Chapter 3
Expanded Categorial Grammar

A. Review

1. Introduction

We first recall several points concerning how we understand the semantic enterprise.

(1) The goal of semantics is to provide a **semantic-analysis** of every admissible phrase in a given language, the **object-language**.

(2) The semantic-analysis of a given phrase ϕ consists of three inter-locking constituents.

- (a) to provide a **semantic-value** for ϕ.
- (b) to provide a **semantic-value** for every **component-phrase** of ϕ.
- (c) to demonstrate how (a) is **computed** from (b).

(3) Every phrase decomposes ultimately into elementary phrases (**morphemes**) whose meanings are provided by the **lexicon**.

(4) We take semantic-evaluation to be **translation** of the object-language into a **target-language**, which is presumed to be better understood.

(5) The target-language we propose is **Loglish**, which is a hybrid of Logic and English, and which at a minimum, contains:

- (a) a first-order logic kernel.
- (b) a type-theoretic super-structure.

(6) The computational component of semantics is characterized by a **compositional-calculus**, which involves various **Rules of Composition**.

In regard to item (5)(b), so far we propose a fairly limited type-theory, based on the following types.

1.	S is a type.	sentences
2.	D is a type.	definite noun phrases
3.	if A and B are types, then so is $(A \rightarrow B)$.	monadic functors
4.	nothing else is a type.	
5.	$\mathsf{C} =_{df} \mathsf{D} \rightarrow \mathsf{S}$ [common-noun phrases]	defined type

In regard to item (6), so far we propose two such rules.

1. Function-Application

a phrase	Λ	of type	$A \rightarrow B$
combines with a phrase	α	of type	A
to produce the phrase	$[\Lambda]\langle\alpha\rangle$	of type	B

2. Conjunction

a phrase	$\lambda\nu\Phi$	of type	$\mathsf{C}\,[\mathsf{D} \rightarrow \mathsf{S}]$
combines with a phrase	$\lambda\nu\Psi$	of type	$\mathsf{C}\,[\mathsf{D} \rightarrow \mathsf{S}]$
to produce the phrase	$\lambda\nu(\Phi\ \&\ \Psi)$	of type	$\mathsf{C}\,[\mathsf{D} \rightarrow \mathsf{S}]$

In regard to item (2), we employ semantic trees. By a **semantic-tree** for a phrase ϕ, we mean a *tree* consisting of the semantic-values of all the sub-phrases of ϕ, plus the original morphemes attached to the terminal nodes.

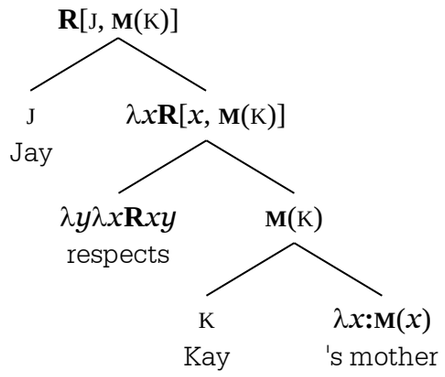
Recall that we employ a special tabular system to present semantic trees, described as follows.

(1) table-cells correspond to nodes.
(2) vertical lines delineate nodes on the same level.
(3) horizontal lines correspond to semantic transformations ("moves").
  (a) bold lines correspond to binary-moves (e.g., function-application).
  (b) non-bold lines correspond to unary-moves (e.g., lambda-conversion).
  (c) intermediate expressions may be omitted.
(4) composition proceeds **down** the page.
  branching proceeds **up** the page.

The following example compares the tabular method with the conventional method of depicting semantic trees.

1. Jay respects Kay's mother

Jay	respects	Kay	's mother
		K	λx:M(x)
	$\lambda y \lambda x \mathbf{R} xy$	M(K)	
J	$\lambda x \mathbf{R}[x,$ M(K)]		
$\mathbf{R}$[J, M(K)]			



2. Positive Examples

The following are successful applications of Basic Categorial Semantics.

2. every woman who is virtuous is happy

every	woman	who	is	virtuous	is	happy
			∅	$\lambda x \mathbf{V} x$	∅	$\lambda x \mathbf{H} x$
		∅	$\lambda x \mathbf{V} x$			
	$\lambda x \mathbf{W} x$	$\lambda x \mathbf{V} x$				
$\lambda P \lambda Q \forall x\{ Px \rightarrow Qx \}$	$\lambda x(\mathbf{W} x \,\&\, \mathbf{V} x)$					
$\lambda Q \forall x\{ (\mathbf{W} x \,\&\, \mathbf{V} x) \rightarrow Qx \}$					$\lambda x \mathbf{H} x$	
$\forall x\{ (\mathbf{W} x \,\&\, \mathbf{V} x) \rightarrow \mathbf{H} x \}$						

3. not every virtuous woman respects Kay

not	every	happy	woman	respects	Kay
		$\lambda x \mathbf{H} x$	$\lambda x \mathbf{W} x$	$\lambda y \lambda x \mathbf{R} xy$	K
	$\lambda P \lambda Q \forall x\{Px \rightarrow Qx\}$	$\lambda x(\mathbf{H} x \,\&\, \mathbf{W} x)$			
	$\lambda Q \forall x\{ (\mathbf{H} x \,\&\, \mathbf{W} x) \rightarrow Qx \}$			$\lambda x \mathbf{R} x$K	
$\lambda P \sim P$	$\forall x\{ (\mathbf{H} x \,\&\, \mathbf{W} x) \rightarrow \mathbf{R} x$K $\}$				
$\sim \forall x\{ (\mathbf{H} x \,\&\, \mathbf{W} x) \rightarrow \mathbf{R} x$K $\}$					

3. Counter-Examples

In this context, by a **counter-example** we mean a string of words that constitute a well-formed and meaningful phrase, but cannot the properly constructed by our (current) account of semantic-composition. The following are just two of many counter-examples.

4. Jay does not respect Kay

Jay	does-not	respect	Kay
		$\lambda y \lambda x \mathbf{R} x y$	K
	$\lambda P \sim P$	$\lambda x \mathbf{R} x K$	
J	$\lambda x \sim \mathbf{R} x K$		
$\sim \mathbf{R} J K$			

5. Jay respects every woman

Jay	respects	every	woman
		$\lambda P\, \lambda Q\, \forall x \{ P x \rightarrow Q x \}$	$\lambda x \mathbf{W} x$
	$\lambda y \lambda x \mathbf{R} x y$	$\lambda Q\, \forall x \{\, \mathbf{W} x \rightarrow Q x \,\}$	
J	$\lambda y\, \forall x \{\, \mathbf{W} x \rightarrow \mathbf{R} y x \,\}$		
$\forall x \{\, \mathbf{W} x \rightarrow \mathbf{R} J x \,\}$			

In each example, the shaded entry offers a plausible hypothesis concerning the identity of the node, but the expression cannot the obtained from the input using currently available rules of composition. Conjunction is not applicable, since the phrases don't have matching appropriate types. Function-application is not applicable, since neither phrase serves as a type-appropriate argument for the other.

B. Expanded Categorial Syntax

1. Introduction

In order to deal with the counter-examples above, we propose to expand the rules of composition. For this purpose, we develop an expanded *calculus of composition*, which we call *Categorial Logic*. We employ the term 'logic' because, in characterizing this system, we utilize (*re-purpose*) logical formalism, and we draw inspiration from extant logical systems – including Classical Logic, Intuitionistic Logic, Relevance Logic, and Linear Logic.

According to Basic Categorial Grammar, there are two rules of composition.

1. Function-Application

a phrase	Λ	of type	$A \rightarrow B$
combines with a phrase	α	of type	A
to produce the phrase	$[\Lambda]\langle\alpha\rangle$	of type	B

2. Conjunction

a phrase	$\lambda \nu \Phi$	of type	$\mathsf{C}\,[\mathsf{D} \rightarrow \mathsf{S}]$
combines with a phrase	$\lambda \nu \Psi$	of type	$\mathsf{C}\,[\mathsf{D} \rightarrow \mathsf{S}]$
to produce the phrase	$\lambda \nu (\Phi\ \&\ \Psi)$	of type	$\mathsf{C}\,[\mathsf{D} \rightarrow \mathsf{S}]$

Let us set aside Conjunction for the moment, since it is not strictly analogous to a logical inference pattern. On the other hand, concerning Function-Application, if we think of arrow as the logical *if-then* connective, then this procedure corresponds to the following inference pattern,

from	if A then B
and	A
infer	B

which is the inference-pattern known as *modus ponens*.

What we propose is that, whereas Basic Categorial-Grammar admits only one logic-like mode of composition, corresponding to *modus ponens*, Expanded Categorial-Grammar admits **infinitely-many** modes of composition, each one corresponding to a valid-inference of Categorial Logic, which is summarized as follows.

Categorial Logic Principle (for composing types)		
Where $\mathcal{A}_0$, …, $\mathcal{A}_k$ are types.		
$\mathcal{A}_1$, …, $\mathcal{A}_k$	combine to form	$\mathcal{A}_0$
IF [1]		
the argument form	$\mathcal{A}_1$, …, $\mathcal{A}_k$ / $\mathcal{A}_0$	
is valid according to Categorial Logic		

2. What is Categorial Logic?

The obvious question then is: what logical system best models grammatical-composition? In this connection, there are several prominent extant logical systems that serve as candidates, including

(1) Classical Logic
(2) Intuitionistic Logic
(3) Relevance Logic
(4) Linear Logic

which we examine in the next few sections.

3. Classical Logic

Classical Logic (CL) is the strongest logic of the group, and accordingly authorizes the maximum number of compositions.[2] Unfortunately, it also authorizes compositions that are grammatically implausible. For example, the following is a principle of Classical Logic.[3]

(c1) $A ; B \vdash A \rightarrow B$

Here, the symbol '$\vdash$' is a meta-logical symbol indicating logical-entailment; for example, (c1) says in effect that:

the argument $\boxed{\text{A ; B ; therefore A}\rightarrow\text{B}}$ is valid.

Accordingly, using Classical Logic to judge grammatical-composition, the following composition-rule is authorized.

$S ; S \vdash S \rightarrow S$

We read this as saying that one may combine two sentences (S) to form a sentential-adverb (S-operator). Since this is grammatically highly implausible, Categorial Logic must reject (c1).

In an important sense, to be explained shortly, (c1) follows from the following inference principle, which is also valid in Classical Logic.

(c2) $B \vdash A \rightarrow B$

This is an inference principle found especially obnoxious by many pioneers of alternative logical systems, including the strict-entailment logics of C.I. Lewis (1912) and the relevant-entailment logics of Anderson and Belnap (1975).

Grammatically-understood, (c2) is equally obnoxious, since the following instance

$S \vdash S \rightarrow S$

authorizes transforming[4] a sentence into a sentential-adverb (S-operator), which seems grammatically implausible.

---

[1] The connective is 'if', not 'if and only if', since we consider other modes of composition to be available. For example, we also admit the Conjunction-Rule.
[2] If a weaker logic validates an argument form $\mathbb{A}$, then a stronger logic also validates $\mathbb{A}$, granting the two logics both pass judgment on $\mathbb{A}$.
[3] Although nothing hinges on this, we use one arrow-symbol '$\rightarrow$' for logic, and another arrow-symbol '$\rightarrow$' for category theory.
[4] A transformation corresponds to a single-premise argument.

In addition to (c1) and (c2), the following principles of Classical Logic also yield implausible composition principles.[5]

(c3) $(A \rightarrow B) \rightarrow B \vdash (B \rightarrow A) \rightarrow A$
(c4) $(A \rightarrow B) \rightarrow A \vdash A$

In light of numerous examples of inadmissible grammatical-compositions based on Classical Logic principles, we conclude that Classical Logic does not properly model grammatical composition.

4. Intuitionistic Logic

Intuitionist Logic (IL) is properly contained in Classical Logic (CL) – every argument deemed valid by IL is also deemed valid by CL, but not conversely. For example, (c3) and (c4) above are rejected by IL. On the other hand, both (c1) and (c2) are accepted by IL. Since the latter yield grammatically-implausible compositions, we conclude that Intuitionistic Logic also does not properly model grammatical composition.

5. No Monotonic Logic Properly Models Grammatical-Composition

A principle that is fundamental to nearly every logical system that has been considered over the past few millennia is the principle of *monotonicity*.[6] The basic idea is that adding premises to a valid argument does not result in an invalid argument. The following meta-logical principle is a special case of monotonicity.[7]

$$\mathcal{B} \vdash C \;\Rightarrow\; \mathcal{A}\,;\mathcal{B} \vdash C$$

In other words, if C follows from $\mathcal{B}$, then C also follows from $\mathcal{A}$ and $\mathcal{B}$.

Next, the following *identity* principle, that a sentence logically entails itself,

$$\mathcal{B} \vdash \mathcal{B}$$

is generally regarded as a minimum requirement of any formal system that presumes to model reasoning. Putting monotonicity and identity together, we obtain the following *simplification principle*.

$$\mathcal{A}\,;\mathcal{B} \vdash \mathcal{B}$$

Suppose we include this principle in the logic of grammatical-composition. Then the following composition is authorized.

$$\mathsf{D}\,;\mathsf{S} \vdash \mathsf{S}$$

In the proposed composition, we compose a sentence by combining a definite-noun-phrase (D) and a sentence (S) – presumably by simply discarding the DNP. However, it seems manifestly plausible that a valid grammatical-composition must meaningfully utilize *all* its inputs in constructing its output. Accordingly, in order for a logic to model grammatical-composition, it cannot contain the simplification principle, and so it cannot be monotonic.

In the following, we examine two prominent non-monotonic logics – Relevance Logic, and Linear Logic.

6. Relevance Logic

As its name suggests, Relevance Logic is characterized by sensitivity to matters of *relevance* – in particular, between premises and conclusions of arguments, and between antecedents and consequents of conditional (if-then) statements. In particular, for Relevance Logic, a conditional statement cannot be true unless there is a relevance-connection between the antecedent and the consequent, and an argument cannot be valid unless there is a relevance-connection between (all) the premises and the conclusion. Many well-known systems of logic do not satisfy either of these *desiderata*.

---

[5] These are prominent examples of valid argument forms whose proof (say, in an intro logic class) cannot be accomplished using only rules pertaining to $\rightarrow$.

[6] This usage comes from the definition of monotonic (increasing) functions. Specifically, a function ϕ is said to be monotonic precisely if $\phi(x) \leqslant \phi(y)$ whenever $x \leqslant y$. In the context of logical systems, the relevant order-relation is set-inclusion, and the relevant function is the consequence function $\mathbb{C}$, defined so that $\mathbb{C}(\Gamma) =_{df} \{\alpha \mid \Gamma \vdash \alpha\}$. Then a logical system is monotonic precisely if its associated consequence-function is monotonic.

[7] Here, we use the symbol '$\Rightarrow$' as the meta-language's if-then connective.

At the same time, it seems that relevance *desiderata* are tailor-made for modelling grammatical-composition. For example, it is presumed that a grammatical functor actually *uses* its input in generating its output, and it is also presumed that a grammatical-composition uses all its input in producing its output.

Relevance Logic does a decent job of modeling grammatical-composition, but it also authorizes several problematic compositions, which we now review.

1. Contraction

$A \rightarrow (A \rightarrow B) \vdash A \rightarrow B$

If we use this to model grammatical-composition, then we obtain the following grammatical principle

$D \rightarrow (D \rightarrow S) \vdash D \rightarrow S$

according to which a transitive-verb *automatically* transforms into an intransitive verb, which seems implausible.

2. Duplication

$A \vdash A \times A$

Here, $\times$ is the multiplicative-counterpart of $\rightarrow$,[8] which is a special connective in all the logics we examine here, and which reduces to conjunction (&) when we move to Intuitionistic and Classical Logic. If we use this to model grammatical-composition, then we obtain the following grammatical principle.

$D \vdash D \times D$

This amounts to saying that a DNP can duplicate itself (repeatedly) and accordingly serve as the input for an unlimited number of functors (e.g., VPs), which seems implausible.

3. Assertion

$(A \rightarrow A) \rightarrow B \vdash B$

If we take this as modelling grammatical-composition, then we obtain the following grammatical principle, where C is the type of common-noun-phrases.

$(C \rightarrow C) \rightarrow (C \rightarrow C) \vdash C \rightarrow C$

From this, we obtain the following composition principle.

$(C \rightarrow C) \rightarrow (C \rightarrow C)\ ;\ C \vdash C$

A *modifier* is a phrase of type $\mathcal{A} \rightarrow \mathcal{A}$, where $\mathcal{A}$ is any type. For example, a common-noun modifier (adjective) is a phrase of type $C \rightarrow C$, and an adjective-modifier is a phrase of type $(C \rightarrow C) \rightarrow (C \rightarrow C)$. According to the grammatical principle proposed above, an adjective-modifier like ‘very’ can be combined with a common-noun-phrase like ‘dog’ to produce a common-noun-phrase ‘very dog’, which seems implausible.

4. Tautology

A tautology is a formula that is logically-true, alternatively a formula that follows from nothing.[9] Every logic has valid argument forms, but not every logic has tautologies. Relevance logic has tautologies, the simplest of which is depicted in the following principle, which we call ‘Tautology’.

$\vdash A \rightarrow A$

Note that Assertion follows from Tautology, so the latter must be rejected by Categorial Logic insofar as the former is rejected. Also, it is implausible to suppose that a phrase can simply enter a semantic-derivation "out of thin air".

---

[8] We later add $\times$ as a further type-forming operator; see Section 2.

[9] When no formulas are in front of $\vdash$, it is understood that the premise set is empty.

7. Linear Logic Without Tautology

Having rejected Relevance Logic as the appropriate logical model of grammatical-composition, we next consider an even weaker logical system, Linear Logic, which is founded on the idea of resource-usage.[10] In particular, whereas Relevance Logic requires that every supposition be used *at least once*, Linear Logic requires that every supposition be used *exactly once*.

This adjustment gets rid of Contraction and Duplication, but it does not get rid of Assertion or Tautology. To accomplish this, we must also ban arguments with no premises, also known as tautologies. This restriction makes sense from a grammatical point of view, since we do not want phrases entering a grammatical-construction "out of thin air". We call the resulting system Linear Logic without Tautology.

By moving to Linear Logic without Tautology, we get rid of many inference principles that make no sense grammatically, but unfortunately we also get rid of inference principles that seem desirable, including the following.

(1) $A \rightarrow B \; ; \; A \rightarrow C \vdash A \rightarrow (B \times C)$ [Conditional Multiplication]
(2) $A \rightarrow (B \rightarrow C) \; ; \; A \rightarrow B \vdash A \rightarrow C$ [Conditional Modus Ponens]
(3) $A \rightarrow B \; ; \; A \rightarrow C \; ; \; (B \times C) \rightarrow D \vdash A \rightarrow D$ [Generalized-Conjunction]

Note that (1) entails the other two in Linear Logic, so it is the crux.

8. Multi-Linear Logic

Based on the considerations of the previous sections, we propose that:

CATEGORIAL LOGIC
is
Linear Logic
minus
Tautology
plus
Conditional-Multiplication

for which we propose the name *Multi-Linear Logic*.

C. Categorial Logic – Formal Presentation

1. Introduction

In this unit, we formally present various logical systems discussed earlier, in particular the fragments of these logics that pertain to the connectives most relevant to categorial-grammar reasoning. The underlying formal language accordingly has a vocabulary consisting of an infinite list of atomic formulas, plus two two-place connectives [$\rightarrow$, $\times$], plus parentheses – the formation rules being given as follows.

(1) every atomic formula is a formula.
(2) if $\mathcal{A}$ and $\mathcal{B}$ are formulas, then so is $(\mathcal{A} \rightarrow \mathcal{B})$.
(3) if $\mathcal{A}$ and $\mathcal{B}$ are formulas, then so is $(\mathcal{A} \times \mathcal{B})$.
(4) nothing else is a formula.

2. Derivations

A logical system [calculus] is a formal apparatus by which all the valid argument-forms are constructed. There are different ways by which this is accomplished.[11] We propose a natural-deduction system, based on derivations, which are schematically depicted as follows.

[10] Linear Logic originates with Jean-Yves Girard (1987).
[11] See Section A.10.

line-number	formula	index	annotation
(1)	Φ_1	I_1	A_1
(2)	Φ_2	I_2	A_2
…			
(m)	Φ_m	I_m	A_m

Line-numbers are cited in the annotation column; a line's annotation indicates exactly how that formula is justified according to the system's rules. On the other hand, a line's index encodes the *suppositions* on which the formula (ultimately) depends. So, for example, in the following,

(7)	P	1+2	3,5, →O

line 7, which is formula 'P', follows from lines 3 and 5 by the inference-rule arrow-out, and depends upon suppositions 1 and 2, composed via the plus-operator.[12]

3. Suppositions

One way in which a formula can enter a derivation is via supposition, which comes in two kinds – premises and assumptions. Whereas premises correspond to the initial input of the reasoning process, assumptions act as *provisional* premises in sub-derivations in connection with various inference-rules. They are governed by the following rules.

1. Premise-Rule

At any point in a derivation, one may write down any **premise**, in accordance with the following schema.[13]

line-number	formula	index	annotation
L	Φ	n (**new**)	Pr

2. Assumption-Rule

At any point in a derivation, one may write down any formula as an **assumption**, in accordance with the following schema.

line-number	formula	index	annotation
L	Φ	n (**new**)	As

3. New

In the above rules, n is any integer that is *new*, which is to say n does not occur earlier in the derivation. Most authors enforce this restriction by insisting that n equals the line-number. The above scheme allows suppositions to be numbered independently.

4. Indices

Indices keep track of *supposition-dependence*,[14] and are officially defined as follows.

(1) every numeral is an index;
(2) if i is an index, and j is an index, then (i+j) is an index;
(3) nothing else is an index.

As mentioned earlier, there are two reasonable ways to index suppositions. (1) One can simply use the supposition's line-number as its index. (2) One can number suppositions independently of formulas; the first supposition is labelled 1, the second one 2, etc. We employ the latter approach.[15]

---

[12] At first, and primarily, we consider just one composition-operator, addition (+). Later we consider a second operator, subtraction (–). By comparison, Restall (2000) posits two modes of composition – extensional-composition, intensional-composition; our two operators distinguish two aspects of intensional-composition. See Section A.10.

[13] One usually writes all the premises first.

[14] Alternatively expressed, indices encode "sub-structural" information; see Section A.11.

[15] Although most examples look the same either way, since most examples start with premises and assumptions.

Notice that compound-indices are, in principle, as diverse as ordered-pairs in set theory. On the other hand, various reduction principles are often postulated, including the following, which collectively amount to treating indices as (encoding) *sets* of formulas.

(1) $i{+}i = i$ Contraction
(2) $i{+}j = j{+}i$ Commutation
(3) $(i{+}j){+}k = i{+}(j{+}k)$ Association
(4) $i{+}\varnothing = i$ Identity [existence of an identity-element]

Notice that these are expressed as identities. On the other hand, more subtle logical distinctions can be made by dividing each identity into two inclusions, written using '$\leqslant$', and subject to the following principles.[16]

(a) $i \leqslant i$ reflexivity
(b) $i \leqslant j \;\&\; j \leqslant k \;.\!\rightarrow\; i \leqslant k$ transitivity
(c) $i \leqslant j \;\&\; j \leqslant i \;.\!\rightarrow\; i = j$ anti-symmetry

To adopt an inclusion principle of the form

(p) $i \leqslant j$

is to adopt the following structural inference principle.

$\mathcal{A}$	i
$\mathcal{A}$	j

Correspondingly, to adopt an identity principle of the form

(p) $i = j$

is to adopt the above inference principle and its converse.

Different logical systems adopt different reduction principles. For example, the system known as R-Mingle adopts (1)-(4). On the other hand, System R replaces (1) with the following weaker contraction principle.

(1a) $i{+}i \leqslant i$ Contraction[17]

On yet another hand, Logics without Tautology reject (4).

There is another key reduction principle – Weakening [aka Monotonicity] – which may be expressed by the following inclusion.

(5) $i \leqslant i{+}j$ Weakening

This corresponds to the idea that adding extraneous premises to a valid argument does not convert it into an invalid argument, which is expressed by the following principle.

(m) $\Gamma \vdash C \;\Rightarrow\; \Gamma{+}A \vdash C$ Weakening/Monotonicity

Both Classical Logic and Intuitionistic Logic are monotonic. On the other hand, as argued earlier in this chapter, monotonicity does not properly model grammatical composition; in grammatical reasoning, no input is extraneous. Categorial Logic is not monotonic, and accordingly rejects (5).

5. Inference Rules – Arrow

In addition to suppositional-rules and structural-rules, a logical system also posits inference-rules, which govern various logical operators. In this connection, it is customary to adopt a methodology according to which every logical-operator is equipped with two rules – an *introduction-rule*, and an *elimination-rule*, although we prefer the more succinct terms 'in' and 'out'.[18]

The following are the rules governing arrow.

[16] These principles amount to the claim that $\leqslant$ is a *partial-ordering*.
[17] Henceforth, unless noted otherwise, by Contraction we will mean the weaker principle given by (1a).
[18] The shorthand terminology "in" [introduction] and "out" [elimination] is borrowed from Pospesel (1974).

1. Arrow-Out (→O)

L_1	$\mathcal{A}\rightarrow\mathcal{B}$	i	
L_2	$\mathcal{A}$	j	
L_3	$\mathcal{B}$	i+j	L_1, L_2, →O

2. Arrow-In (→I)

L_1	$\mathcal{A}$	i	[19]
L_2	$\mathcal{B}$	i+j	
L_3	$\mathcal{A}\rightarrow\mathcal{B}$	j	L_1, L_2, →I

Whereas arrow-out corresponds to the familiar inference principle known as *modus ponens*, arrow-in corresponds to the following key principle about conditionals.

$$\mathcal{A}_1 ; \ldots ; \mathcal{A}_k ; \mathcal{B} \vdash C \;\Rightarrow\; \mathcal{A}_1 ; \ldots ; \mathcal{A}_k \vdash \mathcal{B}\rightarrow C$$

6. Inference Rules – Cross-Operator

A conditional connective → is said to be *residuated* precisely if there is an associated connective × satisfying the following **residual law**.[20]

$$(A\times B)\rightarrow C \;\dashv\vdash\; A\rightarrow(B\rightarrow C)$$

For example, in Classical and Intuitionistic Logic, × is simply *logical-and*.

$$A\times B \;=\; A\&B$$

On the other hand, in Relevance Logic[21] and Quantum Logic[22], the cross-operator corresponds to "compossibility", which may be defined as follows.[23]

$$A\times B \;=_{df}\; \sim(B\rightarrow\sim A) \;[\neq A\&B]$$

Every logical system we consider has a cross-operator with respect to which the arrow-operator is residuated.

The following inference-rules govern ×.

1. Cross-In (×I)

L_1	$\mathcal{A}$	i	
L_2	$\mathcal{B}$	j	
L_3	$\mathcal{A}\times\mathcal{B}$	i+j	L_1, L_2, ×I

2. Cross-Out (×O)

L_0	$\mathcal{A}\times\mathcal{B}$	i	…
L_1-L_2	$\mathcal{A};\mathcal{B} \hookrightarrow C$	j	…
L_3	C	i+j	L_0, L_1-L_2, ×O

---

[19] Usually, the antecedent $\mathcal{A}$ is introduced as an assumption; see later.

[20] One can also call this a *division* principle, based on rewriting A→B as B/A. Then the residual law amounts to the following.

A/(B×C) = (A/B)/C

A divided by (B times C) equals (A divided by B) divided by C

On the other hand, the word 'residual' derives from 'residue', which pertains to subtraction, in which case the residual law is written thus.

A–(B+C) = (A–B)–C

See T.S. Blyth and M.F. Janowitz (1972) for a general mathematical account of residuation.

[21] Dunn (1966).

[22] Hardegree (1981).

[23] The order doesn't matter for relevance logic, in which × is commutative, but it does matter for quantum logic, in which × is not commutative. Also note that, in relevance logic, × is referred to as "fusion", and is usually written using '∘'. See later section +++ for a grammatically motivated logic in which × is not commutative.

Here, $\mathcal{A};\mathcal{B}\hookrightarrow C$ is a sub-derivation of C from $\mathcal{A};\mathcal{B}$ dependent on j,[24] which is a sequence schematically presented as follows.

L_1	$\mathcal{A}$	m (new)	As
L_2	$\mathcal{B}$	n (new)	As
…	…	…	…
L_3	C	$(m+n)+j$	…

7. Derivations of Conclusions from Premises

An argument-form is a collection of formulas one of which serves as the conclusion, and the remainder of which serve as the premises. In order to demonstrate that an argument-form is valid, one constructs a derivation of the conclusion from the premises, which is a derivation such that:

(1) the last line is the conclusion,
which depends upon every premise,[25]
and depends upon no assumption;
(2) every line is one of the following:
  (a) a premise;
  (b) an assumption;
  (c) follows from previous lines by a rule.

8. Example Derivations

We begin with Linear Logic without Tautology, which adopts the principles of Associativity and Commutativity, but rejects Weakening, Contraction, Mingle, and Identity, which are summarized as follows.

(1a)	$i+i \leqslant i$	Contraction	❌
(1b)	$i \leqslant i+i$	Mingle	❌
(2)	$i+j = j+i$	Commutation	✓
(3)	$(i+j)+k = i+(j+k)$	Association	✓
(4)	$i+\varnothing = i$	Identity	❌
(5)	$i \leqslant i+j$	Weakening	❌

Given Associativity and Commutativity, and given that we do not consider any example involving more than 9 suppositions, we can write every index as a simple string of single-digit numerals in numerical order; for example, we write ‘1233’ instead of ‘3+((2+1)+3)’.

1. Linear Logic (Without Tautology)

1. Transitivity

B→C ; A→B ⊢ A→C

(1)	B→C	1	Pr
(2)	A→B	2	Pr
(3)	A	3	As
(4)	B	23	2,3,→O
(5)	C	123	1,4,→O
(6)	A→C	12	3,5,→I

[24] Note that j may be empty, although no actual line is permitted to have an empty-index in logics without tautology.
[25] This restriction is trivially satisfied in a system that admits Weakening. For an example, see Section A.8.5.1.

2. Permutation

$A \rightarrow (B \rightarrow C) \vdash B \rightarrow (A \rightarrow C)$

(1)	$A \rightarrow (B \rightarrow C)$	1	Pr
(2)	B	2	As
(3)	A	3	As
(4)	$B \rightarrow C$	13	1,3,→O
(5)	C	123	2,4,→O
(6)	$A \rightarrow C$	12	3,5,→I
(7)	$B \rightarrow (A \rightarrow C)$	1	2,6,→I

3. Secondary Modus Ponens

$A \rightarrow (B \rightarrow C) \; ; \; B \vdash A \rightarrow C$

(1)	$A \rightarrow (B \rightarrow C)$	1	Pr
(2)	B	2	Pr
(3)	A	3	As
(4)	$B \rightarrow C$	13	1,3,→O
(5)	C	123	2,4,→O
(6)	$A \rightarrow C$	12	3,5,→I

4. Montague's Law [26] [Lifting]

$A \vdash (A \rightarrow B) \rightarrow B$

(1)	A	1	Pr
(2)	$A \rightarrow B$	2	As
(3)	B	12	1,2,→O
(4)	$(A \rightarrow B) \rightarrow B$	1	2,4,→I

5. Inflection [27]

$(A \rightarrow C) \rightarrow D \; ; \; A \rightarrow B \vdash (B \rightarrow C) \rightarrow D$

(1)	$(A \rightarrow C) \rightarrow D$	1	Pr
(2)	$A \rightarrow B$	2	Pr
(3)	$B \rightarrow C$	3	As
(4)	A	4	As
(5)	B	24	2,4,→O
(6)	C	234	3,5,→O
(7)	$A \rightarrow C$	23	4,6,→I
(8)	D	123	1,7,→O
(9)	$(B \rightarrow C) \rightarrow D$	12	3,8,→I

[26] This particular transformational principle is so called because one instance of it [$D \vdash (D \rightarrow S) \rightarrow S$] corresponds to Montague's proposal (1973) to treat DNPs, including proper-nouns, as second-order predicates.

[27] We call this principle ‘inflection’ because it authorizes numerous compositions involving case-inflection, which we introduce in a later chapter.

6. Permutivity [28]

$(A \rightarrow C) \rightarrow D$; $A \rightarrow (B \rightarrow C) \vdash B \rightarrow D$

(1)	$(A \rightarrow C) \rightarrow D$	1	Pr
(2)	$A \rightarrow (B \rightarrow C)$	2	Pr
(3)	B	3	As
(4)	A	4	As
(5)	$B \rightarrow C$	24	2,4,→O
(6)	C	234	3,5,→O
(7)	$A \rightarrow C$	23	4,6,→I
(8)	D	123	1,7,→O
(9)	$B \rightarrow D$	12	3,8,→I
(7)	$A \rightarrow C$	12	3,6,→I

7. Addition

$B \rightarrow C$; $A \times B \vdash A \times C$

(1)	$B \rightarrow C$	1	Pr
(2)	$A \times B$	2	Pr
(3)	A	3	As
(4)	B	4	As
(5)	C	14	1,4,→O
(6)	$A \times C$	134	3,5,×I
(7)	$A \times C$	12	2,3-6,×O

8. Schönfinkel's Law [29] [Residual Law]

$(A \times B) \rightarrow C \dashv\vdash A \rightarrow (B \rightarrow C)$

$\vdash$

(1)	$(A \times B) \rightarrow C$	1	Pr
(2)	A	2	As
(3)	B	3	As
(4)	$A \times B$	23	2,3,×I
(5)	C	123	1,4,→O
(6)	$B \rightarrow C$	12	3,5,→I
(7)	$A \rightarrow (B \rightarrow C)$	1	2,6,→I

$\dashv$

(1)	$A \rightarrow (B \rightarrow C)$	1	Pr
(2)	$A \times B$	2	As
(3)	A	3	2, ×O
(4)	B	4	2, ×O
(5)	$B \rightarrow C$	13	1,3,→O
(6)	C	134	4,5,→O
(7)	C	12	2,3-6,×O
(8)	$(A \times B) \rightarrow C$	1	2,7,→I

[28] A combination of Transitivity and Permutation.

[29] Named after Moses Schönfinkel (1924), who first proposed that one can treat every two-place function as a family of one-place functions.

2. Relevance Logic (Without Tautology)

If we admit Contraction [$i+i \leqslant i$], we obtain the cross-arrow fragment of Relevance Logic,[30] which validates the following additional argument forms.

1. Contraction

$A \rightarrow (A \rightarrow B) \vdash A \rightarrow B$

(1)	A→(A→B)	1	Pr
(2)	A	2	As
(3)	A→B	12	1,2 →O
(4)	B	122	2,3,→O
(5)	B	12	4, Contraction
(6)	A→B	1	2,5,→I

2. Duplication

$A \vdash A \times A$

(1)	A	1	Pr
(2)	A×A	11	1,1,×I
(3)	A×A	1	2, Contraction

3. Conditional Modus Ponens

$A \rightarrow (B \rightarrow C) ; A \rightarrow B \vdash A \rightarrow C$

(1)	A→(B→C)	1	Pr
(2)	A→B	2	Pr
(3)	A	3	As
(4)	B→C	13	1,3,→O
(5)	B	23	2,3,→O
(6)	C	1233	4,5,→O
(7)	C	123	6, Contraction
(8)	A→C	12	3,7,→I

4. Conditional Multiplication

$A \rightarrow B ; A \rightarrow C \vdash A \rightarrow (B \times C)$

(1)	A→B	1	Pr
(2)	A→C	2	Pr
(3)	A	3	As
(4)	B	13	1,3,→O
(5)	C	23	2,3,→O
(6)	B×C	1233	4,5,×I
(7)	B×C	123	6, Contraction
(8)	A→(B×C)	12	3,7,→I

[30] Relevance Logic comprises a rich variety of logical systems. Unless stated otherwise, we concentrate on System R.

3. System R-Mingle

For the sake of thoroughness, we include an example that involves the Mingle rule.

1. Expansion

$A \rightarrow B \vdash A \rightarrow (A \rightarrow B)$

(1)	A→B	1	Pr
(2)	A	2	As
(3)	B	12	1,2,→O
(4)	B	122	3, Mingle
(5)	A→B	12	2,4,→I
(6)	A→(A→B)	1	2,5,→I

4. Linear/Relevant Logic With Tautology

Also, for the sake of thoroughness, we include examples involving the identity index ∅.[31]

1. Tautology

$\vdash A \rightarrow A$

(1)	A	1	As
(2)	A	1+∅	1,Identity
(3)	A→A	∅	1,2,→I

2. Assertion

$(A \rightarrow A) \rightarrow B \vdash B$

(1)	(A→A)→B	1	Pr
(2)	A	2	As
(3)	A	2+∅	2,Identity
(4)	A→A	∅	2,3,→I
(5)	B	1	1,4,→O

5. Intuitionistic Logic

Also, for the sake of thoroughness, we include an example involving Weakening and Identity.

1. Positive Paradox

$\vdash A \rightarrow (B \rightarrow A)$

(1)	A	1	Assumption
(2)	B	2	Assumption
(3)	A	1+2	1, Weakening
(4)	B→A	1	2,3,→Intro
(5)	B→A	∅+1	4, Identity
(6)	A→(B→A)	∅	1,5,→Intro

6. Multi-Linear Logic

Multi-Linear Logic is obtained from Linear Logic without Tautology by adding one further reasoning principle.

1. Conditional-Multiplication (CM)

$A \rightarrow B \; ; \; A \rightarrow C \vdash A \rightarrow (B \times C)$

Alternatively, especially if one wants a pure-arrow logic, one instead hypothesizes the following principle.

[31] As noted earlier in the chapter, Identity is implausible in Categorial Logic.

2. Conditional-Modus-Ponens (CMP)

$A \rightarrow (B \rightarrow C)$; $A \rightarrow B \vdash A \rightarrow C$

Whereas both are provable in Relevance Logic, as seen in the previous section, neither is provable in Linear Logic, although they are inter-derivable, as seen in the following derivations.

(1)	$A \rightarrow B$	1	Pr
(2)	$A \rightarrow C$	2	Pr
(3)	A	3	As
(4)	C	4	As
(5)	B	13	1,3,→O
(6)	$B \times C$	134	4,5,×I
(7)	$C \rightarrow (B \times C)$	13	4,6,→I
(8)	$A \rightarrow [C \rightarrow (B \times C)]$	1	3,7,→I
(9)	$A \rightarrow (B \times C)]$	12	2,8,CMP

(1)	$A \rightarrow (B \rightarrow C)$	1	Pr
(2)	$A \rightarrow B$	2	Pr
(3)	A	3	As
(4)	$A \rightarrow [(B \rightarrow C) \times B]$	12	1,2,CM
(5)	$(B \rightarrow C) \times B$	123	1,3,→O
(6)	$B \rightarrow C$	4	As
(7)	B	5	As
(8)	C	45	6,7,→O
(9)	C	123	5-8,×O
(10)	$A \rightarrow C$	12	3,9,→I

The natural question is how do we adjust our deductive apparatus so that we obtain these principles (and nothing more). There are three basic approaches.

(1) We simply add these as further (ad hoc) inference-rules.[32]
(2) We adjust the existing rules in a way that generates these forms.
(3) We adjust the structural (index) rules in a way that generates these forms.

From the practical point of view, the three approaches are equally good. For example, we can simply add the following rules.

3. Rule – Conditional-Multiplication

L_1	$\mathcal{A} \rightarrow \mathcal{B}$	i	
L_2	$\mathcal{A} \rightarrow C$	j	
L_3	$\mathcal{A} \rightarrow (\mathcal{B} \times C)$	i+j	L_1, L_2, CM

4. Rule – Conditional-Modus-Ponens

L_1	$\mathcal{A} \rightarrow \mathcal{B}$	i	
L_2	$\mathcal{A} \rightarrow (\mathcal{B} \rightarrow C)$	j	
L_3	$\mathcal{A} \rightarrow C$	i+j	L_1, L_2, CMP

Alternatively, we can generalize →O as follows.

[32] Either will do, since they are equivalent in Linear Logic.

5. Multi-Arrow-Out

L_1	$\mathcal{A}\rightarrow\mathcal{B}_1$	i_1	
…	…	…	
L_m	$\mathcal{A}\rightarrow\mathcal{B}_m$	i_m	
L_3	$\mathcal{A}$	k	
L_4	$\mathcal{B}_1\times\ldots\times\mathcal{B}_m$	$i_1 +\ldots+ i_m + k$	L_1-L_m, L_3, $\times\rightarrow$O

Here, the inference begins with m-many conditionals with the same antecedent. When $m = 1$ we obtain the usual arrow-out rule.

Practically speaking, these two approaches work OK, but theoretical considerations prompt us to adjust our structural rules instead. Unfortunately, the structural approach is much more complicated; somehow we have to implement a restricted form of index-contraction. This is accomplished by introducing two forms of index-composition, plus (+) and minus (–), satisfying the following reduction principles.

(1)	$i+(j+k) = (i+j)+k$	+ is associative
(2)	$i+j = j+i$	+ is commutative
(3)	$i–(j–k) = (i–j)–k$	– is associative
(4)	$i–j = j–i$	– is commutative
(5)	$(i–k)+(j–k) \leqslant (i+j)–k$	– distributes over +
(6)	$(i–k)–(j–k) \leqslant (i–j)–k$	– distributes over –
(7)	$i–(j+k) = (i–j)–k$	– and + are residually-related

The plus-operator is associated with the cross-operator; the inference rules for cross remain the same as before. The minus-operator is associated with the arrow operator; the inference rules for arrow are appropriately adjusted as follows.

6. Arrow-Out

L_1	$\mathcal{A}\rightarrow\mathcal{B}$	i	
L_2	$\mathcal{A}$	j	
L_3	$\mathcal{B}$	$i–j$	L_1, L_2, $\rightarrow$O

7. Arrow-In ($\rightarrow$I)

L_1	$\mathcal{A}$	i	
L_2	$\mathcal{B}$	$j–i$	
L_3	$\mathcal{A}\rightarrow\mathcal{B}$	j	L_1, L_2, $\rightarrow$I

We then can derive CMP and CM as follows.

8. Conditional Modus Ponens

$A\rightarrow(B\rightarrow C)$; $A\rightarrow B \vdash A\rightarrow C$

(1)	A→(B→C)	1	Pr
(2)	A→B	2	Pr
(3)	A	3	As
(4)	B→C	1–3	1,3,→O
(5)	B	2–3	2,3,→O
(6)	C	(1–3)–(2–3)	4,5,→O
(7)	C	(1–2)–3	6, Distribution
(8)	A→C	1–2	3,7,→I

9. Conditional Multiplication

$A \rightarrow B \; ; \; A \rightarrow C \vdash A \rightarrow (B \times C)$

(1)	$A \rightarrow B$	1	Pr
(2)	$A \rightarrow C$	2	Pr
(3)	A	3	As
(4)	B	1–3	1,3,→O
(5)	C	2–3	2,3,→O
(6)	$B \times C$	(1–3)+(2–3)	4,5,×I
(7)	$B \times C$	(1+2)–3	6, Distribution
(8)	$A \rightarrow (B \times C)$	1+2	3,7,→I

10. Residual Law – Schönfinkel's Law

For the sake of comparison, we re-do an earlier result using the new indexing scheme.

$(A \times B) \rightarrow C \dashv\vdash A \rightarrow (B \rightarrow C)$

$\vdash$

(1)	$(A \times B) \rightarrow C$	1	Pr
(2)	A	2	As
(3)	B	3	As
(4)	$A \times B$	2+3	2,3,×I
(5)	C	1–(2+3)	1,4,→O
(6)	C	(1–2)–3	5, Residuation
(7)	$B \rightarrow C$	1–2	3,6,→I
(8)	$A \rightarrow (B \rightarrow C)$	1	2,7,→I

$\dashv$

(1)	$A \rightarrow (B \rightarrow C)$	1	Pr
(2)	$A \times B$	2	As
(3)	A	3	As
(4)	B	4	As
(5)	$B \rightarrow C$	1–3	1,3,→O
(6)	C	(1–3)–4	4,5,→O
(7)	C	1–(3+4)	6, Residuation
(8)	C	1–2	2,3-7,×O
(9)	$(A \times B) \rightarrow C$	1	2,8,→I

11. Adding Tautology to MLL produces Relevance Logic

As mentioned earlier, one can recover System R by adding the identity element ∅, which is tantamount to adding tautologies [zero-premise arguments] to the system. The following demonstrate how Contraction and Duplication are demonstrated in the resulting system.

$A \rightarrow (A \rightarrow B) \vdash A \rightarrow B$

(1)	$A \rightarrow (A \rightarrow B)$	1	Pr
(2)	A	2	As
(3)	$A \rightarrow A$	∅	2,2,→I
(4)	$A \rightarrow B$	1+∅	1,3,CMP
(5)	$A \rightarrow B$	1	4, identity

$A \vdash A \times A$

(1)	A	1	Pr
(2)	A	2	As
(3)	$A \rightarrow A$	$\varnothing$	2,2,→I
(4)	$A \rightarrow (A \times A)$	$\varnothing$	3,3,CM
(5)	$A \times A$	1	1,4,→O

9. Summary of Logical Systems Considered

	Inference Principle	Name	Validated by
(1)	$A \rightarrow B ; A \vdash B$	Modus Ponens	C, I, R, L, M
(2)	$B \rightarrow C ; A \rightarrow B \vdash A \rightarrow C$	Transitivity	
(3)	$A \rightarrow (B \rightarrow C) \vdash B \rightarrow (A \rightarrow C)$	Permutation	
(4)	$A \rightarrow (B \rightarrow C) ; B \vdash A \rightarrow C$	Secondary Modus Ponens	
(5)	$A \vdash (A \rightarrow B) \rightarrow B$	Montague's Law	
(6)	$(A \rightarrow C) \rightarrow D ; A \rightarrow B \vdash (B \rightarrow C) \rightarrow D$	Inflection	
(7)	$(A \rightarrow C) \rightarrow D ; A \rightarrow (B \rightarrow C) \vdash B \rightarrow D$	Permutivity	
(8)	$B \rightarrow C ; A \times B \vdash A \times C$	Addition	
(9)	$(A \times B) \rightarrow C \dashv\vdash A \rightarrow (B \rightarrow C)$	Schönfinkel's Law	
(10)	$A \rightarrow B ; A \rightarrow C \vdash A \rightarrow (B \times C)$	Conditional-Multiplication	C, I, R, M
(11)	$A \rightarrow B ; A \rightarrow (B \rightarrow C) \vdash A \rightarrow C$	Conditional-Modus-Ponens	
(12)	$(A \rightarrow A) \rightarrow B \vdash B$	Assertion	C, I, R, L
(13)	$\varnothing \vdash A \rightarrow A$	Tautology	
(14)	$A \rightarrow (A \rightarrow B) \vdash A \rightarrow B$	Contraction	C, I, R
(15)	$A \vdash A \times A$	Duplication	
(16)	$A \times B \vdash A$	Simplification	C, I
(17)	$A \times B \vdash B$	Simplification	
(18)	$A \vdash B \rightarrow B$	Irrelevance	
(19)	$A \rightarrow B \vdash A \rightarrow (A \rightarrow B)$	Expansion	
(20)	$A \vdash B \rightarrow A$	Positive Paradox	
(21)	$(A \rightarrow B) \rightarrow B \vdash (B \rightarrow A) \rightarrow A$	Lukasiewicz's Law[33]	C
(22)	$(A \rightarrow B) \rightarrow A \vdash A$	Peirce's Law[34]	

[33] Named after Jan Łukasiewicz (1878-1956), who invented multi-valued logic. In his system, one can define disjunction in terms of conditional via: $A \vee B =_{df} (A \rightarrow B) \rightarrow B$. See Łukasiewicz (1920).

[34] Named after Charles Sanders Peirce (1839-1914); first presented in (1885).

Sub-Structural Features							
	Logic	**Association**	**Commutation**	**Weakening**	**Contraction**	**Identity**	**Residuation**
C	Classical	✓	✓	✓	✓	✓	✓
I	Intuitionistic	✓	✓	✓	✓	✓	✓
R	Relevance	✓	✓	✗	✓	✓	✓
L	Linear	✓	✓	✗	✗	✓	✓
M	Multi-Linear	✓	✓	✗	✗	✗	✓

10. Gentzen-Style Logic Systems

1. Introduction

We have mentioned sub-structural-logic principles, and we have mentioned that indices in derivations encode sub-structural information. In the present section, we examine the conventional manner of presenting sub-structural logics,[35] and we show how this information is encoded by indices.

Sub-structural logics are customarily presented in terms of Gentzen-style calculi, which are contrasted with Hilbert-style calculi. Both kinds of calculi are generative, which is to say that both posit primitive elements, and both posit methods to construct derivative elements. The difference concerns the underlying elements.

2. Hilbert-Style Calculi

For a Hilbert-style calculus, the underlying elements are formulas, and the calculus endeavors to generate the class of formulas that are logically-true. For example, the arrow-fragment of Intuitionistic Logic can be constructed as follows.

(A1) $\vdash X\rightarrow(Y\rightarrow X)$
(A2) $\vdash [X\rightarrow(Y\rightarrow Z)]\rightarrow[(X\rightarrow Y)\rightarrow(X\rightarrow Z)]$
(MP) if $\vdash X\rightarrow Y$ and $\vdash X$, then $\vdash Y$

Here, the letters ‘X’, ‘Y’, ‘Z’ are schematic variables, understood to be universally quantified, ranging over formulas of the underlying language, and the turnstile symbol ‘$\vdash$’ is a meta-linguistic predicate that may be read ‘… is logically-true (according to the calculus in question)’. A derivation is then a sequence of formulas every line of which is either an axiom or follows from previous lines by a rule. The following is a prominent example.[36]

(1)	$\vdash P\rightarrow((P\rightarrow P)\rightarrow P)$	A1
(2)	$\vdash [P\rightarrow((P\rightarrow P)\rightarrow P)]\rightarrow[(P\rightarrow(P\rightarrow P))\rightarrow(P\rightarrow P)]$	A2
(3)	$\vdash P\rightarrow(P\rightarrow P)$	A1
(4)	$\vdash (P\rightarrow(P\rightarrow P))\rightarrow(P\rightarrow P)$	1, 2, MP
(5)	$\vdash P\rightarrow P$	3, 4, MP

3. Gentzen-Style Calculi

By contrast, for a Gentzen-style calculus, the underlying elements are **sequents**,[37] which are formatted as follows.

$$\Gamma \vdash \Delta$$

Here Γ and Δ are **data-structures** built on formulas, and the turnstile-symbol is a meta-linguistic predicate defined so that we read ‘$\Gamma \vdash \Delta$’ as saying

the argument *from* Γ *to* Δ is valid

---

[35] In this connection, the reader is invited to consult Restall (2000).

[36] It is customary often to drop the turnstile ($\vdash$) and simply write the formula by itself. We include the turnstile in order to indicate exactly how the Hilbert-scheme is related to the Gentzen-scheme.

[37] The term ‘sequent’ is due to Kleene +++, which he proposed as a translation of Gentzen’s ‘sequenz’, which is German for ‘sequence’.

What sort of data-structures are Γ and Δ? According to the simplest and most natural approach, they are both sets (possibly empty), and the sequent

$$\Gamma \vdash \Delta$$

is usually understood *semantically* as saying that whenever *all* of Γ are true, *some* of Δ are true.

There are specializations/simplifications/variations of the general sequent form. For example, one can restrict sequents so that Δ is a singleton formula $\{\phi\}$, in which case a sequent is generically written as follows.

$$\Gamma \vdash \phi$$

This can be further simplified by restricting Γ to be the empty-set, the following being an example.

$$\varnothing \vdash P \rightarrow P$$

This may be read as saying that the formula 'P→P' follows logically from nothing, which is to say it is logically-true. In this way we see that Hilbert-calculi are a special case of Gentzen-calculi.

Other data-structures are worth considering, including multi-sets and sequences. Restall (2000) catalogs a multitude of data-structures, based on two constructors, which he formally writes using semi-colon and comma, which behave mathematically as two-place operators. We concentrate on the semi-colon operator,[38] and we also concentrate on singleton-conclusions, since these are most relevant to Categorial Logic. Then the following are example sequents, where A-D are formulas.

$$A \; ; \; B \vdash C$$
$$(A \; ; \; B) \; ; \; C \vdash D$$

Note the parentheses; we do not presume that the semi-colon operator is associative, nor do we presume that it is commutative. Rather, we take these to be optional structural principles that may characterize a logic. For example, Restall considers the following structural principles, among others, where A-D are arbitrary formulas. Note that double-lines indicate bi-directional processing.

Association 1
… (A;B);C … ⊢ D
… A;(B;C) … ⊢ D

Association 2
… A;(B;C) … ⊢ D
… (A;B);C … ⊢ D

Commutation
… A;B … ⊢ C
… B;A … ⊢ C

Contraction
… A;A … ⊢ B
… A … ⊢ B

Mingle
… A;A … ⊢ B
… A … ⊢ B

Weakening 1
… A … ⊢ C
… A;B … ⊢ C

Weakening 2
… A … ⊢ C
… B;A … ⊢ C

[38] The semi-colon corresponds to our plus-operator.

To these we add the following rule.

Identity-Element 1
… ∅ ; A … ⊢ C
… A … ⊢ C

Identity-Element 2
… A … ⊢ C
… A ; ∅ … ⊢ C

In the above rules, the ellipsis-symbols indicate contexts within which the data-structures are placed, including material on the left and right (both perhaps empty).

Next, there is a fundamental Axiom, known as *Identity*.

A ⊢ A

Here, A is any formula. Instances of Identity are primitive sequents. Derivative sequents are constructed in accordance with rules, including structural-rules mentioned above, and rules governing connectives, including the connectives relevant to Categorial Logic, given as follows.

→ Elimination	
Γ ⊢ A→B	Δ ⊢ A
Γ ; Δ ⊢ B	

→ Introduction
Γ ; A ⊢ B
Γ ⊢ A→B

× Elimination	
Γ ⊢ A×B	… A ; B … ⊢ C
… Γ … ⊢ C	

× Introduction	
Γ ⊢ A	Δ ⊢ B
Γ ; Δ ⊢ A×B	

Next, a derivation consists of a sequence of sequents each one of which is either primitive or is constructed from earlier sequents by a rule. The following are simple examples.

1. Example 1

(1)	A→B ⊢ A→B	Identity
(2)	A ⊢ A	Identity
(3)	A→B ; A ⊢ B	1,2, →Elim

2. Example 2

(1)	A ⊢ A	Identity
(2)	A ; B ⊢ A	1, Weakening 1
(3)	A ⊢ B→A	2, →Intro
(4)	∅ ⊢ A→(B→A)	3, →Intro

4. Natural Deduction Systems

Between Hilbert-Style systems and Gentzen-style systems are natural-deduction systems, which countenance derivations that allow premises and assumptions in derivations. Different schemes are devised for quarantining assumptions. One such scheme uses assumption-dependence indexing. Most authors place the indices to the left of the line-numbers. We place the indices to the right of the formulas. The following example illustrates the difference. Note that we use the semi-colon, rather than plus, in order to render it more similar to Gentzen-style derivations, which employ punctuation-marks instead of algebraic symbols.

(1)	A→B	1	Premise
(2)	A	2	Premise
(3)	B	1;2	1,2, →Elim

1	(1)	A→B	Premise
2	(2)	A	Premise
1;2	(3)	B	1,2, →Elim

Line 1 is a premise, which depends upon line 1 (i.e., itself); line 2 is also a premise, which depends upon line 2 (i.e., itself). Line 3 is built from lines 1 and 2, and so depends upon what they depend upon. Note

Notice that these derivations are isomorphic to Gentzen Example 1 above. The following exhibits the transformation from one to the other.

(1)	A→B ⊢ A→B	Identity
(2)	A ⊢ A	Identity
(3)	A→B ; A ⊢ B	1,2, →Elim

A→B	(1)	A→B	?
A	(2)	A	?
A→B ; A	(3)	B	1,2, →Elim

1	(1)	A→B	Premise
2	(2)	A	Premise
1 ; 2	(3)	B	1,2, →Elim

What about Example 2? This does not convert so easily; extra lines are needed.[39]

(1)	A ⊢ A	Identity
(2)	A ; B ⊢ A	1, Weakening 1
(3)	A ⊢ B→A	2, →Intro
(4)	∅ ⊢ A→(B→A)	3, →Intro

(1)	A	1	Assumption
(2)	B	2	Assumption
(3)	A	1;2	1, Weakening
(4)	B→A	1	2,3,→Intro
(5)	B→A	∅;1	4, Identity-Element 2
(6)	A→(B→A)	∅	1,5,→Intro

5. Examples of Gentzen-Style Derivations

In this section, we exhibit derivations of earlier results, but now using the Gentzen scheme, with the corresponding natural-deduction to the right. Once again, we combine indices using semi-colon instead of plus.

1. Transitivity

B→C ; A→B ⊢ A→C

(1)	B→C ⊢ B→C	Identity
(2)	A→B ⊢ A→B	Identity
(3)	A ⊢ A	Identity
(4)	A→B ; A ⊢ B	2,3,→O
(5)	B→C ; A→B ; A ⊢ C	1,4,→O
(6)	B→C ; A→B ⊢ A→C	5,→I

B→C	1	Pr
A→B	2	Pr
A	3	As
B	2;3	2,3,→O
C	1;2;3	1,4,→O
A→C	1;2	3,5,→I

2. Permutation

A→(B→C) ⊢ B→(A→C)

(1)	A→(B→C) ⊢ A→(B→C)	Identity
(2)	B ⊢ B	Identity
(3)	A ⊢ A	Identity
(4)	A→(B→C) ; A ⊢ B→C	1,3,→O
(5)	A→(B→C) ; A ; B ⊢ C	2,4,→O
	A→(B→C) ; B ; A ⊢ C	Association
(6)	A→(B→C) ; B ⊢ A→C	5,→I
(7)	A→(B→C) ⊢B→(A→C)	6,→I

A→(B→C)	1	Pr
B	2	As
A	3	As
B→C	1;3	1,3,→O
C	1;3;2	2,4,→O
	1;2;3	Ass
A→C	1;2	3,5,→I
B→(A→C)	1	2,6,→I

3. Secondary Modus Ponens

A→(B→C) ; B ⊢ A→C

(1)	A→(B→C) ⊢ A→(B→C)	Identity
(2)	B ⊢ B	Identity
(3)	A ⊢ A	Identity
(4)	A→(B→C) ; A ⊢ B→C	1,3,→O
(5)	A→(B→C) ; A ; B ⊢ C	2,4,→O
	A→(B→C) ; B ; A ⊢ C	Association
(6)	A→(B→C) ; B ⊢ A→C	5,→I

A→(B→C)	1	Pr
B	2	Pr
A	3	As
B→C	1;3	1,3,→O
C	1;3;2	2,4,→O
	1;2;3	Assoc
A→C	1;2	3,5,→I

[39] Note that the slight mis-match between the Gentzen-style derivation and the natural-deduction derivation is not ultimately a problem for Categorial Logic [Multi-Linear Logic], since it does not posit Weakening or Identity.

4. Montague's Law [Lifting]

$A \vdash (A \rightarrow B) \rightarrow B$

(1)	$A \vdash A$	Identity
(2)	$A \rightarrow B \vdash A \rightarrow B$	Identity
(3)	$A \rightarrow B \; ; A \vdash B$	1,2,→O
	$A \; ; A \rightarrow B \vdash B$	Commutation
(4)	$A \vdash (A \rightarrow B) \rightarrow B$	3,→I

A	1	Pr
A→B	2	As
B	2;1	1,2,→O
	1;2	Commu
(A→B)→B	1	2,4,→I

6. Gentzen-Style System for Categorial Logic [Multi-Linear Logic]

Finally, we offer a Gentzen-calculus for Categorial Logic. First, it posits two sub-structural operators – semi-colon and colon[40] – subject to the following principles.[41] Note that double-lines indicate bi-directional reasoning.

Association [:]
… (A**:**B)**:**C … ⊢ D
… A**:**(B**:**C) … ⊢ D

Association [;]
… A;(B;C) … ⊢ D
… (A;B);C … ⊢ D

Commutation [:]
… A**:**B … ⊢ C
… B**:**A … ⊢ C

Commutation [;]
… A;B … ⊢ C
… B;A … ⊢ C

Distribution [:/:]
… (A**:**C)**:**(A**:**C) … ⊢ D
… (A**:**B)**:**C … ⊢ D

Distribution [;/:]
… (A**:**C);(B**:**C) … ⊢ D
… (A;B)**:**C … ⊢ D

Residuation [42]
… A**:**(B;C) … ⊢ D
… (A**:**B)**:**C … ⊢ C

→ Elimination	
Γ ⊢ A→B	Δ ⊢ A
Γ **:** Δ ⊢ B	

→ Introduction
Γ **:** A ⊢ B
Γ ⊢ A→B

× Elimination	
Γ ⊢ A×B	… A;B … ⊢ C
… Γ … ⊢ C	

× Introduction	
Γ ⊢ A	Δ ⊢ B
Γ ; Δ ⊢ A×B	

Finally, we posit a rule that disallows empty premises, which amounts to disallowing tautologies. As mentioned earlier, admitting tautologies corresponds grammatically to allowing phrases to enter a derivation out of thin air, which is grammatically implausible. Note also that, as we in effect demonstrated earlier, if we allow empty premises, then the above logical system reduces to the cross-arrow fragment of Relevance Logic.

---

[40] The colon (:) is often used for proportionality, as in a:b [a is to b], which is analogous to division.

[41] We could use the same symbols [plus and minus] for Gentzen systems as for natural-deduction systems, but then they look like sentence connectives, which causes visual confusion. So we use punctuation-marks between formulas, and we use arithmetic-marks between indices.

[42] Note that Residuation simply reduces to Association if we have just one form of premise-composition governing both × and →.

The following are examples of derivations, done both using the Gentzen-scheme and using the natural-deduction scheme from earlier. Note once again that we employ punctuation-symbols to combine premises, and arithmetic-symbols to combine indices.

1. Residual Law – Schönfinkel's Law

$(A\times B)\rightarrow C \vdash A\rightarrow(B\rightarrow C)$

(1)	$(A\times B)\rightarrow C \vdash (A\times B)\rightarrow C$	Identity
(2)	$A \vdash A$	Identity
(3)	$B \vdash B$	Identity
(4)	$A\ ;\ B \vdash A\times B$	2,3,×Intro
(5)	$(A\times B)\rightarrow C\ :\ (A;B) \vdash C$	1,4,→Elim
(6)	$(A\times B)\rightarrow C\ :\ A\ :\ B \vdash C$	5, Residual
(7)	$(A\times B)\rightarrow C\ :\ A \vdash B\rightarrow C$	3,6,→Intro
(8)	$(A\times B)\rightarrow C \vdash A\rightarrow(B\rightarrow C)$	2,7,→Intro

$(A\times B)\rightarrow C$	1	Pr
A	2	As
B	3	As
$A\times B$	2+3	2,3,×I
C	1–(2+3)	1,4,→O
C	(1–2)–3	5, Residual
$B\rightarrow C$	1–2	3,6,→I
$A\rightarrow(B\rightarrow C)$	1	2,7,→I

$A\rightarrow(B\rightarrow C) \vdash (A\times B)\rightarrow C$

(1)	$A\rightarrow(B\rightarrow C) \vdash A\rightarrow(B\rightarrow C)$	Identity
(2)	$A\times B \vdash A\times B$	Identity
(3)	$A \vdash A$	Identity
(4)	$B \vdash B$	Identity
(5)	$A\rightarrow(B\rightarrow C)\ :\ A \vdash B\rightarrow C$	1,3,→Elim
(6)	$A\rightarrow(B\rightarrow C)\ :\ A\ :\ B \vdash C$	4,5,→Elim
(7)	$A\rightarrow(B\rightarrow C)\ :\ (A\ ;\ B) \vdash C$	6, Residual
(8)	$A\rightarrow(B\rightarrow C)\ :\ (A\times B) \vdash C$	2,7,×Elim
(9)	$A\rightarrow(B\rightarrow C) \vdash (A\times B)\rightarrow C$	8,→Intro

$A\rightarrow(B\rightarrow C)$	1	Pr
$A\times B$	2	As
A	3	As
B	4	As
$B\rightarrow C$	1–3	1,3,→O
C	(1–3)–4	4,5,→O
C	1–(3+4)	6, Residual
C	1–2	3-7,×O
$(A\times B)\rightarrow C$	1	2,8,→I

11. The Lambek Calculus

1. Introduction

The logical systems we have considered so far are formal systems originally intended to model propositional reasoning, not grammatical construction. On the other hand, we have evaluated these systems according to whether their various reasoning principles can plausibly be re-purposed as principles of grammatical composition, more specifically semantic composition.

We conclude this unit by briefly examining a calculus that was designed specifically to model grammatical construction – namely, the Lambek Calculus, originally proposed by Joachim Lambek (1958).

The Lambek Calculus fits into the sub-structural scheme; in particular, it may be characterized sub-structurally as follows.

Logic	Association	Commutation	Weakening	Contraction	Identity	Residuation
Lambek	✓	✗	✗	✗	✗	✓

Thus, the Lambek Calculus is *basically* Linear Logic minus Tautology and minus Commutativity.

On the other hand, since + is not commutative, there are in principle two forms of residuation – left and right, which may be written as follows.

(1) right-residuation $\alpha \dashv (\beta+\gamma) = (\alpha \dashv \beta) \dashv \gamma$

(2) left-residuation $(\alpha+\beta) \vdash \gamma = \alpha \vdash (\beta \vdash \gamma)$

Here we have right-subtraction ($\dashv$) and left-subtraction ($\vdash$). These equations employ plus-minus notation, and pertain to index-composition. There is also cross-slash notation, which Lambek employs, which pertain to formula-composition, and which satisfy corresponding principles concerning multiplication and division.

(1) right-residuation $\alpha \,/\, (\beta\times\gamma) = (\alpha \,/\, \beta) \,/\, \gamma$

(2) left-residuation $(\alpha\times\beta) \backslash \gamma = \alpha \backslash (\beta \backslash \gamma)$

Here we have right-division $[\ /\ ]$ and left-division $[\ \backslash\ ]$.

Reading 'α / β' as "alpha **given** beta", as in *conditional* probability, we can convert slash-notation into back-arrow-notation '$\alpha\leftarrow\beta$' ["α **if** β"]. Reversing slash and back-arrow yields the parallel correspondence between back-slash $[\backslash]$ and arrow $[\rightarrow]$, written as follows.

(1) right-residuation $\alpha\leftarrow(\beta\times\gamma) = (\alpha\leftarrow\beta)\leftarrow\gamma$
(2) left-residuation $(\alpha\times\beta)\rightarrow\gamma = \alpha\rightarrow(\beta\rightarrow\gamma)$

The derivation rules for Lambek Calculus may then be given as follows.[43]

1. Cross-In (×I)

L_1	$\mathcal{A}$	i	
L_2	$\mathcal{B}$	j	
L_3	$\mathcal{A} \times \mathcal{B}$	i+j	L_1, L_2, ×I

2. Cross-Out (×O)

L_0	$\mathcal{A}\times\mathcal{B}$	i	…
L_1-L_2	$\mathcal{A};\mathcal{B} \mapsto C$	j	…
L_3	C	i+j	L_0, L_1-L_2, ×O

3. Right-Arrow-In (→I)

L_1		$\mathcal{A}$	i	
L_2		$\mathcal{B}$	i+j	
L_3		$\mathcal{A}\rightarrow\mathcal{B}$	j	L_1, L_2, →I

4. Right-Arrow-Out (→O)

L_1	$\mathcal{A}$	i	
L_2	$\mathcal{A}\rightarrow\mathcal{B}$	j	
L_3	$\mathcal{B}$	i+j	L_1, L_2, →O

5. Left-Arrow-In (←I)

L_1	$\mathcal{A}$	i	
L_2	$\mathcal{B}$	j+i	
L_3	$\mathcal{B}\leftarrow\mathcal{A}$	j	L_1, L_2, ←I

6. Left-Arrow-Out (←O)

L_1	$\mathcal{A}\leftarrow\mathcal{B}$	i	
L_2	$\mathcal{B}$	j	
L_3	$\mathcal{A}$	i+j	L_1, L_2, ←O

[43] Note carefully that, in the cross-rules, the order of premises matters.

2. Examples of Derivations in the Lambek Calculus

The Lambek Calculus is associative, so we can write indices without parentheses. On the other hand, it is not commutative, so we must carefully maintain the left-right order of indices.

1. Transitivity

$A \rightarrow B \; ; \; B \rightarrow C \vdash A \rightarrow C$

(1)	A→B	1	Pr
(2)	B→C	2	Pr
(3)	A	3	As
(4)	B	31	1,3,→O
(5)	C	312	2,4,→O
(6)	A→C	12	3,5,→I

$A \leftarrow B \; ; \; B \leftarrow C \vdash A \leftarrow C$

(1)	A←B	1	Pr
(2)	B←C	2	Pr
(3)	C	3	As
(4)	B	23	1,3,→O
(5)	A	123	2,4,→O
(6)	A←C	12	3,5,→I

2. Permutation

$A \rightarrow (C \leftarrow B) \dashv\vdash (A \rightarrow C) \leftarrow B$

$\vdash$

(1)	A→(C←B)	1	Pr
(2)	B	2	As
(3)	A	3	As
(4)	C←B	31	1,3,→O
(5)	C	312	2,4,→O
(6)	A→C	12	3,5,→I
(7)	(A→C)←B	1	2,6,←I

$\dashv$

(1)	(A→C)←B	1	Pr
(2)	A	2	As
(3)	B	3	As
(4)	A→C	13	1,3,←O
(5)	C	213	2,4,→O
(6)	C←B	21	3,5,←I
(7)	A→(C←B)	1	2,6,→I

3. Secondary Modus Ponens

$A \rightarrow (C \leftarrow B) \; ; \; B \vdash A \rightarrow C$

(1)	A→(C←B)	1	Pr
(2)	B	2	Pr
(3)	A	3	As
(4)	C←B	31	1,3,→O
(5)	C	312	2,4,←O
(6)	A→C	12	3,5,→I

B ; (B→C)←A ⊢ C←A

(1)	B	1	Pr
(2)	(B→C)←A	2	Pr
(3)	A	3	As
(4)	B→C	23	2,3,←O
(5)	C	123	1,4,→O
(6)	C←A	12	3,5,←I

4. Lifting

A ⊢ B←(A→B)

(1)	A	1	Pr
(2)	A→B	2	As
(3)	B	12	1,2,→O
(4)	B←(A→B)	1	2,3,←I

A ⊢ (B←A)→B

(1)	A	1	Pr
(2)	B←A	2	As
(3)	B	21	1,2,←O
(4)	(B←A)→B	1	2,3,→I

3. Example Syntactic-Trees Based on the Lambek Calculus

The motivation for constructing a *non-commutative* is to formally represent the fundamental syntactic fact that word-order *often* matters to proper phrase-construction. For example, ‘the dog’ is a proper phrase of English, but ‘dog the’ is not. This distinction is represented in the following phrase-structure trees.

the	dog
D←(D→S)	D→S
D	

dog	the
D→S	D←(D→S)
※	

The following is the phrase-structure for a somewhat more complicated phrase.

every	man	respects	Kay
[S←(D→S)]←(D→S)	D→S	(D→S)←D	D
S←(D→S)		D→S	
S			

Notice that some functors take arguments on their right, and others take arguments on their left, which is indicated by the arrow's direction.

The following example admits two parsings.

Jay	respects	Kay
	(D→S)←D	D
D	D→S	
S		

Jay	respects	Kay
D	(D→S)←D	
S←D		D
S		

The composition of ‘Jay respects’ is underwritten by the following derivation.

D ; (D→S)←D ⊢ S←D

(1)	D	1	Pr
(2)	(D→S)←D	2	Pr
(3)	D	3	As
(4)	D→S	23	2,3,←O
(5)	S	123	1,4,→O
(4)	S←D	12	3,5,←I

The following is a very similar example, but alas one that the Lambek Calculus does not properly construct.

Jay	respects	every	woman
		[S←(D→S)]←(D→S)	D→S
	(D→S)←D	S←(D→S)	
D	S←D		
※			

Whereas it is generally agreed that ‘respects every woman’ is a constituent, it is not constructible by the Lambek Calculus. In the above phrase-structure, it looks like the phrase can be constructed using transitivity, but this is illegitimate since the input order is wrong. Consider the following quasi-derivation.

(D→S)←D ; S←(D→S) ⊢ S←D

(1)	(D→S)←D	1	Pr
(2)	S←(D→S)	2	Pr
(3)	D	3	As
(4)	D→S	13	1,3,←O
(5)	S	213	2,4,←O
(6)	S←D	21	3,5,←I

The following alternative parsing does not fare much better.

Kay	respects	every	man
D	(D→S)←D	[S←(D→S)]←(D→S)	D→S
S←D		S←(D→S)	
※			

4. Final Note on Lambek Calculus

The Lambek Calculus is concerned with generating the class of syntactically-admissible strings in a given language. As noted earlier, the string ‘the dog’ is syntactically-admissible in English, whereas ‘dog the’ is not. On the other hand, we are principally concerned with semantics, and hence semantic-admissibility, which is not concerned so much with word-*order* as with word-*grouping*, which is conveyed by the tree-structure of the sentence. A tree-parsing tells us what phrases combine and in what order; for example, ‘brown’ combines with ‘dog’ to make ‘brown dog’ which subsequently combines with ‘the’ to make ‘the brown dog’. Whether ‘brown’ precedes ‘dog’ or the other way around is irrelevant to how they combine *semantically*. On the other hand, word-order often provides clues concerning the underlying *semantic* structure. For example, ‘Jay kisses/kicks/kills Kay’ doesn't mean the same thing as ‘Kay kisses/kicks/kills Jay’. We thoroughly discuss this issue in later sections and later chapters.

D. Expanded Categorial Semantics

1. Introduction

So far, we have a new method to compose *types*, but we do not have a corresponding method to compose semantic-values.[44] In what follows, we develop such a method.

2. Expanded Loglish – Type-Theory With Multiplication

Categorial Logic, as we have presented it above, includes a multiplication connective, $\times$, which corresponds to logical-conjunction for Intuitionistic Logic and Classical Logic, and logical-fusion for Relevance Logic and Linear Logic. This logical-connective gives rise to a corresponding type-operator in Type Theory, characterized as follows.

(1)

if	A	is a type
and	B	is a type
then	$[A \times B]$	is a type

(2)

if	α	is an expression of type	A
and	β	is an expression of type	B
then	$[\alpha \times \beta]$	is an expression of type	$[A \times B]$

Note that we use the same symbol for both the syntactic-operator and for the type-operator. We think of the product $\alpha \times \beta$ as the **un-ordered pair** consisting of α and β.[45]

3. Semantic-Composition

Just as we expand our rules for type-composition, we also expand our rules of semantic-composition, which includes the following replacement for Function-Application.

Categorial Logic Principle (for compositing phrases)		
Where Φ_0 , …, Φ_k are phrases of Loglish.		
Φ_1, …, Φ_k	combine to form	Φ_0
IF		
the argument form	Φ_1 , …, Φ_k / Φ	
is valid according to Categorial Logic **expanded to include phrases of Loglish**		

4. Categorial Logic – Semantic Version

Earlier we presented a formal account of Categorial Logic as it pertains to syntactic-composition. We now do the same thing for semantic-composition. Although semantic-derivations are very similar, they are significantly enlarged, to include semantic-values. The major difference is that, whereas the type-derivation system posits rules pertaining to $\rightarrow$ [and $\times$], the semantic-derivation system posits rules pertaining to λ [and $\times$]. The correspondence is straightforward.

$\rightarrow$ Rules	
$\rightarrow$ Out	*Modus Ponens*
$\rightarrow$ In	Conditional-Generation

λ Rules	
λ Out	Function-Application
λ In	Function-Generation

[44] The connection between proof-theory and the lambda-calculus traces to Haskel Curry and William Alvin Howard , who proposed what is known as the Curry-Howard Isomorphism. See, e.g., W. Howard (1980). Our own calculus is closely allied with the compositional-calculus implicit in the CH-isomorphism, although our lambda-calculus is bigger and our categorial logic is smaller.

[45] Note that, in set theory, A×B is the Cartesian product of sets A and B, which consists of **ordered**-pairs $\langle x,y \rangle$ such that $x \in$ A and $y \in$ B. Our type A×B consists of **un-ordered** pairs. We propose to use a circle-cross symbol $[\otimes]$ for (sets of) ordered-pairs.

5. Definition of Derivation

Earlier we presented a derivation system for type-composition; semantic derivations build on this system by adding a column of Loglish expressions, whose types mimic the "formulas" of our earlier system. In particular, where α_0, …, α_m are expressions of Loglish, with types T_0, …, T_m, a **derivation of** α_0 **from** $\{\alpha_1, \ldots, \alpha_m\}$ is a sequence of lines organized as follows.

line number	expression	type	index	annotation
1	α_1	T_1	1	Pr
…	…	…	…	…
m	α_m	T_m	m	Pr
…	…	…	…	…
…	α_0	T_0	$1+\ldots+m$	…

In particular:

(1)	the first k lines are the premises – α_1, …, α_k.
(2)	every remaining line is either (1) an assumption, or (2) follows from previous lines by an inference-rule.
(3)	the last line is α_0.
(4)	T_m is the type of α_m
(5)	indices are sequences of numerals; + is sequential-sum

Note that we adopt an indexing system from Linear Logic without Tautology, which treats indices as sequences of numerals in numerical order. We treat function-multiplication [the semantic analog of conditional-multiplication] as an additional composition rule on top of the usual IN/OUT rules.[46]

6. Indices

Indices, which are numerical sequences, track assumption-dependence; each new assumption (premise or provisional-assumption) is assigned a new numeral. The sequential-sum operation, +, satisfies the following algebraic principles.

(1)	$(i + j) + k = i + (j + k)$	associative
(2)	$i + j = j + i$	commutative

7. Premise-Rule

At the beginning of a derivation, before any other rule is applied, one may write down any expression as a **premise**, in accordance with the following schema.

L	α (**new**)	A	m (**new**)	Pr

Here, α any open expression of type A, every unbound variable of which is new, which is to say it does not occur unbound earlier in the derivation, and m is any numeral that is new, which is to say it does not occur earlier in the derivation.

[46] In particular, we do not follow the "heroic" pure-sub-structural approach presented in Unit B.

8. Assumption-Rule

At any point in a derivation, one may write down any expression as an **assumption**, which looks thus.

L	α (**new**)	A	m (**new**)	As

Here, α any open expression of type A, every unbound variable of which is new, which is to say it does not occur unbound earlier in the derivation, and m is any numeral that is new, which is to say it does not occur earlier in the derivation.

9. Inference-Rules

1. Lambda-Out (λO) [function-application]

L_1	Λ	$A \rightarrow B$	i	…
L_2	α	A	j	…
L_3	[Λ]⟨α⟩	B	$i+j$	L_1, L_2, λO

Here, Λ is any function-expression [usually a lambda-abstract], α is any expression, and [Λ]⟨α⟩ is the result of applying Λ to α.
See section below on **lambda-conversion**.

2. Lambda-In (λI) [function-generation]

L_1	α	A	i	…
L_2	β	B	$i+j$	…
L_3	λ:α:β	$A \rightarrow B$	j	L_1, L_2, λI

3. Cross-Out (×O)

L_0	α × β	$A \times B$	i	…
L_1-L_2	α, β ↪ γ	…	j	…
L_3	γ	C	$i+j$	L_0, L_1-L_2, ×O

Here, α,β↪γ is a sub-derivation of γ from α, β, dependent on j,[47] which is a sequence schematically presented as follows.

A **sub-derivation** of β from α_1, …, α_k, dependent on j is a sequence arranged as follows.

L_1	α_1	A_1	m_1 (new)	As
…	…	…	…	…
L_k	α_k	A_k	m_k (new)	As
…	…	…	…	…
L_0	β	B	$m_1 + m_k + j$	…

4. Alternative Account of Lambda-In

L_1-L_2	α ↪ β	…	j	…
L_0	λα:β	$A \rightarrow B$	j	L_1-L_2, λI

Here, α↪β is a sub-derivation of β from α, dependent on j.

[47] Note that j may be empty, although no actual line will ever have an empty-index.

5. Cross-In (×I)

L_1	α	A	i	…
L_2	β	B	j	…
L_3	$\alpha \times \beta$	$A \times B$	$i+j$	L_1, L_2, ×I

6. Function-Multiplication

So far, the rules yield the sub-structural logic we call *Linear Logic Without Tautology*. The logical system we propose, which we call *Multi-Linear Logic*, is a strengthening of this obtained by adding the following semantic-composition rule.

L_1	$\lambda\, \alpha : \beta$	$A \rightarrow B$	i	…
L_2	$\lambda\, \alpha : \gamma$	$A \rightarrow C$	j	…
L_3	$\lambda\alpha : \beta \times \gamma$	$A \rightarrow (B \times C)$	$i+j$	L_1, L_2, FM

10. Lambda-Calculus

1. Lambda-Conversion

$[\lambda\nu\mathcal{E}]\langle\sigma\rangle$	//		$\mathcal{E}[\sigma/\nu]$

(1) ν is any variable;
(2) σ is any expression of the **same type** as ν;
(3) $\mathcal{E}[\sigma/\nu]$ results from substituting σ for every occurrence of ν that is **free** in $\mathcal{E}$ **for** σ.

This schema is understood as a bi-directional rule ['//' is like identity.], licensing inter-substitution of the flanking expressions in all contexts.

2. Freedom and Bondage; Open and Closed Expressions

(1) Where $\aleph$ is an abstractor, ν is a variable, and $\mathcal{E}$ is an expression, every *occurrence* of ν in $\aleph\nu\mathcal{E}$ is bound by $\aleph$.[48]

(2) An *occurrence* $\mathbf{o}$ of a variable ν is **free** in expression $\mathcal{E}$ if and only if $\mathbf{o}$ is not bound by an abstractor.

(3) A variable ν is **free** in $\mathcal{E}$ if and only if at least one *occurrence* of ν is free.

(4) A variable ν is **free for** Σ in $\mathcal{E}$ if and only if every variable that is free in Σ is also free in $\mathcal{E}[\Sigma/\nu]$.

(5) An expression $\mathcal{E}$ is **closed** if and only if no variable is free in $\mathcal{E}$.

(6) An expression $\mathcal{E}$ is **open** if and only if it is not closed.

[48] The letter 'ℵ' is the Hebrew aleph, which is a cognate of Greek alpha, both deriving from the Phoenician letter '𐤀' (aleph). It is most famously used by the mathematician Georg Cantor (1845-1918) in his theory of transfinite numbers (1874).

3. Alphabetic Variance (AV)

$\mathcal{E}_1$ // $\mathcal{E}_2$
Here, $\mathcal{E}_1$ and $\mathcal{E}_2$ are alphabetic variants of each other; in other words, there is a permutation (1–1 function) π on the class **V** of variables, whose inverse is π^{-1}, such that, for any variable ν, $\mathcal{E}_1$ results by substituting $\pi(\nu)$ for every *bound* occurrence of ν in $\mathcal{E}_2$, and $\mathcal{E}_2$ results by substituting $\pi^{-1}(\nu)$ for every *bound* occurrence of ν in $\mathcal{E}_1$.
An occurrence of a variable ν is bound precisely if that occurrence lies within the scope of an operator binding ν – i.e., $\forall\nu$, $\exists\nu$, $\iota\nu$, $\lambda\nu$. Otherwise, that occurrence is free. A variable ν is free in an expression $\mathcal{E}$ precisely at least one occurrence of ν in $\mathcal{E}$ is free in $\mathcal{E}$.

11. Further Composition Rules

As noted earlier, Categorial Logic principles do not exhaust the list of composition methods. We also have Conjunction, which was originally introduced for combining one-place predicates [D→S], as follows.

1. Conjunction [Original Form]

L_1	$\lambda\nu\ \Phi$	D→S	i	…
L_2	$\lambda\nu\ \Psi$	D→S	j	…
L_3	$\lambda\nu\ \{\Phi\ \&\ \Psi\}$	D→S	i+j	L_1, L_2, Conj.
ν is a type-D variable; Φ and Ψ are formulas.				

In Expanded Categorial Grammar, just as we expand the logic-looking rules, we also expand Conjunction. First, we posit the following fundamental form.

2. Conjunction [Fundamental Form – VVV] [49]

L_1	$\Phi_1 \times \ldots \times \Phi_k$	S × ... × S	i	…
L_2	$\Phi_1\ \&\ \ldots\ \&\ \Phi_k$	S	i	L_1, Conj
$\Phi_1, \ldots, \Phi_k$ are formulas				

The original Conjunction rule can then be derived, an instance of which goes as follows. Note that function-multiplication [conditional-multiplication] is critical.

(1)	$\lambda x\mathbf{F}x$	D→S	1	Pr
(2)	$\lambda x\mathbf{G}x$	D→S	2	Pr
(3)	x	D	3	As
(4)	$\lambda x[\mathbf{F}x \times \mathbf{G}x]$	D→(S×S)	12	1,2, FM
(4)	$\mathbf{F}x \times \mathbf{G}x$	S×S	123	3,4, λO
(5)	$\mathbf{F}x\ \&\ \mathbf{G}x$	S	123	4, Conj
(4)	$\lambda x(\mathbf{F}x\ \&\ \mathbf{G}x)$	D→S	12	3,5, λI

Indeed, the original rule can be widely generalized as follows.

[49] Our nickname '**VVV**' is short for 'Veni Vidi Vici' [I came, I saw, I conquered] – reputedly said by Julius Caesar after his swift victory against Pharnaces II, King of Pontus and the Kingdom of Cimmerian Bosporus.

3. Conjunction [General Form]

L_1	$\lambda\nu\ \Phi_1$	$A\rightarrow S$	I_1	…
…	…	…		
L_k	$\lambda\nu\ \Phi_k$	$A\rightarrow S$	I_k	…
L_0	$\lambda\nu\ \{\ \Phi_1\ \&\ \ldots\ \&\ \Phi_k\ \}$	$A\rightarrow S$	$I_1 + I_k$	$L_1 - L_k$, Conj.
ν is a variable of any type A; Φ_1, …, Φ_k are formulas.				

12. Examples of Semantic Derivations

1. Simple Function-Composition (Transitivity)

In set theory, if one has two functions

$F : B\rightarrow C$ i.e., F is a function from B into C
$G : A\rightarrow B$ i.e., G is a function from A into C

one can *compose* F and G into a composite-function, $F\circ G$, from A into C, defined (implicitly) as follows.

$$[F\circ G](x) \;=\; F(G(x))$$

From the viewpoint of Categorial Logic, this is simply an instance of the logical principle of transitivity. In particular, the following re-constructs function-composition using Categorial Logic.

(1)	$\lambda x{:}F(x)$	$\mathcal{B}\rightarrow C$	1	Pr
(2)	$\lambda x{:}G(x)$	$\mathcal{A}\rightarrow\mathcal{B}$	2	Pr
(3)	x	$\mathcal{A}$	3	As
(4)	$[\lambda x{:}G(x)]\langle x\rangle$	$\mathcal{B}$	23	2,3,λO
	$G(x)$			λ-conversion
(5)	$[\lambda x{:}F(x)]\langle G(x)\rangle$	C	123	1,4,λO
	$F(G(x))$			λ-conversion
(6)	$\lambda x : F(G(x))$	$\mathcal{A}\rightarrow C$	12	3,5,λI

Notice that the resulting function is exactly what the received set-theoretic definition produces.

2. Schönfinkel's Transform

Moses Schönfinkel (1924) proposed that a two-place function[50]

$$F : (A\times B)\rightarrow C$$

which is a function from the Cartesian-product $\mathcal{A}\times\mathcal{B}$ into C, can be transformed into a "family" of one-place functions,

$$F^* : A\rightarrow(B\rightarrow C)$$

the latter being a function from A into the set $B\rightarrow C$ of functions from B into C. The converse transformation is also admissible.

From the viewpoint of Categorial Logic, these transformations correspond to the following categorial equivalence, which we duly call Schönfinkel's Law.[51]

$$(A\times B)\rightarrow C \;\dashv\vdash\; A\rightarrow(B\rightarrow C)$$

The following derivations reconstruct these two transformations.

---

[50] Note that a two-place function is *ordinarily* regarded as a function that takes an *ordered-pair* as input. We have adopted a weaker, but equally important, notion.

[51] Also, not that, given its historical priority, Heim and Kratzer (1998) propose the term ‘Schönfinkelization’ for this technique, in place of the more frequently used term ‘Currying’ (after Haskell Curry).

(1)	F	$(A\times B)\rightarrow C$	1	Pr
(2)	x	A	2	As
(3)	y	B	3	As
(4)	$x \times y$	$A\times B$	23	2,3,×I
(5)	F$(x \times y)$	C	123	1,4,λO
(6)	λy F$(x \times y)$	$B\rightarrow C$	12	3,5,λI
(7)	$\lambda x\, \lambda y$ F$(x \times y)$	$A\rightarrow(B\rightarrow C)$	1	2,6,λI

Note that if α and β are type-theoretic items, then the product α×β is just their **un**-ordered pair.

(1)	F	$\mathcal{A}\rightarrow(\mathcal{B}\rightarrow C)$	1	Pr
(2)	$x\times y$	$\mathcal{A}\times\mathcal{B}$	2	As
(3)	x	$\mathcal{A}$	3	As (2a)
(4)	y	$\mathcal{B}$	4	As (2b)
(5)	F(x)	$\mathcal{B}\rightarrow C$	13	1,3,λO
(6)	[F(x)](y)	C	134	4,5,λO
(7)	[F(x)](y)	C	12	2,2-6,×O
(8)	$\lambda(x\times y)$ [F(x)](y)	$(\mathcal{A}\times\mathcal{B})\rightarrow C$	12	2,7,λI

Note that the resulting item $\lambda(x\times y)\ \phi(x)(y)$ involves an *expanded* lambda-abstract; in this particular case, it is a function that takes an un-ordered pair $x\times y$ as input. See Chapter 4 for a detailed account of *expanded* lambda-abstraction.

3. Montague's Transform

Richard Montague (1993) proposed that we treat *both* proper-names and quantifier-phrases as second-order predicates of the following type.

$$(\mathsf{D}\rightarrow\mathsf{S})\rightarrow\mathsf{S}$$

This involves a novel approach to QPs, but also a novel approach to proper-names, according to which a name such as 'Kay' does not denote an individual (entity), but rather a set of properties of entities, in this case the set of all properties instantiated by the entity Kay.

From the viewpoint of Categorial Logic, this maneuver transforms an item of type D (entity) into an item of type (D→S)→S, in accordance with the following logical principle, which we call Montague's Law.

$$A \vdash (A\rightarrow B)\rightarrow B$$

The following derivation shows how Categorial Logic reconstructs Montague's transform.

(1)	K	D	1	Pr
(2)	P	D→S	2	As
(3)	PK	S	12	1,2,λO
(4)	λP:PK	(D→S)→S	1	2,3,λI

Notice that this transforms the individual K into the function λP:P(K), which takes a predicate P and yields the result of applying P to K.

Finally, notice that the converse argument is not valid; a quantifier-phrase does not in general give rise to an entity.

4. Generalized-Conjunction [Function Promotion]

Partee and Rooth (1993) proposed that an operator of type

$$[\mathsf{S}\times\mathsf{S}]\rightarrow\mathsf{S}$$

can be transformed into an operator of type,

$$[(A\rightarrow\mathsf{S}) \times (A\rightarrow\mathsf{S})] \rightarrow (A\rightarrow\mathsf{S})$$

where A is any type. Furthermore, by recursion, one can promote the latter to a function of type

$$[(B \rightarrow (A \rightarrow \mathsf{S})) \times (B \rightarrow (A \rightarrow \mathsf{S}))] \rightarrow (B \rightarrow (A \rightarrow \mathsf{S}))$$

where B is any type, and so forth.

This procedure corresponds to what mathematicians call the formation of *function-spaces*; for example, if one can add numbers, then one can, by extension, "add" number-valued functions in accordance with the following (implicit) definition.

$$(\mathrm{F}+\mathrm{G})(x) =_{df} \mathrm{F}(x) + \mathrm{G}(x)$$

The following are the corresponding explicit definitions using lambda-abstraction.

$$\lambda x{:}\mathrm{F}(x) + \lambda x{:}\mathrm{G}(x) =_{df} \lambda x\{\mathrm{F}(x) + \mathrm{G}(x)\}$$
$$\mathrm{F} + \mathrm{G} =_{df} \lambda x\{\mathrm{F}(x) + \mathrm{G}(x)\}$$

For us, function-promotion is just another valid inference in Categorial Logic. For example, the following derivation shows how predicate-conjunction follows from sentence-conjunction as a matter of Categorial Logic. In other words, the following is a Categorial Logic (meta) theorem.

$$\lambda Y \lambda X\{X \,\&\, Y\} \vdash \lambda P \lambda Q \lambda x\{Px \,\&\, Qx\}$$

For the sake of simplifying what rules we need, we render both conjunction-operators in their Schönfinkel forms $[\mathsf{S} \rightarrow (\mathsf{S} \rightarrow \mathsf{S})$ rather than $(\mathsf{S} \times \mathsf{S}) \rightarrow \mathsf{S}]$.

(1)	$\lambda Y \lambda X\{X \,\&\, Y\}$	$\mathsf{S} \rightarrow (\mathsf{S} \rightarrow \mathsf{S})$	1	Pr
(2)	$P \quad [= \lambda x Px]$	$\mathsf{D} \rightarrow \mathsf{S}$	2	As
(3)	$Q \quad [= \lambda x Qx]$	$\mathsf{D} \rightarrow \mathsf{S}$	3	As
(4)	$\lambda x\{Px \times Qx\}$	$\mathsf{D} \rightarrow (\mathsf{S} \times \mathsf{S})$	23	2, 3, C-M
(5)	x	D	4	As
(6)	$Px \times Qx$	$\mathsf{S} \times \mathsf{S}$	234	4, 5, λO
(7)	Px	S	5	As
(8)	Qx	S	6	As
(9)	$\lambda X\{X \,\&\, Qx\}$	$\mathsf{S} \rightarrow \mathsf{S}$	16	1, 8, λO
(10)	$Px \,\&\, Qx$	S	156	7, 9, λO
(11)	$Px \,\&\, Qx$	S	1234	6, 7-10, $\times$O
(12)	$\lambda x\{Px \,\&\, Qx\}$	$\mathsf{D} \rightarrow \mathsf{S}$	123	5, 11, λI
(13)	$\lambda Q \lambda x\{Px \,\&\, Qx\}$	$(\mathsf{D} \rightarrow \mathsf{S}) \rightarrow (\mathsf{D} \rightarrow \mathsf{S})$	12	3, 12, λI
(14)	$\lambda P \lambda Q \lambda x\{Px \,\&\, Qx\}$	$(\mathsf{D} \rightarrow \mathsf{S}) \rightarrow [(\mathsf{D} \rightarrow \mathsf{S}) \rightarrow (\mathsf{D} \rightarrow \mathsf{S})]$	1	2, 13, λI

13. Out of the Frying Pan [52]

Earlier, we exhibited some sentences that challenge Basic Categorial Semantics. We now examine how Expanded Categorial Semantics deals with them.

6. Jay does not respect Kay

<table>
<tr><td>Jay</td><td>does-not</td><td>respect</td><td>Kay</td></tr>
<tr><td rowspan="3">J</td><td rowspan="2">$\lambda P{\sim}P$</td><td colspan="2">$\lambda y\, \lambda x\, \mathbf{R}xy$ | K</td></tr>
<tr><td colspan="2">$\lambda x \mathbf{R}x\mathrm{K}$</td></tr>
<tr><td colspan="3">$\lambda x{\sim}\mathbf{R}x\mathrm{K}$</td></tr>
<tr><td colspan="4">${\sim}\mathbf{R}\mathrm{JK}$</td></tr>
</table>

Note in particular that the problematic (shaded) entry does not result by function-*application*, but rather by function-*composition* (see earlier), being derived as follows.

[52] The titles of this section and the next are in honor of lexicographer [*Oxford English Dictionary*] and philologist J.R.R Tolkien, who uses these titles for adjacent chapters in *The Hobbit*.

(1)	$\lambda x\mathbf{R}x\mathrm{K}$	D→S	1	Pr
(2)	λP∼P	S→S	2	Pr
(3)	x	D_1	3	As
(4)	$\mathbf{R}x\mathrm{K}$	S	13	1,3,λO
(5)	$\sim\mathbf{R}x\mathrm{K}$	S	123	2,4,λO
(6)	$\lambda x\sim\mathbf{R}x\mathrm{K}$	D→S	123	3,5,λI

7. Jay respects every woman

Jay	respects	every	woman
		λP λQ $\forall x\{\mathrm{P}x\rightarrow\mathrm{Q}x\}$	$\lambda x\mathbf{W}x$
	$\lambda y\lambda x\mathbf{R}xy$	λQ $\forall x\{\mathbf{W}x\rightarrow\mathrm{Q}x\}$	
J	$\lambda y\ \forall x\{\mathbf{W}x\rightarrow\mathbf{R}yx\}$		
$\forall x\{\mathbf{W}x\rightarrow\mathbf{R}\mathrm{J}x\}$			

Note in particular that the problematic (shaded) entry does not result by function-application. Nevertheless, it can be systematically constructed from its components using Categorial Logic, as follows.

(1)	λQ $\forall x\{\mathbf{W}x\rightarrow\mathrm{Q}x\}$	(D→S)→S	1	Pr
(2)	$\lambda y\ \lambda x\ \mathbf{R}xy$ $\lambda x\ \lambda y\ \mathbf{R}yx$	D→(D→S)	2	Pr AV
(3)	y	D	3	As
(4)	x	D	4	As
(5)	$\lambda y\ \mathbf{R}yx$	D→S	24	2,4,λO
(6)	$\mathbf{R}yx$	S	234	3,5,λO
(7)	$\lambda x\ \mathbf{R}yx$	D→S	23	4,6,λI
(8)	$\forall x\{\mathbf{W}x\rightarrow\mathbf{R}yx\}$	S	123	1,7,λO
(9)	$\lambda y\ \forall x\{\mathbf{W}x\rightarrow\mathbf{R}yx\}$	D→S	12	3,8,λI

14. Into the Fire [53]

The problem is that we also have the following (simpler!) derivation.

(1)	λQ $\forall x\{\mathbf{W}x\rightarrow\mathrm{Q}x\}$	(D→S)→S	1	Pr
(2)	$\lambda y\ \lambda x\ \mathbf{R}xy$	D→(D→S)	2	Pr
(3)	y	D	3	As
(4)	$\lambda x\ \mathbf{R}xy$	D→S	24	2,4,λO
(5)	$\forall x\{\mathbf{W}x\rightarrow\mathbf{R}xy\}$	S	123	1,7,λO
(6)	$\lambda y\ \forall x\{\mathbf{W}x\rightarrow\mathbf{R}xy\}$	D→S	12	3,8,λI

So the following semantic derivation is equally admissible.

Jay	respects	every	woman
		λP λQ $\forall x\{\mathrm{P}x\rightarrow\mathrm{Q}x\}$	$\lambda x\mathbf{W}x$
	$\lambda y\lambda x\mathbf{R}xy$	λQ $\forall x\{\mathbf{W}x\rightarrow\mathrm{Q}x\}$	
J	$\lambda y\ \forall x\{\mathbf{W}x\rightarrow\mathbf{R}xy\}$		
$\forall x\{\mathbf{W}x\rightarrow\mathbf{R}x\mathrm{J}\}$			
every woman respects Jay			

[53] See note 52.

This illustrates a general problem with expanded categorial semantics. For an even simpler illustration of the problem, we note that the following derivations are equally admissible.

Jay	respects	Kay
	$\lambda y\,\lambda x\,\mathbf{R}xy$	K
J	$\lambda x\,\mathbf{R}x$K	
$\mathbf{R}$JK		
i.e., Jay respects Kay		

Jay	respects	Kay
	$\lambda y\,\lambda x\,\mathbf{R}xy$	K
J	$\lambda x\,\mathbf{R}$Kx	
$\mathbf{R}$KJ		
i.e., Kay respects Jay		

The following are the respective logical derivations.

(1)	$\lambda y\lambda x\mathbf{R}xy$	D→(D→S)	1	Pr
(2)	K	D→(D→S)	2	Pr
(3)	$\lambda x\mathbf{R}x$K	D→S	12	1,2,λO

(1)	$\lambda y\lambda x\mathbf{R}xy$	D→(D→S)	1	Pr
(2)	K	D→(D→S)	2	Pr
(3)	y	D	3	As
(4)	$\lambda x\mathbf{R}xy$	D→S	13	1,3,λO
(5)	$\mathbf{R}$Ky	S	123	3,4,λO
(6)	$\lambda y\mathbf{R}$Ky	D→S	12,	3,5,λI

In other words, according to this account, one reading of 'Jay respects Kay' is that Kay respects Jay!

A similar problem afflicts accusative relative pronouns. Recall that a relative-pronoun – e.g., 'who' – is type-categorized as follows.

type(who) = (D→S)→C

Also recall that, since we currently treat C as a variant of D→S, this in effect renders 'who' as semantically empty. This works perfectly when 'who' serves as the *subject* of the verb 'respects', as in the following analysis.

8. woman who respects Kay

woman	who	respects	Kay
		$\lambda y\lambda x\mathbf{R}xy$	K
	∅	$\lambda x\mathbf{R}x$K	
$\lambda x\mathbf{W}x$	$\lambda x\mathbf{R}x$K		
$\lambda x(\mathbf{W}x$ & $\mathbf{R}x$K)			

But what happens if 'who' serves as the object of the verb, in which case it is often pronounced 'who**m**', as in the following analysis.

9. woman whom Kay respects

woman	whom	Kay	respects
		K	$\lambda y\lambda x\mathbf{R}xy$
	∅	$\lambda x\mathbf{R}x$K	
$\lambda x\mathbf{W}x$	$\lambda x\mathbf{R}x$K		
$\lambda x(\mathbf{W}x$ & $\mathbf{R}x$K)			

The derivation above is in perfect accord with our rules, but it seems to claim that

whom Kay respects
who respects Kay

mean the same thing!

15. What We Need – Case-Marking

By way of avoiding all these pitfalls, in the next two chapters, we propose to enlarge categorial grammar by including case-markers. So, when we combine 'Kay' ['who'] with 'respects', we must indicate whether 'Kay' ['who'] is the subject or the object of the verb, which is exactly what case-marking is designed to do.