!]\vec{\mathcal{R}}$

Case (record-Sel .)

$$\frac{\Gamma \vdash e : P \quad \Gamma \vdash \mathrm{Sel}(P, l) \ \ \mathsf{type}}{\Gamma \vdash e \cdot l : \mathrm{Sel}(P, l)} \ (\text{record-Sel})$$

Similarly to *Case* (λ-Range).

Case ((coer).)

$$\frac{\Gamma \vdash e : S \quad \Gamma \vdash S \leq T}{\Gamma \vdash e|_T : T} \text{ (coer)}$$

By induction hypothesis, $\text{Erase}(e) \in [\![S]\!]\vec{R}$. Since $[\![S]\!]\vec{R} \subseteq [\![T]\!]\vec{R}$, $\text{Erase}(e|_T) = \text{Erase}(e) \in [\![T]\!]\vec{R}$

∎

From these discussions, the main theorem of this section is proved:

Theorem 3.4.16.
For every raw term e if e type-checks, then $\text{Erase}(e) \in SN$

(**Proof**): We now apply Lemma 3.4.14 to any assignment (for example, the constant assignment with value SN) and to the identity substitution, which is legitimate since by (S1), every variable belongs to every saturated set. ∎

3.4.4 Termination of $cF^+_\leq$ Terms

In this section we prove that if the type erasure of an $cF^+_\leq$ term is strongly normalizing then the term itself is strongly normalizing.

We consider the term e and its type erasure $\text{Erase}(e)$.

We prove it by assuming that a typed-term e is not strongly normalizing and $\text{Erase}(e)$ is strongly normalizing and deriving contradictions.

Consider an infinite reduction chain in $cF^+_\leq$. It consists of segments of (β) reduction intermixed with segments of $(\beta 2)$ and (coer*) reductions. Consider the chain of all the type erasures of the elements of the $cF^+_\leq$ chain. To each (β) step in the original chain the same step corresponds in the erased chain, while to each pairs of terms related by a $(\beta 2)$ or (coer*) step in the $cF^+_\leq$ chain two identical terms correspond in the erased chain. This means that collapsing all sequences of identical terms, in the erased chain, into one element, we still obtain a reduction in the type erased $cF^+_\leq$. But each $(\beta 2)$ reduction deletes a Λ_{rec} symbol from the $cF^+_\leq$ term, and then any segment of $(\beta 2)$ reductions has a finite length. Moreover every coercion operator appears and is duplicated only when (β) reduction occurs, and it can move only rightward. when (coer) or (coer2) reduction occurs, and disappear when (coerconst) occurs. So collapsing all segments of $(\beta 2)$ and (coer*) reductions into one term transforms an infinite chain into another infinite chain. This proves that to any infinite reduction chain in $cF^+_\leq$, an infinite reduction in type erased $cF^+_\leq$ corresponds, and so termination for type-erased $cF^+_\leq$ terms implies termination for $cF^+_\leq$ terms.

Then we conclude that every $cF^+_\leq$ term which can type-check is strongly normalizing.

3.5 Example

We discuss the re-usability of programs in $cF^+_\leq$ and the relation between object-oriented languages and $cF^+_\leq$ more deeply.

$cF^+_\leq$, instead of $F_\leq$, fits our intuition that every term has its exact type in a pragmatic programming languages [OBBT89] because in $cF^+_\leq$ every term has its exact type while in $F_\leq$ not. Moreover, we show, from some examples, that we can develop the programming efficiency and the re-usability of programs in $cF^+_\leq$ better than $F_\leq$

Example 3.5.1.

$$m = \Lambda X{\leq}[l{:}T].\lambda x^X.x \cdot l$$

can have the type $\forall X{\leq}[l{:}T].X \rightarrow T$ and its super types in $F_{\leq}$ because of subsumption rule. However, in $cF_{\leq}$, m cannot have its type. But, in $cF^+_{\leq}$, m 's type is uniquely determined as

$$\dfrac{\dfrac{\dfrac{X \leq [l{:}T] \vdash x : X \quad (X \leq [l{:}T])^*(X) \equiv [l{:}T]}{X \leq [l{:}T] \vdash x \cdot l : \mathrm{Sel}(X,l)}}{X \leq [l{:}T] \vdash \lambda x^X.x \cdot l : X \rightarrow \mathrm{Sel}(X,l)}}{\Gamma \vdash \Lambda X{\leq}[l{:}T].\lambda x^X.x \cdot l : \forall X{\leq}[l{:}T].X \rightarrow \mathrm{Sel}(X,l)}$$

Example 3.5.2.

In the system $cF^+_{\leq}$, under an environment Γ, for a term m in Example 3.5.1, a type $[l{:}S]$ where $\Gamma \vdash S \leq T$ holds, and a term $\{l{=}n\}$ of the type $[l{:}S]$ which have been described in section 1, $(m\{[l{:}S]\})(\{l{=}n\})$ has the type S, while, in $F_{\leq}$, $(m\{[l{:}S]\})(\{l{=}n\})$'s minimal type is no more than T though

$$(\lambda x^{[l:S]}.x \cdot l)\{l{=}n\}$$

can have the type S both of in $F_{\leq}$ and in $cF^+_{\leq}$.

3.6 Conclusion

In this chapter, we introduced the calculus $cF^+_{\leq}$, which is a second order λ-calculus with records, coercion operator and bounded quantification including new types $\mathrm{Range}(X)$ and $\mathrm{Sel}(X,l)$. We also proved that $cF^+_{\leq}$ has good properties (unicity of type, Subject Reduction, Church-Rosser, and Strong Normalization) and showed some applications of $cF^+_{\leq}$ to object-oriented languages.

Chapter 4

$\mathrm{cF}^{++}_{\leq}$—A Type System with Type instantiation

4.1 Introduction

In the previous chapter, $cF^{+}_{\leq}$ is introduced to overcome the problem discussed in Section 1.2. However, $cF^{+}_{\leq}$ cannot completely solve it because $cF^{+}_{\leq}$ does not include the type instantiation (as already discussed in Section 3.1).

In this chapter, we give a calculus $cF^{++}_{\leq}$ by adding the type instantiation to $cF^{+}_{\leq}$. In $cF^{++}_{\leq}$, we show that the type system of $cF^{++}_{\leq}$ is well defined, for example, that, under the type instantiation rule $\mathrm{Inst}(\forall X \leq S.T, U) \Rightarrow T[X := U]$, every well-defined type always terminates. We proved it by attributing a "kind" to type. In $cF^{++}_{\leq}$, the treatments of subtype relation are different from that in $cF^{+}_{\leq}$. It is treated only as a binary relation and their meanings are not considered. So the treatments of l.u.b and w.l.u.b. are also different.

4.2 Kind

In this section we describe the kind system of $cF^{++}_{\leq}$, which is a classification of types. Note that the notion of Kind introduced in this chapter is different from that defined in higher-order λ-calculus, F_{ω} [Gir72]. Kind is introduced only to prove the termination of the "reduction" of the type system of $cF^{++}_{\leq}$ and the existence of working least upper bound, and is not related to properties of terms.

Notation 4.2.1.

We do not use only the notations defined in Notation 3.2.1 but also this following notation when no specific notation is declared.

Notation	Metavariable
Kind	$K, L, \ldots$

We firstly define the *Kind*, its subtype relation, and its degree.

Definition 4.2.2. (Kind)

$$\begin{array}{lcll} K & ::= & * & \text{(kind for base types)} \\ & | & \mathrm{Arr}(K) & \text{(kind for } \rightarrow\text{-types)} \\ & | & K_1 \rightarrow K_2 & \text{(kind for } \forall\text{-types)} \\ & | & \Omega & \text{(kind for arbitrary types)} \end{array}$$

Subkind

$$\frac{}{K \leq \Omega}\ (\Omega)$$

$$\frac{K \leq L}{\mathrm{Arr}(K) \leq \mathrm{Arr}(L)}\ (\mathrm{Arr})$$

$$\frac{L_1 \leq K_1 \quad K_2 \leq L_2}{K_1 \rightarrow K_2 \leq L_1 \rightarrow L_2}\ (\rightarrow)$$

Degree of Kind

$$\begin{array}{lcl} |*| & = & 1 \\ |\Omega| & = & 1 \\ |\mathrm{Arr}(K)| & = & |K| + 1 \\ |K \rightarrow L| & = & |K| + |L| + 1 \end{array}$$

4.3 Type

In this section we describe the type system $cF^{++}_{\leq}$. We also assume that there is no subtype relation between base types.

Definition 4.3.1. (Raw Type)
Raw types are defined as follows.

$$\begin{array}{lcll} T & ::= & X & \text{type variable} \\ & | & B & \text{base type} \\ & | & \mathrm{Top} & \text{Top} \\ & | & T \rightarrow T & \text{function type} \\ & | & \forall X{\leq}T.T & \text{(bounded) universal type} \\ & | & \mathrm{Range}(T) & \text{range type} \\ & | & \mathrm{Inst}(T_1, T_1) & \text{instantiation type} \\ & | & \mathrm{Domain}(T) & \text{domain type} \\ & | & \mathrm{Upper}(T) & \text{(upper) bound type} \end{array}$$

The definition of raw types is an extension of that of $F_{\leq}$. Raw types are divided into working types and non-well-working types. *Working types* consist of base types, function types, (bounded) universal types, which are the same as defined in $F_{\leq}$, and Top. Top is the only supertype of all types. *Non-well-working types* consists of type variables, range types, and instantiation types.

Range types have the feature $\mathrm{Range}(P)$, where P has a super type that is a function type, and when a function type $S \rightarrow U$ is substituted for P, $\mathrm{Range}(P)$ is reduced to U. Instantiation types have the feature $\mathrm{Inst}(P,T)$, where P has a super type that is a universal type, and when a universal type $\forall X{\leq}S.U$ is substituted for P, $\mathrm{Inst}(P,T)$ is reduced to $U[X := T]$. Domain types have the feature $\mathrm{Domain}(P)$, where P has a super type that is a function type, and when a function type $S \rightarrow U$ is substituted for P, $\mathrm{Domain}(P)$ is reduced to S. Bound types have the feature $\mathrm{Upper}(P)$, where P has a super type that is a universal type, and when a universal type $\forall X{\leq}S.U$ is substituted for P, $\mathrm{Upper}(P)$ is reduced to S.

Both domain types and bound types are only used to prove the termination of reduction rules of types, and we do not consider the subtype relation and the type assignment to terms of the types $\mathrm{Domain}(P)$ or $\mathrm{Upper}(P)$.

Definition 4.3.2. (Normal Form)
We call a raw type P in *normal form* if P satisfies these conditions:

- P does not contain $\mathrm{Range}(S \to T)$ or $\mathrm{Inst}(\forall X{\leq}S.T, U)$.
- P does not contain Domain , or Upper .

If a raw type P is in normal form and there exist S and T where $P = S \to T$ or $P = \forall X{\leq}S.T$ holds, we call P a *working type*. Otherwise we call P a *non-well-working type*.

Definition 4.3.3. (Type Environment)
In $cF^{++}_{\leq}$, a *(type) environment* is defined as a sequence of pairs of a free type variable and its "upper bound". It is also defined mutually to definition of type good-formation in Definition 4.3.6.

$$\phi \quad \mathsf{env}$$

$$\frac{\Gamma \quad \mathsf{env} \quad \Gamma \vdash T : \Omega \quad X \notin \mathrm{FV}(\Gamma)}{\Gamma, X{:}K, X \leq T \quad \mathsf{env}}$$

Definition 4.3.4.
For a non-well-working type $\mathrm{Range}(P)$ and $\mathrm{Domain}(P)$ (respectively $\mathrm{Inst}(P,U)$ and $\mathrm{Upper}(P)$), a working type $S \to T$ ($\forall X{\leq}S.T$) is substituted for P . The (raw) type $\mathrm{Range}(S \to T)$ and $\mathrm{Domain}(S \to T)$ ($\mathrm{Inst}(\forall X{\leq}S.T, U)$ and $\mathrm{Upper}(\forall X{\leq}S.T)$) need to be reduced to a "normal form".

$$\begin{array}{rcl} \mathrm{Range}(S \to T) & \Rightarrow & T \\ \mathrm{Inst}(\forall X \leq S.T, U) & \Rightarrow & T[X := U] \\ \mathrm{Domain}(S \to T) & \Rightarrow & S \\ \mathrm{Upper}(\forall X \leq S.T, U) & \Rightarrow & S \end{array}$$

Especially, we call the rule $\mathrm{Inst}(\forall X \leq S.T, U) \Rightarrow T[X := U]$ the *type instantiation*. When, after a finite number of reductions, a type T which is not in normal form, is reduced to V which is in normal form, we write down $T \not\Rightarrow V$.

Definition 4.3.5.
For each type T , we want to define the l.u.b. (least upper bound) of T , $\Gamma(T)$. Of course, when T is a working type, we define $\Gamma(T) = T$, Otherwise, that is, when P is a non-well-working type, we define $\Gamma(P)$ as follows:

$$\begin{array}{lcll} \Gamma(X) & \Rightarrow & T & (\text{where } (X \leq T) \in \Gamma) \\ \Gamma(\mathrm{Range}(P)) & \Rightarrow & V & (\text{where } \mathrm{Range}(\Gamma(P)) \not\Rightarrow V) \\ \Gamma(\mathrm{Inst}(P,T)) & \Rightarrow & V & (\text{where } \mathrm{Inst}(\Gamma(P), T) \not\Rightarrow V) \end{array}$$

We have three kinds of type judgements: for the type good-formation $(\Gamma \vdash S : K)$, for the subtype relation $(\Gamma \vdash S \leq T : K)$ and for the type assignment $(\Gamma \vdash e : S)$. We also call the first two kinds of judgements *type judgements*. All kinds of *type judgements* are mutually defined.

4.3.1 Type Judgements

Definition 4.3.6. (Type good-formation ($\Gamma \vdash S : K$))
Type good-formation ($\Gamma \vdash S : K$)
We define the judgement of the type "good-formation", which assigns types to kinds.

$$\frac{X{:}K \in \Gamma}{\Gamma \vdash X : K} \text{ (Var)}$$

$$\frac{}{\Gamma \vdash \ \mathrm{Top} : \Omega} \text{ (Top)}$$

$$\frac{\Gamma \vdash S : \Omega \quad \Gamma \vdash T : L}{\Gamma \vdash S \to T : \mathrm{Arr}(L)} \ (\to)$$

$$\frac{\Gamma \vdash S : \Omega \quad \Gamma, X{:}K, X \leq S \vdash T : L}{\Gamma \vdash \forall X{\leq}S.T : K \to L} \text{ (bound-}\forall\text{)}$$

$$\frac{\Gamma \vdash P : \mathrm{Arr}(L)}{\Gamma \vdash \mathrm{Range}(P) : L} \text{ (Range)}$$

$$\frac{\Gamma \vdash P : K \to L \quad \Gamma \vdash U : K}{\Gamma \vdash \mathrm{Inst}(P, U) : L} \text{ (Inst)}$$

$$\frac{\Gamma \vdash P : \mathrm{Arr}(K)}{\Gamma \vdash \mathrm{Domain}(P) : \Omega} \text{ (Domain)}$$

$$\frac{\Gamma \vdash P : K \to L}{\Gamma \vdash \mathrm{Upper}(P) : \Omega} \text{ (Upper)}$$

$$\frac{\Gamma \vdash T : K \quad K \leq L}{\Gamma \vdash T : L} \text{ (kind-subsumption)}$$

Proposition 4.3.7. (Subject Reduction of type **)**
For a pretype T, if $\Gamma \vdash T : K$ and $T \Rightarrow T'$, then $\Gamma \vdash T' : K$ holds.

(**Proof**): We need only to prove the type instantiation.

$$\mathrm{Inst}(\forall X{\leq}S.T.U) \ \Rightarrow \ T[X := U]$$

$$\frac{\dfrac{\vdots\ \Pi}{\Gamma, X{:}K, X \leq S \vdash T : L} \qquad \vdots}{\dfrac{\Gamma \vdash \forall X{\leq}\, S.T : K \to L \quad \Gamma \vdash U : K}{\Gamma \vdash \mathrm{Inst}(\forall X{\leq}S.T.U) : L}}$$

$$\Downarrow$$

$$\frac{(\text{Substitute } \Gamma \vdash X : K \text{ for } \Gamma \vdash U : K \text{ in } \Pi.)}{\Gamma \vdash T[X := U]}$$

Note that $X \leq S$ is never used in Π. ∎

4.3.2 Subtype Relation

In this section we define the rules of the subtype relation. In $cF^{++}_{\leq}$, this subtype relation is only the binary relation between the types and we need not consider its intuitive semantics.

Definition 4.3.8. (Subtype Relation ($\Gamma \vdash S \leq T : K$))
Subtype Relation ($\Gamma \vdash S \leq T : K$)
The definitions of the subtype relation are listed as below:

$$\frac{\Gamma \vdash T : K}{\Gamma \vdash T \leq T : K} \ (\leq\text{-Refl})$$

$$\frac{\Gamma \vdash X : K \quad \Gamma \vdash T : K \quad (X \leq T) \in \Gamma}{\Gamma \vdash X \leq T : K} \ (\leq\text{-Taut})$$

$$\frac{\Gamma \vdash S \leq T : K \quad \Gamma \vdash T \leq U : K}{\Gamma \vdash S \leq U : K} \ (\leq\text{-Trans})$$

$$\frac{\Gamma \vdash S \leq T : K \quad K \leq L}{\Gamma \vdash S \leq T : L} \ (\text{kind-subsumption})$$

$$\frac{\Gamma \vdash S : K \quad S \Rightarrow T}{\Gamma \vdash S \leq T : K} \ (\text{kind-reduction1})$$

$$\frac{\Gamma \vdash S : K \quad S \Rightarrow T}{\Gamma \vdash T \leq S : K} \ (\text{kind-reduction2})$$

$$\frac{\Gamma \vdash S' \leq S : \Omega \quad \Gamma \vdash T \leq T' : L}{\Gamma \vdash S \to T \leq S' \to T' : \mathrm{Arr}(L)} \ (\leq\text{-}\to)$$

$$\frac{\Gamma \vdash \forall X{\leq}S.T : K \to L \quad \Gamma \vdash S' \leq S : \Omega \quad \Gamma, X{:}K, X \leq S' \vdash T \leq T' : L}{\Gamma \vdash \forall X{\leq}S.T \leq \forall X{\leq}S'.T' : K \to L} \ (\leq\text{-bb}\forall)$$

$$\frac{\Gamma \vdash P \leq Q : \mathrm{Arr}(L)}{\Gamma \vdash \mathrm{Range}(P) \leq \mathrm{Range}(Q) : L} \ (\text{Range})$$

$$\frac{\Gamma \vdash U : K \quad \Gamma \vdash P \leq Q : K \to L}{\Gamma \vdash \mathrm{Inst}(P, U) \leq \mathrm{Inst}(Q, U) : L} \ (\text{Inst})$$

Remark 4.3.9.
When all types and subtype relations are already assigned to kinds, we can have the algorithm of judgement of subtype relation. However, if they are not, it will be difficult to get an algorithm of the judgement.

From the definitions the following properties are satisfied.

Proposition 4.3.10.
If $\Gamma \vdash S \leq T : K$ then $\Gamma \vdash S : K$ and $\Gamma \vdash T : K$ hold.

(**Proof**): Obvious. ∎

Proposition 4.3.11. (Termination of Type)

1. Under an environment Γ, for every type expression T which is well-formed, T is strongly normalizing under the reduction rules defined in Definition 4.3.3.
2. If $(X_i \leq T_i) \in \Gamma$ $(i = 1, \ldots, n)$, we add $X_1 \Rightarrow T_1, \ldots, X_n \Rightarrow T_n$ to "reduction rules" defined in Definition 4.3.3. Then for every well-formed type T, T is strongly normalizing under these reduction rules..

(**Proof**): The proof is similar to that of strong normalization of simply typed λ-calculus[GLT89].

Induction on the size of Γ . For every kind K , we define the candidate of reducibility Red_K

Definition 4.3.12.

$$\begin{array}{lcl}
\mathsf{Red}_* & = & \{T \ : \ \Gamma \vdash T : * \text{ and terminating}\} \\
\mathsf{Red}_{\mathrm{Arr}(L)} & = & \{T \ : \ \mathrm{Domain}(T) \in \mathsf{Red}_\Omega,\ \mathrm{Range}(T) \in \mathsf{Red}_L\} \\
\mathsf{Red}_{K \to L} & = & \{T \ : \ \mathrm{Upper}(T) \in \mathsf{Red}_\Omega \\
 & & \quad \text{and for every } U \in \mathsf{Red}_K, \mathrm{Inst}(T,U) \in \mathsf{Red}_L \text{ holds.}\} \\
\mathsf{Red}_\Omega & = & \{T \ : \Gamma \vdash T : \Omega \text{ and terminating}\}
\end{array}$$

Notation 4.3.13.

We denote $\nu(T)$ as a number which bounds the length of every normalization sequence beginning with T .

Definition 4.3.14.

The type T is *neutral* if T has a shape such as, B (base type), X (type variable), $\mathrm{Domain}(S)$, $\mathrm{Range}(P)$, $\mathrm{Upper}(P)$ and $\mathrm{Inst}(P,U)$.

Lemma 4.3.15.

1. All types in Red_K is terminating.
2. If $T \in \mathsf{Red}_K$ and $T \Rightarrow T'$, $T' \in \mathsf{Red}_K$.
3. T is neutral and for all T' such that $T \Rightarrow T'$, $T' \in \mathsf{Red}_K$ holds, then $T \in \mathsf{Red}_K$.

(**Proof**): Induction of degrees of kinds:

Case (Base type, and Ω .)

Obvious.

Case ($\mathrm{Arr}(K)$.)

Similar to the case $K \to L$. So we omit.

Case ($K \to L$.)

1. Assume $T \in \mathsf{Red}_{K \to L}$. Since $\mathrm{Upper}(T) \in \mathsf{Red}_\Omega$ holds, $\mathrm{Upper}(T)$ is terminating. For a (fresh) type variable $X : K$, X is a normal form, then $\mathrm{Inst}(T,X) \in \mathsf{Red}_L$. Then we have $\nu(T) \leq \nu(\mathrm{Inst}(T,X))$, and therefore T is terminating.
2. $T \in \mathsf{Red}_{K \to L}$, and assume $T \Rightarrow T'$. By induction hypothesis, $\mathrm{Upper}(T) \in \mathsf{Red}_\Omega$ so, $\mathrm{Upper}(T') \in \mathsf{Red}_\Omega$. For every $U \in \mathsf{Red}_K$, from assumptions, $\mathrm{Inst}(T',U) \in \mathsf{Red}_L$ because $\mathrm{Inst}(T,U) \Rightarrow \mathrm{Inst}(T',U)$ holds. So we have $T' \in \mathsf{Red}_{K \to L}$.
3. Assume that T is neutral and that for every T' such that $T \Rightarrow T'$ holds, $T' \in \mathsf{Red}_{K \to L}$ holds. Since $\mathrm{Upper}(T)$ cannot be a redex, and $\mathrm{Upper}(T') \in \mathsf{Red}_\Omega$, since $T' \in \mathsf{Red}_{K \to L}$ holds. So, by induction hypothesis, $\mathrm{Upper}(T) \in \mathsf{Red}_\Omega$ holds. Provide that $U \in \mathsf{Red}_K$. By induction for K , we know that U is terminating; so we can reason about $\mathrm{Inst}(T,U)$ by induction on $\nu(U)$.
   - $\mathrm{Inst}(T',U)$ with $T \Rightarrow T'$; but $T' \in \mathsf{Red}_{K \to L}$, so $\mathrm{Inst}(T',U) \in \mathsf{Red}_L$.
   - $\mathrm{Inst}(T,U')$ with $U \Rightarrow U'$, $U' \in \mathsf{Red}_K$ by induction hypothesis for K , and $\nu(U') < \nu(U)$; so $\mathrm{Inst}(T,U') \in \mathsf{Red}_L$.

   We conclude that $\mathrm{Inst}(T,U) \in \mathsf{Red}_L$, and so $T \in \mathsf{Red}_{K \to L}$.

∎

Proposition 4.3.16.
If $T \in \mathsf{Red}_K$ and $K \leq L$ hold, then $T \in \mathsf{Red}_L$

(**Proof**): We prove these four cases of subkind relation.

Case ($K \leq \Omega$.)
If $T \in \mathsf{Red}_K$, then T is terminating.

Case ($\mathrm{Arr}(K) \leq \mathrm{Arr}(L)$.)
Similar to the following case.

Case ($K_1 \rightarrow K_2 \leq L_1 \rightarrow L_2$.)
Assume that $T \in \mathsf{Red}_{K_1 \rightarrow K_2}$, for every $U \in \mathsf{Red}_{L_1}$. Since $U \in \mathsf{Red}_{K_1}$ holds from hypothesis about $L_1 \leq K_1$, $\mathrm{Inst}(T, U) \in \mathsf{Red}_{K_2}$ and therefore by hypothesis about $K_2 \leq L_2$, $\mathrm{Inst}(T, U) \in \mathsf{Red}_{L_2}$ are satisfied.

∎

Lemma 4.3.17.

1. If for a $S \in \mathsf{Red}_\Omega$ and for all $U \in \mathsf{Red}_K$, $T[X := U] \in \mathsf{Red}_L$ holds, then $\forall X{\leq}S.T \in \mathsf{Red}_{K \rightarrow L}$

(**Proof**): We want to show that $\mathrm{Inst}(\forall X{\leq}S.T, U) \in \mathsf{Red}_{K \rightarrow L}$ holds for all $U \in \mathsf{Red}_K$. We reason by induction on $\nu(S) + \nu(T) + \nu(U)$. Obviously $\mathrm{Upper}(\forall X{\leq}S.T) \in \mathsf{Red}_\Omega$ holds. The type $\mathrm{Inst}(\forall X{\leq}S.T, U)$ is reduced to

- ($T[X := U] \in \mathsf{Red}_L$) From assumptions,
- ($\mathrm{Inst}(\forall X{\leq}S'.T, U)$ with $S \Rightarrow S'$) By induction hypothesis,
- ($\mathrm{Inst}(\forall X{\leq}S.T', U)$ with $T \Rightarrow T'$) By induction hypothesis,
- ($\mathrm{Inst}(\forall X{\leq}S.T, U')$ with $U \Rightarrow U'$) By induction hypothesis,

it is easily proved. ∎

Notation 4.3.18.
We consider the case where, when $X \leq T$ is included in Γ , $X \Rightarrow T$ is included in reduction rules defined in Definition 4.3.4.

Lemma 4.3.19.
Assume $\Gamma \vdash X \leq T : K$. If $T \in \mathsf{Red}_K$, then $X \in \mathsf{Red}_K$

(**Proof**): We prove only the case that $K \rightarrow L$. Firstly we show that $\mathrm{Upper}(X) \in \mathsf{Red}_\Omega$ holds. The type $\mathrm{Upper}(X)$ is reduced to $\mathrm{Upper}(T) \in \mathsf{Red}_K$. Therefore since $\mathrm{Upper}(X)$ is neutral, $\mathrm{Upper}(X) \in \mathsf{Red}_\Omega$.

Secondly we want to show that $\mathrm{Inst}(X, U) \in \mathsf{Red}_L$ holds for all $U \in \mathsf{Red}_K$. Since $\mathrm{Inst}(X, U)$ is neutral, we reason by induction on $\nu(T) + \nu(U)$. The type $\mathrm{Inst}(X, U)$ is reduced to

- ($\mathrm{Inst}(T, U) \in \mathsf{Red}_L$) From assumption,
- ($\mathrm{Inst}(X, U)$ with $U \Rightarrow U'$) By hypothesis,

it is easily proved. ∎

Then we completely proved Proposition 4.3.11. ∎

From that proposition, this property is easily proved.

Proposition 4.3.20.
For T be any type, and suppose all the free type variables of T are among $X_1 \leq T_1, \ldots, X_n \leq T_n$ of kind $K_1, \ldots, K_n$. If $U_i \in \mathsf{Red}_{K_i}$ ($\Gamma \vdash U_i \leq T_i \quad i = 1, \ldots, n$) then $T[\vec{X} := \vec{U}] \in \mathsf{Red}_L$

These properties are needed to prove the subject reduction of $cF^{++}_{\leq}$.

Proposition 4.3.21.
Assume $\Gamma \vdash U \leq S : K$ holds,

1. If $\Gamma, X{:}K, X \leq S, \Delta \vdash T : L$, $T[X := U] = T'$ and $\Delta[X := U] = \Delta'$ hold, then $\Gamma, \Delta' \vdash T' : L$.
2. If $\Gamma, X{:}K, X \leq S, \Delta \vdash T \leq V : L$, $T[X := U] = T'$, $V[X := U] = V'$ and $\Delta[X := U] = \Delta'$ hold, then $\Gamma, \Delta' \vdash T' \leq V' : L$.

(**Proof**): From Proposition 4.3.10, we have $\Gamma \vdash S : K$. Both two cases are easily proved by replacing $X : K$ and $X \leq S : K$ with $S : K$ and $U \leq S : K$ in the original proofs, respectively. ∎

From Proposition 4.3.11, by applying (kind-reduction 1) and (kind-reduction 2), the following properties are easily proved.

Proposition 4.3.22. (Substitution Lemma)
Assume $\Gamma \vdash U \leq S : K$ holds,

1. If $\Gamma, X{:}K, X \leq S, \Delta \vdash T : L$, $T[X := U] \not\equiv T'$ and $\Delta[X := U] \not\equiv \Delta'$ hold, then $\Gamma, \Delta' \vdash T' : L$.
2. If $\Gamma, X{:}K, X \leq S, \Delta \vdash T \leq V : L$, $T[X := U] \not\equiv T'$, $V[X := U] \not\equiv V'$ and $\Delta[X := U] \not\equiv \Delta'$ hold, then $\Gamma, \Delta' \vdash T' \leq V' : L$.

4.4 Term

In the last section, we show that every type which has its kind (= well-defined type) is terminating under the reduction rules defined in Definition 4.3.4. So, in this section, we omit kinds for simplicity. For example, we replace $\Gamma \vdash T : K$ and $\Gamma \vdash S \leq T : K$ with $\Gamma \vdash T$ type and $\Gamma \vdash S \leq T$, respectively. Before discussions, we define working least upper bounds as in Chapter 3.

Definition 4.4.1.
We need to obtain upper bound which is working for all non working type, because we want to know conditions which enable applications. To obtain the working upper bound of non working type P, we recursively define the *working least upper bound (w.l.u.b.)* $\mathrm{Wlub}_\Gamma(P)$ as follows:

$$\begin{array}{rcll} \mathrm{Wlub}_\Gamma(T) & \equiv & T & (T \text{ is a working type}) \\ & | & \mathrm{Wlub}_\Gamma(V) & (T \text{ is a non-well-working type and } \Gamma(T) \not\equiv V) \end{array}$$

From Proposition 4.3.11, these properties are satisfied:

Proposition 4.4.2.

1. Under an environment Γ, for all non working type P, the w.l.u.b. of P $\mathrm{Wlub}_\Gamma(P)$ always exists.
2. For two non-well working types P and Q, if $\Gamma \vdash P \leq Q$ holds, then $\mathrm{Wlub}_\Gamma(P) \equiv \mathrm{Wlub}_\Gamma(Q)$

In this section, we describe the terms of $cF^{++}_{\leq}$. We start by the definition of *raw terms*, among which we distinguish the *terms*, i.e. those raw terms possessing types. Raw terms are divided into several classes: variables, constants, function terms (functions), $\forall$-terms, record terms, record selection terms, and coerced terms. Coercion is introduced to hold the properties of unicity of type and subject reduction.

Definition 4.4.3.

Before considering the term system, we re-consider the definition of type environment, range types and instantiation types, and subtype relation between them.

Type Environment

$$\frac{\Gamma \ \mathsf{env} \quad \Gamma \vdash T \ \mathsf{type} \quad T \text{ is in normal form} \quad X \notin \mathrm{FV}(\Gamma)}{\Gamma, X \leq T \ \mathsf{env}}$$

Type Good-Formation

$$\frac{\Gamma \vdash P \ \mathsf{type} \quad \mathrm{Wlub}_\Gamma(P) \equiv S \to T}{\Gamma \vdash \mathrm{Range}(P) \ \mathsf{type}} \ (\text{Range})$$

$$\frac{\Gamma \vdash P \ \mathsf{type} \quad \mathrm{Wlub}_\Gamma(P) \equiv \forall X{\leq}S.T \quad \Gamma \vdash U \leq S}{\Gamma \vdash \mathrm{Inst}(P,U) \ \mathsf{type}} \ (\text{Inst})$$

Subtype Relation

$$\frac{\Gamma \vdash \mathrm{Range}(P) \ \mathsf{type} \quad \Gamma \vdash \mathrm{Range}(Q) \ \mathsf{type} \quad \Gamma \vdash P \leq Q}{\Gamma \vdash \mathrm{Range}(P) \leq \mathrm{Range}(Q)} \ (\leq\text{-Range})$$

$$\frac{\Gamma \vdash \mathrm{Inst}(P,U) \ \mathsf{type} \quad \Gamma \vdash \mathrm{Inst}(Q,U) \ \mathsf{type} \quad \Gamma \vdash P \leq Q}{\Gamma \vdash \mathrm{Inst}(P,U) \leq \mathrm{Inst}(Q,U)} \ (\leq\text{-Inst})$$

The definition of (Inst) is valid because P and $\forall X{\leq}S.T$ share a kind under Γ.

4.4.1 Term

We define the raw terms and their type-assignment rules:

Definition 4.4.4. (Raw Terms)

e	:=	x^T	(term) variable
	$\mid$	c^T	constant
	$\mid$	$\lambda x^T.e$	λ-abs
	$\mid$	ee	λ-apl
	$\mid$	$\Lambda X{\leq}T.e$	(bounded) Λ-abs
	$\mid$	$e\{T\}$	Λ-apl
	$\mid$	$e\vert_T$	coerced term

We need to define the reduction of types ($\Rrightarrow$) which stick to terms because these sorts of types must be in normal form.

Definition 4.4.5.

x^T	$\Rrightarrow$	$x^{T'}$	where $T \Rrightarrow T$
$\lambda x^T.e$	$\Rrightarrow$	$\lambda x^{T'}.e'$	where $T \Rrightarrow T'$ and $e \Rrightarrow e'$
ef	$\Rrightarrow$	$e'f'$	where $e \Rrightarrow e'$ and $f \Rrightarrow f'$
$\Lambda X{\leq}S.e$	$\Rrightarrow$	$\Lambda X{\leq}S'.e'$	where $S \Rrightarrow S'$ and $e \Rrightarrow e'$
$e\{T\}$	$\Rrightarrow$	$e'\{T'\}$	where $T \Rrightarrow T'$ and $e \Rrightarrow e'$
$e\vert_T$	$\Rrightarrow$	$e'\vert_{T'}$	where $T \Rrightarrow T'$ and $e \Rrightarrow e'$

Definition 4.4.6. (Type assignment ($\Gamma \vdash e : S$))

Terms are selected by the rules below; since term variables are indexed by their type, the rules do not needed assumptions of the term $(x : T)$:

$$\frac{\Gamma \vdash T \ \ \mathsf{type} \quad T \text{ is in normal form.}}{\Gamma \vdash x^T : T} \ (\text{var})$$

$$\frac{\Gamma \vdash e : T \quad S \text{ is in normal form}}{\Gamma \vdash (\lambda x^S.e) : S \to T} \ (\lambda\text{-ab})$$

$$\frac{\Gamma \vdash e : S \to T \quad \Gamma \vdash f : U \quad \Gamma \vdash U \leq S}{\Gamma \vdash ef : T} \ (\lambda\text{-ap})$$

$$\frac{\left\{ \begin{array}{ll} \Gamma \vdash e : P & \Gamma \vdash f : U \quad \Gamma \vdash P \ \ \mathsf{type} \\ \mathrm{Wlub}_\Gamma(P) \equiv (S \to T) & \Gamma \vdash U \leq S \end{array} \right.}{\Gamma \vdash ef : \mathrm{Range}(P)} \ (\lambda\text{-Range})$$

$$\frac{\left\{ \begin{array}{l} \Gamma, X \leq S \vdash e : T \quad S \text{ is in a normal form} \\ \text{no free variable whose type is } X \text{ does not occur in } e \end{array} \right.}{\Gamma \vdash \Lambda X{\leq}S.e : \forall X{\leq}S.T} \ (\text{bound-}\Lambda\text{-ab})$$

$$\frac{\left\{ \begin{array}{l} \Gamma \vdash e : \forall X{\leq}S.T \quad \Gamma \vdash U \leq S \\ U \text{ is in normal form} \\ T[X := U] \ni T' \end{array} \right.}{\Gamma \vdash e\{U\} : T'} \ (\text{bound-}\Lambda\text{-ap-w})$$

$$\frac{\left\{ \begin{array}{l} \Gamma \vdash e : P \quad U \text{ is in normal form} \\ \mathrm{Wlub}_\Gamma(P) \equiv \forall X{\leq}S.T \\ \Gamma \vdash U \leq S \end{array} \right.}{\Gamma \vdash e\{U\} : \mathrm{Inst}(P, U)} \ (\Lambda\text{-ap-nw})$$

$$\frac{\Gamma \vdash e : S \quad \Gamma \vdash S \leq T \quad T \text{ is in normal form}}{\Gamma \vdash e|_T : T} \ (\text{coer})$$

From these definitions, the following property holds.

Proposition 4.4.7.

If $\Gamma \vdash e : T$ holds, T is in normal form.

4.4.2 Reduction

From Proposition 4.3.22, these properties hold:

Proposition 4.4.8. (Substitution Lemma)

1. If $\Gamma \vdash e : S$ and $\Gamma \vdash f : T$ hold, then $\Gamma \vdash e[x^T := f] : S$ holds.
2. If $\Gamma, X \leq S, \Delta \vdash e : T$, and $\Gamma \vdash U \leq S$ hold, then $\Gamma, \Delta' \vdash e' : T'$ holds where $e[X := U] \ni e'$, $T[X := U] \ni T'$, $\Delta[X := U] \ni \Delta'$.

Now we define reduction rules ($\to$).

Definition 4.4.9. (Reduction)

We replace (coer2) with (coer2'), which is obtained by relaxing restrictions of (coer2).

The reduction rules in $cF^{++}_{\leq}$ are listed as follows:

$$\frac{}{\Gamma \vdash (\lambda x^S.e)f \ \rightarrow \ e[x := f|_S]} \ (\beta)$$

$$\frac{e[X := S] \not\ni e'}{\Gamma \vdash (\Lambda X{\leq}S.e)\{T\} \ \rightarrow \ e'} \ (\text{bound } \beta 2)$$

$$\frac{\text{Range}(S) \not\ni V}{\Gamma \vdash (e|_S)f \ \rightarrow \ (ef)|_V} \ (\text{coer})$$

$$\frac{\text{Inst}(P, U) \not\ni V}{\Gamma \vdash (e|_P)\{U\} \ \rightarrow \ (e\{U\})|_V} \ (\text{coer2'})$$

$$\frac{\Gamma \vdash e \text{ is in normal form} \quad S \text{ is a base type.}}{\Gamma \vdash e|_S \ \rightarrow \ e} \ (\text{coerconst})$$

The reduction rules of $cF^{\&}_{\leq}+$ have good properties: Subject Reduction and Church-Rosser:

Theorem 4.4.10. (Subject Reduction)
For each expression e, if $\Gamma \vdash e : T$ and $\Gamma \vdash e \quad \rightarrow \quad e'$ hold, then $\Gamma \vdash e' : T$ holds.

(**Proof**): We prove only the case (coer2'). The other cases are proved similarly to Chapter 3. We replace, for example, " $\text{Wlub}_\Gamma(P) \equiv S \rightarrow T$ " with " $\Gamma \vdash P \leq S \rightarrow T$ ".

Case ((coer2').)

1. (working)
   We first consider the case both P and Q are working, i.e., $P = \forall X{\leq}V.W$ and $Q = \forall X{\leq}S.T$.

$$\frac{\boxed{\mathbf{I}} \qquad \begin{array}{c}\vdots\ \Sigma_1 \\ \Gamma \vdash U \leq S\end{array} \qquad T[X := U] \not\ni T'}{\Gamma \vdash (e|_{\forall X \leq S.T})\{U\} : T'}$$

   where **I** is

$$\frac{\begin{array}{c}\vdots\ \Pi_1 \\ \Gamma \vdash e : \forall X{\leq}V.W\end{array} \qquad \frac{\begin{array}{c}\vdots\ \Sigma_2 \\ \Gamma \vdash \forall X{\leq}V.W \ \ \mathsf{type}\end{array} \qquad \begin{array}{c}\vdots\ \Sigma_3 \\ \Gamma \vdash S \leq V\end{array} \qquad \begin{array}{c}\vdots\ \Sigma_4 \\ \Gamma, X \leq S \vdash W \leq T\end{array}}{\Gamma \vdash \forall X{\leq}V.W \ \leq \forall X{\leq}S.T}}{\Gamma \vdash e|_{\forall X \leq S.T} : \forall X{\leq}\ S.T}$$

$$\Downarrow$$

$$\frac{\boxed{\mathbf{I'}} \qquad \begin{array}{c}\begin{array}{c}\vdots\ \Sigma_4 \\ \Gamma, X \leq S \vdash W \leq T\end{array} \qquad \begin{array}{c}\vdots\ \Sigma_1 \\ \Gamma \vdash U \leq S\end{array} \\ \vdots\ \text{(Subst. Lemma)} \\ \Gamma \vdash W' \leq T'\end{array}}{\Gamma \vdash (e\{U\})|_{T'} : T'}$$

where **I'** is

$$\dfrac{\Gamma \vdash e : \forall X{\leq}V.W \quad \Gamma \vdash \forall X{\leq}V.W \ \ \mathsf{type} \quad \dfrac{\Gamma \vdash U \leq S \quad \Gamma \vdash S \leq V}{\Gamma \vdash U \leq V} \quad W[X := U] \trianglerighteq W'}{\Gamma \vdash e\{U\} : W'}$$

2. (non working and working)
Secondly, we consider the case that P is non working and $Q = \forall X{\leq}S.T$

$$\dfrac{\dfrac{\Gamma \vdash e : P \quad \boxed{\mathbf{I}}}{\Gamma \vdash e|_{\forall X \leq S.T} : \forall X{\leq}S.T} \quad \Gamma \vdash U \leq S \quad T[X := U] \trianglerighteq T'}{\Gamma \vdash (e|_{\forall X \leq S.T})\{U\} : T'}$$

where **I** is

$$\dfrac{\Gamma \vdash P \leq \forall X{\leq}V.W(\text{w.l.u.b}) \quad \dfrac{\Gamma \vdash \forall X{\leq}V.W \ \ \mathsf{type} \quad \Gamma \vdash S \leq V \quad \Gamma, X \leq S \vdash W \leq T}{\Gamma \vdash \forall X{\leq}V.W \leq \forall X{\leq}S.T}}{\Gamma \vdash P \leq \forall X{\leq}S.T}$$

$$\Downarrow$$

$$\dfrac{\boxed{\mathbf{I'}} \quad \dfrac{\boxed{\mathbf{II}} \quad T[X := U] \trianglerighteq T'}{\Gamma \vdash \mathrm{Inst}(P, U) \leq T'}}{\Gamma \vdash (e\{U\})|_{T'} : T'}$$

where **I'** is:

$$\dfrac{\Gamma \vdash e : P \quad \Gamma \vdash P \leq \forall X{\leq}V.W \quad \dfrac{\Gamma \vdash U \leq S \quad \Gamma \vdash S \leq V}{\Gamma \vdash U \leq V}}{\Gamma \vdash e\{U\} : \mathrm{Inst}(P, U)}$$

and **II** is:

$$\dfrac{\Gamma \vdash P \leq \forall X{\leq}S.T \quad \Gamma \vdash \mathrm{Inst}(P, U) \ \ \mathsf{type} \quad \Gamma \vdash \mathrm{Inst}(\forall X{\leq}S.T, U) \ \ \mathsf{type}}{\Gamma \vdash \mathrm{Inst}(P, U) \leq \mathrm{Inst}(\forall X{\leq}S.T, U)}$$

3. (non working and non working)
Third, we consider the case both P and Q are non-working.

$$\dfrac{\dfrac{\Gamma \vdash e : P \quad \Gamma \vdash P \leq Q}{\Gamma \vdash e|_Q : Q} \quad \Gamma \vdash Q \leq \forall X{\leq}S.T(\text{w.l.u.b.}) \quad \Gamma \vdash U \leq S}{\Gamma \vdash (e|_Q)\{U\} : \mathrm{Inst}(Q, U)}$$

$$\Downarrow$$

$$\dfrac{\boxed{\Gamma'} \qquad \dfrac{\Gamma \vdash P \leq Q \quad \Gamma \vdash \mathrm{Inst}(P,U)\ \mathsf{type} \quad \Gamma \vdash \mathrm{Inst}(P,U)\ \mathsf{type}}{\Gamma \vdash \mathrm{Inst}(P,U) \leq \mathrm{Inst}(Q,U)}}{\Gamma \vdash (e\{U\})|_{\mathrm{Inst}(Q,U)} : \mathrm{Inst}(Q,U)}$$

where **Γ'** is

$$\dfrac{\Gamma \vdash e : P \qquad \dfrac{\Gamma \vdash P \leq Q \quad \Gamma \vdash Q \leq \forall X{\leq}S.T}{\Gamma \vdash P \leq \forall X{\leq}S.T} \qquad \Gamma \vdash U \leq S}{\Gamma \vdash e\{U\} : \mathrm{Inst}(P,U)}$$

In the proofs of $\Gamma \vdash \mathrm{Inst}(\cdots, U)$ are very easy and omitted.

∎

Theorem 4.4.11.
$cF^{++}_{\leq}$ satisfies Church-Rosser property.

(**Proof**): We can easily proved as in Chapter 3 using parallel reduction. ∎

4.5 Conclusion

In this chapter, we proposed the calculus $cF^{++}_{\leq}$ by adding the type instantiation rule to $cF^{+}_{\leq}$. We showed that, if the type instantiation rule is added, the reduction of type is always terminating. Thus, $cF^{++}_{\leq}$ is a sound extension of $cF_{\leq}$ and, in $cF^{++}_{\leq}$, many terms which cannot type-check in $cF_{\leq}$ but can do in $F_{\leq}$ can have its type. For example, $\Lambda X{\leq}(\forall Y{\leq}S.S).\lambda x^{X}.x\{S\}$ cannot have its type in $cF_{\leq}$ but have the type $\forall X{\leq}(\forall Y{\leq}S.S).X \to S$ in $F_{\leq}$. In $cF^{++}_{\leq}$ it has the type $\forall X{\leq}(\forall Y{\leq}S.S).X \to \mathrm{Inst}(X,S)$.

Chapter 5

Symbiosis of Parametric and Ad-hoc Polymorphisms

In this chapter we introduce $cF^{\&}_{\leq}+$ a calculus integrating parametric polymorphism and ad-hoc polymorphism by correcting mistakes of $F^{\&}_{\leq}$ and using the techniques of $cF^{+}_{\leq}$ and $cF^{++}_{\leq}$. Among this thesis, we use some informal judgements: for examples $\Gamma \vdash S \equiv T$ stands for "$\Gamma \vdash S \leq T$ and $\Gamma \vdash T \leq S$", and $A_h = \min_{i \in I}\{A_i \mid \Gamma \vdash A \leq A_i\}$ stands for "$\Gamma \vdash A \leq A_h$ and for every A_i such that $\Gamma \vdash A \leq A_i$ holds, $(\Gamma \vdash) A_h \leq A_i$ holds".

5.1 Introduction

$F^{\&}_{\leq}$ [Cas93] seems the first calculus that integrates parametric polymorphism and ad-hoc polymorphism. As in [Cas93], in a programming language a function which performs a dispatch on a type passed as an argument would be probably written as:

```
Fun(X:*) =>
   if X<T1 then exp.1 else if X<T2 then exp.2 ... else if X<Tn then exp.n
```

This function executes exp_1 if we pass to it a type less than or equal to T_1, and so on. If there is more than one candidate we select among them the branch with the least bound. In $F^{\&}_{\leq}$ this function is denoted by:

$$\Lambda X{\leq}T_1.exp_1 \& \Lambda X{\leq}T_2.exp_2 \& \ldots \& \Lambda X{\leq}T_n.exp_n$$

and its type is $\forall X\{T_1.S_1, T_2.S_2, \ldots, T_n.S_n\}$ where $(\, X \leq T_i \vdash exp_i : S_i \,)$. However this type is a rough approximation yet. Indeed, to obtain a coherent and expressive system, we need strong restriction on the T_i's and S_i's.

First of all note that the selected branch may change during the computation. For example take a function f of type $\forall X.\{T_1.S_1, T_2.S_2\}$ with $T_2 \leq T_1$. Consider now the expression $(\forall Y \leq T_1.f[Y])$. Since $Y \leq T_1$ we guess that this expression will be $\forall X{\leq}T_1.S_1$. But if we pass T_2 subtype to the function, then, as Y is bound to T_2, the selected branch will be the second one and the result will have type S_2. If we want reductions to preserve the type in $F^{\&}_{\leq}$, we must require S_2 to be the same type as S_1, which seems to be too strong a restriction. In $F^{\&}_{\leq}$, to ease this restriction, the constraint which is imposed on $F^{\&}_{\leq}$ is that $\vdash T_i \leq T_j$ then $X \leq T_i \vdash S_i \leq S_j$ (we call it the *covariance condition*), not $X \leq T_i \vdash S_j \leq S_i$ (we call it the *contravariance condition*

). However, in this calculus the substitution property of the type system does not hold because, in $F^{\&}_{\leq}$, overloaded types does not satisfy contravariance of the left argument in the subtype relation.

For example,

$$\underline{X} \leq \text{Real} \vdash \forall Y.\{\underline{X}.\text{Int}, \text{Real}.\text{Real}\} \quad \texttt{type}$$

is a valid type in $F^{\&}_{\leq}$. However, when Real is substituted for X, we have to prove

$$\vdash \forall Y.\{\underline{\text{Real}}.\text{Int}, \text{Real}.\text{Real}\} \quad \texttt{type},$$

only to fail.

To overcome this mistake, we must impose contravariance (as well as covariance) of the left argument in the subtype relation on overloaded types of the new calculus $cF^{\&}_{\leq}+$.

We limit our study to the case where the bounds of an overloaded function range over basic types and record types all fields of which are base types. Indeed, the use of arrow types in the bounds posses many non-trivial and complicated problems to solve, due to the covariance and contravariance of the left argument in the subtype relation and the definition of $\cap$-closed.

Definition 5.1.1.

Let Γ an environment. Given a set of base types or record types whose all fields are base types[1] $\{A_i\}_{i\in I}$ we write $\Gamma \vdash \{A_i\}_{i\in I}$ $\cap$*-closed* iff for all $i, j \in I$ if A_i and A_j have a common lower bound, then there exists $h \in I$ where under Γ, A_h is the g.l.b. (abbreviation of "greatest lower bound") of A_i and A_j. We write it as $\Gamma \vdash A_h = A_i \cap A_j$.

Note that $\cap$-closure is quite a draconian restriction. Indeed $\cap$-closed sets of bounds have a very preside form : they are partial downward semi-lattices, i.e. formed by disjoint unions of downward semi-lattices.

5.2 Type

In this section we describe the type system. We first define the pretypes.

Definition 5.2.1. (PreType)

T	:=	P	non-well-working type
	|	W	working type
P	:=	X	type variable
	|	$\text{Range}(P)$	range type
	|	$\text{Sel}(P, l)$	field type
	|	$\text{Inst}(P, B)$	instantiation type
W	:=	B	base type
	|	$T \rightarrow T$	function type
	|	$[l_i{:}T_i]_{i=1,\cdots,n}$	record type
	|	$\forall X.T$	universal type
	|	$\forall X{\leq}T.T$	bounded universal type
	|	$\forall X\{A_1.T_1, \cdots, A_n.T_n\}$	overloaded type
		also denoted by $\forall X\{A_i.T_i\}_{i=1,\cdots,n}$	

We call a (pre)type A a *hereditary basic type* if A is a base type or a record type whose all fields are hereditary basic types, i.e.,

A	=	B	base type
	|	$[l_1{:}A_1, \ldots, l_n{:}A_n]$	record type whose all fields are hereditary basic types

[1] In $cF^{\&}_{\leq}+$, the subtype relation between base types is *not* considered, while it is in $F^{\&}_{\leq}$

A type $\forall X\{A_i.T_i\}_{i=1,\cdots,n}$ represents a *overloaded type*. Overloaded types $\forall X\{A_i.T_i\}_{i=1,\cdots,n}$ must:

1. have bounds ranging over hereditary basic types.
2. have a $\cap$-closed set of bounds.
3. satisfy covariance and contravariance, i.e., if $A_i \leq A_j$ then $X \leq A_i \vdash T_i = T_j$.

Definition 5.2.2.
For a non-well-working type Range(P) (respectively Sel(P,l_i)) , a working type $S \to T$ ($[l_1 : T_1, \cdots, l_i : T_i, \cdots, l_n : T_n]$) is substituted for P . The (raw) type Range($S \to T$) (Sel($[l_1 : T_1, \cdots, l_i : T_i, \cdots, l_n : T_n], l_i$)) need to be reduced to a "normal form".

$$\begin{array}{lll} \Gamma \vdash & \text{Range}(S \to T) & \Rightarrow \quad T \\ \Gamma \vdash & \text{Sel}([l_i{:}T_i]_{i\in I}, l_j) & \Rightarrow \quad T_j \\ \Gamma \vdash & \text{Inst}(\forall X.\{A_i.T_i\}_{i\in I}, A) & \Rightarrow \quad T_h[X := A] \\ & & \quad \text{where } \Gamma \vdash A_h = \min_{i\in I}\{A_i \mid \Gamma \vdash A \leq A_i\} \end{array}$$

Definition 5.2.3. (Type)
We mutually define the type environment and the type good-formation:

$$\frac{\phi \ \ \mathsf{env}}{\Gamma \ \ \mathsf{env} \quad X \notin \mathrm{FV}(()\Gamma)}{\Gamma, X \ \ \mathsf{env}}$$

$$\frac{\Gamma \ \ \mathsf{env} \quad \Gamma \vdash S \ \ \mathsf{type} \quad X \notin \mathrm{FV}(\Gamma)}{\Gamma, X \leq S \ \ \mathsf{env}}$$

$$\frac{X \in FV(\Gamma)}{\Gamma \vdash X \ \ \mathsf{type}} \ (\text{Var})$$

$$\frac{\Gamma \vdash S \ \ \mathsf{type} \quad \Gamma \vdash T \ \ \mathsf{type}}{\Gamma \vdash S \to T \ \ \mathsf{type}} \ (\to)$$

$$\frac{\Gamma, X \vdash T \ \ \mathsf{type}}{\Gamma \vdash \forall X.T \ \ \mathsf{type}} \ (\forall)$$

$$\frac{\Gamma \vdash S \ \ \mathsf{type} \quad \Gamma, X \leq S \vdash T \ \ \mathsf{type}}{\Gamma \vdash \forall X{\leq}S.T \ \ \mathsf{type}} \ (\text{bound-}\forall)$$

$$\frac{\text{For all } i = 1, \cdots, n, \ \Gamma \vdash T_i \ \ \mathsf{type}}{\Gamma \vdash [l_1 : T_1, \cdots, l_n : T_n] \ \ \mathsf{type}} \ (\text{record})$$

$$\frac{\left\{ \begin{array}{c} \Gamma \vdash \forall X{\leq}A_i.T_i \ \ \mathsf{type} \\ \{A_i\}_{i\in I} \cap\text{-closed} \\ \text{If } \Gamma \vdash A_i \leq A_j, \text{ then } \Gamma, X \leq A_i \vdash T_i = T_j \end{array} \right\}}{\Gamma \vdash \forall X.\{A_i.T_i\}_{i\in I} \ \ \mathsf{type}} \ \{\}$$

$$\frac{\Gamma \vdash P \ \ \mathsf{type} \quad \Gamma^*(P) = S \to T}{\Gamma \vdash \text{Range}(P) \ \ \mathsf{type}} \ (\text{Range})$$

$$\frac{\Gamma \vdash P \ \ \mathsf{type} \quad \Gamma^*(P) = [l_1 : T_1 \cdots, l_i : T_i, \cdots, l_n : T_n]}{\Gamma \vdash \text{Sel}(P, l_i) \ \ \mathsf{type}} \ (\text{record-Var})$$

$$\frac{\Gamma \vdash P \ \ \mathsf{type} \quad \Gamma^*(P) = \forall X.\{A_i.T_i\}_{i\in I} \quad \text{there is } h \in I \text{ s.t. } \Gamma \vdash U \leq A_h}{\Gamma \vdash \text{Inst}(P, U) \ \ \mathsf{type}} \ (\text{Inst})$$

Definition 5.2.4. (Subtype Relation)
The rules of the subtype relation are listed as follows:

$$\frac{\Gamma \vdash T \ \ \mathsf{type}}{\Gamma \vdash T \leq T} \ (\leq\text{-Refl})$$

$$\frac{\Gamma \vdash S \leq T \quad \Gamma \vdash T \leq U}{\Gamma \vdash S \leq U} \ (\leq\text{-Trans})$$

$$\frac{\Gamma \vdash P \ \ \mathsf{type} \quad \Gamma(P) \trianglerighteq V \quad \Gamma \vdash V \leq T}{\Gamma \vdash P \leq T} \ (\leq\text{-Taut})$$

$$\frac{\Gamma \vdash S' \leq S \quad \Gamma \vdash T \leq T'}{\Gamma \vdash S \to T \leq S' \to T'} \ (\leq\text{-}\to)$$

$$\frac{\Gamma \vdash \forall X.T \ \ \mathsf{type} \quad \Gamma, X \vdash T \leq T'}{\Gamma \vdash \forall X.T \leq \forall X.T'} \ (\leq\text{-}\forall)$$

$$\frac{\Gamma \vdash \forall X.T \ \ \mathsf{type} \quad \Gamma \vdash S' \ \ \mathsf{type} \quad \Gamma, X \leq S' \vdash T \leq T'}{\Gamma \vdash \forall X.T \leq \forall X{\leq}S'.T'} \ (\leq\text{-b}\forall)$$

$$pt \quad \frac{\Gamma \vdash \forall X{\leq}S.T \ \ \mathsf{type} \quad \Gamma \vdash S' \leq S \quad \Gamma, X \leq S' \vdash T \leq T'}{\Gamma \vdash \forall X{\leq}S.T \leq \forall X{\leq}S'.T'} \ (\leq\text{-bb}\forall)$$

$$\frac{\begin{cases} \Gamma \vdash \forall X.\{A_i.S_i\}_{i\in I} \ \ \mathsf{type} \qquad \Gamma \vdash \forall X.\{B_i.T_i\}_{j\in J} \ \ \mathsf{type} \\ \text{For all } j \in J, \text{ there exists } i \in I \text{ s.t.} \Gamma \vdash B_j \leq A_i \text{ and } \Gamma, X \leq B_j \vdash S_i \leq T_j \end{cases}}{\Gamma \vdash \Gamma \vdash \forall X.\{A_i.S_i\}_{i\in I} \leq \forall X.\{B_j.T_j\}_{j\in J}} \ (\leq\text{-}\{\})$$

$$\frac{\text{For all } i = 1, \cdots, n, \ \Gamma \vdash S_i \leq T_i}{\begin{cases} \Gamma \vdash [l_1{:}S_1, \cdots, l_n{:}S_n, l_{n+1}{:}S_{n+1}, \cdots, l_{n+m}{:}S_{n+m}] \\ \qquad\qquad \leq [l_1{:}T_1, \cdots, l_n{:}T_n] \end{cases}} \ (\leq\text{-record})$$

$$\frac{\Gamma \vdash \mathrm{Range}(P) \ \ \mathsf{type} \quad \Gamma \vdash \mathrm{Range}(Q) \ \ \mathsf{type} \quad \Gamma \vdash P \leq Q}{\Gamma \vdash \mathrm{Range}(P) \leq \mathrm{Range}(Q)} \ (\leq\text{-Range})$$

$$\frac{\Gamma \vdash \mathrm{Sel}(P, l) \ \ \mathsf{type} \quad \Gamma \vdash \mathrm{Sel}(Q, l) \ \ \mathsf{type} \quad \Gamma \vdash P \leq Q}{\Gamma \vdash \mathrm{Sel}(P, l) \leq \mathrm{Sel}(Q, l)} \ (\leq\text{-Sel})$$

$$\frac{\Gamma \vdash \mathrm{Inst}(P, U) \ \ \mathsf{type} \quad \Gamma \vdash \mathrm{Inst}(Q, U) \ \ \mathsf{type} \quad \Gamma \vdash P \leq Q}{\Gamma \vdash \mathrm{Inst}(P, U) \leq \mathrm{Inst}(Q, U)} \ (\leq\text{-Inst})$$

The type system of $cF^{\&}_{\leq}+$ satisfies the following property.

Proposition 5.2.5.
Assume $\Gamma \vdash U \leq S$ holds, then $\Gamma \vdash U \ \ \mathsf{type}$ and $\Gamma \vdash S \ \ \mathsf{type}$ hold.

(**Proof**): Obvious. ∎

Remark 5.2.6.
From these definitions, a set of hereditary basic types has a partial downward semi-latice structure. An example of the structure of $\cap$-closed set is written in Figure 5.2.6.

Proposition 5.2.7. (Substitution Lemma)

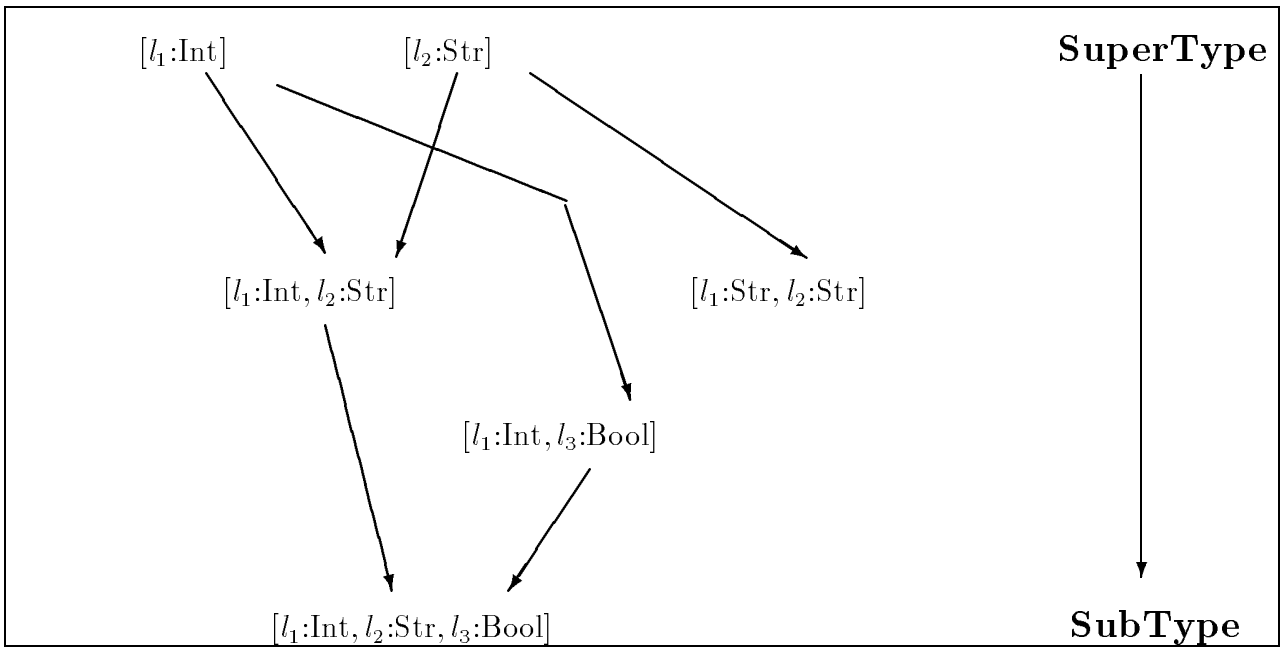

Figure 5.1: Example of the Structure of $\cap$-closed Set

1. Assume $\Gamma \vdash U$ type , $\Gamma \vdash T[X := U] \Rrightarrow T'$, $\Gamma \vdash V[X := U] \Rrightarrow V'$ and $\Gamma \vdash \Delta[X := U] \Rrightarrow \Delta'$ hold,
   (1) If $\Gamma, X, \Delta \vdash T$ type holds, then $\Gamma, \Delta' \vdash T'$ type .
   (2) If $\Gamma, X, \Delta \vdash T \leq V$ type holds, then $\Gamma, \Delta' \vdash T' \leq V'$.
2. Assume $\Gamma \vdash U \leq S$, $\Gamma \vdash T[X := U] \Rrightarrow T'$, $\Gamma \vdash V[X := U] \Rrightarrow V'$ and $\Gamma \vdash \Delta[X := U] \Rrightarrow \Delta'$ hold,
   (1) If $\Gamma, X \leq S, \Delta \vdash T$ type holds, then $\Gamma, \Delta' \vdash T'$ type .
   (2) If $\Gamma, X \leq S, \Delta \vdash T \leq V$ type holds, then $\Gamma, \Delta' \vdash T' \leq V'$.

   (Remark)
   Assume $\Delta \equiv Y_1 \leq S_1, \ldots, Y_n \leq S_n$. If $\Gamma \vdash S_1[X := U] \Rrightarrow S_1'$, $\Gamma, Y_1 \leq S_1' \vdash S_2[X := U] \Rrightarrow S_2'$, ..., and $\Gamma, Y_1 \leq S_1', \ldots, Y_{n-1} \leq S_{n-1}' \vdash S_n[X := U] \Rrightarrow S_n'$ hold, we define $\Delta' \equiv Y_1 \leq S_1', \ldots, Y_{n-1} \leq S_{n-1}', Y_n \leq S_n'$ and write down $\Gamma \vdash \Delta[X := U] \Rrightarrow \Delta'$.

(**Proof**): We prove only the second case:

1. Induction on the structure of the proof of ($\Gamma, X \leq S, \Delta \vdash T$ type) and the size of Δ . We prove only two cases: the first is

   Case ($\Gamma, X \leq S, \Delta \vdash \forall Y.\{A_i.T_i\}_{i \in I}$ type .)

$$\frac{\left\{ \begin{array}{c} \Gamma, X \leq S, \Delta \vdash \forall Y \leq A_i.T_i \text{ type} \\ \{A_i\}_{i \in I} \cap\text{-closed} \\ \Gamma, X \leq S, \Delta \vdash A_i \leq A_j \quad \Rightarrow \quad \Gamma, X \leq S, \Delta, Y \leq A_i \vdash T_i = T_j \end{array} \right\}}{\Gamma, X \leq S, \Delta \vdash \forall Y.\{A_i.T_i\}_{i \in I} \text{ type}} \{\}$$

   Because A_i is closed, i.e., A_i includes no type variable, if $\Gamma, X \leq S, \Delta \vdash A_i \leq A_j$ holds, then $\Gamma, \Delta' \vdash A_i \leq A_j$ holds. The rest of this proof is quote similar to that in Chapter 3.

Case ($\Gamma \vdash \mathrm{Inst}(P, W)$ type .)

$$\frac{\Gamma \vdash P \text{ type} \quad \Gamma^*(P) = \forall Y.\{A_i.T_i\}_{i\in I} \quad \text{there is } h \in I \text{ s.t. } \Gamma \vdash W \leq A_h}{\Gamma \vdash \mathrm{Inst}(P, W) \text{ type}} \ (\mathrm{Inst})$$

We consider two subcases:

(1) ($P[X := U] \Rrightarrow \forall Y.\{A'_j.T'_j\}_{j\in J}$)
Assume $W[X := U] \Rrightarrow W'$. By ,induction hypothesis $\mathrm{Inst}(P, W)[X := U] \Rrightarrow T''_h$ where $\Gamma, \Delta' \vdash A'_h = \min_{j\in J}\{A'_j \mid \Gamma, \Delta' \vdash W' \leq A'_j\}$ and $T'_h[Y := W'] \Rrightarrow T''_h$. Such h exists because $\mathrm{Wlub}_\Gamma(P) \equiv \forall Y.\{B_i.S_i\}_{i\in I}$ holds and k exists such that $\Gamma, X \leq U, \Delta \vdash W \leq B_k$ and therefore we have $\Gamma, \Delta' \vdash W' \leq B'_k$ where $B_k[X := U] \Rrightarrow B'_k$. Then we have from substitution lemma:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Gamma, \Delta', Y \leq A'_h \vdash T'_h \text{ type} & \Gamma, \Delta' \vdash W' \leq A'_h \quad T'_h[Y := W'] \Rrightarrow T''_h \\ & \vdots \ \text{(By induction hypothesis,)} \\ & \Gamma, \Delta' \vdash T''_h \text{ type}\end{array}}{}$$

(2) ($P[X := U] \Rrightarrow Q$)
Easy.

Other cases are proved similarly to Chapter 3.

2. Induction on the structure of the proof of $\Gamma, X \leq S, \Delta \vdash T \leq V$ and the size of Δ . In this chapter we show the proof of two cases:

Case ($\Gamma, X \leq S, \Delta \vdash \mathrm{Inst}(P, W) \leq T$.)

(1) ($P[X := W] \Rrightarrow \forall Y.\{A_j.T_j\}_{j\in J}$)
We have $\Gamma(P)[X := U] \Rrightarrow \forall Y.\{B_i.S_i\}_{i\in I}$ by induction hypothesis and the rules of the subtype relation. Assuming $W[X := U] \Rrightarrow W'$, we have $\mathrm{Inst}(P, W)[X := U] \Rrightarrow T'_h$ where $A_h = \min\{A_j \mid \Gamma, X \leq S, \Delta \vdash W \leq A_j\}$ and $T_h[X := U] \Rrightarrow T'_h$. Such h exists because, from induction hypothesis, there exists k such that $B_k = \min\{B_i \mid \Gamma, X \leq S, \Delta \vdash W \leq B_i\}$ holds.
Therefore we have:

$$\dfrac{\begin{array}{c}\vdots \\ \Gamma, \Delta' \vdash T''_h \leq T'\end{array}}{}$$

(2) ($P[X := W] \Rrightarrow Q$) Assume $W[X := U] \Rrightarrow W'$. Obvious from induction hypothesis, we have

$$\dfrac{\begin{array}{c}\vdots \\ \Gamma, \Delta' \vdash Q' \leq T' \quad \mathrm{Inst}((\Gamma, \Delta')(Q), W') \Rrightarrow Q'\end{array}}{\Gamma, \Delta' \vdash \mathrm{Inst}(Q, W) \leq T'}$$

Case ($\Gamma, X \leq S, \Delta \vdash \mathrm{Inst}(P, W) \leq \mathrm{Inst}(Q, W)$.)
Assume that $W[X := U] \Rrightarrow W'$.

(1) ($P[X := U] \Rrightarrow \forall Y.\{A_j.T_j\}_{j\in J}$, $Q[X := U] \Rrightarrow \forall Y.\{B_i.S_i\}_{i\in I}$)
Since, by induction hypothesis, $\Gamma, \Delta' \vdash \forall Y.\{A_j.T_j\}_{j\in J} \leq \forall Y.\{B_i.S_i\}_{i\in I}$ holds, we have that for all $i \in I$ there exists $j \in J$ such that $\Gamma, \Delta' \vdash B_i \leq A_j$ and $\Gamma, \Delta', Y \leq B_i \vdash T_j \leq S_i$ hold. Then by induction hypothesis and the definition of overloaded types, there exist j_0 and i_0 such that $\Gamma, \Delta' \vdash B_{i_0} \leq A_{j_0}$, $\Gamma, \Delta', Y \leq B_{i_0} \vdash T_{j_0} \leq S_{i_0}$, $B_{i_0} = \min\{B_i \mid \Gamma, \Delta' \vdash W' \leq B_i\}$ and $A_{j_0} = \min\{A_j \mid \Gamma, \Delta' \vdash W' \leq A_j\}$ hold.

Therefore by induction hypothesis, $\Gamma, \Delta' \vdash T'_{j_0} \leq S'_{i_0}$ holds where $T_{j_0}[Y := W'] \Rrightarrow T'_{j_0}$ and $S_{i_0}[Y := W'] \Rrightarrow S'_{i_0}$ hold.

(2) ($P[X := U] \Rrightarrow P'$ and $Q[X := U] \Rrightarrow \forall Y.\{B_i.S_i\}_{i \in I}$)
By induction hypothesis, there exists i_0 such that $B_{i_0} = \min\{B_i \mid \Gamma, \Delta' \vdash W' \leq B_i\}$ holds. Therefore we have $\Gamma, \Delta' \vdash \mathrm{Inst}(P', W') \leq B'_{i_0}$ where $B_{i_0}[Y := W'] \Rrightarrow B'_{i_0}$ holds.

(3) ($P[X := U] \Rrightarrow P'$ and $Q[X := U] \Rrightarrow Q'$)
Obvious.

The other cases are proved similarly to Chapter 3.

■

5.3 Term

In this section we describe the terms of the language. We start by the definition of the *preterms*, among which we distinguish the *terms*, i.e. those preterms that possess a type. Roughly speaking, (pre)terms are defined in three classes: terms of the simply typed λ-calculus, terms for parametric polymorphism ($cF^+_{\leq}$) and terms for overloading[2]. Overloaded functions are built in a list fashion, starting by an *empty* overloaded function ϵ and concatenating new branches by $\&$.

Definition 5.3.1. (PreTerm)

e	$:=$	$x^T \mid c^T \mid \lambda x^T.e \mid ee$	simply-typed λ-calculi
	$\mid$	$\{l_1 = e, \cdots, l_n = e\} \mid e \cdot l$	record calculi
	$\mid$	$\Lambda X.e \mid \Lambda X \leq T.e \mid e\{T\}$	$cF^+_{\leq}$
	$\mid$	$\epsilon \mid e \& f \mid e[A]$	overloading
	$\mid$	$e\vert_T$	coerced term

It is required that the bounds of an overloaded function range over hereditary basic types Therefore, the argument of an overloaded function can be restricted to be a basic hereditary basic types.

Definition 5.3.2. (Term)

$$\frac{\Gamma \vdash T \ \ \mathsf{type}}{\Gamma \vdash x^T : T} \ (\text{var})$$

$$\frac{\Gamma \vdash e : T}{\Gamma \vdash \lambda x^S.e : S \to T} \ (\lambda\text{-ab})$$

$$\frac{\Gamma \vdash e : S \to T \quad \Gamma \vdash f : U \quad \Gamma \vdash U \leq S}{\Gamma \vdash ef : T} \ (\lambda\text{-ap})$$

$$\frac{\Gamma \vdash e : P \quad \Gamma \vdash f : U \quad \Gamma^*(P) = S \to T \quad \Gamma \vdash U \leq S}{\Gamma \vdash ef : \mathrm{Range}(P)} \ (\lambda\text{-Range})$$

$$\frac{\Gamma, X \vdash e : T}{\Gamma \vdash \Lambda X.e : \forall X.T} \ (\Lambda\text{-ab})$$

$$\frac{\Gamma, X \leq S \vdash e : T}{\Gamma \vdash \Lambda X \leq S.e : \forall X \leq S.T} \ (\text{bound-}\Lambda\text{-ab})$$

$$\frac{\Gamma \vdash e : \forall X \leq S.T \quad \Gamma \vdash U \leq S \quad T[X := U] \Rrightarrow T'}{\Gamma \vdash e\{U\} : T'} \ (\text{bound-}\Lambda\text{-ap-w})$$

[2] and records and coercion.

$$\frac{}{\Gamma \vdash \epsilon : \forall X.\{\}} \text{ (overloaded-}\epsilon\text{)}$$

$$\frac{\left\{\begin{array}{l}\Gamma \vdash e : \forall X.\{A_i.T_i\}_{i=1,\ldots,n-1} \quad \Gamma \vdash f : \forall X \leq A_n.T_n \\ \Gamma \vdash \forall X.\{A_i.T_i\}_{i=1,\ldots,n} \ \mathsf{type}\end{array}\right.}{\Gamma \vdash e \& f : \forall X.\{A_i.T_i\}_{i=1,\ldots,n}} \text{ (overloaded-}\Lambda\text{-ab)}$$

$$\frac{\left\{\begin{array}{l}\Gamma \vdash e : \forall X.\{A_i.T_i\}_{i=1,\ldots,n} \\ A_h = \min_{i=1,\ldots,n}\{A_i \mid \Gamma \vdash U \leq A_i\} \\ T_h[X := U] \Rrightarrow T'_h\end{array}\right.}{\Gamma \vdash e[U] : T'_h} \text{ (overloaded-}\Lambda\text{-ap-w)}$$

$$\frac{\Gamma \vdash e : P \quad \Gamma \vdash \mathrm{Inst}(P,U) \ \mathsf{type}}{\Gamma \vdash e[U] : \mathrm{Inst}(P,U)} \text{ (}\Lambda\text{-ap-nw)}$$

$$\frac{\text{For all } i = 1, \cdots, n, \Gamma \vdash e_i : T_i}{\Gamma \vdash \{l_1 = e_1, \cdots, l_n = e_n\} : [l_1 : T_1, \cdots, l_n : T_n]} \text{ (record-const)}$$

$$\frac{\Gamma \vdash e : [\cdots, l{:}T, \cdots]}{\Gamma \vdash e \cdot l : T} \text{ (record-select)}$$

$$\frac{\Gamma \vdash e : P \quad \Gamma \vdash \mathrm{Sel}(P,l) \ \mathsf{type}}{\Gamma \vdash e \cdot l : \mathrm{Sel}(P,l)} \text{ (record-Sel)}$$

$$\frac{\Gamma \vdash e : S \quad \Gamma \vdash S \leq T}{\Gamma \vdash e|_T : T} \text{ (coer)}$$

Proposition 5.3.3.

Assume $\Gamma \vdash U \leq S$, $\Gamma \vdash T[X := U] \Rrightarrow T'$, $\Gamma \vdash e[X := U] \Rrightarrow e'$ and $\Gamma \vdash \Delta[X := U] \Rrightarrow \Delta'$ hold, If $\Gamma, X \leq S, \Delta \vdash e : T$ holds, then $\Gamma, \Delta' \vdash e' : T'$.

(**Proof**): Induction on the structure of the proof of ($\Gamma, X \leq S, \Delta \vdash e : T$).

We prove (overloaded- Λ -ap-w) and from Proposition 5.2.7, the other cases are proved similarly to Chapter 3.

$$\frac{\left\{\begin{array}{l}\Gamma, X \leq S, \Delta \vdash e : \forall Y.\{A_i.T_i\}_{i=1,\ldots,n} \\ A_h = \min_{i=1,\ldots,n}\{A_i \mid \Gamma, X \leq S, \Delta \vdash V \leq A_i\} \\ T_h[Y := V] \Rrightarrow T'_h\end{array}\right.}{\Gamma, X \leq S, \Delta \vdash e[V] : T'_h}$$

When $\Gamma, X \leq S, \Delta \vdash A_h = \min_{i=1,\ldots,n}\{A_i \mid \Gamma, X \leq S, \Delta \vdash V' \leq A_i\}$ holds, then there is some $j \in I$ such that $\Gamma, \Delta' \vdash A_j = \min_{i=1,\ldots,n}\{A_i \mid \Gamma, \Delta' \vdash V' \leq A_i\}$ holds where $V[X := U] \Rrightarrow V'$. By induction hypothesis, $\vdash A_j \leq A_h$ holds, and therefore we have $\Gamma, X \leq S, \Delta, Y \leq A_j \vdash T_h \equiv T_j$ holds. Hence $\Gamma, \Delta, Y \leq A \vdash T'_h \equiv T'_j$ holds where $T_i[X := U] \Rrightarrow T'_i$ ($i = 1, \ldots, n$). Then we have $\Gamma, \Delta \vdash T''_h \equiv T''_j$ where $T'_i[Y := V'] \Rrightarrow T''_i$ ($i = 1, \ldots, n$). ∎

5.4 Reduction

In this section we give the equational theory of the terms of $cF^{\&}_{\leq +}$. We present it under the form of reduction rules.

Definition 5.4.1.

Assume we have global type environment Γ . We call a type T *closed* if no type variable is included in T .

Definition 5.4.2. (Reduction)

$$\frac{}{\Gamma \vdash (\lambda x^S.e)f \quad \rightarrow \quad e[x := f|_S]} \ (\beta)$$

$$\frac{\Gamma \vdash e[X := U] \exists\!\!\!/ e'}{\Gamma \vdash (\Lambda X.e)\{T\} \quad \rightarrow \quad e'} \ (\beta 2)$$

$$\frac{\Gamma \vdash e[X := U] \exists\!\!\!/ e'}{\Gamma \vdash (\Lambda X \leq S.e)\{T\} \quad \rightarrow \quad e'} \ (\text{bound } \beta 2)$$

$$\frac{\left\{ \begin{array}{l} A \text{ is closed.} \quad \Gamma \vdash e\&f : \forall X.\{A_i.T_i\}_{i=1,\ldots,n} \\ n = \max\{h \mid A_h = \min_{i=1,\ldots,n}\{A_i \mid \Gamma \vdash A \leq A_i\}\} \end{array} \right.}{\Gamma \vdash (e\&f)[A] \quad \rightarrow f\{A\}} \ (\beta\text{-overload1})$$

$$\frac{\left\{ \begin{array}{l} A \text{ is closed.} \quad \Gamma \vdash e\&f : \forall X.\{A_i.T_i\}_{i=1,\ldots,n} \\ n \neq \max\{h \mid A_h = \min_{i=1,\ldots,n}\{A_i \mid \Gamma \vdash A \leq A_i\}\} \end{array} \right.}{\Gamma \vdash (e\&f)[A] \quad \rightarrow e[A]} \ (\beta\text{-overload2})$$

$$\frac{\Gamma \vdash f : T \quad \text{Range}(S) \exists\!\!\!/ V}{\Gamma \vdash (e|_S)f \quad \rightarrow \quad (ef)|_V} \ (\text{coer})$$

$$\frac{\Gamma \vdash e : \forall X \leq V.W \quad T[X := U] \exists\!\!\!/ T'}{\Gamma \vdash (e|_{\forall X \leq S.T})\{U\} \quad \rightarrow \quad (e\{U\})|_{T'}} \ (\text{coer2})$$

$$\frac{\Gamma \text{Inst}(P, A) \exists\!\!\!/ P')}{\Gamma \vdash (e|_P)[A] \quad \rightarrow \quad (e[A])|_{P'}} \ (\text{coerInst})$$

$$\frac{e \text{ is in normal form.} \quad S \text{ is a base type.}}{\Gamma \vdash e|_S \quad \Rightarrow \quad e} \ (\text{coerconst})$$

Besides these rules there are the usual rules for the context closure; among these the only one that deserves a note is the rule for Λ. For what it concerns the rules, note that in (β-overload1) and (β-overload2) we require that the types involved in the selection of a branch are closed. In this way we always select the most precise branch. In this chapter, we impose the restriction on the reduction rule (β-overload1) and (β-overload2) because we want $cF^{\&}_{\leq}+$ to satisfy Church-Rosser. However, this restriction is based on the global type environment, and so and, in future, we will have to consider the specific reduction strategy like λv-calculus [Plo75].

$cF^{\&}_{\leq}+$ has there good properties:

Proposition 5.4.3.

1. $cF^{\&}_{\leq}+$ satisfies Subject Reduction.
2. $cF^{\&}_{\leq}+$ satisfies Church-Rosser.

(**Proof**):

1. (Subject Reduction)
   We prove only the case (β-overload1) and (β-overload2). The other cases are trivial or proved similarly as in Chapter 3 and 4.

Case (β-overload1.)

$$\dfrac{\boxed{\mathbf{I}} \quad A_n = \min_{i=1,\ldots,n}\{A_i \mid \Gamma \vdash A \leq A_i\} \qquad T_n[X := A] \Rrightarrow T'_n}{\Gamma \vdash (e\&f)[A] : T'_n}$$

where $\mathbf{I}$ is

$$\dfrac{\Gamma \vdash e : \forall X\{A_i.T_i\}_{i=1,\ldots n-1} \qquad \Gamma \vdash f : \forall X{\leq}A_n.T_n \qquad \Gamma \vdash \forall X.\{A_i.T_i\}_{i=1,\ldots,n} \ \mathsf{type}}{\Gamma \vdash e\&f : \forall X.\{A_i.T_i\}_{i=1,\ldots,n}}$$

$$\Downarrow$$

$$\dfrac{\Gamma \vdash f : \forall X{\leq}A_n.T_n \qquad \Gamma \vdash A \leq A_n \qquad T_n[X := A] \Rrightarrow T'_n}{\Gamma \vdash f\{A\} : T'_n}$$

Case (β-overload2.)

$$\dfrac{\boxed{\mathbf{I}} \quad A_j = \min_{i=1,\ldots,n}\{A_i \mid \Gamma \vdash A \leq A_i\} \qquad T_j[X := A] \Rrightarrow T'_j}{\Gamma \vdash (e\&f)[A] : T'_n}$$

where $\mathbf{I}$ is

$$\dfrac{\Gamma \vdash e : \forall X\{A_i.T_i\}_{i=1,\ldots n-1} \qquad \Gamma \vdash f : \forall X{\leq}A_n.T_n \qquad \Gamma \vdash \forall X.\{A_i.T_i\}_{i=1,\ldots,n} \ \mathsf{type}}{\Gamma \vdash e\&f : \forall X.\{A_i.T_i\}_{i=1,\ldots,n}}$$

$$\Downarrow$$

$$\dfrac{\Gamma \vdash e : \forall X\{A_i.T_i\}_{i=1,\ldots n-1} \qquad A_j = \min_{i=1,\ldots,n-1}\{A_i \mid \Gamma \vdash A \leq A_i\} \qquad T_j[X := A] \Rrightarrow T'_j}{\Gamma \vdash e[A] : T'_j}$$

2. (Church-Rosser)
Use parallel reduction as in Chapter 3 and 4. What to take care of is the reduction (β-overload). Consider a rule the parallel reduction derived from (β-overload1) and (β-

overload2):

$$\frac{\left\{\begin{array}{l} A \text{ is closed.} \quad e \rhd e' \quad f \rhd f' \\ \Gamma \vdash e : \forall X.\{A_i.T_i\}_{i=1,\ldots,n-1} \quad \Gamma \vdash f : \forall Y \leq A_n.T_n \\ n = \max\{h \mid A_h = \min_{i=1,\ldots,n}\{A_i \mid \Gamma \vdash A \leq A_i\}\} \end{array}\right.}{\Gamma \vdash (e\&f)[A] \rhd f'\{A\}} \text{ (PR}\beta\text{-overload1)}$$

$$\frac{\left\{\begin{array}{l} A \text{ is closed.} \quad e \rhd e' \quad f \rhd f' \\ \Gamma \vdash e : \forall X.\{A_i.T_i\}_{i=1,\ldots,n-1} \quad \Gamma \vdash f : \forall Y \leq A_n.T_n \\ n \neq \max\{h \mid A_h = \min_{i=1,\ldots,n}\{A_i \mid \Gamma \vdash A \leq A_i\}\} \end{array}\right.}{\Gamma \vdash (e\&f)[A] \rhd e'[A]} \text{ (PR}\beta\text{-overload2)}$$

However, because A is closed, such m in (PR β-overload1) and (PR β-overload2) need to be fixed, the Church-Rosser of $cF^{\&}_{\leq}+$ is easily proved by parallel reduction.

∎

5.5 Example of Applications to Object-Oriented Languages

We discuss about the re-usability of programs in $cF^{\&}_{\leq}+$ and the relation between object-oriented languages and $cF^{\&}_{\leq}+$ more deeply.

Example 5.5.1.

Classes are subset of a set of all objects, and, because, in this framework, we place objects and terms in the same category, we can regard classes as types. Then a *message* is (an identifier of) a overloaded function whose branches are the methods associated to that message. The methods to be executed are selected according to the type (the class-name) passed as an argument.

We will provide an example, and, in this section, we assume that there is no type variable, i.e., the environment is ϕ (empty).

Let these types defined as follows using record types:

$$\begin{array}{lcl} \mathit{point1D} & = & [x\text{:Int}] \\ \mathit{point2D} & = & [x\text{:Int} \,;\, y\text{:Int}] \\ \mathit{point3D} & = & [x\text{:Int} \,;\, y\text{:Int} \,;\, z\text{:Int}] \\ \mathit{color} & = & [c\text{:Str}] \\ \mathit{(point1D+color)} & = & [x\text{:Int} \,;\, c\text{:Str}] \\ \mathit{(point2D+color)} & = & [x\text{:Int} \,;\, y\text{:Int} \,;\, c\text{:Str}] \\ \mathit{(point3D+color)} & = & [x\text{:Int} \,;\, y\text{:Int} \,;\, z\text{:Int} \,;\, c\text{:Str}] \end{array}$$

From these definitions, for example, $\vdash \mathit{point3D} \leq \mathit{point2D}$, and $\vdash \mathit{(point1D+color)} \leq \mathit{point1D}$ hold. We can define a message Norm working on these types

$$\begin{array}{l} \mathit{Norm} \quad \equiv \\ \quad (\Lambda \text{Mytype} \leq \mathit{point1D}.\lambda \mathit{self}^{\text{Mytype}}.\max\{(\mathit{self}|_{\mathit{point1D}}).x\} \\ \&(\Lambda \text{Mytype} \leq \mathit{point2D}.\lambda \mathit{self}^{\text{Mytype}}.\max\{\mathit{self}|_{\mathit{point2D}}).x, \quad (\mathit{self}|_{\mathit{point2D}}).y\} \quad . \\ \&(\Lambda \text{Mytype} \leq \mathit{point3D}.\lambda \mathit{self}^{\text{Mytype}}. \\ \qquad\qquad \max\{(\mathit{self}|_{\mathit{point3D}}).x, \quad (\mathit{self}|_{\mathit{point3D}}).y, \quad (\mathit{self}|_{\mathit{point3D}}).z\} \end{array}$$

The type of this message is

$$\begin{array}{rcl} S & = & \forall \text{Mytype}.\{\mathit{point1D}.[\text{Mytype} \rightarrow \text{Int}], \\ & & \quad \mathit{point2D}.[\text{Mytype} \rightarrow \text{Int}], \quad . \\ & & \quad \mathit{point3D}.[\text{Mytype} \rightarrow \text{Int}]\} \end{array}$$

We have used the variable *self* to denote the receiver of the message, and, following the notation of [Bru92], the type variable Mytype to denote the type of the receiver.

In the next example, we consider the type which includes type instantiation about overloaded type.

Example 5.5.2.

$$e = \Lambda X{\leq}(\forall Z.\{[l_1{:}\mathrm{Int}].Z \to \mathrm{Int}\}).\lambda x^X.\Lambda Y{\leq}[l_1{:}\mathrm{Int}].\lambda y^Y.x\{Y\}y$$

is a function which applies an ad-hoc polymorphic function to an argument and its type is derived as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{Y \in \mathrm{FV}(\Gamma)}{\Gamma \vdash Y \ \ \mathsf{type}}}{\boxed{\mathbf{I}} \quad \Gamma \vdash y^Y : Y} \qquad \Gamma^*(\mathrm{Inst}(X,Y)) \equiv Y \to \mathrm{Int} \qquad \dfrac{Y \in \mathrm{FV}(\Gamma)}{\Gamma \vdash Y \ \ \mathsf{type}}}{\Gamma \vdash Y \leq Y}}{\Gamma \vdash x^X\{Y\}y^Y : \mathrm{Range}(\mathrm{Inst}(X,Y))}}{\Gamma \vdash \lambda y^Y.x^X\{Y\}y : Y \to \mathrm{Range}(\mathrm{Inst}(X,Y))}}{\left\{\begin{array}{l} X{\leq}(\forall Z.\{[l_1{:}\mathrm{Int}].Z \to \mathrm{Int}\}) \vdash \Lambda Y{\leq}[l_1{:}\mathrm{Int}].\lambda y^Y.x^X\{Y\}y \\ \qquad\qquad : \forall Y{\leq}[l_1 : \mathrm{Int}].Y \to \mathrm{Range}(\mathrm{Inst}(X,Y)) \end{array}\right.}}{\left\{\begin{array}{l} X{\leq}(\forall Z.\{[l_1{:}\mathrm{Int}].Z \to \mathrm{Int}\}) \vdash \lambda x^X.\Lambda Y{\leq}[l_1{:}\mathrm{Int}].\lambda y^Y.x\{Y\}y \\ \qquad\qquad : X \to (\forall Y{\leq}[l_1 : \mathrm{Int}].Y \to \mathrm{Range}(\mathrm{Inst}(X,Y))) \end{array}\right.}}{\left\{\begin{array}{l} \vdash \Lambda X{\leq}(\forall Z.\{[l_1{:}\mathrm{Int}].Z \to \mathrm{Int}\}).\lambda x^X.\Lambda Y{\leq}[l_1{:}\mathrm{Int}].\lambda y^Y.x\{Y\}y \\ \quad : \forall X{\leq}(\forall Z.\{[l_1{:}\mathrm{Int}].Z \to \mathrm{Int}\}).X \to (\forall Y{\leq}[l_1 : \mathrm{Int}].Y \to \mathrm{Range}(\mathrm{Inst}(X,Y))) \end{array}\right.}$$

where **I** is

$$\dfrac{\dfrac{\dfrac{X \in \mathrm{FV}(\Gamma)}{\Gamma \vdash X \ \ \mathsf{type}}}{\Gamma \vdash x^X} \qquad \dfrac{\Gamma^*(X) \equiv (\forall Z.\{[l_1{:}\mathrm{Int}].Z \to \mathrm{Int}\}) \qquad \dfrac{(Y{\leq}[l_1{:}\mathrm{Int}]) \in \Gamma}{\Gamma \vdash Y{\leq}[l_1{:}\mathrm{Int}]}}{\Gamma \vdash \mathrm{Inst}(X,Y) \ \ \mathsf{type}}}{\Gamma \vdash x^X\{Y\} : \mathrm{Inst}(X,Y)}$$

In the proof tree above, we write

$$\Gamma \equiv X{\leq}(\forall Z.\{[l_1{:}\mathrm{Int}].Z \to \mathrm{Int}\}), Y \leq [l_1{:}\mathrm{Int}].$$

When e is applied to a type S and $Norm$ of the type S in Example 5.5.1, $f = e\{S\}Norm$ has the type $\forall \mathrm{Mytype}{\leq}[l_1{:}\mathrm{Int}].\mathrm{Mytype} \to \mathrm{Int}$ because, under an environment $\mathrm{Mytype} \leq [l_1{:}\mathrm{Int}]$, $\mathrm{Range}(\mathrm{Inst}(S, \mathrm{Mytype}))$ is reduced to $\mathrm{Range}(\mathrm{Mytype} \to \mathrm{Int})$ and moreover, to Int.

5.6 Conclusion

In this section, by extending $cF^+_{\leq}$, we proposed the calculus $cF^{\&}_{\leq+}$ integrating symbiosis of parametric polymorphism and a kind of ad-hoc polymorphism and satisfying the property of unicity of type[3]. We also proved that it satisfies the properties of Subject Reduction and Church-Rosser, and provided its applications to object-oriented languages. However, the facility of ad-hoc polymorphism in $cF^{\&}_{\leq+}$ is strictly restricted, and We will have to relax this restriction. The directions of this relaxation will be discussed in Chapter 6.

[3] We think that $cF^{\&}_{\leq+}$ is the first calculus that satisfies unicity of type and Subject reduction and that integrates the two kinds of polymorphisms.

Chapter 6

Conclusion

6.1 Conclusion

In this thesis, by modifying and extending $F_{\leq}$, [Ghe90], we first introduced new calculi $cF^{+}_{\leq}$ and $cF^{++}_{\leq}$, where, for each term expression, its type is uniquely determined and its information is lost only at the coercion operator. In Chapter 3, we introduced the calculus $cF^{+}_{\leq}$, which is a second order λ-calculus with records, coercion operator and bounded quantification including new types Range(X) and Sel(X,l). For a type Range(X), function types may be substituted for X and when a function type $S \rightarrow U$ is substituted for X, Range($S \rightarrow U$) is reduced to the type U. For a type Sel(X,l), record types which include field of label l will be substituted for X and when a record type $[\ldots, l{:}T, \ldots]$ is substituted for X, Sel($[\ldots, l{:}T, \ldots], l$) is reduced to T. We also proved that $cF^{+}_{\leq}$ has good properties (unicity of type, Subject Reduction, Church-Rosser, and Strong Normalization) and showed some applications of $cF^{+}_{\leq}$ to object-oriented languages.

In Chapter 4, we proposed the calculus $cF^{++}_{\leq}$ by adding the type Inst(X,U). For a type Inst(X,U), bounded universal types may be substituted for X and when a universal type $\forall Y{\leq}S.T$ is substituted for X, Inst($\forall Y{\leq}S.T,U$) is reduced to $T[X := U]$. We called this reduction the type instantiation and proved its termination. We also showed that $cF^{++}_{\leq}$ satisfies Subject Reduction and Church-Rosser.

By improving $F^{\&}_{\leq}$, we also provided a calculus $cF^{\&}_{\leq}+$ based on $cF^{+}_{\leq}$ which realizes symbiosis of parametric polymorphism and ad-hoc polymorphism in Chapter 5. It also satisfies good properties such as Subject Reduction and Church-Rosser. In addition, we proposed their applications to object-oriented languages.

6.2 Future Work

We think that the techniques described in this thesis can be applied to calculi integrating intersection types [Pie91, Pie93] and calculi including recursive types and subtyping [CCHMO89]. We also need think how we modify the rules of $cF^{+}_{\leq}$ so that some decidable type checking algorithms exist as in the case of $F_{\leq}$ [Pie92, KS92, CP94]. We also plan to study the interaction between parametricity and subtyping in the system $cF^{++}_{\leq}$ as in [PAC93], for example, by modelling fibration in category theory [Pho92].

The main future work of this thesis is to construct the general framework of calculi which integrate parametric and ad-hoc polymorphisms by extending the notions given in this thesis. There will be two ways to do it. The first way is to extend the notion of Kind introduced in

Chapter 4 such that every kind can include types as in [PT93a] to categorize the type of ad-hoc polymorphic functions. Because, if a calculus included the rule of coercion operator, not the subsumption rule, many terms could not type-check in it, most of calculi integrating parametric and ad-hoc polymorphism include the subsumption rule, not the coercion operator. Therefore they cannot satisfy the property of unicity of type. By using the technique in this thesis, we may be able to overcome the problem (see Section 1.3). The second way is to distinguish equations for ordinary expressions and equations for whole programs as in Plotkin's λv-calculus [Plo75]. Without this kind of distinction, we must restrict ad-hoc polymorphic functions to satisfy Subject Reduction and Church-Rosser property in the framework of second-order λ-calculi. So we must consider the evaluation context to relax the restrictions on ad-hoc polymorphism. We will try extending the $cF^{++}_{\leq}$ to integrate two kinds of polymorphisms, where ad-hoc polymorphic functions will be implemented by overloading [CGL92, Tsu94], merge operator [Tsu92, Tsu93, Tsu94a], or generalizations of extensible records [Car92].

Bibliography

[AM87] Appel, A. W., and D. B. MacQueen. A Standard ML compiler. In *Functional Programming Languages and Computer Architecture*, Lecture Notes in Computer Science 274, pp. 301–324, 1987.

[AM91] Appel, A. W., and D. B. MacQueen. Standard ML of New Jersey. In *Third Inter. Symp. on Prog. Lang. Implementation and Logic Programming*, Springer-Verlag, 1991.

[Bar84] Barendregt, H. P., *The Lambda Calculus — Its Syntax and Semantics*. North Holland, 1984. Revised edition.

[Blo90] Bloom, B., Can LCF be topped?, flat lattice models of typed λ-calculus. *Information and Computation*, Vol. 87, pp. 264-301, 1990.

[Bru92] Bruce, K. B., A paradigmatic object-oriented programming language: Design, static typing and semantics. Technical Report CS-92-01, Williams College, 1992.

[Bru93] Bruce, K. B., Safe type checking in a statically typed object-oriented language. In *Proc., ACM Conference on Principles of Programming Languages*, 1993.

[BDGKK88] Bobrow, D., L. DiMichel, R. Gabriel, S Keene, et al., Common lisp object system specification: X3j13 document 88-002r. *SIGPLAN Notices*, vol. 87, pp. 264–301.

[BTGS90] Breazu-Tannen, C., A Gunter, and A. Scedrov, Computing with coercions. In *Proceedings, ACM Conference on Lisp and Functional Programming*, pp. 44–59, 1990.

[BTGS91] Breazu-Tannen, C., A Gunter, and A. Scedrov, Inheritance as implicit coercion. *Information and Computation*, Vol 93, pp. 172–221, 1991.

[Car88] Cardelli, L., A semantics of multiple inheritance. *Information and Computation*, vol. 76, pp. 148–164, 1988.

[Car92] Cardelli, L., Extensible records in a pure calculus of subtyping, Research report 81, DEC Systems Research Center, 1992. Also in [GM93].

[Cas93] Castagna, G., $F^{\&}_{\leq}$: integrating parametric and "ad hoc" second order polymorphism, *Proc., the 4th International Workshop on Database Programming Languages*, 1993. Full version titled 'Integration of parametric and "ad hoc" second order polymorphism in a calculus with subtyping' is submitted.

[Cas93a] Castagna, G., *Surcharge, Sour-Typage et Liaison Tardive: Fondements Fonctionnels de la Programmation Orientée Objects* (in English). Thèse d'Informatique, Université Paris VII, 1993.

[CCHMO89] Canning, P., W. Cook, W. Vool, W. Hill, J. C. Mitchell, and W. Olthoff, F-bounded quantification for object-oriented programming, in *Functional Programming and Computer Architecture*, pp. 273–280, 1989.

[CG92] Curien, P. L., and G. Ghelli, Coherence of subsumption, minimum typing and the type checking in $F_{\leq}$, *Mathematical Structures in Computer Science*, vol. 2, No. 1, 1992.

[CG93] Curien, P. L., and G. Ghelli, Confluence and decidability of $\beta\eta\mathsf{top} \leq$ reduction in $F_{\leq}$. *Information and Computation*, To appear.

[CGL92] Castagna, G.,G. Ghelli, and G. Longo, A calculus for overloaded functions with subtyping,*Proc., ACM Conference on Lisp and Functional Programming*, pp. 182–192,1992. Full version will appear in *Information and Computation*.

[CGL93] Castagna, G.,G. Ghelli, and G. Longo, A semantics for $\lambda\&$*-early*: a calculus with overloading and early binding. In *International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science, 664, 1993.

[CMMS91] Cardelli, L., S. Martini, J. C. Mitchell, and A. Scedrov, An extension of system F with subtyping, in *Theoretical Aspects of Computer Software*, Lecture Notes on Computer Science, 526, 1991.

[CP94] Castagna, G., and B. Pierce, Decidable Bounded Quantification, To appear in *Proc., ACM Conference on Principles of Programming Languages*, 1994.

[CW85] Cardelli, L. and P. Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, Vol 17, No. 4, pp. 471–522, 1985.

[DG87] DeMichiel, L. G., and R. P. Gabriel. Common lisp object system overview. *Proc., European Conference on Object Oriented Programming*, 1987.

[Gal90] Gallier, J. H., On Girard's "Candidates de Reductibilité", in P. Odifreddi ed., *Logic and Computer Science*, Academic Press, pp. 123–204, 1990

[Ghe90] Ghelli, G., *Proof Theoretic Studies about a Minimal Type System Integrating Inclusion and Parametric Polymorphism*, PhD Thesis, Dipartimento di Informatica, Università di Pisa, 1990. Tech. Rep. TD-6/90.

[Ghe91] Ghelli, G., A static type system for message passing. In *Proc. of Object Oriented Programming, System, and Application '91*, 1991.

[Ghe93] Ghelli, G., Termination of system F-bounded, Manuscript, 1993.

[Gir72] Girard, J. Y., *Interprétation Fonctionnelle et Élimination des Coupures de L'arithmétique D'ordre Supérieur.* Thèse d'Etat, Université Paris VII (1972).

[GLT89] Girard, J. Y., Y. Lafont, and P. Taylor. *Proof and Types.* Cambridge University Press, 1989.

[GM93] Gunter, C. A., and J. C. Mitchell. *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design.* The MIT Press, 1933.

[Has90] Hasegawa, R., Categorical Data Types in Parametric Polymorphism. *Mathematical Structures in Computer Science.* To appear.

[Has91] Hasegawa, R., Parametricity of extensionally collapsed term models of polymorphism and their categorical properties, in *Theoretical Aspects of Computer Software.* Lecture Notes in Computer Science 526, pp.495–512. 1991.

[HW90] Hudak, P., and P. Wadler (ed.) Report on the Programming Language Haskell, 1990.

[KLM94] Katiyar, D., D. Luckham, and J. C. Mitchell, A type system for prototyping languages. In *Proc., ACM Conference on Principles of Programming Languages*, 1994.

[KS92] Katiyar, D., and S. Sankar, Completely bounded quantification is decidable. In *Proc., ACM Workshop on ML and its applications*, pp. 68–77, 1992.

[Mil77] Milner, R., Fully abstract models of typed λ-calculus. *Theoretical Computer Science*, Vol. 4, pp. 1–22, 1977.

[Mit90] Mitchell, J. C., Towards a typed foundation for method specialization and inheritance. In *Proc., ACM Conference on Principles of Programming Languages*, 1990.

[MR91] Ma, Q. M., and J. C. Reynolds, Types, Abstraction, and Parametric Polymorphism, Part 2, in *Proc. of the Mathematical Foundations of Programming Semantics Conference.* Lecture Notes in Computer Science, 1991.

[OB88] Ohori, A., and P. Buneman, Static type inference for parametric classes, *ACM conference on Object Oriented Programming Systems,Languages, and Applications*, pp. 445–455, 1989.

[OBBT89] Ohori, A., P. Buneman, and V. Breazu- Tannen, Database programming in Machiavelli — a polymorphic language with static type inference. *Proc. of ACM SIGMOD Conference*, pp. 46–67, Portlang, Oregon, 1989.

[Pho92] Phoa, W., Using fibration to understand subtypes. In *Applications of Categories in Computer Science*, vol. 177 of L.M.S., pp. 239–254, 1992.

[Pie91] Pierce, B., *Programming with Intersection Types and Bounded Polymorphism.* PhD. Thesis, Computer Science, CMU, 1991, Available as School of Computer Science tech. rep. CMU-CS-91-205.

[Pie92] Pierce, B., Bounded quantification is undecidable, In *Proc., ACM Conference on Principles of Programming Languages*, 1992.

[Pie93] Pierce, B., Intersection Types and Bounded Polymorphism, In *International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science, 664, 1993.

[Plo75] Plotkin, G., Call-by-name, call-by-value, and the λ-calculus. *Theoretical Computer Science*, vol. 1., pp. 125–157, 1975.

[Plo77] Plotkin, G., LCF considered as a programming language. *Theoretical Computer Science*, vol. 5., pp. 223-255, 1977.

[PAC93] Plotkin, G., M. Abadi, and L. Cardelli, Subtyping and Parametricity. 1993.

[PT93] Pierce, B., and D. N. Turner, Simple type-theoretic foundations for object-oriented programming. *Journal of Functional Programming*. To appear.

[PT93a] Pierce, B., and D. N. Turner, Statically Typed Friendly Functions via Partially Abstract Types. *University of Edinburgh technical report* ECS-LFCS-93-256 and *INRIA-Rocquencourt Rapport de Recherche* No. 1899., 1993.

[Rey74] Reynolds, J. C., Towards a theory of type structures. Lecture Notes in Computer Science, 19, pp. 408–425, 1974.

[Rey83] Reynolds, J. C., Types, abstraction, and parametric polymorphism., In *Information Processing 83*, pp. 513–523, 1983.

[Str67] Strachey, C., Fundamental concepts in programming languages, Lecture notes for International Summer School in Computer Programming, Copenhagen, 1967.

[Str88] Stroustrup, B., *The* C++ *Programming Language*, Addison-Wesley, Reading, Mass., 1988.

[Tai75] Tait, W. W., A realizability interpretation of the theory of species, In *Logic Colloquium*, Lecture Notes in Computer Science No. 453, pp. 240–251, 1975.

[Tak89] Takahashi, M., Parallel reductions in λ-calculus. *Journal of Symbolic Computation*, Vol. 7, pp. 113-123, 1989.

[Tak92] Takahashi, M. H., Parallel reductions in λ-calculus. Tech. Rept. C-103, Tokyo Institute of Technology (Revised version of [Tak89]).

[Tsu92] Tsuiki, H., *A Record Calculus with a Merge Operator.* PhD Thesis, Keio University, 1992.

[Tsu93] Tsuiki, H., A record calculus with a merge operator (in Japanese). *Journal of Information Processing Society*, vol. 34, No. 5, pp. 954–962, 1993.

[Tsu94] Tsuiki, H. A normalizing calculus with subtyping and overloading. To appear in *Theoretical Aspects of Computer Software 94*, 1994.

[Tsu94a] Tsuiki, H. A calculus of parametric generic functions. Manuscript, 1994.

[WB89] Wadler, P., and S. Blott. How to make "ad-hoc" polymorphism less "ad-hoc". In *Proc., ACM Conference on Principles of Programming Languages*,pp. 60–76, 1989.

[Wan89] Wand, M., Type inference for record concatenation and multiple inheritance, *Proc, Fourth IEEE Annal Symposium on Logic in Computer Science*, pp. 92–97, 1989.