The Curry-Howard Isomorphism
Project Proposal

Chuangjie Xu (Student ID: 1061493)
Supervisor: Prof. Achim Jung

Project Aims

This is a dissertation-only project.

The *Curry-Howard Isomorphism* refers to the correspondence between systems of formal logic as encountered in *proof theory* and computational calculi as found in *type theory*. The simplest example should be that of intuitionist implicational logic and simply typed lambda calculus.

A great deal of questions about this isomorphism may be asked. For instance, there are various extensions of lambda calculus; therefore, can the corresponding systems of logic be always defined? Apart from logic and types, can such correspondence between them and that of other theories be found?

The purpose of this project is to study the foundations of the Curry-Howard Isomorphism, probe into one of its many extensions.

Lambda Calculus

The *lambda calculus* (λ-calculus) is a family of prototype programming languages. Their main feature is that they are *functional* and *higher-order*.

The *type free λ-calculus*, the simplest of these languages, studies functions and their applicative behavior. Therefore, its first primitive operation is *application*. The expression FA denotes the function F applied to the argument A. Another basic operation is *abstraction*. Let $M \equiv M[x]$ be an expression possibly containing ('depending on') x. Then $\lambda x.M[x]$ denotes the function $x \mapsto M[x]$.

A *λ-term* can be an atom (term-variable), an application, or an abstraction, i.e.

$$M \quad ::= \quad x \;\mid\; MM \;\mid\; \lambda x.M$$

where M is a λ-term and x is a term-variable.

There are three types of *equivalences* in type free λ-calculus:

- *α-conversion*: changing bound variables, i.e. $\lambda x.M =_\alpha \lambda y.M[x := y]$;
- *β-conversion*: applying functions to their arguments, i.e. $(\lambda x.M)N =_\beta M[x := N]$;
- *η-conversion*: extensionality, i.e. $\lambda x.Mx =_\eta M$ where x is not free in M.

Simple Types

The *simply typed lambda calculus* ($\lambda^{\to}$) is a typed interpretation of the lambda calculus with only one type constructor $\to$ that builds function types.

A *type* can be an atom (type-variable) or a composite type ($\sigma \to \tau$, where both σ and τ are types). Types can be assigned to λ-terms. A *type-assignment* is any expression $M{:}\,\tau$

where M is a λ-term and τ is a type.

The *syntax* of $\lambda^{\rightarrow}$ is essentially that of the λ-calculus itself:

$$M \quad ::= \quad x \quad | \quad MM \quad | \quad \lambda x{:}\tau.M$$

where M is a term in $\lambda^{\rightarrow}$, x is a term-variable and τ is a type.

Typing rules for $\lambda^{\rightarrow}$:

- If x is a variable and τ is a type, then $x{:}\tau$ has type τ;
- $(\rightarrow \mathrm{E})$ If M has type $\sigma \rightarrow \tau$ and N has type σ, then application MN has type τ;
- $(\rightarrow \mathrm{I})$ If term M has type τ and variable x has type σ, then abstraction $\lambda x{:}\sigma.M$ has type $\sigma \rightarrow \tau$.

Intuitionistic Logic

Intuitionistic Logic, also called Constructive Logic, is a symbolic logic system which differs from classical logic. In Intuitionistic Logic, the judgments about statements are based on the existence of a proof or "construction" of that statement.

The language of intuitionist propositional logic is same as the one of classical propositional logic. The set Φ of formulas is defined by induction, represented by the following grammar:

$$\Phi \quad ::= \quad \bot \quad | \quad FV \quad | \quad \neg\Phi \quad | \quad (\Phi \wedge \Phi) \quad | \quad (\Phi \vee \Phi) \quad | \quad (\Phi \rightarrow \Phi)$$

where FV is an infinite set of propositional variables.

Syntactically, the law of excluded middle ($\vdash \varphi \vee \neg\varphi$) is not an axiom in intuitionistic logic; therefore, all the rules of natural deduction for the connectives $\bot$, $\neg$, $\wedge$, $\vee$, $\rightarrow$ are adopted except the rule RAA (the "reductio ad absurdum" rule).

Extensions

1. Enriching the correspondence

Such amazing correspondence can also be seen with other theories. Take the Curry-Howard-Lambek Correspondence for example, there is a three-way-isomorphism between intuitionistic logic, typed lambda calculus and Cartesian closed categories, with objects being interpreted as types or propositions and morphisms as terms or proofs.

2. Enriching the type theory

With the enriching of the typed lambda calculus, the correspondence between them and higher order logic can be found. To illustrate, dependent types corresponds to first-order logic, and polymorphic types corresponds to second-order logic, etc.

3. Extending the correspondence to encompass dynamical aspects of the λ-calculus

Basically, λ-terms correspond to constructions in intuitionistic logic. When β-reduction is introduced to λ-calculus, it is obvious that it corresponds to the normalization in systems of logic. From this part of view, there is a correspondence between the "dynamics" of λ-calculus and strong normalization in proof theory.

4. Applications of the Curry-Howard Isomorphism to computing

The Curry-Howard Isomorphism is also known as "Proofs as Programs Correspondence". That's because constructive proofs have computational meanings. This mathematical theorem has been applied in computing, such as programming and computable analysis.

Draft "Table of Contents" for the Dissertation

1. Introduction
2. The basic correspondence
   2.1 Intuitionist implicational logic
   2.2 Simply typed λ-calculus
   2.3 The Curry-Howard Isomorphism
3. Extension (one of the four mentioned above)

   …

N. References

List of Literature to Study

- A. Jung. *A Short Introduction to the Lambda Calculus*.
- H. P. Barendregt. *The lambda calculus: its syntax and semantics*.
- J. R. Hindley. *Basic simple type theorem*.
- M. Huth & M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*.
- D. van Dalen. *Logic and Structure*.
- M. Barr & C. Wells. *Category Theory for Computer Science* (2nd edition).
- The further literature would be discussed with the supervisor.

Timetable & Milestones

Week / Event	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Study extensions																								
Write Part 1 & 2																								
Find problems																								
Inspection																								
Solve problems																								
Holiday																								
Write rest parts																								
Edit dissertation																								
Presentation																								