Ph.D. Thesis

A Theory of Dependent Record Types with Structural Subtyping

Thesis Submitted for the Degree of Doctor of Philosophy

Author       Yangyue Feng
Supervisor   Prof. Zhaohui Luo

Department of Computer Science
Royal Holloway, University of London
January 2011
I declare that this thesis was composed by myself, and the work contained herein is my own, except explicitly stated otherwise.
Part of the result is accepted to appear in Types09 post-proceeding.

Signature :

Date :
Abstract

Type theory stemmed from the philosophical discussion on foundation of logic and set theory, and aroused interest from researchers in logic and formalism who gave it a clear format based on the $\lambda$-calculi. With the growth of the discipline of computing, various forms of type theory have brought in a common interest in developing computer-aided proof assistants. From this multi-disciplinary nature and development of type theory, it has clearly become a pluralistic subject of study that contains elements from philosophy, logic, foundation of mathematics, computing science, and even linguistics, etc.

In this thesis, I present and study a type system of dependent record types. Dependent record types (DRTs) are types of records in which the type of a field may depend on the value of an earlier field. In this thesis, I prove several metatheoretic properties, such as structural properties, subject reduction, strong normalisation and Church-Rosser property of DRTs as defined with the logical framework LF, using the method of typed operational semantics first developed by Goguen.

I also study how structural subtyping may be applied to DRTs, by extending the type theory LF and DRTs with coercive subtyping, and prove that the resulting set of coercions is coherent. This work makes dependent record types a possible theoretical basis for modelling module mechanisms in computer languages.
Acknowledgements

There is a list of people and institutions that I would like to say “thank you!”:

I would like to thank firstly my supervisor Prof. Zhaohui Luo, without whom this study won’t be initiated: Zhaohui has led me into the door of study on type theory, during four years of study I have swum in this nice field and been able to compose this work, as well as received invaluable encouragement and guidance in research from him. I thank also Dr. Seokhyun Han and Dr. Robin Adams for their help on my questions and valuable discussions.

I’d like to thank the Department of Computer Science of Royal Holloway, University of London, who offered me a place to study for four years. Mrs Janet Hales and Ms Jenna Sparkes helped me with multiple daily administrative affairs, the technical support team settled many hardware issues for me. Prof. Alex Gammerman, Dr. Chris Watkins, Dr. Hugh Shanahan and Dr. Alberto Paccanaro gave me many good advice on how to finish the PhD. The amicable environment from the members of staff, from the good friends among the PhD students, and from the PhD and non-PhD friends from the college, have all contributed to my life in UK.

I would like to acknowledge the financial support from Leverhulme Trust, Royal Holloway College Research and the Computer Science Departmental Award, which made my study in UK possible. I thank Dr. Thorsten Altenkirch and Dr. Steffen van Bakel who have confirmed to be the Examiners and come to my Viva, and for their examination of 4 hours on this thesis and the ideas of correction afterwards.

I would like to thank in extra Dr. Alexey Koloydenko for being a kind friend, Sue and Kevin Rundell for offering me a room in Englefield Green during the time for Viva, and Dr. Zhiyuan Luo, Dr. Haixuan Yang for all their help.

Most importantly, I will thank the full love and support from my family, without which this thesis wouldn’t be possible to complete. Thanks also to my nice friends Yvonne Buttkewitz and Michael Harvey for their kind surrounding, and thanks to the meet with Katharina who travelled a long way to my home town Yang Zhou—life’s encountering is always a miracle.
Contents

1 Introduction 1
   1.1 Dependent Type Theory ........................................ 2
   1.1.1 A Description of Type Theory .................................. 3
   1.1.2 Type Theory and its Applications ............................. 8
   1.2 Records and Subtyping ........................................... 13
   1.3 Overview of Thesis ............................................... 17

2 Dependent Record Types 19
   2.1 Introduction to Dependent Record Types ....................... 20
   2.2 The Logical Framework LF ...................................... 22
      2.2.1 Computation and Metatheory .............................. 25
   2.3 The System of IDRT ............................................ 30
      2.3.1 The Extension of LF with DRTs .......................... 30
      2.3.2 Inference Rules of DRTs ................................ 33

3 A Typed Operational Semantics for DRTs 36
   3.1 Basic Introduction ............................................. 37
   3.2 A TOS for Logical Framework ................................. 40
   3.3 A TOS for Dependent Record Types ........................... 42
   3.4 Idea of Using the TOS to Prove Metatheory .................. 44
List of Figures

2.1 Syntactical Components of Logical Framework . . . . . . . . . . . . . 22
2.2 Inference Rules of Logical Framework (LF) . . . . . . . . . . . . . . 24
2.3 Syntactical Components of the System IDRT . . . . . . . . . . . . . 30
2.4 Syntactical Components of Dependent Record Types . . . . . . . . . 31
2.5 Inference Rules of Intensional Dependent Record Types (IDRT) . . 34

3.1 Judgement Forms in Typed Operational Semantics . . . . . . . . . . 39
3.2 Inference Rules of Typed Operational Semantics for LF . . . . . . . 40
3.3 Inference Rules of Typed Operational Semantics for IDRT . . . . . 43

4.1 Compatible Closure $\triangleleft R$ of a Relation $R$ w.r.t. IDRT . . . . 49
4.2 Inf. Rules of Dependent Record Types (indexed with Roman numerals) 52
4.3 Inference Rules of Logical Framework (indexed with Roman numerals) 53
4.4 Parallel Reduction for IDRT . . . . . . . . . . . . . . . . . . . . . . . . . . . 66

5.1 Component-wise Coercion Rules for DRTs . . . . . . . . . . . . . . 78

6.1 TOS Canonical Forms and Weak-Head Reduction for Natural Numbers 95
6.2 TOS Canonical Forms and Weak-Head Reduction for Strong Sum Type 96
Chapter 1

Introduction
1.1 Dependent Type Theory

Organisation of the Section

1.1.1 \(\lambda\)-calculus

Typed \(\lambda\)-calculus

(What is type theory?)

Dependent types

1.1.2 Type theory and its applications

Set Theory
Design Languages
Metamathematics

Constructive Logic
Theorem Proving
1.1.1 A Description of Type Theory

What is a “type theory”, or say, a theory of types? Very roughly speaking, types are collections of special attributed data. This could be understood in two ways: first, types could be seen as representation for a certain genre of objects, they actually contain more general and summarizing information than this object itself, and they collect certain objects of the same specific attributes, two unidentical objects could be of the same type; the second view is, a type for a term is actually an abstract specification of it, and the term, sometimes called an instance of this type, is seen as the implementation of this type, or seen as a proof of this type if we regard the latter as a formula, in this meaning, the term might contain more detailed information than its type.

All the systems being studied in type theory are $\lambda$-calculus-like systems, which means that they are systems containing syntactical terms such as variables, $\lambda$-abstractions (called functions as well) and function applications. No other mathematical constructions such as partial differential equations or open sets are involved.

Some important type theories include PTS (Pure Type Systems) [Bar91], Martin-Löf’s Type Theory [NPS90], ECC (Extended Calculus of Constrictions) [Luo94].

A First Example: What does a type system consist of? An example of a type theory is the Simply Typed $\lambda$-Calculus, which is the most basic type system and has a lot of good computational property such as decidable type-checking, strongly normalizing, etc. The syntax of this system is based on the formal system $\lambda$-Calculus:

$\lambda$-Terms are defined as a set of variables, applications and abstractions:

$$M :: = \ x \mid MN \mid \lambda x. M$$

Computation in $\lambda$-Calculus The computation in $\lambda$-Calculus includes two
forms: the $\beta$ and $\eta$ (if extensionality is allowed) reductions.

\[(\lambda x.f) \ a \rightarrow_\beta f[a/x]\]

\[\lambda x.f \ x \rightarrow_\eta f \ (x \not \text{ free in } f)\]

where $f[a/x]$ is the capture-avoiding substitution of term $a$ for the free occurrences of variable $x$ in $f$, which is defined as usual will all possible changes of bound variables.

**Types** are defined as a set of primitive types $\iota$ \(^1\) and function types (or called arrow types):

\[\tau ::= \iota \mid \tau \rightarrow \tau'\]

**Typed $\lambda$-Calculus** The typed $\lambda$-terms are defined as a set of typed variables, typed application and (functionally) typed abstractions, with the symbol \\
\[\vdash\] to notify a term of its type. Informally, They are typed variables $x : \sigma$ ($x$ of type $\sigma$), application $MN : \tau$ if $M : \sigma \rightarrow \tau$ and $N : \sigma$, abstraction $\lambda x : \sigma.P : \sigma \rightarrow \tau$ if $P : \tau$.

**Inference Rules of Typed $\lambda$-Calculus** The three syntactical conditions above could be formally described with a set of *inference rules* (or typing rules), where $\Gamma$ is a typing context which is a set of typing assumptions. A typing assumption has the form $x : \sigma$, meaning $x$ has type $\sigma$.

\[
\begin{align*}
\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma} & \quad \frac{\Gamma, x : \sigma \vdash P : \tau}{\Gamma \vdash \lambda x : \sigma.P : \sigma \rightarrow \tau} & \quad \frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}
\end{align*}
\]

In type theory, one of the main concerns is on the relation between terms and types, and such problems are respectively:

- type checking problems (TC): “Is a certain type assertion $M:\sigma$ correct ?”
- type assignment problems (TA): “How to find a type for a given term $M$ ?”
- type inhabitation problems (TI): “Given a type $\sigma$, how to find a term of this

\(^1\)Here in the brief introduction we use the Greek letters $\iota$ to represent primitive types, in our later work regarding the record types they will be specified; and we will use letters $A, B$ for arbitrary types instead of $\sigma, \tau$ in the later texts.
Dependent Type Theory

1.1. Dependent Type Theory

For the simply typed $\lambda$-calculus, all of these problems are decidable, which means that there exist terminating algorithms to give a solution to these problems if the solution exists, otherwise return a failure if the solution does not exist. And the TI problem corresponds to the provability in some logics and is often undecidable for more sophisticated systems.

Another important concern in type theory is the metatheoretic properties of it, or called the study of the metatheories. Such properties are the Church-Rosser theorem, type uniqueness, strong normalisation, subject reduction, etc. This aspect of a type system will be one of our main foci in this thesis.

Dependent Types

The typed lambda calculus with only two simple types: the primitive types and the function types (or called arrow types); now in this section, we will introduce some more complicated types, which are generally called the dependent types. The arrow type is one among them, which is known as the dependent product type $\Pi$, and we will add now another dependent type that is called the dependent sum type $\Sigma$ (which could be also written as $\times$, in correspondence with the $\to$), with the definition of the set of types as follows:

Definition 1.1.1 (Dependent Product and Sum Types) Types are now defined as a set of primitive types, dependent product type (or called $\Pi$-type) and dependent sum type (or called $\Sigma$-type):

$$\tau ::= \iota \mid \Pi x:M.N \mid \Sigma x:M.N$$

And accordingly the set of terms defined now as:

Definition 1.1.2 ($\lambda$-Calculus with Dependent $\Pi$ and $\Sigma$ Types) The $\lambda$-terms are now defined as a set of variables, applications, abstractions, dependent sum, first

---

2The TC and TA problems are sometimes equivalent since to solve $MN:\sigma$ one has to solve $N:?:$. 
projection and second projection:

\[ M :: = x | MN | \lambda x: \sigma. M | (M, N)_\sigma | \pi_1(M) | \pi_2(M) \]

Another example of dependent types is the type of vectors:

The \( n \)-tuple of Reals :: = Vec(R, n)

in which the type \( \text{Vec}(R, n) \) depends on the value of \( n : N \), where \( N \) is the type of natural numbers.

In dependent type theory, type-checking is more complicated because of the dependency. Computations are involved to decide equality of dependent types in a term. If arbitrary values are allowed in dependent types, then deciding type equality may involve deciding whether two arbitrary programs produce the same result, that’s why type-checking becomes undecidable in general.

H. Barendregt has developed the lambda cube \(^3\) in [Bar91] to classify the type systems from three axes: the dependency, the order, and the logical connectives. In this cube, eight formal systems have been synthesised. It forms a fine structure to describe these eight systems hierarchically enhanced in their power along three dimensions, including from the simple typed \( \lambda \)-calculus as the simplest case to Coquand’s calculus of constructions [Coq88] as the most complicated case.

The first and lowest point is the simple typed lambda calculus (STT), as the original vertex, stemming from which three ways to extend STT reach out, finally all to Coquand’s calculus of constructions (CIC) which is the higher order dependently-typed polymorphic \( \lambda \)-calculus, as its diametric opposite vertex. The three dimensions are marked with index:

- “P” : the dependency, i.e. predicate logic;
- “\( \omega \)” : the type variable or polymorphism, i.e. the logical connectives;

\(^3\)The “lambda” from the Greek letter \( \lambda \).
1.1. Dependent Type Theory

• “2” : the higher order, i.e. quantification over types (sets).

On the cube there are six faces. At the bottom face we have all propositional logic, then to the right-hand side face we have predicates, on the back face we have type variables, and only at the top face we can quantify over types and have higher order logic.

And there is a way to describe different dependency simply by an \((s_1, s_2)\) rule where \(s_i\) could be either \(*\) or \(\circ\) which denotes level of function abstraction, we have, symmetrically:

• \((*, *)\) : systems where term depends on term;
• \((*, \circ)\) : systems where type depends on term;
• \((\circ, \circ)\) : systems where type depends on type;
• \((\circ, *)\) : systems where term depends on type.

Examples in the cube are such:

- Simply typed lambda calculus \((\lambda \rightarrow)\) \(\alpha : * \vdash (\alpha \rightarrow \alpha) : *\)

- System F \((\lambda 2)\) \(\alpha : * \vdash (\lambda \alpha : \alpha. \alpha) : \alpha \rightarrow \alpha\)

- System \(F_\omega\) \((\lambda \omega)\) \(\vdash (\lambda \alpha : \ast. \alpha \rightarrow \alpha) : (\ast \rightarrow \ast) : \circ\)

- Edinburgh logical framework LF \((\lambda \mathcal{P})\) \(\alpha : * \vdash (\alpha \rightarrow *) : \circ\)

- Second order dependent type theory \((\lambda \mathcal{P}2)\) Obtained from \(\lambda \mathcal{P}\) by allowing quantification over type constructors.

- Higher order dependently typed polymorphic lambda calculus \((\lambda \mathcal{P}_\omega)\)
  The Calculus of Constructions, obtained from \(\lambda \mathcal{P}2\) by allowing type variables and thus allowing polymorphism.
1.1. Dependent Type Theory

1.1.2 Type Theory and its Applications

We here make an analogue between types and several other scientific disciplines such as mathematics and computer science, after we have firstly discussed briefly some technical aspects of type theories.

Types and Sets

Types and sets are both studied in the early part of the 1900’s (c.f. work by Russell and Whitehead [WR1910,12,13]), and formal set theory appeared not much earlier than type theory. The central idea of the study of both, however, was trying to find a powerful enough basis for describing abstract objects in mathematics, and reasoning on these objects could be carried on. Once is a “full” formalization for this underlying basis built, the well-foundness of mathematics and the proof correctness could be trusted, with a certain reliability.

In early 20th century Russell first discovered that sets have their fault and that naive set theory was paradoxical, which has led to the study of formal set theory, e.g. such as Zermelo-Fränkel set theory with the axiom of choice to come forth. Along with the development of computer science, from a more modern view, naive sets are not suitable for characterizing computation with programming.

Types have been studied from the initial motivation to improve this problematic paradoxical structure of sets, and were found to be much closer with the notion of computation, especially with the so-called Curry-Howard isomorphism being put forward.

The significance of types compared to sets is thus the following:

- Type theory aims at capturing both the notion of logic (via the formula-as-types embedding) and the notion of computation (via the inclusion of functional programs written in typed lambda-calculus). In some sense, it offers a more organized information description mechanism, for example, the introduction of hierarchical universe structures. Meanwhile, logical reasoning and
data types are “built-in”, and could be thus more easily applied to computing.

• Almost all advanced mathematics are built on top of the notion of sets, types could, very similarly, as a counterpart of sets, be another foundation for building mathematics. This is reflected by the research on formalizing mathematics with type theories and computer-aided proofs.

• There have been important results in using automated proof checkers in both academic and industrial research, coming out with logic and types, such as type checking, protocol verification, computer-aided proof check, etc. The study relates closely to type theory, especially type theory would incorporate important concepts such as induction and recursion.

Types and Constructive Logic

Constructive type theory based on Martin-Löf’s contribution [Mar84] is intrinsically liased with constructive logics. What connects type systems to logical justification and verification was the Curry-Howard isomorphism (or called the formulæ-as-types embedding or propositions-as-types embedding), observed and developed by the mathematician H. Curry and the logician W. Howard in the late 1950’s until 1970’s. This isomorphism (or preferably to be called a conceptual correspondence by some researchers, in order to separate with the real meaning of “isomorphism” in mathematics) is a one-to-one correspondence between logical systems and type systems

\(^4\) in the sense that:

“Propositions are mapped to Types”

“Proofs are mapped to Terms”

This one-to-one correspondence between propositions and types means that for every proposition in the logic, whenever it is provable, i.e., there exists a proof of it, then for the corresponding type, there exists a term to inhabit the type.

This provides actually two readings of “\(M : \sigma\)”: first, from programming point of view, we can see types as a specification and the terms as programs or algorithms

\(^4\)In the original paper of Howard [How80], the correspondence was made between logical systems and lambda calculi.
for it; thus $M$ is a function of the type $\sigma$. Second, from logical point of view, a type could be regarded as a proposition, and a term as its proof; thus $M$ is a proof of proposition $\sigma$.

Similarly, the consistency of a logical system is the same as in a type system. A logical system is consistent if and only if $A \land \neg A$ is not provable, which is equivalent to the existence of an unprovable formula. Correspondingly, a type system is consistent when there exists an uninhabited type. For example, the $\forall \alpha: \ast. \alpha$ is the uninhabited type in System F (the second order $\lambda$-calculus, which is consistent).

**Types and Language Design**

Remarkably, type theory could be regarded as an abstract theory or a foundational study for computer programming languages. Although a real computer programming system looks far more complicated, many constructions of real programming languages have their prototype in type theory; and many ideas in high-level programming languages and compiler designs could be sought for support from type theory. An example of a short java-like program segment that uses dependent array type is as follows:

```java
class InpUtArrayC {
    ... 
    int extent; // variable not evaluated 
    int[] myArray; 
    ... // read in extent from BufferedReader 
    myArray = new int[extent]; // dependent type 
    ... 
}
```

We can list some applications of type theory with regards to language design such as:

- Module mechanisms are important component for modern computer programming. From type theory, constructions such as $\Sigma$-types and record types could be used to model module mechanisms and formalize Object-Oriented programming.
1.1. Dependent Type Theory

- Type theory develops along with functional programming languages, such as ML in the late 1970s; also in the 1980s, with motivation in combining proof theory (developed by Gentzen, Gödel and Herbrand in the 1930s) and computing, a proof system extending the ideas of the automatic proof system LCF has been designed by adopting the ML as an implementation language. This research led to the Caml language family and Ocaml which is the implementation language for the Coq proof assistant [BC04]. This will be more elaborated in the next paragraph.

Type-theoretic Tools for Interactive Theorem-proving

Interactive theorem proving is the field of computer science and mathematical logic combined with tools to develop formal proofs by man-machine collaboration. The original goal was in the attempt of automating the construction of mathematical proofs. The result is the so-called proof assistant: an interactive proof editor, with which the user can guide the automatic search for proofs. Proof assistants could be used also to specify and verify computer programs developed with other modern programming languages.

There exist quite a few well-developed proof assistants, namely, Coq, Lego, Plastic, Agda, Nuprl, HOL, Isabelle, Epigram, Mizar, Isar, PVS, ACL2, Matita, etc. Some of these systems work with proof scripts that are a list of tactics which are needed to verify the validity of statements.

These proof assistants are based from different logical systems. Coq and Lego are based on the Calculus of Constructions extended with inductive types. They both allow impredicative statements as used in actual mathematics, but as argued by Poincaré, Weyl and Martin-Löf potentially unreliable [Poi1906, Wey1918, Wey87, Mar84]. Agda is similar to Coq and Lego except that it is based on Martin-Löf type theory in which impredicative quantifications are not allowed.

Nuprl is based on an extensional type system, which means type checking is no longer decidable [Hof95]. HOL is based on Church’s simple type theory (being based on classical logic), and is a classical system of higher order logic. Isabelle is based on intuitionistic simple type theory, but it uses this logic as a meta-logic, and many

5All these systems and tools have their online websites where references could be sought at.
other logics such as first-order predicate logic, systems in lambda cube and higher order logic are described *internally*.

Mizar is based on Tarski-Grothendieck set theory, which is ZFC extended with the axiom of existence of arbitrary large cardinals. A nice feature of Mizar is that the proof-scripts are close to an ordinary proof in mathematics (which are internally represented as proofs in set theory) and a rich result library exists. ACL2 is based on classical primitive recursive arithmetic, it does not support inductive types but supports LISP-style recursive function executing in the system. PVS is based on classical simple type theory and it allows all kind of rewriting for numeric and symbolic equalities. These three systems are more user-friendly and they are more oriented to “mathematical” use.

**Type Theory as Foundation of Mathematics**

Type theories could be a metalanguage to formalize mathematics. A formalization of a mathematical system with a type theory is that we treat this mathematical system (that we called the target system or the target language) as the object of the type theory (which we call the metalanguage or the metamathematics), and interpret the elements of the target mathematical system into the type theoretic terms. A formal proof, is the proof constructed with some interactive theorem-prover supported by some type theory, i.e. in the metalanguage; and this formal proof talks about some property (or some theorems) in the original target mathematics, and could be automatically checked with the theorem-prover. There are many interesting related issues and questions in the formalization of a theory: first, we might ask, to what extend could mathematics in general be formalized with type theories or type-theoretic tools. Second, we might be interested in the correctness and how exactly the formalization could help in theorem-proving.

For example, the Four-Colour Theorem has been formalized with the proof assistant Coq [BC04] and proven [Gon05]. And Weyl’s predicative mathematics has been formalized with the proof assistant Plastic [Cal06] in [AL07].
## 1.2 Records and Subtyping

In this section we introduce two things that are studied in this thesis: the dependent record types and subtyping for record types.

What are dependent record types?

\[
\langle l_1 : A_1, \ l_2 : A_2, \ ... \ l_n : A_n \rangle
\]

What is subtyping for them?

\[
R_1 \leq R_2 \ \text{?} \ \ R_1 < R_2 \ ?
\]

**Record Types** Record types are collections of labelled types, which could be presented as:

\[
Record\ types \quad R := \langle l_1 : A_1, \ l_2 : A_2, \ ... \ l_n : A_n \rangle
\]

where \( l_i \)'s are labels and \( A_i \)'s are types.

Dependent record types (DRTs for short) are record types with dependency, i.e., they are types of records in which the type of a field may depend on the value of an earlier field. In a formal way, we could write as:

\[
Dependent\ record\ types \quad R := \langle l_1 : A_1, \ l_2 : A_2, \ ... \ l_n : A_n \rangle
\]

where \( A_j \) may depend on \( A_i \) (\( 0 \leq i < j \leq n \)).
Dependent record kinds (which are called dependent record types in a system in which kinds of a logical framework are called types) have been studied in several previous work [BT98, CPT05], and dependent record types in [HL94], aiming at providing formal treatment to module systems such as in functional programming and in proof languages. We will discuss further the difference of dependent record kinds and dependent record types in Section 2.3.1 (to Page 32).

MacQueen has first suggested in [Mac86] a module system using dependent Σ-types as module signatures, which combines the first ideas coming from ML on data abstraction and from Pebble on dependent Sigma types. Later, Harper and Lillibridge have proposed in [HL94] a representation of records in order to treat the ML-style sharing by equation, or called sharing by coherence conditions; but in their formulation the bounded quantification used is proven to be undecidable, mainly caused by the FORGET-rule which loses type information, the proof is close to Pierce’s proof for undecidability of $F_{\leq}$ [Pie94].

Betarte and Tasistro have proposed in [BT98] a system with record kinds as an extension of Martin-Löf’s Logical Framework, with type inclusion to represent subtyping. These systems all studied record kinds, however, record kinds are not as expressive to capture module mechanisms as a system of record types [Pol02, Luo09].

In our current work, an improved formulation of dependent record types has been put forward, in the context of using so called intensional manifest fields (to Page 82) in the dependent record types [Luo08]; and this formulation has been explored with several case studies in modelling OO programs.

Metatheorems and the approaches to prove them are of a great research interest, since they study how formal systems internally behave. Thus, whether or not this formulation has good metatheoretic properties is of our interest, as we would like to know if our current formulation of dependent record types is correct, concerning with decidability, and also with other aspects such as subject reduction, Church-Rosser property and so on. If these properties hold, our current formulation of dependent record types serves well as foundation for modelling module mechanisms, and pro-
vides a good basis for implementation.

The main effort of this thesis is to show the metatheorems of this novel formalism of dependent record types, through a pure proof-theoretic method called the typed operational semantics developed by Goguen in his Ph.D thesis [Gog94] and for Logical Framework [Gog99].

**Subtyping in Type Theories** One of the important differences between type theory and set theory is that in the former we do not have a notion of subtype that corresponds to the notion of subset in the latter. This limitation has urged people to look for the right notion of subtypes, i.e., to develop a mechanism that can express $A \leq B$ where $A$ and $B$ are two types. We would like to work on such a mechanism which will first match to the intrinsic concepts and structures of type theory, and secondly should be convenient and suitable for type-theoretic applications, such as modelling of programming languages with inheritance.

Let us first look onto two notions of subtypes in type theories: these are related to two different views on types and their objects [Luo10]. The first one is called subsumptive subtyping, which is the “ordinary” notion of subtyping, with the subsumption rule as follows, for type assignment systems such as the polymorphic calculi in programming languages:

$$
\frac{a : A \quad A \leq B}{a : B}
$$

Types are seen as an assignment to objects. Objects are considered to exist first, and they constitute a pre-given universe, with the types assigned to them. The subtyping relation is obtained by overloading object terms.

Under this viewpoint, objects are independent of their types, and the types are then assigned to them. Polymorphism may exist by overloading $\lambda$-terms, which may reside in different types. Examples are such as ML-like programming languages.
For example, the typing relation $\lambda x.x : \alpha \to \alpha$ where $\alpha$ can be $Nat$, $Int \to Int$, $Int \to (Int \to Int)$, ..., etc.

However, this notion will not suit for type theories with canonical objects, the reason being the incompatibility of subsumption with canonical objects, since canonicity is lost in subsumptive subtyping [AM06].

The limitation of subsumptive notion of subtyping has turned us to the second notion of subtyping, which is named as the coercive subtyping, for the type theories with canonical objects such as Martin-Löf’s type theory.

The coercive subtyping view, i.e. the “canonical object view” is where types are seen as collections of their canonical objects. Types and their objects are considered to co-exist, but one does not exist without the other.

Under this viewpoint, types are given inductively by introduction - elimination - computation rules, and there is the embedded consistent logic and unique typing. Examples are such as Martin-Löf’s type theory and UTT (Unifying Theory of dependent Types) [Luo94]. Subtyping could be regarded as a mechanism of abbreviations.

For example, 0 and $succ(n)$ are canonical natural numbers and $1 + 2 \to 3$ (i.e., $succ(succ(succ(0)))$, where the uncanonical object 3 is seen as the abbreviation for the canonical term under the coercive “enforcement”, or the coercive mapping).

We shall in this work study how structural subtyping of DRTs can be considered in the framework of coercive subtyping [Luo97, Luo99]. And we will discuss and study the metatheoretic issues over this framework in the later Chapter 5.
1.3 Overview of Thesis

The main trunk of this thesis is organised as follows:

Chapter 2
IDRT

Chapter 3
TOS

Chapter 4
Metatheory of TOS

Section 4.6
Metatheoretic properties of IDRT

Chapter 5
An application of metatheory: structural subtyping

Chapter 6
Conclusions & future work
In Chapter 2, we will give the syntax and inference rules of intensional dependent record types (IDRT), as an extension of the logical framework LF;

In Chapter 3, we develop a typed operational semantics (TOS) for this system, with brief introduction on how to prove metatheories with this semantics;

In Chapter 4, there are our main results, i.e. the metatheories of typed operational semantics, including structural properties, adequacy with respect to untyped reduction, completeness and soundness, subject reduction and strong normalisation;

And especially in Section 4.6, we cluster those properties for the system of IDRT, in order to be used by some results from the next Chapter;

In Chapter 5, we study coercive subtyping for dependent record types, with the result proven that a set of coercions is coherent, and we present a way of structural subtyping for IDRT with intensional manifest fields;

In Chapter 6, we conclude our work and summarize further study on the topic.
Chapter 2

Dependent Record Types
2.1 Introduction to Dependent Record Types

A dependent record type is a type of labelled tuples whose fields may have types that depend on the values of earlier fields. For instance, if \( \text{Nat} \) and \( \text{Vect}(n) \) are the types of natural numbers and vectors of length \( n \), respectively, \( \langle n : \text{Nat}, v : \text{Vect}(n) \rangle \) is the dependent record type with objects (called records) such as \( \langle n = 2, v = [5, 6] \rangle \), where dependency is respected: the vector \([5, 6]\) must be of type \( \text{Vect}(2) \).

Formally, dependent record type (short as DRT) is formulated as an extension of an intensional type theory, such as Martin-Löf’s type theory or UTT. Basically, a record type is a type which is a composition of data items (called fields), like the \( \Sigma \)-types, but with labels (that belong to a new syntactic category \( \mathcal{L} \)) associated with those fields. And dependent record types are record types whose component types could depend on the values of previous fields. The formal representations of a record type and a record are given as follows:

\[
R :: = \langle \rangle | \langle R, l : A \rangle \\
r :: = \langle \rangle | \langle r, l = a : A \rangle
\]

With the symbol \( \langle \rangle \) overloaded for both empty record types and empty record terms. There are two operations associated with record terms: restriction (or first projection) \([r]\) that removes the last component of record \( r \), and field selection \( r.l \) that selects the field labelled \( l \).

The reason why DRTs are of interest of studying is that it can be used to provide type-theoretic models for mechanisms in modern programming languages, such as module mechanisms, and Object-Oriented programming paradigms. What’s more, because the DRT is purely an extension of an intensional type theory, some constructions in those mechanisms, such as sharing constructions, could be modelled within the intensional type theory, which has not been able to achieve before without using extensional components [Luo08, Ler94]. As in extensional type theories (ETTs)
type-checking and well-formedness of propositions are undecidable [Hof95]; thus, this methodology to model things with only pure intensional type constructions is of great interest.

**Modelling Object-Oriented Program with DRT** With the idea “Classes as Types”, we can use dependent record types to model OO classes and interfaces. The way how a class could be modelled as a type is that with a parameterised unit type “(A, a)” and a *coercion* function (ξ) defined as “(A, a) → a”. A class, represented by the unit type (to Page 78), could be mapped through the coercion into one of its instance object, represented by this element a. And similarly, a method M in the class could also be represented by certain unit types, and be mapped into the runtime correct parameterisation on an instance with a. In this way, the important OO features such as subtype polymorphism and dynamic dispatch, are correctly captured in the type system.

**Formulation of DRT as an Extension of LF** The inference rules of record types and records, with a new category of labels L defined, are given in Section 2.3. Because we combine the DRTs together with a logical framework LF as its extension, we first give an introductory section to LF.
2.2 The Logical Framework LF

The Logical Framework LF [Luo94] is the typed version of Martin-Löf’s logical framework [NPS90]. It is itself a type system that serves as a meta-language to specify type theories, which are called the object type theories, in contrast with the meta-system. The type theories being specified with LF are such as Martin-Löf’s intensional type theory [NPS90], the Unifying Theory of dependent Types (UTT) [Luo94], and the logic-enriched type theories (LTT) [AL07, Luo07, AL09]. Here, we give only a brief introduction to fix the notations appearing in the thesis.

Conceptually, LF and its object systems could be seen as a hierarchical world: the kinds in LF (i.e. the types of LF itself) and the types in the object type theory. With the mechanism of inductive schemata, various inductive types could be defined to construct rich type systems. For example, the type for Natural numbers $\text{Nat}$, the product type $\Pi$-type, the sum type $\Sigma$-type, the type for vectors $\text{Vector}$, etc. For details of how inductive types can be specified, see Part III of [NPS90] or Chapter 9 of [Luo94].

In LF, the syntactical entities include contexts, kinds and terms, which are of the following forms listed in the Figure 2.1 below:

\[
\begin{align*}
\text{Contexts} & \quad C :: = () \mid \Gamma, x : A \\
\text{LF Kinds} & \quad K :: = \text{Type} \mid \text{El}(A) \mid (x:K)K' \\
\text{LF Terms} & \quad M :: = x \mid [x:K]M \mid M(M')
\end{align*}
\]

Figure 2.1: Syntactical Components of Logical Framework

Since LF is itself a type system, the types in LF are called kinds, including:

- $\text{Type}$ – the kind representing the collection of all types ($A$ is a type if $A : \text{Type}$);
- $\text{El}(A)$ – the kind of objects of type $A$ (we often omit El); and

\footnote{However, the dependent record types here to be extended to the system of LF are not specified with inductive schemata, but are given separately, see the following section.}
2.2. The Logical Framework LF

- \((x:K)K'\) (or simply \((K)K'\) when \(x \notin FV(K')\)) – the kind of dependent functional operations \(f\) which can be applied to an object \(k\) of kind \(K\) to form \(f(k)\) of kind \([k/x]K'\). When \(f \equiv [x:K]k\), \(f(a)\) is computationally equal to \([a/x]k\). \(^2\)

Notation-wise, the dependent product kind \((x:K)K'\) is previously written in our introductory part as \(\Pi x:K.K'\). \([x:K]M\) is the \(\lambda\)-abstraction and was written as \(\lambda x:K.M\); \(M(M')\) is the function application and one could also understand it as \(f(k)\) in a traditional symbolic convention for functions.

We use \(\equiv\) to denote the syntactical identity (up to \(\alpha\)-conversion) and write \(M\{x\}\) to indicate that \(x\) may occur free in \(M\) and subsequently write \(M\{a\}\) for the substitution \([a/x]M\).

The judgement forms in LF include, for example,

- \(\Gamma \vdash k : K\), which asserts that \(k\) is an object of kind \(K\); and
- \(\Gamma \vdash k = k' : K\), which asserts that \(k\) and \(k'\) are (computationally) equal objects of kind \(K\).

The inference rules to define the typing relation and judgemental equality of Logical Framework are shown in the Figure 2.2. In particular, \(\beta\eta\)-equal objects are computationally equal. For instance, an abstraction \([x:K]k\) can be applied to form \(([x:K]k)(a)\) that is computationally equal to \([a/x]k\).

\(^2\)The definition of capture-avoiding substitution (prior to Page 4), noted here as \([N/x]M\), is the substitution of term \(N\) for the free occurrences of variable \(x\) in \(M\), which is defined as usual will possible changes of bound variables.
2.2. The Logical Framework LF

<table>
<thead>
<tr>
<th>Contexts and assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset ) valid \quad \Gamma \vdash K \text{ kind } x \notin FV(\Gamma) \quad \Gamma, x:K \vdash x : K \quad \Gamma, x:K, \Gamma' \vdash x : K |</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General equality rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash K \text{ kind } ) \quad \Gamma \vdash K = K' \quad \Gamma \vdash K = K' \Gamma \vdash K' = K'' \quad \Gamma \vdash K = K''</td>
</tr>
<tr>
<td>( \Gamma \vdash k : K ) \quad \Gamma \vdash k = k' : K \quad \Gamma \vdash k = k : K \quad \Gamma \vdash k = k' = k'' : K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equality typing rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash k : K \Gamma \vdash K = K' \Gamma \vdash k : K' ) \quad \Gamma \vdash k = k' = k'' : K'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Substitution rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma, x:K, \Gamma' \vdash K' \text{ kind } ) \quad \Gamma \vdash k : K \quad \Gamma, [k/x] \Gamma' \vdash [k/x]K' \text{ kind } \quad \Gamma, x:K, \Gamma' \vdash k : K' \quad \Gamma \vdash k = k' : K</td>
</tr>
<tr>
<td>( \Gamma, [k/x] \Gamma' \vdash [k/x]K' \quad \Gamma, x:K, \Gamma' \vdash k : K \quad \Gamma, [k/x] \Gamma' \vdash [k/x]k' : [k/x]K' \quad \Gamma, x:K, \Gamma' \vdash k : K \quad \Gamma, [k/x] \Gamma' \vdash [k/x]k' = [k/x]k'' : [k/x]K''</td>
</tr>
<tr>
<td>( \Gamma, [k/x] \Gamma' \vdash [k/x]K' = [k/x]K'' \quad \Gamma, x:K, \Gamma' \vdash k' : K' \quad \Gamma \vdash k : K \quad \Gamma, [k/x] \Gamma' \vdash [k/x]k' = [k/x]k'' : [k/x]K''</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The kind Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash A : Type \quad \Gamma \vdash A = B : Type \quad \Gamma \vdash \text{El}(A) \text{ kind } \quad \Gamma \vdash \text{El}(A) = \text{El}(B) |</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dependent product kinds</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash K \text{ kind } \quad \Gamma \vdash x:K \vdash K' \text{ kind } \quad \Gamma \vdash K_1 = K_2 \quad \Gamma, x:K_1 \vdash K'_1 = K'_2 \quad \Gamma \vdash (x:K)K' = (x:K_2)K'_2 |</td>
</tr>
<tr>
<td>( \Gamma \vdash x:K \vdash k : K' \quad \Gamma \vdash K_1 = K_2 \quad \Gamma, x:K \vdash k_1 = k_2 : K \quad \Gamma \vdash [x:K_1]k_1 = [x:K_2]k_2 : (x:K_1)K'</td>
</tr>
<tr>
<td>( \Gamma \vdash f : (x:K)K' \quad \Gamma \vdash k : K \quad \Gamma \vdash f = f' : (x:K)K' \quad \Gamma \vdash k_1 = k_2 : K \quad \Gamma \vdash f(k) = f'(k_2) : [k_1/x]K'</td>
</tr>
<tr>
<td>( \Gamma, x:K \vdash k' : K' \quad \Gamma \vdash k : K \quad \Gamma \vdash f : (x:K)K' \quad x \notin FV(f) \quad \Gamma \vdash (x:K')k'(k) = [k/x]k' : [k/x]K' \quad \Gamma \vdash [x:K]f(x) = f : (x:K)K'</td>
</tr>
</tbody>
</table>

Figure 2.2: Inference Rules of Logical Framework (LF)
2.2.1 Computation and Metatheory

In the introduction and the above, we have given a brief image of what a type system presents and what LF is; in this subsection, we are going to introduce the notion of computation of a type system, and what a metatheory contains.

In formal logic, a formal system consists of a formal language of expressions together with some rules for deduction. The deduction is used to derive an expression from one or more other expressions antecedently expressed in the system. A formal system may be formulated and studied for its intrinsic properties (we call them metatheories), or it may be intended as a description (i.e. a model) of external phenomena. We will give some informal discussions about it in this paragraph.

Sentences in the language of a type theory, i.e., the expressions being deduced are judgements, and the formal correctness of which are given by derivability.

We call two forms of judgements: non-hypothetical and hypothetical ones:

- Non-hypothetical judgements are like ⊢ a : A, which assert that the term a computes to a value that is of type A.

- Hypothetical judgements are like Γ ⊢ a : A, where a context Γ ≡ x_i : A_i (i = 1, ..., n) is viewed as an assumption that x_i is an arbitrary object of type A_i, and the judgement Γ ⊢ a : A asserts that “a is of type A in Γ”.

**Derivation** A derivation of a judgement J (in the form Γ ⊢ M : A) is a finite sequence of judgements J_1, ..., J_n (n ≥ 1, J_n ≡ J) such that for 1 ≤ i ≤ n, J_i is the conclusion of some instance of an inference rule whose premises are in {J_j | j < i}.

In type theories, the correct judgements are derivable judgements.

---

3It is the metatheoretic properties, such as subject reduction & strong normalisation, of the computation defined here that justify that a does compute to its value of type A if the above judgement is derivable.
As we mention about correctness, note that there are usually two views of seeing the correctness of a sentence: the model-theoretic tradition will think it important to give a model to the formal system to reason about correctness; while the syntactic one will take that the deductive system itself, i.e. the inference rules and/or axioms, is giving meaning to sentences. Here we tend to be of the second view, since for type theories their complete models do not exist in general. And the correct judgements are just thought of as the derivable ones.

**Derivable judgements** A judgement \( J \) is derivable if there is a derivation of \( J \). We write \( \Delta \Rightarrow_R J \) if \( \Delta \) is the derivation sequence for \( J \) by applying some instances of the inference rules in set \( R \).

**Remark** \( \Delta \) could include \( J \) itself, and in this case the derivation is superfluous.

**Inference rules** A (structural) inference rule\(^4\) is given by a pair \((\Delta, J)\), (usually written as \( \frac{\Delta}{J} \)) where \( \Delta \equiv \{ J_1, ..., J_n \} \) \((n \geq 0)\) is a finite set of judgements (called premises) and \( J \) is a judgement (called conclusion). There may be one or more side conditions on the application of an inference rule.
An instance of the rule is \( (\sigma \Delta \equiv \{ \sigma J_1, ..., \sigma J_n \}, \sigma J) \) for a substitution \( \sigma \).
In the case when \( n = 0 \) the rule is called an axiom, i.e. a rule without any premise.

A rule could be redundant in the sense that it is derivable or admissible.

**Derivable rules** The rule \( r \equiv (\Delta, J) \) is derivable in a formal system if \( \Delta \Rightarrow^+_R J \) from a set of inference rules \( R \) where \( \Rightarrow^+_R \) is the additive closure of \( \Rightarrow_R \).\(^5\) Intuitively, a derivable rule is one whose conclusion can be derived from its premises using the other rules.

\(^4\)The word “structural” refers to the fact that a rule could be seen as a rule scheme, which is different from the meaning of “structural” for rules in a sequent calculus which concern about structures of \( \Gamma \) such as Weakening, Exchange and Contraction rules.

\(^5\)We do not need to consider the instances of this rule, because all the rules are rule schemes that include the instances.
2.2. The Logical Framework LF

**Admissible rules** In logic, a rule \( r \equiv (\Delta, J) \) is admissible if the set of theorems of the system is closed\(^6\) under the rule; i.e., for any instance of the rule for a substitution \( \sigma \), \( \sigma J \) is derivable whenever all judgements from \( \sigma \Delta \) are. Intuitively, an admissible rule is one whose conclusion holds (i.e. the judgement \( J \) is derivable) whenever the premises hold.

**Remark** Admissible rules could be thought of as theorems of the deductive system. Every derivable rule is admissible, but not vice versa in general.

**Example of Admissibility and Derivability** \(^7\) As an example, consider the set of rules which defines Natural Numbers: \( R \equiv \{ \text{rule}_1 \equiv (\emptyset, 0 : \text{nat}), \text{rule}_2 \equiv (n : \text{nat}, s(n) : \text{nat}) \} \).

Then, the rule \( (n : \text{nat}, s(s(n)) : \text{nat}) \) is derivable, because the conclusion \( s(s(n)) : \text{nat} \) is derivable from the premise \( n : \text{nat} \) by applying twice of \( \text{rule}_2 \) with proper substitutions.

While the rule \( (s(n) : \text{nat}, n : \text{nat}) \) is only admissible, not derivable. To prove it is admissible, i.e., whenever \( s(n) : \text{nat} \) holds then \( n : \text{nat} \) holds (†), this could be done via induction on \( n \). For \( n = 0 \), by \( \text{rule}_1 n : \text{nat} \) holds; suppose in the case of \( n \) (†) holds, one assumes a derivation of \( s(n) : \text{nat} \), and in the case of \( n + 1 \), one could produce a derivation of \( s(n) : \text{nat} \) by the induction hypothesis. Thus (†) is proved.

However, it is not derivable, because there is no rule in \( R \) that can be applied to derive the conclusion from the premise. □

\(^6\)Closure in the sense that the set of theorems in this formal system \( Them \equiv Them \cup Adm \) where \( Adm \) denotes the set of admissible rules.

\(^7\)The subtle difference between derivable rules and admissible rules could be explained by the notion of stability: the derivability is stable under addition of the deductive system (i.e. the set of inference rules and/or axioms) while the admissibility ones are not. For instance, if we add another rule in the \( R \) above: \( \text{rule}_3 \equiv (\emptyset, s(-3) : \text{nat}) \). Then, the first rule \( (n : \text{nat}, s(s(n)) : \text{nat}) \) is still derivable following the same proof. But the second rule \( (s(n) : \text{nat}, n : \text{nat}) \) is no longer admissible because the fact \( s(-3) : \text{nat} \) holds (derived from \( \text{rule}_3 \)) couldn’t infer that \( -3 : \text{nat} \) holds (it is not derivable). A logic is called structurally complete if every admissible rule is derivable.
Having given the basic conceptions related to the notion of computation, we introduce some basic concepts of metatheories of a type theory. The metatheory is the intrinsic properties of a formal system (which could already be called as a theory, that is why the word "meta" comes into being). We take the type system Extended Calculus of Constructions (short as ECC, see [Luo94]) as an example for giving the notions of computation and metatheory.

In ECC, computation and the computational equality are defined by means of reduction and conversion, respectively:

Reduction in ECC The one-step reduction, notation $\triangleright_1$, is the compatible closure (Page 49) of the two contraction schemes $\triangleright^\beta$ and $\triangleright^\sigma$ – given below:

- $(\lambda x : A . M) N \triangleright^\beta [N/x] M$ (β contraction scheme)
- $\pi_i((M_1, M_2))_A \triangleright^\sigma M_i$ (σ contraction scheme)

And a reduction is a finite (possibly empty) series of one-step reductions. A term $M$ is said to be reducible from a term $N$ if there exists a reduction from $M$ to $N$.

Conversion in ECC The conversion over two terms, notation $\simeq$, is the smallest equivalence relation generated by the transitive closure of the one-step reduction relation $\triangleright_1$.

Lemma 2.2.1 $M$ is convertible to $M'$, notation $M \simeq M'$, if and only if $\exists M_1, ..., M_n$ s.t. $M \equiv M_1$, $M_n \equiv M'$ and $M_i \triangleright_1 M_{i+1}$ or $M_{i+1} \triangleright_1 M_i$ for $i = 1, ..., n - 1$.

Metatheories of ECC The metatheories of ECC include: Church-Rosser theorem (i.e. if there are two distinct reductions starting from two computationally equal terms, then there exists a term which is reducible from the two computationally equal terms), Strong Normalisation (i.e. every reduction sequence is finite), Subject Reduction (i.e. the typing relation of a term is preserved when it reduces to another term), Substitution or Cut (i.e. a type-preserving substitution in a judgement preserves the derivability of the judgement), etc.
There are many ways to prove metatheories; basically, these methods fall into three categories: first, use reduction-oriented semantics for (untyped) terms, and study the properties reflected on reductions between untyped raw terms, as is done for proving the metatheories for ECC. Second, set up judgemental equality-oriented semantics (existing method called typed operational semantics) for the well-typed terms, and reason the metatheories of the type system with those of this semantics (usually also represented as a type system) via soundness proof. A third method could be seen as an indirect or relative proof: if one has set up reduction-oriented semantics and has proven certain meta-properties for a type theory $T_1$, and also could build up a mapping from constructions in a target type theory $T_2$ to those in $T_1$, and could meanwhile prove that $T_2$ preserves the properties as shown in $T_1$; then, by a relative proof, one could arrive at the meta-properties for $T_2$.

More contents related to the metatheories in type theory will be studied in detail in Chapter 3 and 4, we will give out the second approach, i.e. a typed operational semantics for a type system of dependent records.
2.3 The System of IDRT

In this study, dependent record types have been formulated as an extension of an intensional type theory such as Martin-Löf’s type theory or UTT, which is specified in the logical framework LF. Through showing the Soundness of the semantics to its original system, together with the Strong Normalisation property of terms regarding to the semantics, we succeed to prove metatheory such as the Strong Normalisation Theorem of the dependent record types. Because our system of dependent record types is intensional, we refer to this system of LF extended with DRTs as IDRT.

In the system IDRT, the syntactical entities will include four categories, compared to the ones of LF: contexts $C$, kinds $K$, terms $M$ and record types $R$, listed in the Figure 2.3 below, where we will use $K_{LF}, M_{LF}$ to denote the LF kinds and terms (prior to Page 22), and $K_R, r$ for the kinds of record types and records:

\[
\begin{align*}
\text{Contexts} \quad C &::= \emptyset \mid \Gamma, x : A \\
\text{Kinds} \quad K &::= K_{LF} \mid K_R \\
\text{Record Types} \quad R &::= \emptyset \mid \langle R, l : A \rangle \ (\text{Labels } l \in \mathcal{L}) \\
\text{Terms} \quad M &::= M_{LF} \mid r \mid [M] \mid M.l
\end{align*}
\]

Figure 2.3: Syntactical Components of the System IDRT

2.3.1 The Extension of LF with DRTs

We have extended the LF with DRTs, which is the type system Logical Framework ([Luo94]) extended with dependent record types ([Luo08, Luo09]), the latter is a novel formulation from previous studies on record types ([CPT05, HL94, Pol02]). Dependent record types are record types whose component types may depend on the values of previous fields. The syntax of this type system is given as follows in Figure 2.4, where $\mathcal{L}$ is an (infinite) set of labels, $l \in \mathcal{L}$ and $L \subset \mathcal{L}$ is finite:
2.3. The System of IDRT

\[
\text{Kinds of Record Types} \quad K_R :: = \text{RType} \mid \text{RType}[L]
\]

\[
\text{Record Types} \quad R :: = \langle \rangle \mid \langle R, l : A \rangle \quad (\text{Labels } l \in \mathcal{L})
\]

\[
\text{Records} \quad r :: = \langle \rangle \mid \langle r, l = a : A \rangle
\]

Figure 2.4: Syntactical Components of Dependent Record Types

The inference rules of IDRT consist of the rules for LF (Figure 2.2) and the additional rules in Figure 2.5 of the next section. Here are some informal explanations.

- We add new kinds \(\text{RType}\) and \(\text{RType}[L]\) of record types. Intuitively, \(\text{RType}[L]\) is the kind of the record types whose (top-level) labels are all in \(L\), a finite set of labels. Naturally, if \(L \subseteq L'\), every record type in \(\text{RType}[L]\) is also in \(\text{RType}[L']\). The kind \(\text{RType}\) is the kind of all record types and could conceptually be understood as ‘\(\text{RType}[\mathcal{L}]\)’. Finally, every record type is also a type. These are formally reflected in the rules for the kinds of record types in Figure 2.5.

- Record types are types of the form \(\langle \rangle\) or \(\langle R, l : A \rangle\). Intuitively, a record type is of the form \(\langle l_1 : A_1, ..., l_n : A_n \rangle\),\(^8\) where each \(l_i : A_i\) is a field labelled by \(l\). An object of this record type is a labelled tuple \(\langle l_1 = a_1 : A_1, ..., l_n = a_n : A_n \rangle\), where \(a_i\) is of the type of the corresponding field. Note that, formally, each \(A_i\) in the record type is not a type, but a family of types; this is how dependency is incorporated – we have dependent record types.

- There are two operations on records: restriction (or first projection) \([r]\) that removes the last component of record \(r\) and field selection \(r.l\) that selects the value of the field labelled by \(l\). For instance, intuitively, for the record \(r \equiv \langle l_1 = a_1 : A_1, l_2 = a_2 : A_2, l_3 = a_3 : A_3 \rangle\) of type \(\langle l_1 : A_1, l_2 : A_2, l_3 : A_3 \rangle\), we have \([r] = \langle l_1 : A_1, l_2 : A_2 \rangle\) and \(r.l_2 = [r].l_2 = a_2\). These are formally reflected in the introduction, elimination and computation rules in Figure 2.5.

- The congruence rules for record types and records in Figure 2.5 propagate the computational equality through the term structure. Also, we do not include

\(^8\)We overload the \(\langle \ldots \rangle\) notation for records and their types. It is always possible to distinguish between the two.
the weakly extensional equality rules as considered in [Luo08], which we will discuss further in Section 6.1.1. Therefore, we call the system the type system for intensional DRTs.

We shall adopt the following terminology: the terms of the form \( (r, l = a : A) \) will be called pair-records. (For example, we shall use this terminology in specifying the TOS-rules for record types in Figure 3.3 in Chapter 3.)

**Record Types v.s. Record Kinds**

It is worth pointing out that our type system contains dependent record *types* (as studied by Pollack [Pol02], Luo [Luo08, Luo09] and the current thesis), rather than dependent record *kinds* (as studied by Betarte and Tasistro [BT98] and Coquand, Pollack and Takeyama [CPT05]). We would like to distinguish these two notions clearly: in a type theory with inductive types, types include those such as \( \text{Nat} \) of natural numbers and \( \Sigma \)-types of dependent pairs, while the examples of kinds include, for example, the kind \( \text{Type} \) of all types. They exist at two completely different levels and have rather different structures and properties.

In general, types have a much more sophisticated and richer structure than kinds. For instance, it is easy to show that a kind is of the form either \( \text{Type} \) or \( (x : K)K' \), but types are not (e.g., a type may be of the form \( f(a) \)). To appreciate the difference, let us consider the issue of ensuring label distinctness. If one considers only record kinds, it is easy to guarantee that the labels in the same record kind are distinct because of the limited syntactic forms of kinds (see, for example, [CPT05]). However, this is not easy at all for record types (think, for example, how one ensures that a label does not occur in a type of the form \( f(a) \)). In our case, we have to introduce the kinds \( RType[L] \) to ensure that it is the case that the (top-level) labels in the same record type are distinct. In other words, intuitively, \( l \neq l' \) for any record type \( \langle ..., l : A, ..., l' : A', ... \rangle \). This is guaranteed by means of the side condition \( l \notin L \) of the second formation rule in Figure 2.5.

That a type system with record types is more powerful than one with only record kinds can be understood from another angle when one wants to introduce universes of

---

9 Types in the terminology of Martin-Löf’s type theory are what we call kinds in this paper. Therefore, the so-called record types in [BT98] and [CPT05] are really record kinds.
2.3. The System of IDRT

record types. It is possible to introduce type universes for dependent record types, as shown in [Luo09]; this, however, cannot be done for record kinds. Therefore, record types are more useful than record kinds (for example, in representing module types in data refinement [Luo09]).

Since types have a more sophisticated structure than kinds, it is more difficult to study the metatheoretic properties of a system with record types, as compared with a metatheoretic study of record kinds. As we show in this paper, the approach of using typed operational semantics can be used in this endeavour.

2.3.2 Inference Rules of DRTs

The inference rules of Dependent Record Types are as follows in the Figure 2.5, which consist of:

- The rules for kinds for record types. The first two rules basically say that both \( R\text{Type} \) and \( R\text{Type}[L] \) belong to the kind level; the later three rules indicate three subkinding relations (i.e. an inclusion over the kinds) between the \( R\text{Type}[L'] \) (\( L \subseteq L' \)) and \( R\text{Type}[L], R\text{Type}[L] \) and \( R\text{Type}, R\text{Type} \) and \( Type \) respectively.

- Formation rules for record types. These two formation rules say how to form the empty record type and the non-trivial record type.

- Introduction rules for records. These two introduction rules indicate the typing relation between the records and their record types; note that an empty record is of type of the empty record type, with overloaded symbol \( \langle \rangle \).

- Elimination rules for records. These three elimination rules say how to decompose a record of a non-trivial record type; note that because in our setting labels are forced to be distinct, there is an elimination rule on different label selection.

- Computational rules for records. These three computational rules say how to conduct the two operations on records; similarly there is a computational rule on different label selection.
2.3. The System of IDRT

<table>
<thead>
<tr>
<th>Kinds RType</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅ valid RType[∅]</td>
</tr>
<tr>
<td>∅ valid RType[]</td>
</tr>
</tbody>
</table>

Formation rules

<table>
<thead>
<tr>
<th>Γ valid RType[L]</th>
<th>Γ valid RType[L]</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊢ R : RType[L]</td>
<td>⊢ R : RType[L]</td>
</tr>
<tr>
<td>⊢ R : RType[L]</td>
<td>⊢ R : RType[L]</td>
</tr>
<tr>
<td>⊢ R : RType[L]</td>
<td>⊢ R : RType[L]</td>
</tr>
<tr>
<td>⊢ R : RType[L]</td>
<td>⊢ R : RType[L]</td>
</tr>
</tbody>
</table>

Introduction rules

<table>
<thead>
<tr>
<th>Γ valid RType[L]</th>
<th>Γ valid RType[L]</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊢ ⟨⟩ : RType[L]</td>
<td>⊢ ⟨⟩ : RType[L]</td>
</tr>
<tr>
<td>⊢ ⟨⟩ : RType[L]</td>
<td>⊢ ⟨⟩ : RType[L]</td>
</tr>
<tr>
<td>⊢ ⟨⟩ : RType[L]</td>
<td>⊢ ⟨⟩ : RType[L]</td>
</tr>
<tr>
<td>⊢ ⟨⟩ : RType[L]</td>
<td>⊢ ⟨⟩ : RType[L]</td>
</tr>
</tbody>
</table>

Elimination rules

<table>
<thead>
<tr>
<th>Γ valid RType[L]</th>
<th>Γ valid RType[L]</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊢ r : ⟨⟩</td>
<td>⊢ r : ⟨⟩</td>
</tr>
<tr>
<td>⊢ r : ⟨⟩</td>
<td>⊢ r : ⟨⟩</td>
</tr>
<tr>
<td>⊢ r : ⟨⟩</td>
<td>⊢ r : ⟨⟩</td>
</tr>
<tr>
<td>⊢ r : ⟨⟩</td>
<td>⊢ r : ⟨⟩</td>
</tr>
</tbody>
</table>

Computation rules

<table>
<thead>
<tr>
<th>Γ valid RType[L]</th>
<th>Γ valid RType[L]</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊢ [r] : R</td>
<td>⊢ [r] : R</td>
</tr>
<tr>
<td>⊢ [r] : R</td>
<td>⊢ [r] : R</td>
</tr>
<tr>
<td>⊢ [r] : R</td>
<td>⊢ [r] : R</td>
</tr>
<tr>
<td>⊢ [r] : R</td>
<td>⊢ [r] : R</td>
</tr>
</tbody>
</table>

Congruence rules for record types

<table>
<thead>
<tr>
<th>Γ valid RType[L]</th>
<th>Γ valid RType[L]</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊢ R = R : RType[L]</td>
<td>⊢ R = R : RType[L]</td>
</tr>
<tr>
<td>⊢ R = R : RType[L]</td>
<td>⊢ R = R : RType[L]</td>
</tr>
<tr>
<td>⊢ R = R : RType[L]</td>
<td>⊢ R = R : RType[L]</td>
</tr>
<tr>
<td>⊢ R = R : RType[L]</td>
<td>⊢ R = R : RType[L]</td>
</tr>
</tbody>
</table>

Congruence rules for records

<table>
<thead>
<tr>
<th>Γ valid RType[L]</th>
<th>Γ valid RType[L]</th>
</tr>
</thead>
</table>

Figure 2.5: Inference Rules of Intensional Dependent Record Types (IDRT)
• Congruence rules for record types. These two congruence rules say that empty record types are regarded as computationally equal, and two non-trivial record types are computationally equal if their components are.

• Congruence rules for records. Same as above; also if two records are computationally equal, the restriction and field selection results are computationally equal as well.

Some Discussion on Field-selection Rules

We studied the two forms of computation rules for the DRTs and show that they are equal if some form of extensional equality is allowed, indicated by the following comment:

Equal Forms of Two Field-Selection Rules Consider the following two forms of computational rules on record field selection, Prove that (1) and (2) are equal if we have \( r =_\eta ([r], l = r.l : A) : (R, l : A) \).

\[
\begin{align*}
\Gamma \vdash r : (R, l : A) \quad \Gamma \vdash [r].l' : B \quad l \neq l' \\
\Gamma \vdash r.l' = [r].l' : B \quad \text{(1)}
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash \langle r, l = a : A \rangle : (R, l : A) \quad \Gamma \vdash r.l' : B \quad l \neq l' \\
\Gamma \vdash \langle r, l = a : A \rangle .l' = r.l' : B \quad \text{(2)}
\end{align*}
\]

**Proof.** \( (\Rightarrow) \) Obvious. Apply field selection of \( l' \) to both records \( \langle r, l = a : A \rangle \) and \( r \).

\( (\Leftarrow) \) This can be derived by using the fact that \( r =_\eta ([r], l = r.l : A) : (R, l : A) \) (we note this equality as \( (\eta_R) \)) up to the weakly extensional equality. \( \square \)

We have also the following two lemmas, which could be proved by induction over structure of terms, to make us understand more on the kind \( RType \):

**Lemma 2.3.1 (Well-typeness of Field-Selection to a Random Record)** If \( R : RType[\{l\}] \), then \( r : R \vdash r.l : B \) for some \( B \).

**Lemma 2.3.2 (Syntactic Composition of Record Types)** If \( \Gamma \vdash R : RType[L] \), then either \( R \in FV[\Gamma] \) or \( R \equiv \langle R', l : A \rangle \) for \( l \in L \) up to syntactic identity.
Chapter 3

A Typed Operational Semantics for DRTs
3.1 Basic Introduction

The typed operational semantics (abbreviated as TOS) is a proof-theoretic method to prove the metatheoretic properties of type theories. It was developed by H. Goguen in his PhD thesis [Gog94], where he studied the metatheory of UTT and proved that UTT has the nice properties such as Church-Rosser, Subject Reduction and Strong Normalisation.

In this thesis, the TOS approach is applied to study the metatheory of dependent record types. After a brief informal introduction of the approach, we develop the typed operational semantics for the system IDRT of intensional DRTs and show that it has the soundness and completeness properties. The metatheoretic properties of TOS for dependent record types are studied in Chapter 4.

The TOS Approach

For a type theory, its typed operational semantics captures its computational behaviour, usually given by its (untyped) reduction relation. For example, in TOS, the following judgement

\[ \Gamma \vdash M \rightarrow N \rightarrow P : A \]

informally asserts that, among other things, \( N \) and \( P \) are the weak-head normal form and the normal form of the term \( M \), respectively.\(^1\) For the logical framework LF, for example, its corresponding TOS has been studied [Gog99] and its inference rules are given in Section 3.2. Since many metatheoretic properties of a type theory are concerned with its computational behaviour, it is not a surprise that TOS provides an effective approach to the metatheory of type theories.\(^2\)

The TOS and its corresponding type theory are related to each other by means of the soundness and completeness theorems. Using the judgement \( \Gamma \vdash M : A \) to abbreviate \( \Gamma \vdash M \rightarrow N \rightarrow P : A \) for some \( N \) and \( P \), we can state the soundness and completeness properties as follows:

---

\(^1\)Formally, the reduction relation and the TOS are related to each other by means of the ‘adequacy theorems’ such as Lemmas 4.3.4 and 4.3.5 for IDRT in Section 4.3.2.

\(^2\)It is worth noting that, although it is useful to study the metatheory for many type theories, the TOS approach would not be suitable for non-normalising type theories. See [Gog94] for discussions.
3.1. Basic Introduction

- Soundness: $\Gamma \vdash M : A$ implies $\Gamma \models M : A'$ (for some $A'$ that is normal form of $A$).

- Completeness: $\Gamma \models M : A$ implies $\Gamma \vdash M : A$.

Based on soundness and completeness, we can prove many metatheoretic properties of the type theory. For example, it can be shown that, if $\Gamma \models M : A'$, then $M$ is strongly normalizable. Therefore, strong normalisation, the property that every well-typed term is strongly normalizable, can be proved by means of such a fact together with the soundness property, as pictured as follows:

\[
\begin{array}{c}
\Gamma \vdash M : A \quad \text{Strong Normalisation} \quad M \text{ is SN} \\
\text{Soundness} \quad \Gamma \models M : A' \quad \text{SN for TOS}
\end{array}
\]

As shown in Chapter 4 and 5, for dependent record types, the SN property for the corresponding TOS is proven in Theorem 4.5.6. Then, by the Soundness Theorem (Theorem 4.4.2), we can show that strong normalisation for IDRT (Corollary 4.6.2).

Note that, to implement such ideas is not a simple matter: it requires one to prove:

- that the TOS is “adequate” w.r.t. the (untyped) reduction relation,
- that the TOS is sound and complete w.r.t. the original type theory, and
- that the TOS satisfies some specific metatheoretic properties (e.g., strong normalisation).

Then, one can transfer the results to the original type theory to show that it has nice metatheoretic properties. This is what we shall do for IDRT, the type theory with dependent record types.

TOS for Dependent Record Types

The typed operational semantics for dependent record types is described in this section. The judgement forms in a TOS are given in Figure 3.1, three of which are

3.1. Basic Introduction

**Basic forms:**

- $\Gamma \vdash \Delta$
- $\Gamma \vdash A \rightarrow B$
- $\Gamma \vdash M \rightarrow N \rightarrow P : A$

**Abbreviated forms:**

- $\Gamma \vdash \text{ok}$
- $\Gamma \vdash M \rightarrow_w N : A$
- $\Gamma \vdash M \rightarrow_n P : A$
- $\Gamma \vdash M : A$

Figure 3.1: Judgement Forms in Typed Operational Semantics

the basic forms of judgements whose informal meanings are:

- $\Gamma \vdash \Delta$: the context $\Gamma$ has context $\Delta$ as its normal form;

- $\Gamma \vdash A \rightarrow B$: the kind $A$ is well formed in context $\Gamma$ and has normal form $B$; and

- $\Gamma \vdash M \rightarrow N \rightarrow P : A$: the terms $M$, $N$, $P$ are well formed in context $\Gamma$ of kind $A$ and $M$ has weak-head normal form $N$ and normal form $P$.

From these basic judgements, one can define other forms of judgements, including the following:

- $\Gamma \vdash \text{ok}$ stands for “$\Gamma \vdash \Delta$ for some $\Delta$”;

- $\Gamma \vdash M \rightarrow_w N : A$ stands for “$\Gamma \vdash M \rightarrow N \rightarrow P : A$ for some $P$”;

- $\Gamma \vdash M \rightarrow_n P : A$ stands for “$\Gamma \vdash M \rightarrow N \rightarrow P : A$ for some $N$”; and

- $\Gamma \vdash M : A$ stands for “$\Gamma \vdash M \rightarrow N \rightarrow P : A$ for some $N$ and $P$”.

The typed operational semantics for the type system IDRT of intensional DRTs is the extension of that for LF (Figure 3.2) with the inference rules given in Figure 3.3.
3.2 A TOS for Logical Framework

This is the typed operational semantics Goguen developed in [Gog99] for the logical framework LF:

\[
\begin{align*}
\text{Contexts} & \quad \Gamma \vdash () \rightarrow () & EMP \quad \models \Gamma \rightarrow \Delta & \quad \Gamma \models A \rightarrow B & \quad x \notin \operatorname{dom}\{\Gamma\} & \quad \text{W\textsc{eak}} \\
\text{Kinds} & & \Gamma \models \text{ok} & \quad \Gamma \models \text{Type} \rightarrow \text{Type} & \quad \Gamma \models M \rightarrow N \rightarrow P : \text{Type} & \quad \EL \\
& & \Gamma \models A_1 \rightarrow B_1 & \quad \Gamma, x:A_1 \models A_2 \rightarrow B_2 & \quad \Gamma \models (x:A_1)A_2 \rightarrow (x:B_1)B_2 & \quad \PI \\
\text{Terms} & \quad \Gamma_0, x:A, \Gamma_1 \models A \rightarrow B & \quad \Gamma_0, x:A, \Gamma_1 \models x \rightarrow x : B & \quad \VAR \\
& & \Gamma \models A_1 \rightarrow B_1 & \quad \Gamma, x:A_1 \models M_0 \rightarrow P_0 : B_2 & \quad [x:B_1]P_0 \text{ not } \eta \text{- redex} & \quad \LAM \\
& & \Gamma \models [x:A_1]M_0 \rightarrow [x:A_1]M_0 \rightarrow [x:B_1]M_0 \rightarrow (x:B_1)B_2 & \quad \Gamma \models P \rightarrow P \rightarrow P : (x:B_1)B_2 & \quad \ETA \\
& & \Gamma \models A_1 \rightarrow B_1 & \quad \Gamma, x:A_1 \models M_0 \rightarrow P(x) : B_2 & \quad \Gamma \models [x:A_1]M_0 \rightarrow [x:A_1]M_0 \rightarrow P : (x:B_1)B_2 & \quad \BASE \\
& & \Gamma \models [M_2/x]B_2 \rightarrow C & \quad N_1 \text{ not abstraction} & \quad \Gamma \models M_1 \rightarrow N_1 \rightarrow N_1(M_2) \rightarrow P_1(P_2) : C & \quad \BETA \\
& & \Gamma \models M_1 \rightarrow [x:A_1]N_0 : (x:B_1)B_2 & \quad \Gamma \models M_2 : B_1 & \quad \Gamma \models [M_2/x]N_0 \rightarrow P \rightarrow Q : C & \quad \Gamma \models [M_2/x]B_2 \rightarrow C & \quad \BETA \\
& & \Gamma \models M_1(M_2) \rightarrow P \rightarrow Q : C
\end{align*}
\]

Figure 3.2: Inference Rules of Typed Operational Semantics for LF
Following these rules of the semantics, regarding to LF rules (in Figure 2.2) in the previous Chapter, it’s not hard to see that the rules correspond to all the possible normalisation processes in the LF. Because the syntactical components of LF are contexts, kinds, and terms, the TOS rules are of these three groups too:

- Rules for contexts normalisation: there are two rules for empty context and for non-empty context for which the normalisation process is defined inductively;

- Rules for LF kinds: three rules respectively for the kinds $TYPE$, $EL(M)$ and dependent product kind $PI$, which are quite straightforward;

- Rules for LF terms: there are two groups of rules, one for the $\lambda$-reduction wise, and the other for the $\eta$-reduction wise (remember that in LF there is $\eta$-reduction, see Definition 4.2.2 on Page 50, at the second entry; but there is no $\eta$-reduction in DRT); for each group, there are two sorts of rules, the $BASE$ rules are for the case when reduction actually doesn’t occur during the normalisation process, and the other two rules are for the actual $\beta\eta$-reductions during normalisation process.

From the description above on the TOS for LF (which shall be referred as $TOS_{LF}$ when specifically needed in the later text), which is developed in H. Goguen’s thesis [Gog94], and the large solid space that he has set up for this theoretical approach, it’s convincing for us that this nicely defined mechanism could work also with a high-level of validity for the intensional DRTs.
3.3 A TOS for Dependent Record Types

We extend the typed operational semantics for LF given in the last section, and develop a TOS for the dependent record types.

Formally, the typed operational semantics for DRTs is defined inductively by the inference rules shown in the following Figure 3.3.

As corresponding to Figure 2.5, the TOS rules for DRTs (TOS\textsubscript{DRT}) consist of:

- The TOS rules for kinds of record types \textit{RType} and \textit{RType}[L].
- The TOS rules for record types: the empty record type and the non-trivial record type.
- The TOS rules for pair-records: the empty records and the non-trivial records.
- The TOS rules for both computations, i.e. restriction and field-selection operations on records. Intuitively, the BASE-rules are for situations when the computation does not happen, and the other ones are for those when computation happens. For field selection, there is a rule for different label selection, because in our setting labels in a record are meant to be distinct.

Most of the rules are self-explanatory. We only mention that, besides using the abbreviated forms of judgement (see above) in the rules, we also use the terminology of “pair-record” as introduced on Page 32. For example, in (BASE\textsubscript{RESTR}), we require that \(p\) or \(q\) be not a pair-record, for otherwise, for instance, \([p]\) could be a redex and would not be in normal form.

We should point out that the informal meaning of the TOS judgements in Section 3.1 are different from the formal definition of the TOS here through the inference rules. However, the Adequacy Lemma and the Completeness and Soundness Theorem in next Chapter 4 show that these two coincide.
### Kinds of Record Types

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \models \text{ok} )</td>
<td>( \text{RTYPE} ) ( \Gamma \models \text{RType} \rightarrow \text{RType} ) ( \Gamma \models \text{RTYPE} ) ( \Gamma \models \text{RType}[L] \rightarrow \text{RType}[L] )</td>
</tr>
</tbody>
</table>

### Record Types

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \models \text{ok} )</td>
<td>( \text{EMP}_{\text{RCDT}} ) ( \Gamma \models \langle \rangle \rightarrow \langle \rangle \rightarrow \langle \rangle : \text{RType}[\emptyset] ) ( \Gamma \models R \rightarrow_n P : \text{RType}[L] ) ( \Gamma \models A \rightarrow_n B : (P) \text{Type} ) ( l \notin L ) ( \Gamma \models \langle R, l : A \rangle \rightarrow \langle R, l : A \rangle \rightarrow \langle P, l : B \rangle : \text{RType}[L \cup {l}] )</td>
</tr>
</tbody>
</table>

### Pair-records

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \models \text{ok} )</td>
<td>( \text{EMP}_{\text{RCD}} ) ( \Gamma \models \langle \rangle \rightarrow \langle \rangle \rightarrow \langle \rangle : \langle \rangle ) ( \Gamma \models \langle R, l : A \rangle \rightarrow_n \langle P, l : B \rangle : \text{RType} ) ( \Gamma \models r \rightarrow_n p : P ) ( \Gamma \models A(r) \rightarrow_n C : \text{Type} ) ( \Gamma \models a \rightarrow_n b : C ) ( \Gamma \models \langle r, l = a : A \rangle \rightarrow \langle r, l = a : A \rangle \rightarrow \langle p, l = b : B \rangle : \langle P, l : B \rangle )</td>
</tr>
</tbody>
</table>

### Restrictions

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \models r \rightarrow q \rightarrow p : \langle P, l : B \rangle ) ( p, q \text{ not pair-records} )</td>
<td>( \Gamma \models [r] \rightarrow [q] \rightarrow [p] : P ) ( \text{BASE}_{\text{RESTR}} ) ( \Gamma \models r \rightarrow w \langle p, l = b : A \rangle : \langle P, l : B \rangle ) ( \Gamma \models p \rightarrow s \rightarrow t : P )</td>
</tr>
</tbody>
</table>

### Selections

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \models r \rightarrow q \rightarrow p : \langle P, l : B \rangle ) ( p, q \text{ not pair-records} )</td>
<td>( \Gamma \models [r] \rightarrow q.r[l] \rightarrow p.l : C ) ( \text{BASE}_{\text{FLDSEL}} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \models r \rightarrow w \langle p, l = b : A \rangle : \langle P, l : B \rangle ) ( \Gamma \models b \rightarrow c \rightarrow d : C )</td>
<td>( \Gamma \models A(p) \rightarrow_n C : \text{Type} ) ( \Gamma \models r.l \rightarrow c \rightarrow d : C ) ( \text{FLDSEL} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \models r \rightarrow s : \langle P, l : B \rangle ) ( \Gamma \models [r].l' \rightarrow c \rightarrow d : C ) ( l \neq l' )</td>
<td>( \Gamma \models r.l' \rightarrow c \rightarrow d : C ) ( \text{FLDSL}' )</td>
</tr>
</tbody>
</table>
3.4 Idea of Using the TOS to Prove Metatheory

As we have shown such a picture in Section 3.1 to sketch out the idea of using a typed operational semantics to prove metatheory for a type system, we will illustrate further this idea in the shape of several Theorems, in order to give a better understanding of our major proofs in this thesis.

\[
\begin{align*}
\Gamma \vdash M : A & \xrightarrow{\text{Strong Normalisation}} M \text{ is SN} \\
\text{Soundness} & \quad \Gamma \vdash M : A' \\
\Gamma \vdash M = N : A & \xrightarrow{\text{SN for TOS}} M \text{ and } N \text{ are both strongly normalizable to some term } P
\end{align*}
\]

The reason that we use TOS to prove metatheory for IDRT lies in the following: the TOS serves as a semantical control to the terms in IDRT. It describes reductive behaviours of the system IDRT, thus is called operational. The well typeability of terms in TOS indicates the strong normalizability of terms in IDRT. However, to implement this idea is not as simple: one has to first prove this semantics is sound and complete to the original system, and adequate to the untyped reduction in IDRT; secondly, one proves the metatheoretic properties of the TOS, then, by transference, indirectly proves those properties for IDRT.

To give an example, if one wants to prove that in IDRT, \( \Gamma \vdash M = N : A \) implies that \( M \) and \( N \) are both strongly normalizable to some term \( P \), we will have to first prove that in TOS, \( \Gamma \vdash M \rightarrow_n N : A \) implies \( M \) and \( N \) are both strongly normalizable to \( P \). Then, by soundness, one derives the previously stated theorem. And to do the latter proof, we shall show the adequacy of TOS with respect to the untyped reduction in IDRT.

We explain this by listing three main theorems in the following, i.e. the three arrows in the above picture:

First, the TOS describes reductive behaviours of IDRT, and could satisfy certain formal properties due to a careful typing control. The typing in TOS declares the well-behavenss of strongly normalizing terms which is going to be defined on Page 50, and thus good metatheoretic properties could be guaranteeingly proved, such as \textit{Strong Normalisation}, whose proof will be shown in Chapter 4, Section 4.5.
3.4. Idea of Using the TOS to Prove Metatheory

**Strong Normalisation of TOS:**

1. If $\Gamma \vdash A \rightarrow B$ then $A$ is strongly normalizable;
2. If $\Gamma \vdash M \rightarrow N : P : A$ then $M$ is strongly normalizable.

Second step, these good properties of TOS could be transferred to the IDRT, by the Soundness Theorem shown below, for which the proof will be demonstrated in Chapter 4, Section 4.4.

**Soundness of TOS with respect to IDRT:**

1. If $\Gamma$ valid then $\Gamma \vdash \text{ok}$;
2. If $\Gamma \vdash A$ kind then there exists $B$ such that $\Gamma \vdash A \rightarrow B$;
3. If $\Gamma \vdash A = B$ then there exists $C$ such that $\Gamma \vdash A \rightarrow C$ and $\Gamma \vdash B \rightarrow C$;
4. If $\Gamma \vdash M : A$ then there exist $P, B$ such that $\Gamma \vdash A \rightarrow B$ and $\Gamma \vdash M \rightarrow_n P : B$;
5. If $\Gamma \vdash M = N : A$, then both $M$ and $N$ are strongly normalizable to some $P$, $A$ is strongly normalizable to some $B$ such that $P : B$ in $\Gamma$;
6. If $\Gamma \vdash M = N : A$, then both $M$ and $N$ are strongly normalizable to some $P$, $A$ is strongly normalizable to some $B$ such that $P : B$ in $\Gamma$.

Finally, this approach of using the typed operational semantics to prove metatheory in IDRT lies in binding the two above, which leads to a proof-theoretic result of metatheories of IDRT, among which crucially the strong normalisation of IDRT. Having the above Strong Normalisation of TOS and Soundness of TOS proven, we can derive metatheory such as *Strong Normalisation* of the system IDRT shown as the following, which will appear in a hub in Chapter 4, Section 4.6.

**Strong Normalisation of IDRT:**

1. If $\Gamma valid$ then $\Gamma \vdash \text{ok}$;
2. If $\Gamma \vdash A$ kind then $A$ is strongly normalizable;
3. If $\Gamma \vdash A = B$ then both $A$ and $B$ are strongly normalizable to some $C$;
4. If $\Gamma \vdash M : A$, then $M$ is strongly normalizable to some $P$, $A$ is strongly normalizable to some $B$ such that $P : B$ in $\Gamma$;
5. If $\Gamma \vdash M = N : A$, then both $M$ and $N$ are strongly normalizable to some $P$, $A$ is strongly normalizable to some $B$ such that $P : B$ in $\Gamma$.

All these are just to give a very general idea, the actual proofs to arrive at these three major Theorems are much more complicated as many Lemmas are needed to be shown and some extra formalisms (e.g. Parallel Reduction, to Page 65) are to be constructed, we shall elaborate this work in the next Chapter 4.
Chapter 4

Metatheoretic Properties of TOS for DRTs
4.1 Basic Structural Properties

Structural properties on the judgements are first part of the metatheory for the typed operational semantics, and they could be proved by induction on derivations in the typed operational semantics.

**Lemma 4.1.1 (Subcontexts)** Any derivation \( \Gamma_0, \Gamma_1 \models J \) has a (not necessarily strict) sub-derivation of \( \Gamma_0 \models \text{ok} \).

*Proof.* By induction on derivations. Consider cases of last TOS rules applied:
- **EMP**, easy to see both \( \Gamma_0 \) and \( \Gamma_1 \) are (\), immediate.
- **WEAK**, consider subcases on \( \Gamma_0, \Gamma_1 \): (1) \( \Gamma_0 \equiv \Gamma \) and \( \Gamma_1 \equiv x:A \), immediate; (2) \( \Gamma_0 \equiv \Gamma, x:A \) and \( \Gamma_1 \equiv () \), immediate; (3) \( \Gamma \equiv \Gamma_0, \Gamma_2 \) and \( \Gamma_1 \equiv \Gamma_2, x:A \), since \( \Gamma_2 \) is subcontext of \( \Gamma_1 \), by using induction hypothesis on \( \Gamma \) we have \( \Gamma_0 \models \text{ok} \).
- For kinds TYPE, EL, PI, RTYPE and RTYPE[L], by induction hypothesis.
- For all terms, all by induction hypothesis. \( \square \)

**Lemma 4.1.2 (Contexts)** If \( \Gamma \models J \) then \( \Gamma \) is consistent w.r.t \( J \), i.e. :
1. If \( \Gamma \models A \rightarrow B \) then \( \text{FV}(A) \cup \text{FV}(B) \subseteq \text{dom}\{\Gamma}\} \);
2. If \( \Gamma \models M \rightarrow N \rightarrow P : A \) then \( \text{FV}(M) \cup \text{FV}(N) \cup \text{FV}(P) \cup \text{FV}(A) \subseteq \text{dom}\{\Gamma}\};
And particularly (3) If \( \models \Gamma \Rightarrow \Delta \) then \( \text{dom}\{\Delta}\} = \text{dom}\{\Gamma}\}.

*Proof.* By induction on derivations. Consider the following cases:
- **EMP**, immediate.
- **WEAK**, by induction hypothesis \( \text{dom}\{\Delta}\} = \text{dom}\{\Gamma}\} \), \( \text{FV}(A) \cup \text{FV}(B) \subseteq \text{dom}\{\Gamma}\} \), and \( x \notin \text{dom}\{\Gamma}\}, hence \( \text{dom}\{\Delta, x:B\} = \text{dom}\{\Delta\} \cup \{x\} \cup \text{FV}(B) = \text{dom}\{\Gamma\} \cup \{x\} = \text{dom}\{\Gamma\} \cup \{x\} \cup \text{FV}(A) = \text{dom}\{\Gamma, x:A\} \).
- The other cases are similarly by induction hypothesis. \( \square \)

**Lemma 4.1.3 (Renaming)** Suppose \( \gamma \) is a renaming from \( \Delta \) to \( \Gamma \). If \( \Gamma \models J \), then \( \Delta \models J[\gamma] \) (where \( J[\gamma] \) is the \( \gamma \)-renaming from \( J \), i.e. \( A[\gamma] \rightarrow B[\gamma] \) or \( M[\gamma] \rightarrow 

\(^1\)Subcontexts lemma could be also called Context Validity.
\[ N[\gamma] \rightarrow P[\gamma] : B[\gamma] \] when \( J \) appears as the form \( A \rightarrow B \) or \( M \rightarrow N \rightarrow P : B \) respectively).

\textit{Proof.} By induction on derivations. Use Contexts (lemma 4.1.2) for the cases of \( PI, \ LAM \) and \( ETA \).

\] Lemma 4.1.4 (Weakening) If \( \Gamma \models J \) and \( \Gamma \subseteq \Delta \), then \( \Delta \models J \).

\textit{Proof.} By induction on derivations.

\] Lemma 4.1.5 (Strengthening) Suppose \( z \notin FV(\Gamma_1) \cup FV(A) \cup FV(M) \) (\( A \) and \( M \) are first symbol in \( J \), i.e. if \( J \) appears as the form \( A \rightarrow B \) or \( M \rightarrow N \rightarrow P : B \) respectively). If \( \Gamma_0, z:C, \Gamma_1 \models J \) then \( \Gamma_0, \Gamma_1 \models J \).

\textit{Proof.} By simultaneous induction on derivations. Use Contexts (lemma 4.1.2) for the case of \( ETA \).
4.2 Prerequisite Definitions

In this section we will give some important definitions related to our proofs, namely, the definition of untyped reduction over terms in IDRT, and the definitions of how a term is weak-head normal, normal and strongly normalizable.

We shall show in Section 4.3.2 that the notion of computation captured in TOS is adequate with respect to the usual (untyped) reduction relation, which is defined in the following definitions.

Definition 4.2.1 (Compatible Closure) Let $R$ be a relation on terms. The compatible closure of $R$, notation $\triangleright_R$, for the system IDRT, is the least relation satisfying the following rules shown in Figure 4.1:

$$
\frac{M \sim R N}{M \triangleright_R N (R - Inc)}
$$

$$
\frac{A_1 \triangleright_R B_1}{(x:A_1)A_2 \triangleright_R (x:B_1)A_2 (\Pi - L)}
$$

$$
\frac{A_2 \triangleright_R B_2}{(x:A_1)A_2 \triangleright_R (x:A_1)B_2 (\Pi - R)}
$$

$$
\frac{R_1 \triangleright_R R_2}{\langle R_1, l : A \rangle \triangleright_R \langle R_2, l : A \rangle (RCD_{fst})}
$$

$$
\frac{A_1 \triangleright_R B_1}{\langle R, l : A_1 \rangle \triangleright_R \langle R, l : B_1 \rangle (RCD_{snd})}
$$

$$
\frac{x:A_1]M_0 \triangleright_R x:B_1]M_0 (\lambda - L)}{\langle x:A]M \triangleright_R [x:A]N (\xi)}
$$

$$
\frac{\langle r, l = a : A_1 \rangle \triangleright_R \langle r, l = a : B_1 \rangle (rcd)}{a_1 \triangleright_R a_2}
$$

$$
\frac{r_1 \triangleright_R r_2}{\langle r_1, l = a : A \rangle \triangleright_R \langle r_2, l = a : A \rangle (rcd_{fst})}
$$

$$
\frac{M \triangleright_R N}{M(P) \triangleright_R N(P) (App - L)}
$$

$$
\frac{\langle r, l = a : A \rangle \triangleright_R \langle r, l = a : A \rangle (App - R)}{a_1 \triangleright_R a_2}
$$

$$
\frac{\langle r, l = a : A \rangle \triangleright_R \langle r, l = a : A \rangle (rcd_{snd})}{\langle r, l = a : A \rangle \triangleright_R \langle r, l = a : A \rangle (rcd_{snd})}
$$

$$
\frac{M \triangleright_R N}{P(M) \triangleright_R P(N) (App - R)}
$$

$$
\frac{\langle r, l = a : A \rangle \triangleright_R \langle r, l = a : A \rangle (rcd_{snd})}{\langle r, l = a : A \rangle \triangleright_R \langle r, l = a : A \rangle (rcd_{snd})}
$$

$$
\frac{M.l \triangleright_R N.l (Sel)}{\langle M \rangle \triangleright_R \langle N \rangle (Res)}
$$

Figure 4.1: Compatible Closure $\triangleright_R$ of a Relation $R$ w.r.t. IDRT
Definition 4.2.2 (Untyped Reduction for IDRT) The (untyped) one-step reduction over terms, notation $\to$, is the compatible closure of the relation given by the following rules:

$\beta$ \hspace{1cm} $([x:A]M)N \to [N/x]M$

$\eta$ \hspace{1cm} $[x:A]M(x) \to M$ \hspace{1cm} $(x \notin FV(M))$

$\pi_1$ \hspace{1cm} $[\langle r, l = a : A \rangle] \to r$

$\pi_2$ \hspace{1cm} $\langle r, l = a : A \rangle.l \to a$

$\pi_2'$ \hspace{1cm} $\langle r, l = a : A \rangle.l' \to r.l'$ \hspace{1cm} $(l \neq l')$

We write $\to^+$ and $\to^*$ to be the corresponding transitive closure and reflexive and transitive closure, respectively.

A term of the form on the left of an arrow is called a redex. For example, a $\pi_2$-redex is a term of the form $\langle r, l = a : A \rangle.l$.

Remark (Extensionality and Untyped $\eta$-Reduction for DRT) In particular, if the system of DRTs is extensional, an $\eta$-like untyped reduction could be introduced to the above definition. An $\eta$-redex for DRT objects is a term of the form $<[r], l = r.l : A>$. And a corresponding untyped $\eta$-reduction rule would be like:

$\eta_R$ \hspace{1cm} $[\langle r \rangle, l = r.l : A] \to r$

The field-selection and restriction over empty records, i.e. $\langle \rangle.l$ and $[\langle \rangle]$, are normal, so in the reduction rule $\eta_R$ we do not need the $r$ to be non-empty. More discussion of extensionality and the $\eta$-rules will be given in Chapter 6.

Definition 4.2.3 (Weak-Head Normal Forms and Normal Forms)

- A term $M$ is weak-head normal if
  - $M \equiv x$ is a variable;
  - $M \equiv [x:K]M'$;
4.2. Prerequisite Definitions

- \( M \equiv M'(M'') \), where \( M' \) is weak-head normal and not an abstraction;
- \( M \equiv \langle \rangle \);
- \( M \equiv \langle r, l = a : A \rangle \); or
- \( M \equiv [M'] \) or \( M \equiv M'.l \), where \( M' \) is weak-head normal and not a pair-record.

- A term \( M \) is normal if
  - \( M \equiv x \) is a variable;
  - \( M \equiv [x:K]M' \), which is not an \( \eta \)-redex, and \( K \) and \( M' \) are normal;
  - \( M \equiv M'(M'') \) and \( M' \) and \( M'' \) are normal and \( M' \) not an abstraction;
  - \( M \equiv \langle \rangle \);
  - \( M \equiv \langle r, l = a : A \rangle \) and \( r, a \) and \( A \) are normal.
  - \( M \equiv [M'] \) or \( M \equiv M'.l \), where \( M' \) is normal and not a pair-record.

The notions of weak-head normal forms and normal forms are lifted to record types, kinds and contexts in the usual way.

Definition 4.2.4 (Strongly Normalizable) A term in the system \( IDRT \) is strongly normalizable iff every reduction sequence starting from this term is finite.

Another equivalent way of defining strong normalisation for a term \( A \), written \( SN(A) \), is the least predicate closed under the following rule:

\[(SN - I) \text{ if for all term } B, A \rightarrow B \text{ implies } SN(B), \text{ then } SN(A)\]

For convenience, we attach again below the two Figures for inference rules of LF and of DRT, with each rule enumerated with Roman numerals, for these rules to be referred in the later proofs.
4.2. Prerequisite Definitions

<table>
<thead>
<tr>
<th>Kinds RType</th>
<th>( \Gamma \vdash \text{RType kind} )</th>
<th>( \Gamma \vdash \text{RType kind}_{II} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash \text{RType}[L] ) ( L \subseteq L' )</td>
<td>( \Gamma \vdash \text{RType}[L'] )</td>
<td>( \Gamma \vdash \text{RType} )</td>
</tr>
<tr>
<td>( \Gamma \vdash \text{RType} )</td>
<td>( \Gamma \vdash \text{RType} )</td>
<td>( \Gamma \vdash \text{Type} )</td>
</tr>
</tbody>
</table>

**Formation rules**

<table>
<thead>
<tr>
<th>( \Gamma \vdash \langle \rangle )</th>
<th>( \Gamma \vdash \text{RType}[\emptyset] )</th>
<th>( \Gamma \vdash A : (R)\text{Type} ) ( l \notin L )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash \langle R, l : A \rangle ) : ( \text{RType} )</td>
<td>( \Gamma \vdash \langle r, A \rangle ) : ( \langle r, A \rangle )</td>
<td>( \Gamma \vdash \langle r, l = a : A(r) \rangle )</td>
</tr>
</tbody>
</table>

**Introduction rules**

<table>
<thead>
<tr>
<th>( \Gamma \vdash \langle \rangle )</th>
<th>( \Gamma \vdash \langle R, l : A \rangle )</th>
<th>( \Gamma \vdash \langle r, l : A \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash \langle [r], A \rangle ) : ( \text{RType} )</td>
<td>( \Gamma \vdash \langle r, {l} \rangle : \text{RType}[{l}] )</td>
<td>( \Gamma \vdash \langle r, l = a : A(r) \rangle )</td>
</tr>
</tbody>
</table>

**Elimination rules**

<table>
<thead>
<tr>
<th>( \Gamma \vdash \langle r, l : A \rangle )</th>
<th>( \Gamma \vdash \langle R, l : A \rangle )</th>
<th>( \Gamma \vdash [r], A ) : ( {l} ) : ( \langle r, l : A \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash \langle {r}, l : A \rangle ) ( l \neq l' )</td>
<td>( \Gamma \vdash \langle r, l = a : A(r) \rangle ) : ( {l} )</td>
<td>( \Gamma \vdash \langle r, l = a : A(r) \rangle ) : ( {l} )</td>
</tr>
</tbody>
</table>

**Computation rules**

| \( \Gamma \vdash \langle \{r\}, l : A \rangle \) |
|---------------------------------|---------------------------------|
| \( \Gamma \vdash \langle r, l = a : A(r) \rangle \) : \( \{l\} \) | \( \Gamma \vdash \langle r, l = a : A(r) \rangle \) : \( \{l\} \) |

**Congruence rules for record types**

<table>
<thead>
<tr>
<th>( \Gamma \vdash \langle \rangle )</th>
<th>( \Gamma \vdash \langle \rangle ) : ( \text{RType}[\emptyset] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash \langle R, l : A \rangle ) : ( \langle R, l : A \rangle )</td>
<td>( \Gamma \vdash \langle R, l : A \rangle ) : ( \langle R, l : A \rangle )</td>
</tr>
</tbody>
</table>

**Congruence rules for records**

<table>
<thead>
<tr>
<th>( \Gamma \vdash \langle \rangle )</th>
<th>( \Gamma \vdash \langle \rangle ) : ( \text{RType}[\emptyset] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash \langle r, l = a : A(r) \rangle ) : ( \langle r, l = a : A(r) \rangle )</td>
<td>( \Gamma \vdash \langle r, l = a : A(r) \rangle ) : ( \langle r, l = a : A(r) \rangle )</td>
</tr>
</tbody>
</table>

Figure 4.2: Inf. Rules of Dependent Record Types (indexed with Roman numerals)
4.2. Prerequisite Definitions

Contexts and assumptions

\[ \Gamma \vdash K \quad \text{kind} \quad x \notin \text{FV}(\Gamma), \quad \Gamma, x : K, \Gamma' \vdash \text{valid} \]

\[ \frac{\text{(i)}}{} \quad \frac{\text{(ii)}}{} \quad \frac{\text{(iii)}}{} \]

General equality rules

\[ \frac{\Gamma \vdash K \quad \text{kind}}{\Gamma \vdash K = K} \quad \frac{\Gamma \vdash K = K'}{\Gamma \vdash K' = K''} \quad \frac{\Gamma \vdash K = K'}{\Gamma \vdash K' = K''} \]

\[ \frac{\Gamma \vdash k : K}{\Gamma \vdash k = k : K} \quad \frac{\Gamma \vdash k = k' : K}{\Gamma \vdash k' = k' : K} \]

Equality typing rules

\[ \frac{\Gamma \vdash k : K \quad \Gamma \vdash k = k : K'}{\Gamma \vdash k : K'} \quad \frac{\Gamma \vdash k = k' : K \quad \Gamma \vdash k' = k' : K'}{\Gamma \vdash k = k'' : K} \]

Substitution rules

\[ \frac{\Gamma, x : K, \Gamma' \vdash \text{valid} \quad \Gamma \vdash k : K}{\Gamma, [k/x] \Gamma' \vdash \text{valid}} \quad \frac{\Gamma, x : K, \Gamma' \vdash \text{kind} \quad \Gamma \vdash k = k' : K}{\Gamma, [k/x] \Gamma' \vdash [k/x] \text{K' kind}} \]

\[ \frac{\Gamma, x : K, \Gamma' \vdash k : K'}{\Gamma, [k/x] \Gamma' \vdash [k/x] \text{K' kind}} \quad \frac{\Gamma, x : K, \Gamma' \vdash k = k' : K'}{\Gamma, [k_1/x] \Gamma' \vdash [k_1/x] \text{K' kind} = [k_2/x] \text{K' kind}} \]

\[ \frac{\Gamma, x : K, \Gamma' \vdash K' = K''}{\Gamma, [k/x] \Gamma' \vdash [k/x] \text{K' kind} = [k/x] \text{K'' kind}} \quad \frac{\Gamma, x : K, \Gamma' \vdash k = k' : K'}{\Gamma, [k/x] \Gamma' \vdash [k/x] \text{K' kind} = [k/x] \text{K'' kind}} \]

The kind Type

\[ \Gamma \vdash \text{valid} \quad \frac{\Gamma \vdash A : \text{Type}}{\Gamma \vdash \text{El}(A) \quad \text{kind}} \quad \frac{\Gamma \vdash A = B : \text{Type}}{\Gamma \vdash \text{El}(A) = \text{El}(B) \quad \text{kind}} \]

Dependent product kinds

\[ \frac{\Gamma \vdash K \quad \text{kind}}{\Gamma \vdash (x : K) \text{K' kind}} \quad \frac{\Gamma \vdash K_1 = K_2 \quad \Gamma, x : K_1 \vdash k_1 = K_2}{\Gamma \vdash (x : K_1) K'_1 = (x : K_2) K'_2} \]

\[ \frac{\Gamma \vdash x : K \quad \Gamma \vdash k : K'}{\Gamma \vdash [x : K] k : (x : K) \text{K' kind}} \quad \frac{\Gamma \vdash K_1 = K_2 \quad \Gamma, x : K_1 \vdash k_1 = K_2}{\Gamma \vdash [x : K_1] k_1 = [x : K_2] k_2 : (x : K_1) K'} \]

\[ \frac{\Gamma \vdash f : (x : K) K' \quad \Gamma \vdash k : K}{\Gamma \vdash f(k) : [k/x] K'} \quad \frac{\Gamma \vdash f : (x : K) K' \quad \Gamma \vdash k_1 = K_2}{\Gamma \vdash f(k_1) = f(k_2) : [k_1/x] K'} \]

\[ \frac{\Gamma \vdash x : K \quad \Gamma \vdash k' : K' \quad \Gamma \vdash k : K}{\Gamma \vdash (x : K) [k'] k = [k/x] k' : [k/x] \text{K'' kind}} \quad \frac{\Gamma \vdash f : (x : K) K' \quad x \notin \text{FV}(f)}{\Gamma \vdash [x : K] f(x) = f : (x : K) \text{K'' kind}} \]

Figure 4.3: Inference Rules of Logical Framework (indexed with Roman numerals)
4.3 Completeness and Adequacy

The TOS we have studied is sound and complete w.r.t. the type system IDRT of dependent record types. In the informal introduction to TOS in Section 3.1, we have over-simplified the situation. In fact, what we shall do is to show that completeness holds for a simpler system $IDRT^-$ (with judgements of the form $\Gamma \vdash J$), which is obtained from IDRT by removing the seven substitution rules of LF in Figure 2.2. Therefore, the soundness and completeness may be pictured as follows:

\[
\begin{array}{ccc}
\vdash & \vdash \\
\text{Soundness} & \text{Completeness}
\end{array}
\]

In this Section 4.3, we will first show the completeness proof, and the soundness proof will follow in the next section.

4.3.1 Completeness

We have proved the completeness and soundness of $IDRT$ with respect to its typed operational semantics that we have defined. Proving Completeness Theorem we have worked on a weaker system than the original $IDRT$: the $IDRT^-$ which is $IDRT$ without structural rules, namely without the substitution rules in the LF.

**Theorem 4.3.1 (Completeness of TOS with respect to $IDRT^-$)**  

1. If $\Gamma \models \text{ok}$ then $\vdash \Gamma$ valid;  
2. If $\Gamma \models A \rightarrow B$ then $\Gamma \vdash \text{A kind}$ and $\Gamma \vdash A = B$;  
3. If $\Gamma \models M \rightarrow N \rightarrow P : A$ then $\Gamma \vdash M : A$, $\Gamma \vdash M = N : A$, $\Gamma \vdash M = P : A$ and $\Gamma \vdash A = A$.  

**Proof.** By simultaneous induction on derivations and examining each case of TOS inference rules in Figure 3.2 and Figure 3.3. Consider the following cases, with reference to DRT rules and LF rules in Figure 4.2 and Figure 4.3:

---

$\vdash$ is $\vdash$ without considering structural rules in $IDRT$, namely the “substitution rules” in LF.

For clarity in the following proof we will mark these four conditions as (a), (b), (c) and (d).
4.3. Completeness and Adequacy

- EMP, from LF first rule (i) we’ve got $\vdash ()$ valid.

- WEAK, by induction hypothesis $\Gamma \vdash A$ kind, by condition $x \notin dom\{\Gamma\}$ and from LF second rule (ii) we have $\vdash \Gamma, x:A$ valid. Again by induction hypothesis, $\vdash \Gamma, x:A = \Delta, x:B$.

- TYPE, by induction hypothesis we have $\vdash \Gamma$ valid, so $\Gamma \vdash Type$ kind by LF first kind rule (xix), and $\Gamma \vdash Type = Type$ by LF equality rule (iv).

- EL, by induction hypothesis $\Gamma \vdash M = P : Type$, thus $\Gamma \vdash El(M) = El(P)$ by LF third kind rule. Also $\Gamma \vdash M : Type$ by induction hypothesis, so $\Gamma \vdash El(M)$ kind by LF second kind rule (xx).

- PI, by induction hypothesis on first premise, $\Gamma \vdash A_1$ kind and $\Gamma \vdash A_1 = B_1$; on second premise, $\Gamma, x:A_1 \vdash A_2$ kind and $\Gamma, x:A_1 \vdash A_2 = B_2$. By LF first dependent product kinds rule (xxii) $\Gamma \vdash (x:A_1)A_2$ kind, by LF second dependent product kinds rule (xxiii) $\Gamma \vdash (x:A_1)A_2 = (x:B_1)B_2$.

- VAR, by induction hypothesis $\Gamma_0, x:A, \Gamma_1 \vdash A$ kind and $\Gamma_0, x:A, \Gamma_1 \vdash A = B$. By Subcontext (lemma 4.1.1) $\Gamma_0, x:A, \Gamma_1 \models ok$, so by induction hypothesis $\vdash \Gamma_0, x:A, \Gamma_1$ valid. By LF term variable rule (iii) $\Gamma_0, x:A, \Gamma_1 \vdash x : A$, and by LF term equality rule (x) $\Gamma_0, x:A, \Gamma_1 \vdash x : B^{(a)}$. Furthermore by LF term reflexivity rule (vii) $\Gamma_0, x:A, \Gamma_1 \vdash x = x : B^{(b,c)}$. Finally by LF kind symmetry and transitivity rules (v, vi) $\Gamma_0, x:A, \Gamma_1 \vdash B = B^{(d)}$.

- LAM, by induction hypothesis on first premise, $\Gamma \vdash A_1$ kind and $\Gamma \vdash A_1 = B_1$; on second premise, $\Gamma, x:A_1 \vdash M_0 : B_2$ and $\Gamma, x:A_1 \vdash M_0 = P_0 : B_2$ and $\Gamma, x:A_1 \vdash B_2 = B_2$. Thus by LF third dependent product kinds rule (xxiv) $\Gamma \vdash [x:A_1]M_0 : (x:A_1)B_2$, and by LF second dependent product kinds rule (xxiii) $\Gamma \vdash (x:A_1)B_2 = (x:B_1)B_2$, so by LF term equality rule (x) $\Gamma \vdash [x:A_1]M_0 : (x:B_1)B_2^{(a)}$. By LF kind symmetry and transitivity rules (v, vi) $\Gamma \vdash (x:B_1)B_2 = (x:B_1)B_2^{(d)}$. Also, by LF term reflexivity rule (vii) $\Gamma \vdash [x:A_1]M_0 = [x:A_1]M_0 : (x:A_1)B_2$ thus by LF term equality rule (xi) $\Gamma \vdash [x:A_1]M_0 = [x:A_1]M_0 : (x:B_1)B_2^{(b)}$. Similarly by LF
4.3. Completeness and Adequacy

fourth dependent product kinds rule (xxv) \( \Gamma \vdash [x:A_1]M_0 = [x:B_1]P_0 : (x:A_1)B_2 \),
again by LF term equality rule (xi) \( \Gamma \vdash [x:A_1]M_0 = [x:B_1]P_0 : (x:B_1)B_2 \).

- **ETA**, by induction hypothesis on first premise, \( \Gamma \vdash A_1 \) kind and \( \Gamma \vdash A_1 = B_1 \);
on second premise, \( \Gamma, x:A_1 \vdash M_0 : B_2 \) and \( \Gamma, x:A_1 \vdash M_0 = P(x) : B_2 \) and \( \Gamma, x:A_1 \vdash B_2 = B_2 \); on third premise, \( \Gamma \vdash P : (x:B_1)B_2 \) and \( \Gamma \vdash (x:B_1)B_2 = (x:B_1)B_2 \). It’s similar to prove \(^{(a,b,d)}\) as the case **LAM**, furthermore \( \Gamma \vdash [x:B_1]P(x) = P : (x:B_1)B_2 \) by LF \( \eta \)-rule (xxix), so by LF term transitivity rule (ix) \( \Gamma \vdash [x:A_1]M_0 = P : (x:B_1)B_2 \).

- **BASE**, by induction hypothesis on first premise, \( \Gamma \vdash M_1 : (x:B_1)B_2 \) and \( \Gamma \vdash M_1 = N_1 : (x:B_1)B_2 \) and \( \Gamma \vdash M_1 = P_1 : (x:B_1)B_2 \) and \( \Gamma \vdash (x:B_1)B_2 = (x:B_1)B_2 \); on second premise, \( \Gamma \vdash M_2 : B_1 \) and \( \Gamma \vdash M_2 = N_2 : B_1 \) and \( \Gamma \vdash M_2 = P_2 : B_1 \) and \( \Gamma \vdash B_1 = B_1 \); on third premise, \( \Gamma \vdash [M_2/x]B_2 \) kind and \( \Gamma \vdash [M_2/x]B_2 = C \). By LF application rule (xxviii) \( \Gamma \vdash M_1M_2 : [M_2/x]B_2 \), by term transitivity (x) \( \Gamma \vdash M_1M_2 : C \). By LF application equality rule (xxvii) \( \Gamma \vdash M_1M_2 = N_1M_2 : C \) using term transitivity and reflexivity (vii, ix). Similarly \( \Gamma \vdash M_1M_2 = P_1P_2 : C \). By LF kind symmetry and transitivity (v, vi) \( \Gamma \vdash C = C \).

- **BETA**, by induction hypothesis on first premise, \( \Gamma \vdash M_1 : (x:B_1)B_2 \) and \( \Gamma \vdash M_1 = [x:A_1]N_0 : (x:B_1)B_2 \); on second premise, \( \Gamma \vdash M_2 : B_1 \); on third premise, \( \Gamma \vdash [M_2/x]N_0 = P : C \) and \( \Gamma \vdash [M_2/x]N_0 = Q : C \); on fourth premise, \( \Gamma \vdash [M_2/x]B_2 = C \). From the fact that \( \Gamma \models M_1 \rightarrow_w [x:A_1]N_0 : (x:B_1)B_2 \) we know \( \Gamma, x:A_1 \models N_0 \rightarrow_n P_0 : B_2 \) and \( \Gamma \models A_1 \rightarrow B_1 \) as subderivations (check the cases inductively of **LAM**, **ETA** and **BETA** to get this lemma proven); so by further induction hypothesis we have \( \Gamma \vdash A_1 = B_1 \) and \( \Gamma, x:A_1 \vdash N_0 : B_2 \).

Now we start to prove the four conditions: Similar to the case **BASE** we have \( \Gamma \vdash M_1M_2 : C \). By LF application equality rule (xxvii) \( \Gamma \vdash M_1M_2 = ([x:A_1]N_0)M_2 : [M_2/x]B_2 \), by term equality (x) \( \Gamma \vdash M_2 : A_1 \), so by LF \( \beta \)-rule (xxviii) \( \Gamma \vdash ([x:A_1]N_0)M_2 = [M_2/x]N_0 : [M_2/x]B_2 \). Hence by term transitivity (ix) \( \Gamma \vdash M_1M_2 = [M_2/x]N_0 : [M_2/x]B_2 \), by kind transitivity (xi) we have \( \Gamma \vdash M_1M_2 = [M_2/x]N_0 : C \). From this \(^{(a)}\) and using term transitivity (ix) we get \( \Gamma \vdash M_1M_2 = P : C \) and \( \Gamma \vdash M_1M_2 = Q : C \). By LF kind symmetry
and transitivity \((v, vi)\) \(\Gamma \vdash C = C^{(d)}\).

- \(\text{RTYPE}\), by induction hypothesis \(\vdash \Gamma \text{ valid}\), by \(\text{DRT}\) first rule (I) \(\Gamma \vdash \text{RTYPE} \; \text{kind}\) and \(\Gamma \vdash \text{RTYPE} = \text{RTYPE}\) by kind equality (iv). Similar for \(\text{RTYPE}[L]\).

- \(\text{EMP}_{\text{RCDT}}\) and \(\text{EMP}_{\text{RCD}}\), similar to above case using \(\text{DRT}\) first formation rule (VI) (or first introduction rule (VIII)) and \(\text{DRT}\) congruence rules (XVI, XVIII).

- \(\text{RCDT}\), by induction hypothesis on first premise, \(\Gamma \vdash R : \text{RType}[L]\) and \(\Gamma \vdash R = P : \text{RType}[L]\); on second premise, \(\Gamma \vdash A : (P)\text{Type}\). By LF dependent kind equality rule (xxiii) we have \(\Gamma \vdash A : (R)\text{Type}\), and from the condition \(l \notin L\), by \(\text{DRT}\) formation rule (VII) \(\Gamma \vdash \langle R, l : A \rangle : \text{RType}[L \cup \{l\}]^{(a)}\). Similar to prove \((b,c)\) using \(\text{DRT}\) congruence rule (XVII). Also by LF first general equality rule (iv) and \((a)\) \(\Gamma \vdash \text{RType}[L \cup \{l\}] = \text{RType}[L \cup \{l\}]^{(d)}\).

- \(\text{RCD}\), by induction hypothesis on first premise, \(\Gamma \vdash \langle R, l : A \rangle : \text{RType}\) and \(\Gamma \vdash \langle R, l : A \rangle = \langle P, l : B \rangle : \text{RType}\); on second premise, \(\Gamma \vdash r : P\) and \(\Gamma \vdash r = p : P\) and \(\Gamma \vdash \text{El}(P) = \text{El}(P)\); on third premise, \(\Gamma \vdash A(r) : \text{Type}\) and \(\Gamma \vdash A(r) = C : \text{Type}\) and \(\Gamma \vdash \text{Type} = \text{Type}\); on fourth premise, \(\Gamma \vdash a : C\) and \(\Gamma \vdash a = b : C\) and \(\Gamma \vdash \text{El}(C) = \text{El}(C)\). From the fact that \(\Gamma \models \langle R, l : A \rangle \to_n \langle P, l : B \rangle : \text{RType}\) we know \(\Gamma \models R \to_n P : \text{RType}[L]\) for some \(L\) and \(\Gamma \models A \to_n B : (P)\text{Type}\) as subderivations; so by further induction hypothesis we have \(\Gamma \vdash R : \text{RType}[L]\) and \(\Gamma \vdash R = P : \text{RType}[L]\) and \(\Gamma \vdash A = B : (P)\text{Type}\). Now to prove the four conditions: from \(\text{DRT}\) introduction rule for records (IX) we know \(\Gamma \vdash \langle r, l : a : A \rangle : \text{El}(\langle P, l : A \rangle)\) and by LF kind transitivity (vi) and \(\text{DRT}\) record congruence (XIX) we have \(\Gamma \vdash \langle r, l : a : A \rangle : \langle P, l : B \rangle^{(a)}\). The conditions \((b,c)\) are achieved by using \(\text{DRT}\) congruence rules again and by LF kind rules and \(\text{DRT}\) record type congruence to get \((d)\).

- \(\text{BASE}_{\text{RESTR}}\), by induction hypothesis, \(\Gamma \vdash r : \langle P, l : B \rangle\) and \(\Gamma \vdash r = q : \langle P, l : B \rangle\) and \(\Gamma \vdash r = p : \langle P, l : B \rangle\) and \(\Gamma \vdash \text{El}(\langle P, l : B \rangle) = \text{El}(\langle P, l : B \rangle)\). By \(\text{DRT}\) first elimination rule (X) \(\Gamma \vdash [r] : P^{(a)}\). To prove \((b,c)\) using the last two \(\text{DRT}\) record congruence (XX, XXI), and using kind equality and then record type
congruence (XVII) on $\Gamma \vdash \langle P, l : B \rangle = \langle P, l : B \rangle : \text{RT}_\text{ype}$ to get \((d)\). Same for $\text{BASE}_{\text{FLDSEL}}$.

\(-\text{RESTR}\), by induction hypothesis on first premise, $\Gamma \vdash r : \langle P, l : B \rangle$ and $\Gamma \vdash r = \langle p, l = b : B \rangle : \langle P, l : B \rangle$ and $\Gamma \vdash \langle P, l : B \rangle = \langle P, l : B \rangle$; on second premise, $\Gamma \vdash p : P$ and $\Gamma \vdash p = s : P$ and $\Gamma \vdash p = t : P$ and $\Gamma \vdash \text{El}(P) = \text{El}(P)$ \((d)\). By DRT first elimination rule (X) $\Gamma \vdash [r] : P$ \((a)\). Since we know $\Gamma \vdash [\langle p, l = b : B \rangle] = p : P$ by computation rule (XIII), by congruence rule (XX) and term transitivity (ix) we get $\Gamma \vdash [r] = s : P$ \((b)\), similarly get \((c)\). Same pattern of proof for $\text{FLDSEL}$ and $\text{FLDSL}'$.

\textbf{Theorem 4.3.2 (Completeness of TOS with respect to IDRT)}

1. If $\Gamma \vdash \text{ok}$ then $\Gamma \vdash \Gamma$ valid;
2. If $\Gamma \vdash A \rightarrow B$ then $\Gamma \vdash A$ kind and $\Gamma \vdash A = B$;
3. If $\Gamma \vdash M \rightarrow N \rightarrow P : A$ then $\Gamma \vdash M : A$, $\Gamma \vdash M = N : A$, $\Gamma \vdash M = P : A$ and $\Gamma \vdash A = A$.

\textbf{Proof.} By Theorem 4.3.1 and the inclusion of $\text{IDRT}^-$ in $\text{IDRT}$, i.e., the former is a subsystem of the latter. \qed
4.3.2 Adequacy and Determinacy

Now we deal with the two important aspects on the metatheory of the typed operational semantics: adequacy lemma shows that the notion of computation captured in TOS is adequate with respect to the untyped reduction and the associated notions of normal forms; and determinacy lemma, or called unique normal form, claims that these normal form and weak-head normal form are unique.

In [Gog94], Goguen has shown an adequacy lemma of TOS for LF with respect to the untyped $\beta\eta$-reduction (which could be comprehended as $\to^*_{\beta\eta}$) over terms in LF, as follows:

**Lemma 4.3.3 (Adequacy of TOS$_{LF}$ for Untyped $\beta\eta$-Reduction)**

(1) If $\Gamma \models A \to C$ then there exists $B$ such that $A \to^*_\beta B \to^*_\eta C$;

(2) If $\Gamma \models M \to N \to P : A$ then there exists $N'$ such that $M \to^*_\beta N \to^*_\eta N' \to^*_\eta P$.

**Proof.** By induction on derivations. Consider the following cases:

- **TYPE**, we have $Type \to^*_\eta Type \to^*_\eta Type$.
- **EL**, by induction hypothesis, we know that there exists $N'$ such that $M \to^*_\beta N' \to^*_\eta P$, thus $El(M) \to^*_\beta El(N') \to^*_\eta El(P)$.
- **PI**, by induction hypothesis, we know that there exist $A'_1, A'_2$ such that $A_1 \to^*_\beta A'_1 \to^*_\eta B_1$ and $A_2 \to^*_\beta A'_2 \to^*_\eta B_2$, thus $(x:A_1)A_2 \to^*_\beta (x:A'_1)A'_2 \to^*_\eta (x:B_1)B_2$.
- **VAR**, we have $x \to^*_\beta x \to^*_\beta x \to^*_\eta x$.
- **ETA**, by induction hypothesis, we know that there exist $B'_1, N'$ such that $A_1 \to^*_\beta B'_1 \to^*_\eta B_1$ and $M_0 \to^*_\beta N \to^*_\eta N' \to^*_\eta P(x)$, and by (FV) condition $x \notin FV(P)$, thus $[x:A_1]M_0 \to^*_\beta [x:A_1]M_0 \to^*_\beta [x:B'_1]N' \to^*_\eta [x:B_1]P(x) \to^*_\eta P$.
- **LAM**, by induction hypothesis, we know that there exist $B'_1, N'_0$ such that $A_1 \to^*_\beta B'_1 \to^*_\eta B_1$ and $M_0 \to^*_\beta N_0 \to^*_\beta N'_0 \to^*_\eta P_0$, thus $[x:A_1]M_0 \to^*_\beta [x:A_1]M_0 \to^*_\beta [x:B'_1]N_0 \to^*_\eta [x:B_1]P_0$ ($[x:B_1]P_0$ not an $\eta$-redex by the rule).
- **BASE**, by induction hypothesis, we know that there exist $N'_1, N'_2$ such that $M_1 \to^*_\beta N_1 \to^*_\beta N'_1 \to^*_\eta P_1$ and $M_2 \to^*_\beta N_2 \to^*_\beta N'_2 \to^*_\eta P_2$, thus $M_1(M_2) \to^*_\beta N_1(M_2) \to^*_\beta N'_1(N'_2) \to^*_\eta P_1(P_2)$.
- **BETA**, by induction hypothesis, we know that $M_1 \to^*_\beta [x:A_1]N_0$ and there exists $P'$ such that $[M_2/x]N_0 \to^*_\beta P \to^*_\beta P' \to^*_\eta Q$, thus $M_1(M_2) \to^*_\beta ([x:A_1]N_0)(M_2) \to^*_\beta [M_2/x]N_0 \to^*_\beta P \to^*_\beta P' \to^*_\eta Q$. \qed
In our work, we have also two adequacy lemmas, which show the relation from well-formed judgements in the \( TOS \), to untyped reduction (which could be comprehended as \( \rightarrow_{\beta R}^* \) over terms in \( IDRT \), and to the WHN/NFs in the \( IDRT \).

**Lemma 4.3.4 (Adequacy for Untyped Reduction)**

1. If \( \Gamma \models A \rightarrow C \) then there exists \( B \) such that \( A \rightarrow_{\beta R}^* B \rightarrow_{\eta}^* C \);
2. If \( \Gamma \models M \rightarrow N \rightarrow P : A \), then there exists \( N' \) such that \( M \rightarrow_{\beta R}^* N \rightarrow_{\beta R}^* N' \rightarrow_{\eta}^* P \).

**Proof.** By induction on derivations. Similar to the above Theorem 4.3.3, with considerations to record reductions in Definition 4.2.2. \( \square \)

**Lemma 4.3.5 (Adequacy for Normal Forms and WHNFs)**

1. If \( \models \Gamma \rightarrow \Delta \) then \( \Delta \) is normal;
2. If \( \Gamma \models A \rightarrow B \) then \( B \) is normal;
3. If \( \Gamma \models M \rightarrow N \rightarrow P : A \) then \( N \) is weak-head normal and \( P \) and \( A \) are normal.

**Proof.** By induction on derivations. Consider the following cases:

- **TYPE**, by definition.
- **PI**, by induction hypothesis, \( B_1, B_2 \) are normal, thus \( (x:B_1)B_2 \) is normal.
- **EL**, by induction hypothesis, \( P \) is normal, thus \( El(P) \) is normal.
- **VAR**, \( x \) is weak-head normal and normal by definition.
- **ETA**, \([x:A_1]M_0 \) is weak-head normal by definition, and \( P \) is normal by induction hypothesis.
- **LAM**, \([x:A_1]M_0 \) is weak-head normal by definition, and \([x:B_1]P_0 \) is not an \( \eta \)-redex by the rule, thus it is normal by definition (\( B_1, P_0 \) are normal by induction hypothesis).
- **BASE**, by induction hypothesis, \( N_1 \) is weak-head normal, thus \( N_1(M_2) \) is weak-head normal because \( N_1 \) is not an abstraction by the rule; also by induction hypothesis, \( P_1, P_2 \) are normal, and since \( N_1 \) is not an abstraction we have that \( P_1 \) is not an abstraction, thus \( P_1(P_2) \) is normal.
- **BETA**, by induction hypothesis \( P \) is weak-head normal and \( Q \) is normal.

\footnote{This lemma could also be called the Postponement Lemma, because basically it says that the \( \eta \)-reductions (here for \( LF \) terms only) could be postponed until the last.}
4.3. Completeness and Adequacy

- \textit{RTYPE} and \textit{RTYPE}[L], by definition.
- \textit{RCDT}, \langle R, l : A \rangle \text{ is weak-head normal by definition, and } \langle P, l : B \rangle \text{ is normal by definition } (P, B \text{ are normal by induction hypothesis}).
- \textit{RCD}, \langle r, l = a : A \rangle \text{ is weak-head normal by definition, and } \langle p, l = b : B \rangle \text{ is not an } \eta\text{-redex by the rule and thus is normal } (p, b, B \text{ are normal by induction hypothesis}).
- \textit{RESTR}, \textit{FLDSEL}, and \textit{FLDSL}', by induction hypothesis, \(s, c\) are weak-head normal and \(t, d\) are normal.
- \textit{BASE}_{\text{RESTR}}, by induction hypothesis, \(q\) is weak-head normal and not a pair-record, thus \([q]\) is weak-head normal by definition; similarly \([p]\) is normal.
- \textit{BASE}_{\text{FLDSEL}}, similar to the case \textit{BASE}_{\text{RESTR}}. \(\Box\)

\textbf{Observation} We have that \textit{RType}[L] \equiv \textit{RType}[L'] if \(L = L'\).

\textbf{Lemma 4.3.6 (Determinacy (Unique Normal Form))}

1. If \(\vdash \Gamma \rightarrow \Delta, \vdash \Gamma \rightarrow \Phi\), then \(\Delta \equiv \Phi\);
2. If \(\Gamma \mid A \rightarrow B, \Gamma \mid A \rightarrow C\), then \(B \equiv C\);
3. If \(\Gamma \mid M \rightarrow N \rightarrow P : B, \Gamma \mid M \rightarrow Q \rightarrow R : C\), then \(N \equiv Q, P \equiv R, B \equiv C\).

\textit{Proof}. By simultaneous induction on derivations. We consider the following cases:
- \textit{EMP}, \textit{RTYPE} and \textit{RTYPE}[L], trivial.
- \textit{WEAK} and \textit{VAR}, simply by induction hypothesis.
- \textit{PI}. Suppose that we have both \(\Gamma \mid (x:A_1)A_2 \rightarrow (x:B_1)B_2\) and \(\Gamma \mid (x:A_1)A_2 \rightarrow (y:C_1)C_2\). Then by inversion and Renaming (lemma 4.1.3) we know that \(\Gamma \mid A_1 \rightarrow B_1 ; \Gamma, x:A_1 \mid A_2 \rightarrow B_2\) and \(\Gamma \mid A_1 \rightarrow C_1 ; \Gamma, y:A_1 \mid [y/x]A_2 \rightarrow C_2\). By induction hypothesis we have \(B_1 \equiv C_1\), and using Renaming we have \(\Gamma, x:A_1 \mid A_2 \rightarrow [x/y]C_2\) and again by induction hypothesis \(B_2 \equiv [x/y]C_2\), and so \((x:B_1)B_2 \equiv (y:C_1)C_2\).
- \textit{LAM} and \textit{ETA}, using Renaming similarly as \textit{PI}.
- \textit{RCDT} and \textit{RCD}, \textit{RESTR} and \textit{FLDSEL}, all simply by induction hypothesis. For example, for \textit{RCDT}, suppose that we have both \(\Gamma \mid (R, l : A) \rightarrow_n (P, l : B) : \textit{RType}[L]\) and \(\Gamma \mid (R, l : A) \rightarrow_n (P', l : B') : \textit{RType}[L']\) for some \(L'\), by induction hypothesis, \(L \equiv L'\), thus \textit{RType}[L] \equiv \textit{RType}[L']\) by convention; by inversion and induction hypothesis then we know \(P \equiv P'\) and \(B \equiv B'\), thus \(\langle P, l : B \rangle \equiv \langle P', l : B' \rangle\). \(\Box\)
4.4 Soundness

The Soundness Theorem is much harder to prove, and we have to consider all the inference rules of IDRT including the structural rules.

Lemma 4.4.1 Suppose $\Gamma \vdash N \rightarrow Q \rightarrow S : A'$ and $\Gamma \vdash A \rightarrow A'$. Then:

1. Substitution is admissible:
   (a) If $\vdash \Gamma, x : A, \Delta \rightarrow \Phi$ then there is a $\Psi$ such that $\vdash \Gamma, [N/x]\Delta \rightarrow \Psi$.
   (b) If $\Gamma, x : A, \Delta \vdash B \rightarrow C$ then there is a $D$ such that $\Gamma, [N/x]\Delta \vdash [N/x]B \rightarrow D$.
   (c) If $\Gamma, x : A, \Delta \vdash M \rightarrow P \rightarrow R : B$ then there are $T, U$ and $C$ such that $\Gamma, [N/x]\Delta \vdash [N/x]M \rightarrow T \rightarrow U : C$, $\Gamma, [N/x]\Delta \vdash [N/x]P \rightarrow T \rightarrow U : C$ and $\Gamma, [N/x]\Delta \vdash [N/x]B \rightarrow C$.

2. Application is admissible: if $\Gamma \vdash M \rightarrow P \rightarrow R : \Pi x : A'. B$ then there are $T, U$ and $C$ such that $\Gamma \vdash M(N) \rightarrow T \rightarrow U : C$ and $\Gamma \vdash [N/x]B \rightarrow C$.

Proof. The proof followed by simultaneous induction on the length of $A'$, i.e. $|A'|$.

Theorem 4.4.2 (Soundness of TOS with respect to IDRT)

1. If $\Gamma \vdash \text{ok}$ then there exists $\Delta$ such that $\vdash \Gamma \rightarrow \Delta$;
2. If $\Gamma \vdash A$ kind then there exists $B$ such that $\Gamma \vdash A \rightarrow B$;
3. If $\Gamma \vdash M : A$ then there exist $P, B$ such that $\Gamma \vdash A \rightarrow B$ and $\Gamma \vdash M \rightarrow P : B$;
3'. If $\Gamma \vdash M = N : A$ then there exist $P, B$ such that $\Gamma \vdash A \rightarrow B$, and $\Gamma \vdash M \rightarrow P : B$, $\Gamma \vdash N \rightarrow P : B$.

Proof. By induction on derivations. We shall examine each case of LF and DRT inference rules in Figure 4.2 and Figure 4.3, and here we consider all the structural rules:

- (xx). By induction hypothesis there exist $P, B$ such that $\Gamma \vdash M \rightarrow P : B$ and $\Gamma \vdash \text{Type} \rightarrow B$. By inversion of TYPE rule we know $\text{Type} \equiv B$, so using the $\text{EL}_{\text{TOS}}$ rule we have $\Gamma \vdash \text{El}(M) \rightarrow \text{El}(P)$.

5We note in this proof $\text{EL}_{\text{TOS}}$ to stand for the rule $\text{EL}$ in the TOS, in order to achieve better clearness. The other rule names of TOS similarly apply.
– (iii). By induction hypothesis $\Gamma_0, x:A, \Gamma_1 \vdash ok$. By Context Validity (Lemma 4.1.1) $\Gamma_0, x:A \vdash ok$. By inversion of $\text{WEAK}_{TOS}$ rule there exists $B$ such that $\Gamma_0 \vdash A \rightarrow B$. By Weakening (Lemma 4.1.4) $\Gamma_0, x:A, \Gamma_1 \vdash A \rightarrow B$. So using $\text{VAR}_{TOS}$ we have $\Gamma_0, x:A, \Gamma_1 \vdash x \rightarrow x \rightarrow x : B$.

– (xxiv). By induction hypothesis there exist $P_0, B_2$ such that $\Gamma, x:A_1 \vdash M_0 \rightarrow_n P_0 : B_2$ and $\Gamma, x:A_1 \vdash A_2 \rightarrow B_2$. By Subcontext and inversion of $\text{WEAK}_{TOS}$ rule there exists $B_1$ such that $\Gamma \vdash A_1 \rightarrow B_1$. Using $P_{I_{TOS}}$ we have $\Gamma \vdash (x:A_1)A_2 \rightarrow (x:B_1)B_2$, there are two subcases here: First, $P_0 \equiv P(x)$ with $x \notin FV(P)$, using $\text{ETA}_{TOS}$ we have $\Gamma \vdash [x:A_1]M_0 \rightarrow_n P : (x:B_1)B_2$; Second, $[x:B_1]P_0$ is not an $\eta$-redex, using $LAM_{TOS}$ we have $\Gamma \vdash [x:A_1]M_0 \rightarrow_n [x:B_1]P_0 : (x:B_1)B_2$.

– (xxvii). By induction hypothesis there exist $P_1, P_2, B_1'$ such that $\Gamma \vdash M_1 \rightarrow_n P_1 : B$, $\Gamma \vdash N_1 \rightarrow_n P_1 : B$ and $\Gamma \vdash (x:A_1)A_2 \rightarrow B$; also $\Gamma \vdash M_2 \rightarrow_n P_2 : B_1'$, $\Gamma \vdash N_2 \rightarrow_n P_2 : B_1'$ and $\Gamma \vdash A_1 \rightarrow B_1'$. By inversion of $P_{I_{TOS}}$ rule and the fact that $\Gamma \vdash (x:A_1)A_2 \rightarrow B$ we know $B \equiv (x:B_1)B_2$, $\Gamma \vdash A_1 \rightarrow B_1$ and $\Gamma, x:A_1 \vdash A_2 \rightarrow B_2$. By Determinacy (Lemma 4.3.6) and (1),(2) we have $B_1 \equiv B_1'$. From Lemma 4.4.1 Case 2 we know $\Gamma \vdash M_1(M_2) \rightarrow_n U : C$, $\Gamma \vdash [M_2/x]B_2 \rightarrow C$; $\Gamma \vdash N_1(N_2) \rightarrow_n U' : C'$, $\Gamma \vdash [N_2/x]B_2 \rightarrow C'$. By Adequacy (Lemma 4.3.5) and Subject Reduction (Lemma 4.5.4) $\Gamma \vdash P_1(P_2) \rightarrow_n U : C$, $\Gamma \vdash [P_2/x]B_2 \rightarrow C$ and $\Gamma \vdash P_1(P_2) \rightarrow_n U' : C'$, $\Gamma \vdash [P_2/x]B_2 \rightarrow C'$. Hence by Determinacy and (3),(4) we have $U \equiv U'$, $C \equiv C'$. Finally, we know $\Gamma, x:A_1 \vdash A_2 \rightarrow B_2$, so from Lemma 4.4.1 Case 1 $\Gamma \vdash [M_2/x]A_2 \rightarrow C''$. By Adequacy and Subject Reduction $\Gamma \vdash [M_2/x]B_2 \rightarrow C''$. By Determinacy and (5),(6) $C \equiv C''$.

– Substitution rules. We consider the classical substitution rule (xvii), where other structural rules are similar. By induction hypothesis $\Gamma_0, x:A, \Gamma_1 \vdash M \rightarrow_n P : D$ with $\Gamma_0, x:A, \Gamma_1 \vdash B \rightarrow D$; and $\Gamma_0 \vdash N \rightarrow_n Q : C$ with $\Gamma_0 \vdash A \rightarrow C$. By Lemma 4.4.1 Case 1 we know $\Gamma_0, [N/x]\Gamma_1 \vdash [N/x]M \rightarrow_n R : E$ and $\Gamma_0, [N/x]\Gamma_1 \vdash [N/x]D \rightarrow E$ for some $R$ and $E$; and $\Gamma_0, [N/x]\Gamma_1 \vdash [N/x]B \rightarrow F$ for some $F$. By Adequacy and Subject Reduction $\Gamma_0, [N/x]\Gamma_1 \vdash [N/x]D \rightarrow F$. By Determinacy and (7),(8) $E \equiv F$. 
4.4. Soundness

- Kinding rules. We show for $\text{RType}$ (I): By induction hypothesis $\Gamma \models \text{ok}$; using $\text{RTYPE}_{\text{TOS}}$ we have $\Gamma \models \text{RType} \rightarrow \text{RType}$.

- Formation rules for $\text{RType}[.]$ (VII). We show for the second one: By induction hypothesis there exist $P, B'$ such that $\Gamma \models \text{RType}[L] \rightarrow B'$ (so by inversion $B' \equiv \text{RType}[L]$), and $\Gamma \models R \rightarrow_n P : \text{RType}[L]$. Also by induction hypothesis there exist $B$ such that $\Gamma \models A \rightarrow B : (P)\text{Type}$. Together with the condition $l \notin L$, using $\text{RCDT}_{\text{TOS}}$ we have $\Gamma \models \langle R, l : A \rangle \rightarrow_n \langle P, l : B \rangle : \text{RType}[L \cup \{l\}]$.

- Introduction rules. We show for the second one (IX): By induction hypothesis there exist $P, B$ such that $\Gamma \models \langle R, l : A \rangle \rightarrow_n \langle P, l : B \rangle : \text{RType}$ (9); also that there exist $s, P'$ such that $\Gamma \models r \rightarrow_n s : P'$ and $\Gamma \models R \rightarrow_n P' : \text{RType}[L]$ for some $L$. By inversion of $\text{RCDT}_{\text{TOS}}$ and by Determinacy we know $P \equiv P'$. Also by induction hypothesis there exist $b, B'$ such that $\Gamma \models a \rightarrow_n b : B'$ and $\Gamma \models A(r) \rightarrow B'$ (10).

Here, consider two subcases when we construct the new normal form for the small record $\langle r, l = a : A \rangle$:

First subcase, $\langle s, l = b : B \rangle$ is not an $\eta$-redex. By inversion on (9), we have $\Gamma \models A \rightarrow_n B : (P)\text{Type}$ and $\Gamma \models r \rightarrow_n s : P$. Use the $\text{TOS}_{\text{LF}}$ rule for application, we have $\Gamma \models A(r) \rightarrow_n B(s) : \text{Type}$. Use the $\text{TOS}_{\text{LF}}$ rule for $\text{El}$, we have $\Gamma \models \text{El}(A(r)) \rightarrow \text{El}(B(s))$ (11). Compare (10) and (11), by Determinacy we have $\text{El}(B(s)) \equiv \text{El}(B')$. By inversion on $\text{LF}'$s $\text{El}$ rule we have $\Gamma \models B(s) \equiv B' : \text{Type}$. Apply induction hypothesis again and by the fact that $B'$ is normal (Adequacy for normal forms, Lemma 4.3.5), we have $\Gamma \models B(s) \rightarrow_n B' : \text{Type}$. Finally, with all these conditions got, we apply $\text{RCDT}_{\text{TOS}}$ rule and get $\Gamma \models \langle r, l = a : A \rangle \rightarrow_n \langle s, l = b : B \rangle : \langle P, l : B \rangle$.

Second subcase, $\langle s, l = b : B \rangle$ is an $\eta$-redex. This means there exists $r'$ such that $[r'] \equiv s$ and $r'.l \equiv b$. Similarly to the above subcase, by inversion on $\text{BASE}_{\text{RESTR}}$ and $\text{BASE}_{\text{FLDSEL}}$ rules of $\text{TOS}_{\text{DRT}}$, we have $\Gamma \models r' \rightarrow_n p : \langle P, l : B \rangle$ for some $p$.

- Elimination rules. We show for the third one (XII) (the first two could be easily deduced from inverting the $\text{BASE}_{\text{TOS}}$ rules): By induction hypothesis there exist $p, b, P, B$ such that $\Gamma \models r \rightarrow_n \langle p, l = b : B \rangle : \langle P, l : B \rangle$ and $\Gamma \models \langle R, l : A \rangle \rightarrow \langle P, l : B \rangle$; also that there exist $c$ such that $\Gamma \models [r]', l' \rightarrow_n c : C$ and $\Gamma \models B \rightarrow C$. There are
two subcases here, apply an inversion either on the the rule \textit{FLDSEL}_{TOS} or on the rule \textit{BASE}_{TOS}, we will have $\Gamma \models B(P) \to C : Type$. Together with the condition $l \neq l'$, using $FLDSL'_{TOS}$ we have $\Gamma \models r.l' \to_n c : C$.

- Computation rules. We show for the third one (XV): From the induction hypothesis and by inversion on the rule $RESTR_{TOS}$ and using the rule $FLDSL'_{TOS}$ it will be easily derived that $\Gamma \models r.l' \to_n s : C$ and $\Gamma \models [r].l' \to_n s : C$ and $\Gamma \models B \to C$ for some $s, C$.

- Congruence rules. We show for the one for $RType[L \cup \{l\}]$ (XVII): Using induction hypothesis and the rule $RCDT_{TOS}$, also by Determinacy, we will derive that $\Gamma \models \langle R, l : A \rangle \to \langle P, l : B \rangle : RType[L \cup \{l\}]$, $\Gamma \models \langle R', l : A' \rangle \to \langle P, l : B \rangle : RType[L \cup \{l\}]$ for some $P, B$ and $\Gamma \models RType[L \cup \{l\}] \to RType[L \cup \{l\}]$. $\square$
4.5 Strong Normalisation and Other Metatheories

The strong normalisation property of the TOS says that, if a term \( M \) is well-typed in the TOS (i.e., \( \Gamma \vdash M : A \)), then \( M \) is strongly normalizable. This result, together with the soundness theorem, will then be enough to show that the original type theory has the property of strong normalisation.

The strong normalisation property of the TOS by introducing a notion of parallel reduction and showing that it has the so-called Parallel Subject Reduction property.

We first define parallel reduction:

**Definition 4.5.1 (Parallel Reduction)** Parallel reduction \( \Rightarrow \) is defined as the least relation closed under the following rules of inference in Figure 4.4, and is extended in an obvious way to kinds and contexts.

\[
\begin{align*}
\text{VAR} & \quad x \Rightarrow x \\
\lambda & \quad \frac{A \Rightarrow A'}{M \Rightarrow M'} \quad \frac{[x:A]M \Rightarrow [x:A']M'}{[x:A]M \Rightarrow [x:A']M'} \\
\text{APP} & \quad \frac{M \Rightarrow M' \quad N \Rightarrow N'}{M(N) \Rightarrow M'(N')}
\end{align*}
\]

\[
\begin{align*}
\beta & \quad \frac{M \Rightarrow M' \quad N \Rightarrow N'}{} \\
& \quad \frac{M_0 \Rightarrow M'(x) \quad x \notin \text{FV}(M')}{[x:A]M_0 \Rightarrow M'}
\end{align*}
\]

\[
\begin{align*}
\text{RCDT}_{\text{EMP}} & \quad \langle \rangle \Rightarrow \langle \rangle \\
\text{RCDT} & \quad \frac{R \Rightarrow R' \quad A \Rightarrow A'}{\langle R, l : A \rangle \Rightarrow \langle R', l : A' \rangle}
\end{align*}
\]

\[
\begin{align*}
\text{RCD}_{\text{EMP}} & \quad \langle \rangle \Rightarrow \langle \rangle \\
\text{RCD} & \quad \frac{r \Rightarrow r' \quad a \Rightarrow a' \quad A \Rightarrow A'}{\langle r, l = a : A \rangle \Rightarrow \langle r', l = a' : A' \rangle}
\end{align*}
\]

\[
\begin{align*}
\text{BASE}_{\text{RESTR}} & \quad \frac{r \Rightarrow r'}{[r] \Rightarrow [r']}
\end{align*}
\]

\[
\begin{align*}
\text{BASE}_{\text{FLDSEL}} & \quad \frac{r \Rightarrow r'}{r, l \Rightarrow r', l}
\end{align*}
\]

\[
\begin{align*}
\text{RESTR} & \quad \frac{r \Rightarrow r'}{[r, l = a : A] \Rightarrow r'}
\end{align*}
\]

\[
\begin{align*}
\text{FLDSEL} & \quad \frac{a \Rightarrow a'}{\langle r, l = a : A \rangle, l \Rightarrow a'}
\end{align*}
\]

\[
\begin{align*}
\text{FLDSL}' & \quad \frac{r \Rightarrow r' \quad l \neq l'}{\langle r, l = a : A \rangle, l' \Rightarrow r', l'}
\end{align*}
\]

Figure 4.4: Parallel Reduction for IDRT
Remark Parallel reduction has some simple properties. First, $M \Rightarrow M$ for all $M$. Furthermore, if $M \rightarrow N$ then $M \Rightarrow N$, and if $M \Rightarrow N$ then $M \rightarrow^* N$. Second, any term parallelly reduces to itself. ($\text{id}$). Finally, if $M \Rightarrow M'$ and $N \Rightarrow N'$ then $[N/x]M \Rightarrow [N'/x]M'$ (proved by induction on $M \Rightarrow M'$).

Lemma 4.5.2 (Parallel Reduction $\subset$ Untyped Reduction)
If $M \Rightarrow N$ then $M \rightarrow^* N$.

Proof. By structural induction on the terms. From Definition 4.4 we know that no $\rightarrow$-redex was created by parallel reduction $\Rightarrow$. 

Remark As consequence of Lemma 4.5.2, we have $\rightarrow \subset \Rightarrow \subset \rightarrow^* \subset \Rightarrow^*$, Church-Rosser holds for both $\rightarrow^*$ and $\Rightarrow^*$.

(Proof: the first $\subset$, evident; the second $\subset$, by that no new $\rightarrow^*$-redex is created during $\Rightarrow$; the third $\subset$, by taking $\Rightarrow$-operation to the both sides of first $\subset$. Thus the Church-Rosser property of $\Rightarrow$ implies that of $\Rightarrow^*$, which is equal to the Church-Rosser property of $\rightarrow^*$ by observing that $\rightarrow^{**} = \rightarrow^*$ due to transitive closure.)

Lemma 4.5.3 (Parallel Subject Reduction)

(1) If $\Gamma \vdash \Delta$ and $\Gamma \Rightarrow \Gamma'$ then $\Gamma' \vdash \Delta$.

(2) If $\Gamma' \vdash A \rightarrow B$, $\Gamma \Rightarrow \Gamma'$ and $A \Rightarrow A'$ then $\Gamma' \vdash A' \rightarrow B$.

(3) If $\Gamma \vdash M \rightarrow N \rightarrow P : A$, $\Gamma \Rightarrow \Gamma'$ and $M \Rightarrow M'$ then there exist $N'$ and $N''$ such that $\Gamma' \vdash M' \rightarrow N'' \rightarrow P : A$, $N \Rightarrow N'$ and $\Gamma' \vdash N' \rightarrow N'' \rightarrow P : A$.

Proof. By simultaneous induction on derivations in the TOS.

We prove (3) according to the definition of parallel reduction (Definition 4.5.1).

– VAR. By Definition 4.5.1, we let the $M', N', N''$ all be $x$.

– LAM. By induction hypothesis and inversion, we get $A_1 \Rightarrow B_1$, $M_0 \Rightarrow P_0$, so by Definition 4.5.1 $[x:A_1]M_0 \Rightarrow [x:B_1]P_0$. Also we have $[x:A_1]M_0 \Rightarrow [x:A_1]M_0$ by (id). We let $M'$ be $[x:A_1]P_0$, $N', N''$ both be $[x:B_1]P_0$ and the induction step is proved.

– ETA. Similar to the above case, by induction hypothesis and Definition 4.5.1 we get $[x:A_1]M_0 \Rightarrow P$. We let $M'$ be $[x:A_1]M_0$, $N', N''$ be $P$ and the induction step is proved.

– BETA. By induction hypothesis and inversion, using both APP and $\beta$ in Definition 4.5.1, we get $M_1(M_2) \Rightarrow [M_2/x]N_0$. We let $M'$ be $[M_2/x]N_0$, $N', N''$ both be
$P$ and the induction step is proved.

- **BASE.** By induction hypothesis and inversion, using $APP$ in Definition 4.5.1, we get $M_1(M_2) \Rightarrow N_1(M_2)$. We let $M', N', N''$ all be $N_1(M_2)$ and the induction step is proved.

- **$EMP_{RCDT}$ and $EMP_{RCD}$.** By Definition 4.5.1, we let $M', N', N''$ all be $\langle \rangle$.

- **RCDT.** Similar to $LAM$, we let the $M'$ be $\langle R, l : A \rangle$, the $N', N''$ both be $\langle P, l : B \rangle$ so the induction step is proved.

- **RCD.** Similar to **RCDT**, we let the $M'$ be $\langle r, l = a : A \rangle$, the $N', N''$ both be $\langle p, l = b : B \rangle$, and the induction step is proved.

- **$BASE_{RESTR}$ and $BASE_{FLDSEL}$.** We show only for $BASE_{RESTR}$: similar to **BASE**, we have $[r] \Rightarrow [q]$, and we let $M', N', N''$ all be $[q]$, and the induction step is proved.

- **$RESTR$ and $FLDSEL$.** We show only for $RESTR$: similar to $BETA$, by induction hypothesis and inversion, using both $BASE_{RESTR}$ and $RESTR$ in Definition 4.5.1, we get $[r] \Rightarrow p$. We let $M'$ be $p$, $N', N''$ both be $s$ and the induction step is proved.

- **$FLDSL'$.** By induction hypothesis and inversion, we have $[r].l' \Rightarrow c$, with the condition that $l \neq l'$, using $FLDSL'$ in Definition 4.5.1, we get $r.l' \Rightarrow c$. We let $M', N', N''$ all be $c$ and the induction step is proved.

$\square$

The proof of subject reduction is very similar to that of the previous Parallel Subject Reduction (Lemma 4.5.3).

**Lemma 4.5.4 (Subject Reduction)**

If $\Gamma \models M \rightarrow N \rightarrow P : A$, $\Gamma \Rightarrow \Gamma'$ and $M \Rightarrow M'$ then there exists $N'$ such that $\Gamma' \models M' \rightarrow N' \rightarrow P : A$ and $N \rightarrow^* N'$.

**Proof.** The proof of this lemma follows directly from Lemma 4.5.3 (3) and Lemma 4.5.2. $\square$

The proof of strong normalisation of the TOS (Theorem 4.5.6) uses also the following lemma:
Lemma 4.5.5 (Subject Reduction w.r.t. Weak-head Normal)

(1) If $M$ is weak-head normal and not an abstraction and $M \to^* N$ then $N$ is weak-head normal and not an abstraction.

(2) If $M$ is weak-head normal and not a pair-record and $M \to^* N$ then $N$ is weak-head normal and not a pair-record.

Proof. By induction on length of reduction for untyped terms. \qed

Now we present the theorem and proof of strong normalisation for TOS.

Theorem 4.5.6 (Strong Normalisation of TOS)

(1) If $\Gamma \vdash A \to B$ then $A$ is strongly normalizable;

(2) If $\Gamma \vdash M \to N \to P : A$ then $M$ is strongly normalizable.

Proof. By simultaneous induction on derivations. We consider the TOS rules:

– **TYPE.** By definition the kind $\text{TY PE}$ is normal thus strongly normalizable.

– **EL.** By induction hypothesis $M$ is strongly normalizable, so by definition $\text{EL}(M)$ is strongly normalizable.

– **PI.** By induction hypothesis $A_1, A_2$ are strongly normalizable, since every reduct from $(x:A_1)A_2$ will be in the form $(x:A'_1)A'_2$ where $A'_1$ is the reduct of $A_1$ and $A'_2$ is the reduct of $A_2$. Thus the strong normalizability of both $A_1$ and $A_2$ implies the strong normalizability of $(x:A_1)A_2$; this could be shown by induction on the maximal length of reductions for $A_1$ and $A_2$.

– **VAR.** By definition $x$ is normal thus is strongly normalizable.

– **LAM and ETA.** Similar to the case PI, by induction hypothesis $A_1$ and $M_0$ are strongly normalizable, thus $[x:A_1]M_0$ is strongly normalizable.

– **BASE.** By induction hypothesis $M_1$ and $M_2$ are strongly normalizable, and by assumption $\Gamma \vdash M_1 \to_w N_1 : (x:B_1)B_2$ where $N_1$ is not an abstraction. By induction on the maximal length of reductions for $M_1$ and $M_2$, we show that (*) if $M_1, M_2$ are strongly normalizable, $\Gamma \vdash M_1 \to_w N_1 : (x:B_1)B_2$ where $N_1$ is not an abstraction then $M_1(M_2)$ is strongly normalizable. We also show that (***) if $M_1$ is strongly normalizable and $M_1 \Rightarrow M'_1$ then $M'_1$ is strongly normalizable.

Then, by $\text{SN} - I$, we need to prove that if $M_1(M_2) \Rightarrow P$ then $P$ is strongly normalizable. Consider the two possible sub-reductions:
First, if $M_1 \Rightarrow M'_1$. Using Parallel Subject Reduction (Lemma 4.5.3), from the assumptions that $\Gamma \models M_1 \rightarrow N_1 \rightarrow P_1 : (x:B_1)B_2$ and $M_1 \Rightarrow M'_1$, there exist $N'_1, N''_1$ such that $\Gamma \models N'_1 \rightarrow_w N''_1 : (x:B_1)B_2$, $\Gamma \models N'_1 \rightarrow_w N''_1 : (x:B_1)B_2$ and $N_1 \Rightarrow N'_1$. From $N_1 \Rightarrow N'_1$ using Lemma 4.5.2 we have $N_1 \rightarrow^* N'_1$; from Lemma 4.5.5 $N'_1$ is weak-head normal and not an abstraction. From $\Gamma \models N'_1 \rightarrow_w N''_1 : (x:B_1)B_2$ using Adequacy Lemma for Untyped Reduction (Lemma 4.3.4) we have $N'_1 \rightarrow^* N''_1$; from Lemma 4.5.5 $N''_1$ is not an abstraction. Also by induction hypothesis, using $(\ast)$ we know that $M'_1$ is strongly normalizable, furthermore using $(\ast)$ we have that $M'_1(M_2)$ is strongly normalizable;

Second, if $M_2 \Rightarrow M'_2$, similarly by induction hypothesis.

- $\textit{BETA}$. Use Parallel Subject Reduction, the closure of Strong Normalisation under reduction and the closure of reduction under substitution, respectively.
- $\textit{RTYPE} \text{ and } \textit{RTYPE}[L]$. By definition the two kinds are normal thus strongly normalizable.
- $\textit{EMP}_{\textit{RCDT}} \text{ and } \textit{EMP}_{\textit{RCD}}$. By definition the empty record type and the empty record object are normal thus strongly normalizable.
- $\textit{RCDT}$. By induction hypothesis $R, A$ are both strongly normalizable, thus the record type $\langle R, l : A \rangle$ is strongly normalizable similar to the case $\textit{PI}$.
- $\textit{RCD}$. By induction hypothesis, $r, a$ are strongly normalizable, and $\langle R, l : A \rangle$ is strongly normalizable. Using one step of inversion of the rule $\textit{RCDT}$ we get $A$ is strongly normalizable too. Similar to the above case we have $\langle r, l = a : A \rangle$ is strongly normalizable.
- $\textit{BASE}_{\textit{RESTR}} \text{ and } \textit{BASE}_{\textit{FLDSEL}}$. We show for $\textit{BASE}_{\textit{RESTR}}$:

Consider possible reduction $r \Rightarrow r_1$: using Parallel Subject Reduction, there exist $r', r''$ such that $\Gamma \models r_1 \rightarrow_w r'' : \langle P, l : B \rangle$, $\Gamma \models r' \rightarrow_w r'' : \langle P, l : B \rangle$ and $q \Rightarrow r'$. Similarly we show that $(\ast\ast\ast)$ if $r$ is strongly normalizable, $\Gamma \models r \rightarrow_w r' : \langle P, l : B \rangle$ where $r'$ is not a pair-record then $[r]$ is strongly normalizable.

By induction hypothesis, using $(\ast)$ $r_1$ is strongly normalizable; From Lemma 4.5.2, Lemma 4.5.5 and Adequacy Lemma for Untyped Reduction similarly we have $r''$ is not a pair-record, thus using $(\ast\ast\ast)$ we have $[r_1]$ is strongly normalizable.

- $\textit{RESTR}, \textit{FLDSEL}$ and $\textit{FLDSL}'$. Similar to the case $\textit{BETA}$, using Parallel Subject Reduction, the closure of Strong Normalisation under record reductions implies the strong normalizability of $[r]$, $r.l$ and $r.l'$, respectively.
And finally, we arrive at the following main Corollary, which is strong normalisation of our original system of dependent record types \( \text{IDRT} \):

**Corollary 4.5.7 (Strong Normalisation of \( \text{IDRT} \))**

1. If \( \Gamma \) valid then \( \Gamma \) is strongly normalizable;
2. If \( \Gamma \vdash A \) kind then \( A \) is strongly normalizable;
3. If \( \Gamma \vdash A = B \) then both \( A \) and \( B \) are strongly normalizable to some \( C \);
4. If \( \Gamma \vdash M : A \), then \( M \) is strongly normalizable to some \( P \), \( A \) is strongly normalizable to some \( B \) such that \( P : B \) in \( \Gamma \);
5. If \( \Gamma \vdash M = N : A \), then both \( M \) and \( N \) are strongly normalizable to some \( P \), \( A \) is strongly normalizable to some \( B \) such that \( P : B \) in \( \Gamma \).

**Proof.** Derived from Theorem 4.4.2 (Soundness of \( \text{TOS} \) w.r.t. \( \text{IDRT} \)) and Theorem 4.5.6 (Strong Normalisation of \( \text{TOS} \)).

Other metatheory such as the Church-Rosser property, uniqueness of normal form, etc. could be derived in a similar way as strong normalisation and subject reduction, and we give these theorems in the following Section 4.6 collectively.
4.6 Metatheoretic Properties of IDRT

From the properties of the TOS that have been proved in the above sections, the metatheoretic properties of IDRT, the type theory for dependent record types, can be proved. Here, we give the theorems for Subject Reduction, Church-Rosser and Strong Normalisation.

**Corollary 4.6.1 (Subject Reduction for IDRT)** If $\Gamma \vdash M : A$ and $M \rightarrow N$, then $\Gamma \vdash N : A$.

*Proof.* By Soundness (Theorem 4.4.2) we have $\Gamma \mid= M \rightarrow_n P : A$ for some $P$. By consequence of Lemma 4.5.2 $M \rightarrow N$ implies $M \Rightarrow N$. From these two and by Lemma 4.5.4 we have $\Gamma \mid= N \rightarrow_n P : A$. Again by Completeness (Theorem 4.3.2) we have $\Gamma \vdash N : A$.  \qed

**Corollary 4.6.2 (Strong Normalisation for IDRT)**

1. If $\Gamma$ valid, then $\Gamma$ is strongly normalizable.

2. If $\Gamma \vdash A$ kind, then $A$ is strongly normalizable.

3. If $\Gamma \vdash A = B$, then both $A$ and $B$ are strongly normalizable to some $C$.

4. If $\Gamma \vdash M : A$, then $M$ and $A$ are strongly normalizable and $\Gamma \vdash P : B$, where $P$ and $B$ are the normal forms of $M$ and $A$, respectively.

5. If $\Gamma \vdash M = N : A$, then both $M$ and $N$ are strongly normalizable to some $P$, $A$ is strongly normalizable to some $B$ such that $\Gamma \vdash P : B$.

*Proof.* By the Soundness Theorem 4.4.2 and Theorem 4.5.6. \qed

**Corollary 4.6.3 (Church-Rosser Property for IDRT)** If $\Gamma \vdash M = N : A$, then $\exists P$ such that $M \rightarrow^* P$ and $N \rightarrow^* P$. 

2
Proof. By Soundness (Theorem 4.4.2), there exist $P, B$ such that $\Gamma \vdash M \rightarrow_n P : B$ and $\Gamma \vdash N \rightarrow_n P : B$. By Adequacy (Lemma 4.3.4) we prove that $M \rightarrow^* P$ and $N \rightarrow^* P$. 

**Corollary 4.6.4 (Unicity of Types)** If $\Gamma \vdash M : A$ and $\Gamma \vdash M : B$ then $\Gamma \vdash A = B : Type$.

Proof. By Soundness (Theorem 4.4.2), Uniqueness of Normal Forms (Theorem 4.3.6) and Completeness (Theorem 4.3.2).

**Corollary 4.6.5 (Kind Correctness)** If $\Gamma \vdash M : A$ then $\Gamma \vdash A$ kind.

Proof. By Soundness (Theorem 4.4.2) and Completeness (Theorem 4.3.2).

Until here we have introduced and set up a full metatheory for the TOS of IDRT, and given the proven results on metatheory of the dependent record types, which is the main result of this thesis. From the next chapter, we would start on a new topic, which is on the coercive subtyping for the DRTs. We will make use of some of the metatheories proved before, and the next chapter on coercive subtyping could be regarded as an application of DRTs with metatheoretic properties, like Church-Rosser property and type uniqueness.
Chapter 5

Structural Subtyping
5.1 Coercive Subtyping: an Introduction

We study in this chapter structural subtyping for dependent record types in the framework of coercive subtyping [Luo99]. After a brief introduction to coercive subtyping in this section, we consider the structural coercions for dependent record types and prove that, together with a special coercion from the unit types, they are coherent, using the metatheoretic results of IDRT studied in the previous chapters. As an application of this coherence result, we will study intensional manifest fields in the DRTs.

Coercive subtyping is a general approach to abbreviation and subtyping in dependent type theories [Luo97, Luo99]. In coercive subtyping, $A$ is a subtype of $B$ if there is a coercion $c : (A)B$, expressed by $\Gamma \vdash A \xrightarrow{c} B : Type$. The main idea is expressed by the following coercive definition rule in [Luo97]:

$$
\frac{f : B \to D \quad a : A \quad A \xrightarrow{c} B}{f(a) = f(c(a)) : D}
$$

which says that if $a : A$ ($a$ is an object of type $A$), and $A \xrightarrow{c} B$ (type $A$ is subtype of type $B$ via coercion $c$), then $a$ could be regarded as $c(a)$ which is an object of type $B$. We say that $a$ stands for $c(a)$, or $a$ is the abbreviation of $c(a)$.

Under the more traditional notion of subtyping where the following subsumption rule is used:

$$
\frac{a : A \quad A \leq B}{a : B}
$$

the property of canonicity is lost.\footnote{Definition(Canonicity, [AM06]): Any closed object of an inductive type is definitionally equal to a canonical object of that type.} A counterexample is: with the parameterised inductive type $List$, suppose if we have:

$$
\frac{A \leq B}{List(A) \leq List(B)}
$$
then by the subsumption rule we would have $\text{nil}(A) : \text{List}(B)$. However, $\text{nil}(A)$ is not definitionally equal to any canonical object of type $\text{List}(B)$, not to $\text{nil}(B)$ or $\text{cons}(B, b, l)$ for some $b, l$. Thus, canonicity is lost here.

A possible solution to this incompatibility is to amend the elimination rules to treat the new objects created by subtyping, but it introduces new mechanisms such as bounded quantification which is complicated and may lead to undecidability [Luo10].

However, with the notion of coercive subtyping, the property of canonicity is preserved. In our previous example,

$$f : \text{List}(B) \rightarrow D \quad \text{nil}(A) : \text{List}(A) \quad \text{List}(A) \xrightarrow{\text{map}(c)} \text{List}(B)$$

$$f(\text{nil}(A)) = f(\text{map}(c)(\text{nil}(A))) : D$$

where $\text{map}(c)$ coerces the canonical object $\text{nil}(A)$ to $\text{nil}(B)$ which is a canonical object of $\text{List}(B)$.

We would like to separate two sorts of coercive subtyping rules: those for structural subtyping and those for non-structural subtyping. Structural subtyping is syntax-directed for inductive types and record types. For example, the above coercive subtyping rule about the types of lists are a typical rule for structural subtyping. The general rules for structural subtyping and the associated transitivity elimination results could be found in [LA08]. In the following section, we shall study the structural subtyping rules for dependent record types.

Non-structural subtyping can be useful in many applications. Examples of such include the rule $(\xi)$ that we shall also introduce and study in the next section and, together with the structural subtyping rules for DRTs, it allows one to introduce manifest field in record types, an interesting application we shall study in the last section of this chapter.
5.2 Structural Subtyping for DRTs

In this section, we will introduce a formal system with coercive subtyping. This system is called \( IDRT[\mathcal{R}] \), the type system of dependent record types with the set \( \mathcal{R} \) of coercions, which satisfied certain coherence property. We shall study the coherence property of this system, which is informally, that the set of coercions \( \mathcal{R} \) is coherent such that there are no unidentical coercions in \( \mathcal{R} \) which map between the same pair of two types.

5.2.1 The system \( IDRT[\mathcal{R}] \)

Briefly, if \( T \) is a type theory specified in the logical framework LF and \( R \) is a set of basic subtyping rules (i.e., coercion rules for coercive subtyping), then \( T[R] \) is the type theory \( T \) extended with coercive subtyping generated by the rules in \( R \).

If we take \( T \) to be the system of dependent record types \( IDRT \) and \( R \) the set of coercion rules \( \mathcal{R} \) consisting of the rules \((\xi)\) and \((d_i)^R\) \((i = 1, 2, 3)\) to be specified below, this results in our system \( IDRT[\mathcal{R}] \).

We give explanations to these coercions rules in more detail:

Structural Coercion Rules for DRTs. The structural subtyping rules for dependent record types include the so-called component-wise rules, as given in Figure 5.1 below. These rules propagate subtyping through the dependent record types. For instance, in \((d_3)^R\), \(d_3^R \) maps \((r, l = a)\) to \((c(r), l = c'(r)(a))\), propagating the subtyping relations represented by the coercions \( c \) and \( c' \) through the record types.

\(^2\)For the component-wise coercion, we originally only have a single rule \( (d) \) which is the following:

\[
\begin{align*}
\Gamma \vdash A : (R)Type & \quad \Gamma \vdash A' : (R')Type \\
\Gamma \vdash R \xrightarrow{C} R' : RType & \quad \Gamma, x : R \vdash A(x) \xrightarrow{C[x]} A'(c(x)) : Type \\
\Gamma \vdash (R, l : A) & \xrightarrow{d} \langle R', l : A' \rangle : RType
\end{align*}
\]

with the function \( d \) defined in term with \( c \) and \( c' \) as \( d(r : \langle R, l : A \rangle) =_{d} \langle c[r], c'[r](r, l) \rangle \).

However, with the lost of weak extensionality in our DRTs, some properties do not hold any more for proving the Coherence, which results a change into three distinct \( d \)-rules.
5.2. Structural Subtyping for DRTs

\[
\frac{\Gamma \vdash R : R\text{Type}[L]}{\Gamma \vdash R' : R\text{Type}[L]} \quad \frac{\Gamma \vdash A' : (R)\text{Type}}{\Gamma \vdash R \xrightarrow{c} R' : R\text{Type}} \quad l \notin L
\]

\[
\Gamma \vdash \langle R, l : A' \circ c \rangle \xrightarrow{d_R^1} \langle R', l : A' \rangle : R\text{Type}
\]

where \(d_R^1 = [x : \langle R, l : A \rangle](c([x]), l = x.l)\).

\[
\frac{\Gamma \vdash R : R\text{Type}[L]}{\Gamma \vdash R' : R\text{Type}[L]} \quad \frac{\Gamma \vdash A : (R)\text{Type}}{\Gamma \vdash A' : (R)\text{Type}} \quad \frac{\Gamma \vdash x : R \vdash A(x) \xrightarrow{\epsilon(x)} A'(x) : \text{Type}}{\Gamma \vdash \langle R, l : A \rangle \xrightarrow{d_R^2} \langle R', l : A' \rangle : R\text{Type}}
\]

where \(d_R^2 = [x : \langle R, l : A \rangle]([x], l = \epsilon([x])(x.l))\).

\[
\frac{\Gamma \vdash R \xrightarrow{c} R' : R\text{Type}}{\Gamma \vdash A(x) \xrightarrow{\epsilon(x)} A'(c(x)) : \text{Type}}
\]

\[
\Gamma \vdash \langle R, l : A \rangle \xrightarrow{d_R^3} \langle R', l : A' \rangle : R\text{Type}
\]

where \(d_R^3 = [x : \langle R, l : A \rangle](c([x]), l = \epsilon([x])(x.l))\).

Figure 5.1: Component-wise Coercion Rules for DRTs

We also clarify that in this piece of work for structural subtyping of dependent record types, we do not consider the other form of record structural subtyping in the intuitive sense of “smaller records”, which is \(R \leq \langle R, l : A \rangle\).

The Coercion Rule \((\xi)\) for Unit Types. A typical example of non-structural rules of subtyping is the \((\xi)\)-rule about the unit types, as studied in [Luo08].

A unit type \(\text{Unit}(A, a)\) is parameterised by an arbitrary type \(A\) of the kind \(\text{Type}\) together with an object \(a\) of this type \(A\). It is the inductive type with only one object \(\text{unit}(A, a)\). It can be specified in LF by declaring the following constants:

\[
\begin{align*}
\text{Unit} & : (A: \text{Type})(x:A) \ \text{Type} \\
\text{unit} & : (A: \text{Type})(x:A) \ \text{Unit}(A, x)
\end{align*}
\]
\[
E : (A : Type)(x : A) (C : (Unit(A, x)) Type)(c : C(unit(A, x))(z : Unit(A, x))) C(z)
\]

and the following computation rule:

\[
E(A, x, C, c, unit(A, x)) = c : C(unit(A, a)).
\]

For instance, the declaration of \(Unit\) gives rise to the following rule of formation for
the unit types:

\[
\Gamma \vdash A : Type \quad \Gamma \vdash a : A \\
\Gamma \vdash Unit(A, a) : Type
\]

The coercion \(\xi_{A,a}\) is from \(Unit(A, a)\) to \(A\) and maps every object of \(Unit(A, a)\)
to \(a\). This is reflected by means of the following basic subtyping rule:

\[
(\xi) \\
\Gamma \vdash A : Type \quad \Gamma \vdash a : A \\
\Gamma \vdash Unit(A, a) \xi_{A,a} A : Type
\]

where \(\xi_{A,a} = [x : Unit(A, a)]a\).

In the following, we show that the coercion rule \((\xi)\) and the component-wise
coercion rules are coherent together and hence can be used together to provide
means to express manifest fields in DRTs.

### 5.2.2 Coherence

Coherence basically means that there do not exist two different coercions that map
between the same pair of types. That is, if in our system there are such two coercive
rules added in the same time: \(\Gamma \vdash A \xrightarrow{c} B : Type\) and \(\Gamma \vdash A \xrightarrow{c'} B : Type\), then
we can derive that \(\Gamma \vdash c = c' : (A)B\).

Formally, let \(T[R]_0\) be the type system that extends \(T\) with the following:

- the subtyping judgements of the form \(\Gamma \vdash A \xrightarrow{c} B : Type\), and
- the basic subtyping rules in \(R\) together with the congruence, substitution and
  transitivity rules for the subtyping judgements.

Here is the formal definition of coherence.
Definition 5.2.1 (Coherence) Let $\Gamma \vdash_0$ to denote derivability of judgements in $T[R]_0$. A set $\mathcal{R}$ of coercion rules is coherent if

1. For any type $\Gamma$, $A$ and $c$, $\Gamma \not\vdash_0 A \xrightarrow{c} A : Type$.
2. The following rule is admissible:

$$
\begin{array}{c}
\Gamma \vdash A \xrightarrow{c} B : Type \\
\Gamma \vdash A \xrightarrow{c'} B : Type
\end{array}
\Rightarrow
\Gamma \vdash c = c' : (A)B
$$

Remark The notion of coherence is a crucial property in coercive subtyping. Incoherence would imply that the extension with coercive subtyping is not conservative in the sense that more judgements of the original type theory $T$ can be derived. When the coercions are coherent, one can always insert coercions correctly into a derivation in the extension to obtain a derivation in the original type theory.

It can be shown, based on the metatheoretic results of IDRT studied in this thesis, that the rules for the component-wise coercions and the coercion rule $(\xi)$ are coherent together.

Theorem 5.2.2 (Coherence) Let $\mathcal{R} = \{(\xi), (d_1^R), (d_2^R), (d_3^R)\}$. Then $\mathcal{R}$ is coherent.

Proof. We prove the second clause of coherence (about admissibility), by showing that the following more general rule is admissible:

$$
\begin{array}{c}
\Gamma \vdash A \xrightarrow{c} B : Type \\
\Gamma \vdash A' \xrightarrow{c'} B' : Type \\
\Gamma \vdash A = A' : Type \\
\Gamma \vdash B = B' : Type
\end{array}
\Rightarrow
\Gamma \vdash c = c' : (A)B
$$

This is proved by induction on derivations. For example, in the case that the last rules to derive $\Gamma \vdash A \xrightarrow{c} B : Type$ and $\Gamma \vdash A' \xrightarrow{c'} B' : Type$ are both $(\xi)$ with $c \equiv \xi_{C, a}$ and $c' \equiv \xi_{C', b}$, we have that

$$
Unit(C, a) \equiv A = A' \equiv Unit(C', b).
$$
Then, by Church-Rosser (Theorem 4.6.3), $C = C'$, $a = b$, and $\xi_{C,a}(x) = a = b = \xi_{C',b}(x)$ for any $x : \text{Unit}(C,a)$. Therefore, by the following LF-rules in Figure 2.2:

\[
\begin{align*}
\Gamma \vdash K_1 & = K_2 \quad \Gamma, x : K_1 \vdash k_1 = k_2 : K \\
\Gamma \vdash [x : K_1]k_1 & = [x : K_2]k_2 : (x : K_1)K \\
\Gamma \vdash f : (x : K)K' & \text{ } x \notin \text{FV}(f) \\
\Gamma \vdash [x : K]f(x) & = f : (x : K)K'
\end{align*}
\]

we have $\xi_{C,a} = \xi_{C',b}$.

The other case is that the last rules applied in a derivation are $(d^i_R)$ ($i = 1, 2, 3$). Suppose two coercions $d^i_R$ and $d^i_R'$ are used, apply induction and by congruence rules of records, we have $d^i_R \equiv d^i_R'$, respectively for $i = 1, 2, 3$. \qed
5.3 Manifest Fields in DRTs

Manifest fields for dependent record types may be represented in an intensional type theory by means of the coercions \((\xi)\) and \((d_R^i)\) \((i = 1, 2, 3)\). This was explained in [Luo08], where it was made clear that, in order for the representational scheme to work, these coercions need to be coherent. In the previous section, we have shown that they are coherent (Theorem 5.2.2); therefore, they can be employed to represent intensional manifest fields in DRTs as conceived in [Luo08]. We shall give a brief explanation of this in this section.

Manifest Fields in DRTs. A field in a dependent record type can be abstract, of the form ‘\(v : A\)’, in the sense that the data expected in that field can be any object of \(A\), or manifest, of the form ‘\(v = a : A\)’, if the data expected in that field is not only of type \(A\) but the ‘same’ as the specific object \(a\) of type \(A\). Manifest fields provide a powerful representation mechanism which intuitively allows an internal expression of definitional entries at the level of types: one can use it to express the typing constraint that a field expects a particular object of a type rather than anyone of that type. For instance, one can use \(\Sigma\)-types or dependent record types with manifest fields to express the powerful module mechanism with the so-called ML-style sharing (or sharing by equations) [Mac86].

Extensional Manifest Fields. Manifest fields may be introduced directly by means of introducing extensional equalities or represented by other extensional constructs such as the extensional equality in Martin-Löf’s extensional type theory. However, such an extensionality often introduces difficulties in metatheoretic studies and sometimes could even introduce undecidability of type checking. (See [Luo08] for a more detailed summary of this and further references.) As we said above, it is possible to represent manifest fields in an intensional type theory by means of coercive subtyping.

Intensional Manifest Fields for DRTs. Intensional manifest fields (IMFs) in DRTs are introduced as follows. For \(A : (R)Type\) and \(a : (r:R)A(r)\),

\[ l \sim a : A \quad \text{stands for} \quad l : [r:R]Unit(A(r), a(r)). \]
In the simpler situation, for $A : \text{Type}$ and $a : A$, $l \sim a : A$ stands for $l : \text{Unit}(A, a)$. In records, for $b : B$,

$$l \sim_B b \text{ stands for } l = \text{unit}(B, b).$$

The IMFs in record types work as intended under the assumption that we have the following two coercions studied in the previous section:

- The coercion $\xi$.
- The component-wise coercions $d^i_R$ ($i = 1, 2, 3$).

Examples of IMFs and how they may be used can be found in [Luo08].

**with-clauses.** Intensional manifest fields in DRTs used to define the with-clauses [Pol02] and other associated operations.

- For $R \equiv \langle l_1 : A_1, \ldots, l_n : A_n \rangle$, $i \in \{1, \ldots, n\}$ and $a : (x : R_{i-1})A_i(x)$, where $R_{i-1} \equiv \langle l_1 : A_1, \ldots, l_{i-1} : A_{i-1} \rangle$, the with-clause is defined as follows:

$$R \text{ with } l_i \text{ as } a =_{df} \langle l_1 : A_1, \ldots, l_{i-1} : A_{i-1}, l_i : A_i, l_{i+1} : A_{i+1}, \ldots, l_n : A_n \rangle.$$

- For $r : R$,

$$r|_{l_i} =_{df} \langle l_1 = r.l_1, \ldots, l_{i-1} = r.l_{i-1}, l_i \sim_{A_i(r_{i-1})} r.l_i, l_{i+1} = r.l_{i+1}, \ldots, l_n = r.l_n \rangle,$$

where $r_{i-1} \equiv \langle l_1 = r.l_1, \ldots, l_{i-1} = r.l_{i-1} \rangle$.

The above definitions of the with-clause and the $|$-operation behave as intended, shown by the following lemma.

**Theorem 5.3.1** Let $R \equiv \langle l_1 : A_1, \ldots, l_n : A_n \rangle$ and $r : R$. Then, $r.l_i = a(r_{i-1})$ if and only if $r|_{l_i} : (R \text{ with } l_i = a)$.

**Proof.** ($\Rightarrow$) By induction on $n$.

In the case of $n = 1$, where $R \equiv \langle l_1 : A_1 \rangle$, and $r.l_1 = a$. 
We have \( r|_{l_i} =_d \langle l_1 \sim_{A_i} r.l_1 \rangle =_d \langle l_1 = \text{unit}(A_1, r.l_1) \rangle \), by congruence rule of units and \( r.l_1 = a \) we have \( \text{unit}(A_1, r.l_1) = \text{unit}(A_1, a) \) and again by congruence rule of records we have \( r|_{l_i} = \langle l_1 = \text{unit}(A_1, a) \rangle \). We also have \((R \text{ with } l_i = a) =_d \langle l_1 : \text{Unit}(A_1, a) \rangle \). Since \( \text{unit}(A_1, a) : \text{Unit}(A_1, a) \), by introduction rule of records \( r.l_i = (R \text{ with } l_i = a) \).

Suppose in the case of \( n \Rightarrow \) holds. Then in the case of \( n + 1 \), where \( R \equiv \langle l_1 : A_1, \ldots, l_n : A_n, l_{n+1} : A_{n+1} \rangle \), and \( r.l_i = a(r_{i-1}) \). We have \( r|_{l_i} =_d \langle l_1 = r.l_1, \ldots, l_i \sim_{A_i(r_{i-1})} r.l_i, \ldots, l_n = r.l_n, l_{n+1} = r.l_{n+1} \rangle \)

\( =_d \langle l_1 = r.l_1, \ldots, l_i = \text{unit}(A_i(r_{i-1}), r.l_i), \ldots, l_n = r.l_n, l_{n+1} = r.l_{n+1} \rangle \)

where \( r_i : R_i = \langle l_1 : A_1, \ldots, l_i : A_i \rangle \) for \( i \leq n + 1 \).

Also we have \((R \text{ with } l_i = a) =_d \langle l_1 : A_1, \ldots, l_i \sim a : A_i, \ldots, l_n : A_n, l_{n+1} : A_{n+1} \rangle \)

\( =_d \langle l_1 : A_1, \ldots, l_i : [r_{i-1}; R_{i-1}] \text{Unit}(A_i(r_{i-1}), a(r_{i-1})), \ldots, l_n : A_n, l_{n+1} : A_{n+1} \rangle \) for \( i \leq n + 1 \).

This contains two subcases: For \( i \leq n \) we can easily deduce \( r|_{l_i} \equiv \langle r_n|_{l_i}, l_{n+1} = r.l_{n+1} \rangle \), by induction hypothesis we have \( r_n|_{l_i} : (R_n \text{ with } l_i = a) \), thus by introduction rule we have \( r|_{l_i} : (R_n \text{ with } l_i = a, l_{n+1} : A_{n+1}) \equiv (R \text{ with } l_i = a) \).

For \( i = n + 1 \), similar to the base case, \( r|_{l_i} =_d \langle l_1 = r.l_1, \ldots, l_{n+1} = \text{unit}(A_{n+1}(r_n), r.l_{n+1}) \rangle \), by introduction rule \( r|_{l_i} : \langle l_1 : A_1, \ldots, l_{n+1} : [r_{n}; R_n] \text{Unit}(A_{n+1}(r_n), r.l_{n+1}) \rangle \), by congruence \( r|_{l_i} : \langle l_1 : A_1, \ldots, l_{n+1} : [r_{n}; R_n] \text{Unit}(A_{n+1}(r_n), a(r_n)) \rangle =_d (R \text{ with } l_i = a) \).

\((\Leftarrow)\) Let \( r|_{l_i} : (R \text{ with } l_i = a) \), which means, from definition of \( | \) and with-clause

\( \langle l_1 = r.l_1, \ldots, l_i = \text{unit}(A_i(r_{i-1}), r.l_i), \ldots, l_n = r.l_n \rangle \) is of type

\( \langle l_1 : A_1, \ldots, l_i : [r_{i-1}; R_{i-1}] \text{Unit}(A_i(r_{i-1}), a(r_{i-1})), \ldots, l_n : A_n \rangle \). \hspace{1cm} (1)

And we also know from definition that \( r|_{l_i} \) is of type

\( \langle l_1 : A_1, \ldots, l_i : [r_{i-1}; R_{i-1}] \text{Unit}(A_i(r_{i-1}), r.l_i), \ldots, l_n : A_n \rangle \). \hspace{1cm} (2)

By type uniqueness (Corollary 4.6.4) we have \((1) = (2)\), and by Church-Rosser property (Corollary 4.6.3) we have \( r.l_i = a(r_{i-1}) \). \hfill \Box
Chapter 6

Conclusions and Further Work
6.1 Conclusion of Thesis

- Modelling Object-oriented Programming
- Modelling Module Mechanisms
- Incorporating Subtyping

DRT

- A Better Understanding w.r.t. non-SN Calculi
- Extended to Inductive Types and Universes
- Compared to Proof via Conservativity of Σ-types
- η-rules or Weakly Extensional Rules Discussed

TOS
6.1.1 Conclusion with the Dependent Record Types

Dependent record types have been studied here in this thesis and in other works [Luo08, Luo08-2, Luo08-3, Luo09, Luo09-2, FL10], with possible applications in modelling Object-Oriented Programs and in modelling module mechanisms. This thesis emphasises on the metatheoretic properties of this sort of types, and has used a typed operational semantics way to prove some of the metatheory of DRTs. Meanwhile, these metatheories guarantee the other metatheoretic properties while DRTs are applied with other type structures, for instance as in Chapter 5, we studied DRTs with coercive subtyping and proved that coercions are coherent.

We will discuss here extensionality, namely the weakly extensional rules of DRTs and their impact to metatheoretic proofs.

Discussion on Extensionality

The weakly extensional equality rules (WERs for short) are the $\eta$-rules for the dependent record types. Systems with WERs are called the extensional DRTs (EDRT).

\[
\begin{align*}
\text{Weakly extensional equality rules} \\
\Gamma \vdash r : \langle \rangle & \quad \Gamma \vdash r : \langle R, l : A \rangle \\
\Gamma \vdash r = \langle \rangle : \langle \rangle & \quad \Gamma \vdash [r] = [r'] : R \\
\Gamma \vdash r.l = r'.l : A([r]) & \quad \Gamma \vdash r = r' : \langle R, l : A \rangle (WER1) \\
\end{align*}
\]

These two rules basically say that two records are computationally equal if their components are. For instance, from the second rule above, we would have $\langle r, l = r.l \rangle = r$ for any $r$ of type $\langle R, l : A \rangle$. However, in our later study, we have discovered that including the $\eta$-rules will make some metatheoretic property fail to hold. Thus, we have chosen to exclude these two weakly extensional equality rules, and make our system of dependent record types intensional.

The weakly extensional equality rules (WERs) could cause the Church-Rosser property, i.e., two distinct reductions starting from the same term commute, to fail. This is generally problematic with systems with $\eta$-rules and $\eta$-equality, and is going to be illustrated with the following example.
An example for why Church-Rosser fails: in a system with both Π-type, Σ-type and unit type 1, suppose we allow η-rules for Π, Σ and for unit types,

\[
\begin{align*}
(\eta_\Pi) & \quad \lambda x:A.f(x) \to f : B \ (\text{if } x \notin \text{FV}(f)) \\
(\eta_\Sigma) & \quad \langle \pi_1(x), \pi_2(x) \rangle \to x : A \\
(\eta_1) & \quad x \to * : 1
\end{align*}
\]

then, a term \( \lambda x:A.f(x) \) with \( f : A \to 1 \) could have two reductions by \((\eta_\Pi)\) and by \((\eta_1)\) that do not commute: \( \lambda x:A.f(x) \to_{\eta_\Pi} f \) and \( \lambda x:A.f(x) \to_{\eta_1} \lambda x:A : * \). Note that \( \lambda x:A : * \) is already a canonical form.

Normally, we would hope the Church-Rosser property to hold, in the sense that:

\[
\begin{align*}
\lambda x:A.\langle \pi_1(x), \pi_2(x) \rangle & \to_{\eta_\Pi} \langle \pi_1(x), \pi_2(x) \rangle \to_{\eta_\Sigma} x, \text{ and} \\
\lambda x:A.\langle \pi_1(x), \pi_2(x) \rangle & \to_{\eta_\Sigma} \lambda x:A.x \to_{\eta_\Pi} x
\end{align*}
\]

Similarly for a term \( \langle \pi_1(x), \pi_2(x) \rangle : 1 \times A \), it reduces to both \( x \) and to \( \langle *, \pi_2(x) \rangle \) under \((\eta_\Sigma)\) and \((\eta_1)\) respectively, which do not commute.

We distinguish two types of η-reductions: one is the structural ηs such as \((\eta_\Pi)\) or \((\eta_\Sigma)\), the other one is the non-structural or “terminating” ηs such as \((\eta_1)\) (they are called “terminating” because the calculations are towards a canonical form). It appears to us that the non-structural η-rules have caused the failure of the Church-Rosser property. Further proofs of the impact on η-rules to CR are required.
6.1.2 Conclusion with the Metatheory by Typed Operational Semantics

Metatheory is the theoretical aspect or properties of a system. In general, the hold of some meta-properties (for example, strong normalisation, Church-Rosser property) could ensure the good functionality (e.g., soundness, confluence, or coherence, etc) of this system, thus the study carries certain importance. There is another view of metatheory, through the decidability of type checking in the type system. And many approaches for metatheories are using a semantical analysis (operational, denotational, or algorithmic), or a transference to another system (when conservativity holds), to prove certain metatheory for a type system.

Unlike Coquand, Pollack and Takeyama’s work in [CPT05] in suggesting a proof of termination of type-checking for the record types, we have shown that Goguen’s method to prove strong normalisation with a typed operational semantics could also be applied to the dependent record types given in [Luo08, Luo09]. As in Goguen’s work in [Gog94, Gog99] of establishing a full metatheory for the UTT and for the LF, we have also managed to do so for the DRTs with a same technique. Nevertheless, our work has led to a further understanding on the following three things: First, the DRTs with intensional manifest fields developed in [Luo08, Luo09] could have confidently good metatheoretic properties, which would enhance the DRTs themselves as a modelling mechanism for OO-style programming and as a component to be implemented in proof assistants; Second, the essence of this technique of using a typed operational semantics is to capture the subtle relation between well-formedness and well-behaviour with typing control in the operational semantics; Third, this technique could be extended to wider range of type systems such as the Logical Framework with inductive schemata, but in the proofs of metatheory there, pure proof-theoretic proofs are hard to achieve. i.e., without any model interpretation. Thus, this work could be seen as a simple yet nontrivial application of Goguen’s technique which succeeds in ensuring the good metatheory of dependent record types.

Nevertheless, this work has led us to the following thoughts on canonical normal-
6.1. Conclusion of Thesis

isation of formal calculi, that how TOS succeeds to capture canonical normalisation reductions via the typed control, and thus provided the purely proof-theoretic approach; however, for the ill-behaving (e.g. non-SN) calculi, we would like to know whether or not TOS could also work, and if not, we’d also like to know why TOS fails to work for them.

On the other hand, on the non-canonical normalisation of formal calculi, we are interested to find out if TOS works also for the weak-head normalisation. These are the open problems that wait for our future work.

Discussion on Approaches of Metatheoretic Proofs

Note that there are usually several different ways (or say, “flavours”) of studying the meta theoretic things for a typed system. On of them is to set up reduction-oriented semantics for untyped terms, and another is to set up judgemental equality-oriented semantics (or called typed operational semantics) for well-typed terms. And sometimes it is just a mix of usage of both ways. Here we principally want to work with the second method.

- Reduction between two raw terms (for example, the proofs of metatheories for ECC in [Luo94]).
- Typed operational semantics (for example, the proofs of metatheories for UTT in [Gog94], for LF in [Gog99], and for $F_w \leq$ in [CG97]).
- The approach to “relatively” prove metatheories for $LF + RType[L]$ with mapping RType to Sigma types.

More contents and open problems might include different ways of proving metatheory for "LF extended with DRTs", these are left to our future work:

- Way via typed operational semantics: as in this thesis, and based crucially on the soundness of the TOS.
- Way via Conservativity with Sigma-types: an alternative approach (but not replaceable with the above one). See also the part “Future Work”.
6.2 Future Work

In this section about future work to do, we will summarise from three aspects that the work has reflected difficult or interesting.

These three aspects are: first, the possibility of using DRTs to model Object-Oriented Programming; second, the TOS approach enlarged to more complicated type structures, such as TOS for inductive types and possible metatheoretic proof by conservativity; third, the TOS approach used for other not well-behaving type theories such as non-strongly normalizable type systems.

Applications of DRTs

We come up with some other conclusions to the possible applications regarding dependent record types presented in this thesis:

- “Classes and Interfaces as Types”.
  Is it possible to provide a type-theoretic foundation for the modern programming language such as the Object-Oriented Programming? If we see at the links between the two, there are two different understanding of types: the first, types are denotations, objects exist in the first place and types are assigned to them (examples: ML-like programming languages); the second, types are collections of canonical objects, types and their objects co-exist (examples: type systems like Martin-Löf’s TT and UTT). If we enlarge the second view, and intuitively see the classes and interfaces in OOP as types, and the instance of the class as the object of the type; the two paradigms coincide. Moreover, let the inheritance features of OOP be implemented through coercive subtyping. These are a basic sketch of the first application that DRTs could be associated. Conceptually, we need a model that is (1) coherent and clear in mapping every idea in concrete programming, and also (2) itself decidable, expressive and adequate for the concrete programming.

- Polymorphism: Unit type with structural subtyping. The reason why a class could be modelled as a type, is that a parameterised unit type “(A, a)” and a
coercion \((\xi) \text{“}(A, a) \rightarrow a\)" are used. Hence, a method \(M\) in one class could be represented by \((A, a)\), which could both stand for the class specification and for an instantiated object. Due to meta-properties of coercions, important OOP features (such as subtype polymorphism, dynamic dispatch, etc.) could be reflected. All the modelling uses intensional type theory which does not need extensional constructions, this could be seen as an atout for better satisfiability of meta-properties like decidability, and better machine-verifiably supported, etc. The difference between using usual manifest types and using coercions with intensional manifest fields in DRTs may be that, the latter could go closer to the implementation of dynamic typing / dynamic dispatching. If coercive functions could be generated run-time, the system is more powerful than an extensional type system with built-in \(\eta\)-like rules for manifests.

**TOS Approach for Other Type Theories**

A few problems are left open during the work of the thesis, which are:

- **Typed operational semantics to non-SN calculi**: For strong normalisation, we are interested to know further how TOS would succeed to capture canonical normalisation reductions via typing control. And we would also like to know whether TOS could work for ill-behaving (e.g. non-SN) calculi. For weak-head normalisation, we are interested to know whether TOS works for the WHN (non-canonical) of dependent record types.

- **Adequacy of type-theoretic model to object-oriented programming and module mechanism**: We put the question that, how is a type-theoretic model for object-oriented programming adequate? There is a possible way of judging it: take a formal semantics of object-oriented programming, e.g. a denotational semantics (which uses mathematical functions in specifications, unlike structural operational semantics as uses rules to specify the syntactical components); then study the relation between the formal semantics and our model. This is a way to see how it is reflected a model captures some features of object-oriented programming such as inheritance and dynamic dispatch.
6.2. Future Work

TOS for Inductive Types

We have also briefly tried to extend the TOS approach with more type-theoretic constructions, such as the inductive type schemata [Luo94] and the universes. We will give an example of TOS rules for inductive types $N$ and $\Sigma$ here.

**Definition 6.2.1 (Inductive Schemata)** Let $\Gamma$ be a valid context and $\bar{\Theta} \equiv (\Theta_1, ..., \Theta_n)$ ($n \in \omega$) a sequence of inductive schemata in $\Gamma$. Then, $\bar{\Theta}$ generates a $\Gamma$-type which is introduced by declaring the following constant expressions w.r.t. $\bar{\Theta}$:

\[
\begin{align*}
\mathcal{M}[\bar{\Theta}] & : \text{Type} \\
t_i[\bar{\Theta}] & : \Theta_i[\mathcal{M}[\bar{\Theta}]] \quad (i = 1, ..., n) \\
E[\bar{\Theta}] & : (C:(\mathcal{M}[\bar{\Theta}]) \text{Type}) \\
& \quad (f_1: \Theta_1^c[\mathcal{M}[\bar{\Theta}], C, t_1[\bar{\Theta}]] ... (f_n: \Theta_n^c[\mathcal{M}[\bar{\Theta}], C, t_n[\bar{\Theta}]])) \\
& \quad (z:\mathcal{M}[\bar{\Theta}])C(z)
\end{align*}
\]

(6.3)

where $E$ is called the elimination constant, and $\Theta_i^c$ is the kind constructor corresponding to inductive schemata sequence $\Theta_i$.

Natural Numbers could be defined with inductive schemata, where the eliminator is defined as $\text{Rec}_N = d f E_N[\bar{\Theta}_N]$ in the Definition 6.2.2.

\[
\begin{align*}
N & : \text{Type} \\
0 & : N \\
succ & : (N)N \\
\text{Rec}_N & : (C:(N)\text{Type})(c:C(0))(f:(x:N)(C(x))C(\text{succ}(x))) \\
& \quad (n:N)C(n)
\end{align*}
\]

**Definition 6.2.2 (Natural Numbers)** With $\bar{\Theta}_N \equiv (X, (X)X)$, the natural number type $N = d f \mathcal{M}[X, (X)X]$ is defined as:

\[
\begin{align*}
N & = d f \mathcal{M}[\bar{\Theta}_N] : \text{Type} \\
0 & = d f t_1[\bar{\Theta}_N] : N \\
succ & = d f t_2[\bar{\Theta}_N] : (N)N
\end{align*}
\]
\[ E_N : (C:(N)Type) \ \ (C(0)) \ ((n:N)(C(n))(C(succ(n)))) \ (z:N)C(z) \]

\[ \text{Rec}_N = \df C:Type \ [c:C] \ [f:(N)(C)C] \ E_N([n:N]C, c, f) \]

\( \Sigma \)-types are intuitively types for pairs (unlabelled records in some sense). They could be formally constructed in the logical framework LF, as done in [Luo94] as part of the composition of UTT. The \( \Sigma \)-types have such known properties:

- First, metatheories hold for \( \Sigma \)-types in a type theory constructed within LF, which could be directly proven using (untyped) reduction-oriented semantics.
- Second, however, the first and second projections for pairs have been found to be incoherent.

We could also define \( \Sigma \)-types with inductive schemata in the following Definition 6.2.3 [Luo94].

**Definition 6.2.3 (\( \Sigma \)-Type)** With \( \bar{\Theta}_{\Sigma(A,B)} \equiv (x:A)(B(x))X \), the strong sum type \( \Sigma = \df [A:Type][B:(A) Type]M[(x:A)(B(x))X] \) is defined as:

\[ \Sigma = \df M[\bar{\Theta}_{\Sigma(A,B)}] : (A:Type)(B:(A) Type) Type \]

\[ \text{pair} = \df \iota_1[\bar{\Theta}_{\Sigma(A,B)}] : (A:Type)(B:(A) Type)((x:A)B(x))\Sigma(A, B) \]

\[ E_{\Sigma} : (A:Type)(B:(A) Type)(C:(\Sigma(A, B)) Type) \]

\[ ((g:(x:A)B(x))C(\text{pair}(A, B, g))) \ (z:\Sigma(A, B))C(z) \]

\[ \pi_1 = \df [A:Type][B:(A) Type][z:\Sigma(A, B)] \]

\[ E_{\Sigma}(A, B, [z:\Sigma(A, B)]A, [x:A][y:B(x)]x, z) \]

\[ \pi_2 = \df [A:Type][B:(A) Type][z:\Sigma(A, B)] \]

\[ E_{\Sigma}(A, B, [z:\Sigma(A, B)]B(\pi_1(A, B, z)), [x:A][y:B(x)]y, z) \]

With a TOS for the general inductive schemata defined in [Gog94], the TOS for Natural Numbers could be defined by rules in the following Figure 6.1:
6.2. Future Work

Figure 6.1: TOS Canonical Forms and Weak-Head Reduction for Natural Numbers

Note that for $N = df \mathcal{M}[X,(X)X]$, $SCH_{T;X}(\Theta_N)$ actually means that $X \rightarrow_n X'$ and $X(X) \rightarrow_n X'(X')$, and by the rules set of TOS we know that $X \equiv X'$ and $X(X) \equiv X'(X')$ so among the premises some are omitted due to this admissibility.

The TOS rules for $\Sigma$-types could be defined in a similar manner in Figure 6.2:

---

$SCH^S_{T;X}(\Theta)$ means that $\vdash \Gamma, PSCH_X(\Theta)$ and $\Gamma, X:Type \vdash \Theta_i \rightarrow_n \Theta'_i$ for $1 \leq i \leq n$. $PSCH_X(\Theta)$ means that $\Theta$ well-formed schemata in UTT. We omit the full definition of the TOS for inductive schemata here, see [Gog94] for formal details.
6.2. Future Work

\[ \Gamma \vdash \Sigma \rightarrow_n \Sigma : (A:\text{Type})(B:(A)\text{Type})\text{Type} \quad (S - \Sigma) \]

\[ \Gamma \vdash \text{pair} \rightarrow_n \text{pair} : (A:\text{Type})(B:(A)\text{Type})((x:A)B(x))\Sigma(A,B) \quad (S - \text{pair}) \]

\[ \Gamma \vdash E_\Sigma \rightarrow_w E_\Sigma : (A:\text{Type})(B:(A)\text{Type})(C:(\Sigma(A,B))\text{Type}) \]
\[ ((g:(x:A)B(x))C(\text{pair}(A,B,g))) \]
\[ (z:\Sigma(A,B))C(z) \quad (S - E_\Sigma) \]

\[ \Gamma \vdash A : \text{Type} \quad \Gamma \vdash B : (A)\text{Type} \]
\[ \Gamma \vdash C : (\Sigma(A,B)\text{Type}) \quad \Gamma \vdash C(\text{pair}(A,B,z)) \rightarrow_n D \]

\[ \Gamma \vdash E_\Sigma(A,B,\text{pair}(A,B,z)) \rightarrow_w [z:\Sigma(A,B)]A : (\Sigma(A,B))A \quad (W - E_\Sigma - \pi_1) \]

\[ \Gamma \vdash A : \text{Type} \quad \Gamma \vdash B : (A)\text{Type} \]
\[ \Gamma \vdash C : (\Sigma(A,B)\text{Type}) \quad \Gamma \vdash C(\text{pair}(A,B,z)) \rightarrow_n D \]

\[ \Gamma \vdash E_\Sigma(A,B,\text{pair}(A,B,z)) \rightarrow_w [z:\Sigma(A,B)]B(\pi_1(A,B,z)) : (\Sigma(A,B))(A)B \quad (W - E_\Sigma - \pi_2) \]

Figure 6.2: TOS Canonical Forms and Weak-Head Reduction for Strong Sum Type

This definition would be useful for our future work on a relative proof on conservativity of Σ-types over dependent record types and the TOS approach applied on Σ-types. As mentioned in Section 2.1, there exists an indirect method in the treatment of metatheories, or called the relative proof: for a type system \(T_1\), if one has got its reduction-oriented semantics and proven certain meta-properties; and also for a target type system \(T_2\), one could develop a mapping from constructions in \(T_1\) to \(T_2\), and prove that the meta-properties preserve; then, by the relative proof one gets meta-properties for \(T_2\).

In this way, if we could build a mapping from DRTs to the Σ’s, and try to prove the conservativity in TOS. Then we could similarly get metatheories for DRTs proven. This relative proof approach is quite of our interest, and we are currently on research for establishing the mappings from DRTs to Σ-types, and a possible relative proof of the metatheories for DRTs via those for Σ-types.
Bibliography


Z. Luo. On Subtyping for Type Theories with Canonical Objects. Seminar talk in Fun in the afternoon, Feb 2010.


[Wey1918] H. Weyl. *Das Kontinuum.* Kritische Untersuchungen über die Grundlagen der Analysis, Leipzig, 1918. (Translated as [Wey87])
