Substructural Calculi with Dependent Types[*] (Extended Abstract)

Zhaohui Luo[†]

Royal Holloway, Univ of London
zhaohui.luo@hotmail.co.uk

1 Introduction

In this paper, we investigate how to introduce dependent types into the substructural calculi such as the Lambek calculus [8] and linear logic [6]. The motivations of such a move include facilitating a closer correspondence between syntax and semantics in natural language analysis, especially when modern type theories are employed in formal semantics [14, 10, 2], and developing promising applications such as that to concurrency through dependent session types (cf. [5] among others).

Combining resource sensitive types with dependent types has been an interesting but difficult research topic and many researchers have worked on this, mainly with motivations to apply such calculi to programming and verification problems in computer science (see early work such as [1] and recent developments such as [7, 16]). However, how such a combination should be done is still widely open, on the one hand, and very much depends on the motivations in their applications, on the other.

We shall present two substructural calculi with dependent types: the first containing dependent Lambek types and the second dependent linear types. Technically, the former adheres to the usual assumption that types do not depend on substructural variables (in this case, the Lambek variables), which makes the technical development easier, while the latter allows type dependency on linear variables, which makes the development more challenging as well as more interesting in applications.

2 Dependent Lambek Types

Dependent types have been used in various contexts in computational linguistics (see, for example, [14, 3, 12] among others). In [15] that formalises the Lambek calculus in (a proof assistant that implements) Martin-Löf's type theory, Ranta discussed the idea of introducing type dependency into directed types and gave an inspiring example to represent directed types of quantifiers, although the paper did not formally propose such an extension. De Groote et al [4] studied how to extend the underlying type system for

[*]Linearity & TLLA (affiliaed with FLoC), Oxford. 2018.
[†]Partially supported by EU COST Action CA15123.

ACGs by (intuitionistic) Π-types so that type families can be used to represent syntactic categories indexed by linguistic features, but they did not study substructural Π-types.

We present a Lambek calculus with dependent types. Besides the type constructors in the Lambek calculus [8], we shall introduce directed types for dependent products (Π^r and Π^l) and dependent sums ($\Sigma^\sim$ and Σ°). Also considered is how to introduce type universes into the calculus. These are some of the type constructors in a modern type theory found useful in formal semantics and, therefore, their introduction at the syntactic level helps to facilitate a closer syntax-semantics tie as mentioned above. A focus of the current paper is to formulate the rules for these types in ordered contexts so that their meanings are correctly captured.

2.1 The Lambek Calculus

Introducing dependent types into a Lambek calculus, we consider a calculus with contexts having two parts:

$$\Gamma;\ \Delta$$

where Γ is an intuitionistic context (whose variables can be used for any times in a term) and Δ is a Lambek context (or ordered context). Types may only depend on ordinary variables in Γ, but not on Lambek variables in Δ.[1] Since we are going to introduce dependent types in which objects may occur, we need the following equality typing rule which says that computationally equal types have the same objects:

$$\frac{\Gamma;\Delta \vdash a : A \quad \Gamma;\ * \vdash A = B}{\Gamma;\Delta \vdash a : B}$$

Contexts of the above form obey the following validity rules where, in the last two rules, $x \notin FV(\Gamma, \Delta)$:

$$\frac{}{*;*\ valid} \qquad \frac{\Gamma;\Delta\ valid \quad \Gamma;\ * \vdash A\ type}{(\Gamma, x{:}A);\ \Delta\ valid} \qquad \frac{\Gamma;\Delta\ valid \quad \Gamma;\ * \vdash A\ type}{\Gamma;\ (\Delta, x{:}A)\ valid}$$

For variables, we have

$$\frac{\Gamma, x{:}A, \Gamma';\ *\ \ valid}{\Gamma, x{:}A, \Gamma';\ * \vdash x : A} \quad \frac{\Gamma;\ y{:}A\ valid}{\Gamma;\ y{:}A \vdash y : A}$$

We can now present the directed types in the Lambek calculus. The rules for the directed type B/A are given in Figure 1. The directed type $A \setminus B$ has the term constructor $\backslash x{:}A.b$ and its rules are symmetric with those for A/B and are omitted (and so are the rules for ordered conjunctions).

[1] As mentioned in the Introduction, this design choice follows most of the existing work on introducing dependent types into resource sensitive calculi and makes the resulting calculus simpler to study. This is not the case anymore when we consider dependent linear types in §3.

$$(/\text{-F})\ \frac{\Gamma;\ * \vdash A\ \mathit{type} \quad \Gamma;\ * \vdash B\ \mathit{type}}{\Gamma;\ * \vdash B/A\ \mathit{type}} \qquad (/\text{-I})\ \frac{\Gamma;\ (\Delta, x{:}A) \vdash b : B \quad \Gamma;\ * \vdash B/A\ \mathit{type}}{\Gamma;\ \Delta \vdash /x{:}A.b : B/A}$$

$$(/\text{-E})\ \frac{\Gamma;\ \Delta_1 \vdash f : B/A \quad \Gamma;\ \Delta_2 \vdash a : A}{\Gamma;\ (\Delta_1, \Delta_2) \vdash f\ a : B} \qquad (/\text{-C})\ \frac{\Gamma;\ (\Delta_1, x{:}A) \vdash b : B \quad \Gamma;\ \Delta_2 \vdash a : A}{\Gamma;\ (\Delta_1, \Delta_2) \vdash (/x{:}A.b)\ a = [a/x]b : B}$$

Figure 1: Rules for directed Lambek types B/A.

$$(\Pi^r\text{-F})\ \frac{\Gamma;\ * \vdash A\ \mathit{type} \quad \Gamma, x{:}A;\ * \vdash B\ \mathit{type}}{\Gamma;\ * \vdash \Pi^r x{:}A.B\ \mathit{type}} \qquad (\Pi^r\text{-I})\ \frac{\Gamma, x{:}A;\ \Delta \vdash b : B \quad \Gamma;\ * \vdash \Pi^r x{:}A.B\ \mathit{type}}{\Gamma;\ \Delta \vdash \lambda^r x{:}A.b : \Pi^r x{:}A.B}$$

$$(\Pi^r\text{-E})\ \frac{\Gamma;\ \Delta \vdash f : \Pi^r x{:}A.B \quad \Gamma;\ * \vdash a : A}{\Gamma;\ \Delta \vdash \mathit{app}^r(f, a) : [a/x]B} \qquad (\Pi^r\text{-C})\ \frac{\begin{array}{c}\Gamma, x{:}A;\ \Delta \vdash b : B \quad \Gamma;\ * \vdash a : A \\ \Gamma;\ * \vdash \Pi^r x{:}A.B\ \mathit{type}\end{array}}{\Gamma;\ \Delta \vdash \mathit{app}^r(\lambda^r x{:}A.b,\ a) = [a/x]b : [a/x]B}$$

Figure 2: Directed Π^r-types.

2.2 Dependent Lambek Types: Examples

Directed Dependent Products. Dependent product types (Π-types) are split into directed dependent products (Π^r and Π^l). The rules for Π^r-types are given in Figure 2. The rules for Π^l-types, omitted here, are symmetric with term constructors $\lambda^l x{:}A.b$ and $\mathit{app}^l(a, f)$.

Type Universes. In type theory, a type universe is a type whose objects are (names of) types. For instance, in formal semantics in modern type theories [14, 10, 2], common nouns are interpreted as types (rather than predicates). In a similar fashion, we may introduce a universe CN of common nouns:

$$\frac{}{*; * \vdash \mathrm{CN}\ \mathit{type}} \qquad \frac{\Gamma;\ * \vdash A : \mathrm{CN}}{\Gamma;\ * \vdash T_{\mathrm{CN}}(A)\ \mathit{type}}$$

where T_{CN} maps any common noun to a type (we often omit T_{CN} and just write A for $T_{\mathrm{CN}}(A)$). Note that CN is closed under several type constructors including the directed dependent sum types below.

Example 2.1. *Here is a simple example in syntactic analysis (as in categorial grammar). Consider the following sentence (1):*

(1) Every student works.

The words in the sentence can be given the following types:

(2) $\mathit{every} : \Pi^r X{:}\mathrm{CN}.\ S/(X \setminus S)$

$$(\Sigma^{\sim}\text{-F}) \qquad \frac{\Gamma;\ * \vdash A\ type \quad \Gamma, x{:}A;\ * \vdash B\ type}{\Gamma;\ * \vdash \Sigma^{\sim}x{:}A.B\ type}$$

$$(\Sigma^{\sim}\text{-I}) \qquad \frac{\Gamma; * \vdash a : A \quad \Gamma;\Delta \vdash b : [a/x]B}{\Gamma;\Delta \vdash pair(a,b) : \Sigma^{\sim}x{:}A.B}$$

$$(\Sigma^{\sim}\text{-E}) \qquad \frac{\Gamma;\Delta \vdash p : \Sigma^{\sim}x{:}A.B \quad \Gamma, x{:}A;\ \Delta', y{:}B \vdash e : C \quad \Gamma;\ * \vdash C\ type}{\Gamma;\ (\Delta,\Delta') \vdash \mathbf{let}\ pair(x,y) = p\ \mathbf{in}\ e\ :\ C}$$

$$(\Sigma^{\sim}\text{-C}) \qquad \frac{\Gamma; * \vdash a : A \quad \Gamma;\Delta \vdash b : [a/x]B \quad \Gamma, x{:}A;\ \Delta', y{:}B \vdash e : C \quad \Gamma;\ * \vdash C\ type}{\Gamma;\ (\Delta,\Delta') \vdash \mathbf{let}\ pair(x,y) = pair(a,b)\ \mathbf{in}\ e\ =\ [a/x, b/y]e\ :\ C}$$

Figure 3: Rules for $\Sigma^{\sim}$-types.

(3) $student : \mathrm{CN}$

(4) $works : human \setminus S$

where S is the type of sentences and student is a subtype of human (and, hence by contravariance, $human \setminus S$ is a subtype of $student \setminus S$). It is then straightforward to derive that

$$app^r(every, student)\ works\ :\ S$$

In other words, (1) is a sentence.

Directed Dependent Sums Dependent sum types (Σ-types) are split into reverse dependent sums ($\Sigma^{\sim}$) and concatenation dependent sums (Σ°). The rules for $\Sigma^{\sim}$-types are given in Figure 3, while the rules for Σ°-types are symmetric and omitted. We remark that the universe CN is closed under $\Sigma^{\sim}$ and Σ°. For example, directed dependent sum types may be used to analyse modified common nouns when the modifying adjectives are intersective or subsective. To illustrate this with an example, assuming that $B : A \setminus S$, let's use $\Sigma^{\sim}(A,B)$ to abbreviate $\Sigma^{\sim}x{:}A.(x\ B)$ and $\Sigma^{\circ}(A,B)$ to abbreviate $\Sigma^{\circ}x{:}A.(x\ B)$. Now, with $diligent : human \setminus S$, we can use $\Sigma^{\sim}(student, diligent)$ to describe the modified common noun *diligent student*, and $\Sigma^{\circ}(student, diligent)$ to analyse *student who is diligent*.

3 Dependent Linear Types

When introducing dependent types into a substructural calculus, a most challenging decision to make is whether to allow types to depend on substructural variables. For instance, for $f : A \multimap A$, the equality type $Eq_A(f\ x,\ x)$ depends on the linear variable x of type A. In most of the existing research so far except [13, 11], types are only allowed to depend on intuitionistic variables, but not on substructural variables such

Intuitionistic Π-*types* (Conversion: $(\lambda x{:}A.b)(a) \simeq [a/x]b$)

$$\frac{\Gamma,\ x : A \vdash B\ type}{\Gamma \vdash \Pi x{:}A.B\ type} \qquad \frac{\Gamma,\ x : A \vdash b : B}{\Gamma \vdash \lambda x{:}A.b : \Pi x{:}A.B} \qquad \frac{\Gamma \vdash f : \Pi x{:}A.B \quad \bar{\Gamma} \vdash a : A}{\Gamma \vdash f(a) : [a/x]B}$$

Linear Π-*types* (Conversion: $(\lambda x{::}A.b)\ a \simeq [a/x]b$)

$$\frac{\Gamma,\ x{::}A \vdash B\ type}{\Gamma \vdash \Pi x{::}A.B\ type} \qquad \frac{\Gamma,\ x{::}A \vdash b : B}{\Gamma \vdash \lambda x{::}A.b : \Pi x{::}A.B} \qquad \frac{\Gamma \vdash f : \Pi x{::}A.B \quad \Delta \vdash a : A \quad Merge(\Gamma;\Delta)\downarrow}{Merge(\Gamma;\Delta) \vdash f\ a : [a/x]B}$$

Figure 4: Intuitionistic and linear Π-types

as the linear variables (or Lambek variables, as in the calculus described in the previous section). In this section, we follow [11] to present LDTT, a linear dependent type theory where types may depend on linear variables.

A context in LDTT may contain two forms of entries: the usual (intuitionistic) entries $x{:}A$ and the linear ones $y{::}A$, where y is called a linear variable. For any term t, $FV(t)$ is the set of free variables occurring in t and, for any context Γ, if $FV(t) \subseteq FV(\Gamma)$, then $D_\Gamma(t)$ is the set of dependent variables of t w.r.t. Γ, defined as follows: (1) $FV(t) \subseteq D_\Gamma(t)$, and (2) for any $x \in D_\Gamma(t)$, $FV(\Gamma_x) \subseteq D_\Gamma(t)$. In LDTT, we have the following variable typing rule:

$$(V) \qquad \frac{\Gamma,\ x \bar{:} A,\ \Gamma'\ valid \quad (\text{for all } y{::}\Gamma_y \in \Gamma,\ y \in D_\Gamma(x)) \quad \Gamma' \text{ intuitionistic}}{\Gamma,\ x \bar{:} A,\ \Gamma' \vdash x : A} \qquad (\bar{:} \in \{:, ::\})$$

where 'Γ′ intuitionistic' means that Γ′ does not contain any linear entries.

We have two forms of Π-types: the intuitionistic $\Pi x{:}A.B$ and linear $\Pi x{::}A.B$, whose rules are given in Figure 4. For both, the formation and introduction rules are not unusual, although their elimination rules need some explanations. For intuitionistic Π-types, in order to type $f(a)$ under context Γ, a is required to be typable in $\bar{\Gamma}$, the intuitionistic part of Γ, obtained from Γ by removing all the entries whose variables are in $FV_{LD}(\Gamma)$, the set of linear dependent variables in Γ.[2] The elimination rule for linear Π-types involves the operation $Merge(\Gamma;\Delta)$, which is only defined, notation $Merge(\Gamma;\Delta)\downarrow$, if $\bar{\Gamma} \equiv \bar{\Delta}$ and $FV_{LD}(\Gamma) \cap FV_{LD}(\Delta) = \emptyset$: (1) $Merge(\langle\rangle;\langle\rangle) = \langle\rangle$; (2) If $x \in FV_{LD}(\Gamma,\Delta)$, $Merge(\Gamma, x\bar{:}A;\ \Delta) = Merge(\Gamma;\ \Delta, x\bar{:}A) = Merge(\Gamma;\Delta), x\bar{:}A$, where $\bar{:}$ is either : or ::; and (3) $Merge(\Gamma, x{:}A;\ \Delta, x{:}A) = Merge(\Gamma;\Delta), x{:}A$.

LDTT also contains equality types $Eq_A(a,b)$, whose rules are given in Figure 5. The Eq-formation rule involves another context merge operation $merge(\Gamma;\Delta)$, which is only defined, notation $merge(\Gamma;\Delta)\downarrow$, under the condition that, if $x\bar{:}A \in \Gamma$ and $x\bar{:}B \in \Delta$, then (1) $\bar{:}$ is either both : or both :: and (2) $A \equiv B$: (1) $merge(\Gamma;\langle\rangle) = \Gamma$, and (2) for

[2]Formally, $FV_{LD}(\Gamma)$ is defined as follows: (1) $FV_{LD}(\langle\rangle) = \emptyset$; (2) $FV_{LD}(\Gamma, x{::}A) = FV_{LD}(\Gamma) \cup \{x\}$; (3) if $FV(A) \cap FV_{LD}(\Gamma) = \emptyset$, then $FV_{LD}(\Gamma, x{:}A) = FV_{LD}(\Gamma)$; otherwise, $FV_{LD}(\Gamma, x{:}A) = FV_{LD}(\Gamma) \cup \{x\}$.

Equality types (Conversion: $subst(refl(a), q) \simeq q$)

$$\frac{\Gamma \vdash a : A \quad \Delta \vdash b : A \quad merge(\Gamma;\Delta)\downarrow}{merge(\Gamma;\Delta) \vdash Eq_A(a,b)\ type} \qquad \frac{\Gamma \vdash a : A}{\Gamma \vdash refl(a) : Eq_A(a,a)}$$

$$\frac{\Gamma \vdash p : Eq_A(a,b) \quad \Delta \vdash q : B[a] \quad Merge(\Gamma;\Delta), x\bar{:}A \vdash B[x]\ type\ (\bar{:} \in \{:, ::\}) \quad Merge(\Gamma;\Delta)\downarrow}{Merge(\Gamma;\Delta) \vdash subst(x.B, p, q) : B[b]}$$

Figure 5: Equality types

$\bar{:}$ being either : or ::, $merge(\Gamma;\ x\bar{:}A, \Delta)$ is equal to (i) $merge(\Gamma;\Delta)$, if $x \in FV(\Gamma)$, and (ii) $merge(\Gamma, x\bar{:}A;\ \Delta)$, otherwise. Its elimination rule involves the $Merge$-operation defined earlier.

In linear logic, every linear variable occurs free for exactly once in a typed term. In LDTT, every linear variable *occurs essentially* for exactly once in a typed term – a property we call *weak linearity*. More precisely, for $\Gamma \vdash a : A$, the multiset of variables essentially occurring in a under Γ, $E_\Gamma(a)$, is defined by induction on derivations (we omit the part of the definition for Eq-types): (1) for (V) above, $E_{\Gamma,\ x\bar{:}A,\ \Gamma'}(x) = D_{\Gamma,\ x\bar{:}A,\ \Gamma'}(x)$; (2) for the λ-typing rules, $E_\Gamma(\lambda x\bar{:}A.b) = E_{\Gamma,x\bar{:}A}(b)\backslash\{x\}$, where $\bar{:} \in \{:, ::\}$; (3) for intuitionistic applications, $E_\Gamma(f(a)) = E_\Gamma(f) \cup E_{\overline{\Gamma}}(a)$; (4) for linear applications, $E_{Merge(\Gamma;\Delta)}(f\ a) = E_\Gamma(f) \cup E_\Delta(a)$.

Theorem (weak linearity) If $\Gamma \vdash a : A$ and $x{::}\Gamma_x \in \Gamma$, then $x \in E_\Gamma(a)$ only once. □

4 Conclusion

The developments in this paper are based on the author's work reported in previous talks [9, 11] about this topic. Future work includes studies in two fronts: one is to employ dependent Lambek types in the study of NL syntactical analysis and related issues in the syntax-semantics interface with MTT-semantics, and the other is to apply dependent linear types to the study of dependent session types for further development in theory of concurrency and concurrent programming languages.

References

[1] I. Cervesato and F. Pfenning. A linear logical framework. *Information and Computation*, 179, 2002.

[2] S. Chatzikyriakidis and Z. Luo. *Formal Semantics in Modern Type Theories.* Wiley & ISTE Science Publishing Ltd., 2018. (to appear).

[3] P. de Groote and S. Maarek. Type-theoretic extensions of abstract categorial grammars. *Proc. of the Workshop on New Directions in Type-theoretic Grammars. ESSLLI 2007*, 2007.

[4] P. de Groote, S. Maarek, and R. Yoshinaka. On two extensions of abstract categorial grammars. *LPAR 2007, LNAI 4790*, 2007.

[5] S. Gay. Session types: Achievements and challenges. Invited Talk at TYPES16, May 2016.

[6] J.-Y. Girard. Linear logic. *Theoret. Comput. Sci.*, 50, 1987.

[7] N. Krishnaswami, P. Pradic, and N. Benton. Integrating dependent and linear types. *POPL 2015.*

[8] J. Lambek. The mathematics of sentence structure. *The American Mathematical Monthly*, 65(3), 1958.

[9] Z. Luo. A Lambek Calculus with Dependent Types. *TYPES 2015.*

[10] Z. Luo. Formal semantics in modern type theories with coercive subtyping. *Linguistics and Philosophy*, 35(6):491–513, 2012.

[11] Z. Luo and Y. Zhang. A linear dependent type theory. *TYPES 2016. Novi Sad*, 2016.

[12] S. Martin and C. Pollard. A dynamic categorial grammar. *Formal Grammar 2014*, 2014.

[13] C. McBride. I got plenty ońuttin׳ *In S. Lindley and C. McBride and P. Trinder and D. Sannella (eds.) A List of Successes That Can Change the World, LNCS 9600*, 2016.

[14] A. Ranta. *Type-Theoretical Grammar.* Oxford University Press, 1994.

[15] A. Ranta. Syntactic calculus with dependent types. *J of Logic, Language and Information*, 7, 1998.

[16] M. Vákár. A categorical semantics for linear logical frameworks. *FoSSaCS 2015.*