Reduction as Deduction

John Darlington Yi-ke Guo Martin Kohler
Department of Computing
Imperial College
180 Queen's Gate, London SW7 2BZ, U.K.
E-mail: jd, yg,mk@doc.ic.ac.uk

Functional Languages Implementation Workshop. Paper 10

1 Introduction

In this paper. we present a logical encoding of reduction strategies of lambda calculus in terms of linear logic. First of all, we present a fragment of intuitinistic linear logic to formalise the behaviours of concurrent computation. The logic, which can be regarded as a process calculi, is then used as a meta language to specify the behaviors of various reduction strategies of the calculus. The encoding maps a lambda expressions into linear logic formulae by taking into account its computational behaviours. With such encoding, reductions are sysmatically mappped into a uniform framework of linear deduction.

2 A Linear Logic Model of Computation

Linear logic was proposed by Girard as a logic for modelling computational systems [Gir87]. In linear logic, the structural rules for weakening and contraction are dropped. Therefore, it is not possible to copy or discard arbitrary formulae during a deduction. Logical formulae become resources which may be consumed by deduction. This property makes the logic ideal for specifying computational systems, and particularly concurrent computation. In the appendix, we present linear logic by its sequent system where $\multimap$ is adopted as a basic logical operation rather than a derived operation. A comprehensive description of linear logic can be found in [Gir87].

Consider the sequent $\Gamma \vdash \Delta$ as consuming the resource of Γ to meet the requirement of Δ. A linear proof can be regarded as a *resource construction/consumption* procedure where the root of the proof tree describes the initial state of a computation which proceeds by expanding the frontier of the partially constructed tree. The collection of leaves presents the current state of the computation. Following this view, the multiset of formulae Γ in a sequent $\Gamma \vdash \Delta$ is viewed as a multiset of resources. These resources may be transformed into new resources by applying the inference rules (left-rules). Since there has no sequentiality imposed on transforming these resources, the formulae in Γ can be regarded as concurrent processes. Logic operators (connectives and quantifiers) become combinators of the processes. Regarding each sequent as a state of the computation, the operational semantics of these combinators are provided by identifying state transitions of the computation with the inferences of the logic. The proof tree as a whole records the history of computation.

Thus, we can select an appropriate fragment of the logic as a process calculus. From a logical point of view, any statement of the calculus is a logical formula, whereas, from a computational point of view, a formula is a process. The concurrent computation of the language is modelled by deduction in the logic. The following analogies arise:

Logic		**Computation**
Formula	$\Longleftrightarrow$	**Process**
Sequent	$\Longleftrightarrow$	**Configuration**
Deduction	$\Longleftrightarrow$	**Concurrent Computation**

Following this principle of "formulae as processes, proofs as computations", basic constructions for concurrent computation can be straightforwardly related to an operational interpretation of linear deduction. A detailed description can be found in [Guo93].

We will now present a logical process calculus as a meta language for the operational behaviour of the λ-calculus. It is based on the intuitionistic linear logic [Abr90]. We first present the syntax of the language.

Let A, x, t, c be the syntactical variables ranging over agents, channel symbols, terms and conditional expressions respectively. The language has the following syntax:

A	$\triangleq$	1	unit
	$\mid$	$x(t)$	atomic communication
mid	$c \multimap A$	condition	
	$\mid$	$\forall \bar{x}.\ m \multimap A$	communicating method
	$\mid$	$A \otimes A$	parallel composition
	$\mid$	$A \& A$	choice
	$\mid$	$\exists x.A$	hiding
	$\mid$	$!A$	storing

where the syntactical variable m ranges over a set of *templates* defined as

m	$\triangleq$	$x(t)$	receiver
	$\mid$	$m \otimes m$	multiple receiver

A condition c in an implication $!c \multimap A$ is generally a preinterpreted formula. We assume that a built-in theory to define the meaning of the symbols of c.

Communicating methods are generally of the form

$$(\forall \overline{x}.p_1(\overline{x_1}) \otimes \ldots \otimes p_n(\overline{x_n}) \multimap A).$$

They are also called *multiheaded clauses* and the atomic propositions $p_i(\overline{x_i})$ are called the *heads* of the clause. Computation is to take these methods as rewriting rules to reduce a multiset of formulae into a store of consistent bindings. Each application of a method forms a *multiset rewriting step*.

The concurrency of the computation is reflected by using the multiplicative conjunction $\otimes$ as an operator for parallel composition. Its deduction rule

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \otimes B \vdash \Delta}$$

can be understood as decomposing the resource $A \otimes B$ into the resources A and B in the context Γ.

Consider the deduction rule $(\& \vdash)$:

$$(\& \vdash 1) \quad \frac{\Gamma, A \vdash \Delta}{\Gamma, A \& B \vdash \Delta} \qquad (\& \vdash 2) \quad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \& B \vdash \Delta}$$

The deduction can be regarded as choosing one of the component of $A \& B$ to verify the sequent within the context Γ. Since computation is to search for a successful proof, the choice may be influenced by other formulae in Γ (i.e. the environment). This corresponds to the "external choice" in concurrent programming [BG90]. Thus, don't care non-determinism (committed-choice), for which the arrival nondeterminism is a special case, can be uniformly modelled in terms of additive conjunction.

Communication between processes is realized by passing messages, as outlined in the following deduction:

$$\cfrac{\cfrac{\cfrac{\Gamma, B, A[t_1/y] \vdash \Delta}{\Gamma, B, x(t_1), x(t_1) \multimap A[t_1/y] \vdash \Delta}(\multimap \vdash)}{\Gamma, B, x(t_1), \forall y.\ x(y) \multimap A \vdash \Delta}(\forall \vdash)}{\Gamma, B \otimes x(t_1), \forall y.\ x(y) \multimap A \vdash \Delta}(\otimes \vdash)$$

The formula $x(t_1)$ can be understood as the sending of a message (t_1) via a channel x. Thus, the formula $B \otimes x(t_1)$ sends the message $x(t_1)$ and then behaves like B. The formula $\forall y.\ x(y) \multimap A$ receives the message

$x(t_1)$ and behaves like $A[t_1/x]$. In general, synchronization between processes in the calculus is modelled by linear modus ponens. Consider the following deduction rule for linear implication:

$$\frac{\Gamma_1 \vdash A, \Delta_1 \qquad \Gamma_2, B \vdash \Delta_2}{\Gamma_1, \Gamma_2, A \multimap B \vdash \Delta_1, \Delta_2}$$

A special case of the inference rule is that the succedent of the lower sequent (the global requirement of computation) isn't split during deduction. That is:

$$\frac{\Gamma_1 \vdash A \qquad \Gamma_2, B \vdash \Delta}{\Gamma_1, \Gamma_2, A \multimap B \vdash \Delta}$$

The deduction step can be interpreted as transforming the current state, represented as $\Gamma_1, \Gamma_2, A \multimap B \vdash \Delta$, to the state $\Gamma_2, B \vdash \Delta$ whenever the local requirement condition A is met by Γ_1 (i.e. $\Gamma_1 \vdash A$) [1]. Then the deduction, called *linear modus ponens*, can be presented as:

$$\frac{\Gamma_2, B \vdash \Delta}{\Gamma_1, \Gamma_2, A \multimap B \vdash \Delta} \qquad \text{if } \Gamma_1 \vdash A$$

With this interpretation, linear modus ponens forms a uniform synchronisation mechanism where the implication $A \multimap B$ can be regarded as a process which will become B whenever the environment contains enough resources to meet the synchronisation condition (local requirement) represented by the premise A. Synchronisation is logically modelled as verifying the entailment between the resource Γ_1 and the local requirement A. In the case of communication, the local requirement A can be regarded as a *message pattern* waiting for matching the corresponding messages. Thus, linear modus ponens provides a concise means to model asynchronous communication.

The condition c in a formula $!c \multimap A$ states the synchronization condition between the environment and A. A can proceed only when c is justified with respect to recently computed variable bindings. The effect can be written as $\vdash_S c$ where S denotes the built-in theory asserting the truth of the formula c (Thus, the theory should be represented in classical logic). The modality ! allows the formula c to be accepted by the built-in theory in classical logic.

Variables local to a process can be introduced via existential quantification, because an existentially quantified variable can not be free outside the scope of $\exists$.

$$\frac{\Gamma, A}{\Gamma, \exists x.A}\ x \text{ is not free in } \Gamma$$

The following derived rule, called CD because it combines contraction and dereliction, indicates that $!A$ can be understood as storing the process A:

$$\frac{!A, A, \Delta \vdash \Gamma}{!A, \Delta \vdash \Gamma}\ \mathsf{CD-Rule}$$

The inference is read as "making a copy of A and putting it into the environment". In the calculus, we use ! to store resources.

The operational model of the process calculus is given as a special form of deduction in linear logic. It can be presented as a transition system where a configuration Γ is a multiset of agents. We use Con to denote the set of all configurations. A derivation relation $\xrightarrow{c} \subseteq Con \times Con$ represents the transition between configurations, where $\vartheta(\Gamma)$ denotes the free variables in the agents Γ.

The rules of the transition system are defined as follows:

$$\Gamma, 1 \xrightarrow{c} \Gamma,$$

$$\Gamma, A \otimes B \xrightarrow{c} \Gamma, A, B$$

$$\Gamma, A \& B \xrightarrow{c} \Gamma, A$$

$$\Gamma, A \& B \xrightarrow{c} \Gamma, B$$

$$\Gamma, !c \multimap A \xrightarrow{c} \Gamma, A \qquad \text{if } \vdash_S c$$

$$\Gamma, x_1(t_1), x_2(t_2), \ldots, x_n(t_n), (\forall \bar{y}.\ \otimes_{i=1}^{n} x_i(y_i) \multimap A) \xrightarrow{c} \Gamma.A[t_i/y_i]$$

[1] The interpretation imposes a sequentiality of verifying the left branch first.

$$\Gamma.\exists x.A \xrightarrow{c} \Gamma.A \qquad x \notin \vartheta(\Gamma)$$
$$\Gamma, !A \xrightarrow{c} \Gamma, A, !A$$

For any initial formula A, the initial configuration is $\Gamma_0 : \{A\}$. The computation transforms the initial configuration into a simpler system Γ_n such that no more communication is possible. The transition rules are deduction steps in linear logic. As proved in [DG92], the transition system realizes a special form of proofs in linear logic. The computation is complete in the sense that it constructs a proof that is canonical for all possible evaluations of A. The soundness and completeness results can be summarized in the following theorem where we use $\vdash_{\mathcal{L}}$ to denote the provability in linear logic[2].

Theorem 2.1 *For any agent A, a term t and $x \in \vartheta(A)$, $A \vdash_{\mathcal{L}} x(t)$ iff there exists a derivation $\Gamma_0 : \{A\} \xrightarrow{c}{}^{*} \Gamma_n$ such that $\Gamma_n \vdash_{\mathcal{L}} x(t)$*

3 Encoding Call by Name Reduction in Linear Logic

In this section, we present a simple linear logic encoding of the call by name reduction model of λ-calculus by encoding each expression into a linear logic formula. Reduction in λ-expressions thus corresponds to deduction of the formulae. With this encoding, a call by name evaluation model is presented within the linear deduction framework. The encoding of call by name lambda reduction is:

$$\begin{array}{lcl}
[x]^{\circ}_{y} & = & x(y) \\
[(\lambda x.M)]^{\circ}_{z} & = & \forall x, y.(z(x,y) \multimap [M]^{\circ}_{y}) \\
[(MN)]^{\circ}_{z} & = & \exists x, y.([M]^{\circ}_{x} \otimes x(y,z) \otimes [y := N]^{\circ}) \\
 & & x, y \text{ is not free in } M, N \\
[x := N]^{\circ} & = & !(\forall m.x(m) \multimap [N]^{\circ}_{m})
\end{array}$$

According to the concurrent semantics of the calculus, the atomic formula $x(y)$ is read as "sending the channel name y on the channel x". The atomic formula $x(y,z)$ is understood as the transmission of a tuple (y,z) along the channel x. The binding $y := N$ can be interpreted as adding an entry into an environment, which is built up during the computation and contains a set of communication methods.

The intuition underlying our translation is to model λ-expressions directly as communicating processes, i.e. the formula $[e]^{\circ}_{x}$ is interpreted as a process which computes e and associates the channel x with the result of the computation.

Function as Servers: The translation of functions (λ-abstractions)

$$[\lambda x.M]^{\circ}_{z} \quad = \quad \forall x, y.\ z(x,y) \multimap [M]^{\circ}_{y}$$

associates the abstraction with the channel z and translates it into a communication method $\forall x, y.\ z(x,y) \multimap [M]^{\circ}_{y}$. This formula behaves like a server waiting for a message to arrive along the channel z indicating the name of the channel along which the argument of the function will be passed when the function is applied and the name of the channel to where the result will be sent. Thus, in this translation, function names, variables and communication channels are identified. This approach directly follows Milner's translation of the λ-calculus into the π-calculus [MPD89]. β-reduction corresponds to a communication which is realised by a combination of universal instantiation and linear modus ponens as indicated by the following deduction:

$$\dfrac{\dfrac{\Gamma, [M[u/x]]^{\circ}_{v} \vdash \Delta}{\Gamma, z(u,v), z(u,v) \multimap [M[u/x]]^{\circ}_{v} \vdash \Delta}\,(\multimap \vdash)}{\Gamma, z(u,v), (\forall x, y.\ z(x,y) \multimap [M]^{\circ}_{y}) \vdash \Delta}\,(\forall \vdash)$$

Following the identity of function names and communication channels, the formula $z(u,v)$ can be understood as either sending the tupled channel names (u,v) via the channel z, or a function call formed by applying function z to u with channel v transmitting the result. Combined, they allow the deduction of a formula that represents the body of the original λ-abstraction, with the argument identifier x substituted by the argument name u. This substitution creates a specific instance of the function body M which is equal in meaning to $[M[u/x]]^{\circ}_{v}$. The result of this specialised function body will therefore be computed using u and the result will be transmitted along the channel v, as desired. The deduction shows that linear modus ponens provides a logical interpretation of λ-rewriting.

[2] More precisely, we assume linear logic integrated with a built-in theory for justifying conditions.

Application as Parallel Composition: The translation of function application:

$$[(MN)]^{\circ}_{z} \quad = \quad \exists x, y.([M]^{\circ}_{x} \otimes [y := N]^{\circ} \otimes x(y, z))$$

takes the behaviour of call by name reduction into account. Two intermediate names x, y are created for private channels that will be used as a function name and for the transmission of the argument, respectively. The formulae for the function and the argument expression are linked via parallel composition but the evaluation of the argument expression is suspended and stored in a "logical environment" until a demand is received. The third formula: $x(y, z)$ "feeds" the name y of the argument into the function x and ensures that the result of the function application is sent onto the channel z. Moreover, the argument N of the application is associated with the name y as its environment entry. It will only be evaluated if and when its value is required for the computation of the result of the application. The demand for evaluation is propagated by sending to the environment the channel on which the value of N is expected. Thus, an environment entry can be logically formalised as:

$$[y := N]^{\circ} = !(\forall m.y(m) \multimap [N]^{\circ}_{m})$$

which can be understood as a suspension for the expression N, waiting for a demand transmitted on channel y. The demand has the form of a message $y(m)$ where m is the channel over whichthe result of N should be transmitted. The view of an environment as a set of suspensions is consistent with the notion of environments in the lazy functional programming context where a suspended computation is encoded as a stored closure. Note that whenever the argument is needed, it will be copied and be reevaluated. Thus, the encoding reflects the "call-by-name" mechanism for parameter passing.

The following translation theorem (theorem 3.0.1), which is proved in [Guo93], states the soundness of the encoding.

Theorem 3.0.1 (Translation Theorem) *Let e_1, e_2 be two λ-expressions and the relation $\rightarrow_{\beta}$ denote a call by name β-conversion step. Then:*

$$e_1 \rightarrow_{\beta} e_2 \qquad \Rightarrow \qquad [e_1]^{\circ}_{z} \vdash_{LL} [e_2]^{\circ}_{z}$$

4 A General Encoding of Lambda Reduction Models

A general encoding of lambda reduction models is presented as follows:

$[x]^{\circ}_{z} = x(z)$
Name passing

$[(\lambda x.M)]^{\circ}_{z} = \exists f.z(f) \otimes !(\forall x, y.(f(x, y) \multimap [M]^{\circ}_{y}))$
Abstraction as server

$[x := N]^{\circ} = !(\forall m.x(m) \multimap [N]^{\circ}_{m})$
Stored environment entry

$[(MN)]^{\circ}_{z} = \exists x, y, u.([N]^{\circ}_{u} \otimes (\forall v.u(v) \multimap [M]^{\circ}_{x} \otimes (\forall f.x(f) \multimap f(y, z) \otimes [y := v]^{\circ})))$
Call by value reduction

$[(MN)]^{\circ}_{z} = \exists x, y.([M]^{\circ}_{x} \otimes [y := N]^{\circ} \otimes (\forall f.x(f) \multimap f(y, z)))$
Call by name ction

$[(MN)]^{\circ}_{z} = \exists x, y, u.([M]^{\circ}_{x} \otimes (\forall f.x(f) \multimap f(y, z) \otimes \forall r.(y(r) \multimap [N]^{\circ}_{u} \otimes \forall v.(u(v) \multimap r(v) \otimes [y := v]^{\circ}))))$
Call by need reduction

$[(MN)]^{\circ}_{z} = \exists x, y, u.([M]^{\circ}_{x} \otimes [N]^{\circ}_{u} \otimes (\forall f, v.x(f) \otimes u(v) \multimap f(y, z) \otimes [y := v]^{\circ}))$
Data flow Model

References

[Abr90] Samson Abramsky. Computational Interpretation of Linear Logic. Technical report, Dept. of Computing, Imperial College, Oct. 1990.

[BG90] Gerard Boudol and G.Berry. The chemical abstract machine. In *Proc. of the 17th Annual ACM symposium on Principle of Programming Languages.* ACM, 1990.

[DG92] J. Darlington and Y. Guo. Definitional constraint programming for parallel computing: An introduction. In *Proc. of the ICOT workshop on Future Direction of Parallel Programming and Architecture*, June 1992. ICOT TM-1185.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1), 1987.

[Guo93] Yike Guo. *Definitional Constraint Programming.* PhD thesis, Dept. of Computing, Imperial College, December 1993.

[MPD89] R. Milner, J. Parrow, and D.Walker. A calculus of mobile processes. Technical report, Dept. of Computer Science, University of Edinburgh, 1989.

A Linear Logic

Linear logic is presented in Figure 1 in the form of a Gentzen-style sequent calculus. Since the structural rules for weakening and contraction are abolished in linear logic, a collection of formulae in a deduction should be regarded as a multiset rather than a set. The feature of resource-consciousness results from the linearity of the proofs which requires that each assumption must be used exactly once. For a sequent $\Gamma_1..\Gamma_n \vdash \Delta_1..\Delta_m$ the assumptions Γ_i can be viewed as resources and the conclusions Δ_j as requirements that have to be met by spending the given resources.

The linearity necessitates two different versions of conjunction: the *multiplicative conjunction* (`tensor`, $\otimes$) and the *additive conjunction* (`with`, &). If the multiplicative conjunction $\otimes$ is used in an assumption Γ_i of a sequent, then this reflects the fact that both components of the conjunction must make contributions to the proof derivation within the same environment. On the other hand, the use of the additive conjunction & reflects the fact that once the formula is used in the proof, either the first or the second component must be chosen for the further derivation. Dually, two versions of disjunction are required. The *multiplicative disjunction* (`par`, @) causes the splitting of its environment, whereas *additive disjunction* (`plus`, $\oplus$) causes the environment to be copied for the derivation of both components.

Another important connective is *linear implication* ($\multimap$), which can be used to localize inference. It can be derived by the logical equivalence $A \multimap B = A^{\perp}@B$ where $A^{\perp}$ is the linear negation of A. Each sequent $\Gamma_1..\Gamma_n \vdash \Delta_1..\Delta_m$ has the intended meaning that $\otimes_{i=1}^{n}\Gamma_i \multimap @_{j=1}^{m}\Delta_j$ is valid.

The "of course" modality (!) is introduced to allow a controlled form of weakening and contraction in the language. Its dual (?) is called "why not".

Due to the symmetry of linear logic, the same multiset of formulae can be treated either as assumptions, using conjunction, or as consequences, using disjunction. Here, we concentrate on the interpretation of formulae as resources and there only deal with the rules for the left-hand side of sequents.

For a comprehensive description of linear logic see [Gir87].

Axiom:

$$\text{(Id)}\quad \frac{}{A \vdash A}$$

Cut :

$$\frac{\Gamma_1 \vdash G, \Delta_1 \qquad \Gamma_2, G \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}$$

Structure Rule:

$$\text{(Exchange Left)}\quad \frac{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Gamma_1, B, A, \Gamma_2 \vdash \Delta} \qquad \text{(Exchange Right)}\quad \frac{\Gamma \vdash \Delta_1, A, B, \Delta_2}{\Gamma \vdash \Delta_1, B, A, \Delta_2}$$

Logic Rules :

$(1 \vdash)$	$\frac{\Gamma \vdash \Delta}{\Gamma, 1 \vdash \Delta}$	$(\vdash 1)$	$\frac{}{\vdash 1}$
$(0 \vdash)$	$\frac{}{\Gamma, 0 \vdash \Delta}$	$(\vdash \top)$	$\frac{}{\Gamma \vdash \top, \Delta}$
$(\bot \vdash)$	$\frac{}{\bot \vdash}$	$(\vdash \bot)$	$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}$
$(^\perp \vdash)$	$\frac{\Gamma \vdash A, \Delta}{\Gamma, A^\perp \vdash \Delta}$	$(\vdash^\perp)$	$\frac{\Gamma, A \vdash \Delta}{\Gamma \vdash A^\perp, \Delta}$
$(\otimes \vdash)$	$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \otimes B \vdash \Delta}$	$(\vdash \otimes)$	$\frac{\Gamma_1 \vdash A, \Delta_1 \qquad \Gamma_2 \vdash B, \Delta_2}{\Gamma_1, \Gamma_2 \vdash A \otimes B, \Delta_1, \Delta_2}$
$(\& \vdash)$	$\frac{\Gamma, A \vdash \Delta}{\Gamma, A \& B \vdash \Delta}, \frac{\Gamma, B \vdash \Delta}{\Gamma, A \& B \vdash \Delta}$	$(\vdash \&)$	$\frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \& B, \Delta}$
$(@ \vdash)$	$\frac{\Gamma_1, A \vdash \Delta_1, \qquad \Gamma_2, B \vdash \Delta_2}{\Gamma_1, \Gamma_2, A @ B \vdash \Delta_1, \Delta_2}$	$(\vdash @)$	$\frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A @ B, \Delta}$
$(\oplus \vdash)$	$\frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \oplus B \vdash \Delta}$	$(\vdash \oplus)$	$\frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \oplus B, \Delta} \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \oplus B, \Delta}$
$(\multimap \vdash)$	$\frac{\Gamma_1 \vdash A, \Delta_1 \qquad \Gamma_2, B \vdash \Delta_2}{\Gamma_1, \Gamma_2, A \multimap B \vdash \Delta_1, \Delta_2}$	$(\vdash \multimap)$	$\frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \multimap B, \Delta}$
(Contraction !)	$\frac{\Gamma, !A, !A \vdash \Delta}{\Gamma, !A \vdash \Delta}$	(Contraction ?)	$\frac{\Gamma \vdash ?A, ?A, \Delta}{\Gamma \vdash ?A, \Delta}$
(Weakening !)	$\frac{\Gamma \vdash \Delta}{\Gamma, !A \vdash \Delta}$	(Weakening ?)	$\frac{\Gamma \vdash \Delta}{\Gamma \vdash ?A, \Delta}$
(Promotion ?)	$\frac{!\Gamma, A \vdash ?\Delta}{!\Gamma, ?A \vdash ?\Delta}$	(Promotion !)	$\frac{!\Gamma \vdash A, ?\Delta}{!\Gamma \vdash !A, ?\Delta}$
(Dereliction !)	$\frac{\Gamma, A \vdash \Delta}{\Gamma, !A \vdash \Delta}$	(Dereliction ?)	$\frac{\Gamma \vdash A, \Delta}{\Gamma \vdash ?A, \Delta}$
$(\exists \vdash *)$	$\frac{\Gamma, A \vdash \Delta}{\Gamma, \exists x.A \vdash \Delta}$	$(\vdash \exists)$	$\frac{\Gamma \vdash A[t/x], \Delta}{\Gamma \vdash \exists x.A, \Delta}$
$(\forall \vdash)$	$\frac{\Gamma, A[t/x] \vdash \Delta}{\Gamma, \forall x.A \vdash \Delta}$	$(\vdash \forall *)$	$\frac{\Gamma \vdash A, \Delta}{\Gamma \vdash \forall x.A \vdash \Delta}$

* x is not free in lower sequent.

Figure 1: Linear Logic