Chapter 3
Expanded Categorial Grammar

Chapter 3 Expanded Categorial Grammar ............................................................................................. 1
A. Expanded Categorial Syntax .................................................................................................................. 2
  1. Review ................................................................................................................................... 2
  2. Positive Examples .................................................................................................................. 3
  3. Counter-Examples .................................................................................................................. 3
  4. Expanded Grammatical Composition .................................................................................... 4
  5. What is Categorial Logic? ..................................................................................................... 5
  6. Classical Logic ....................................................................................................................... 5
  7. Intuitionistic Logic ................................................................................................................. 6
  8. No Monotonic Logic Properly Models Grammatical-Composition ...................................... 6
  9. Relevance Logic ..................................................................................................................... 6
  10. Linear Logic Without Identity ............................................................................................... 8
  11. Multi-Linear Logic ................................................................................................................. 8
  12. Summary of Logical Systems Considered ............................................................................. 8
B. Expanded Categorial Semantics ..................................................................................................... 9
  1. Introduction............................................................................................................................ 9
  2. Expanded Loglish – Type-Theory With Multiplication ........................................................ 9
  3. Semantic-Composition ......................................................................................................... 10
  5. Examples of Semantic Derivations ...................................................................................... 10
  6. Out of the Frying Pan .......................................................................................................... 13
  7. Into the Fire ......................................................................................................................... 14
  8. What We Need – Case-Marking .......................................................................................... 15
C. Appendices ................................................................................................................................... 16
  1. Expanded Grammatical Types ............................................................................................. 16
  2. Multi-Linear Logic – Formal Presentation ............................................................................ 16
  3. Derivation System for Lambda-Composition ..................................................................... 21
  4. Lambda-Calculus ................................................................................................................. 24
  5. Further Composition Rules .................................................................................................. 25
A. Expanded Categorial Syntax

1. Review

Let us 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 $\phi$ consists of three inter-locking constituents.

(a) to provide a semantic-value for $\phi$.
(b) to provide a semantic-value for every component-phrase of $\phi$.
(c) to demonstrate how (1) is computed from (2).

(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/algorithmic component of semantics is characterized by a compositional-calculus, which involves various Rules of Composition.

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) S is a type.</td>
<td>sentences</td>
</tr>
<tr>
<td>(2) D is a type.</td>
<td>definite noun phrases</td>
</tr>
<tr>
<td>(3) if $A$ and $B$ are types, then so is $(A \rightarrow B)$.</td>
<td>monadic functors</td>
</tr>
<tr>
<td>(4) nothing else is a type.</td>
<td></td>
</tr>
<tr>
<td>(5) $C \cong D \rightarrow S$ [common-noun phrases]</td>
<td>defined type</td>
</tr>
</tbody>
</table>

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

1. Function-Application

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a phrase $\Lambda$ of type $A \rightarrow B$</td>
<td></td>
</tr>
<tr>
<td>combines with a phrase $\alpha$ of type $A$</td>
<td></td>
</tr>
<tr>
<td>to produce the phrase $<a href="%5Calpha">\Lambda</a>$ of type $B$</td>
<td></td>
</tr>
</tbody>
</table>

2. Conjunction

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a phrase $\lambda \nu \Phi$ of type $C \mid D \rightarrow S$</td>
<td></td>
</tr>
<tr>
<td>combines with a phrase $\lambda \nu \Psi$ of type $C \mid D \rightarrow S$</td>
<td></td>
</tr>
<tr>
<td>to produce the phrase $\lambda \nu (\Phi &amp; \Psi)$ of type $C \mid D \rightarrow S$</td>
<td></td>
</tr>
</tbody>
</table>

In regard to item 2, we employ semantic trees. By a semantic-tree for a phrase $\phi$, we mean a tree consisting of the semantic-values of all the sub-phrases of $\phi$, plus the original morphemes attached to the terminal nodes. Recall that we employ a special tabular system by which semantic trees are presented.
The following example illustrates the two methods.

1. Jay respects Kay's mother

\[
\begin{array}{|c|c|c|}
\hline
\text{Jay} & \text{respects} & \text{Kay's mother} \\
\hline
\lambda y \lambda x Rxy & \lambda x : M(x) & M(k) \\
\hline
J & \lambda x R[x, M(k)] & R[J, M(k)] \\
\hline
\end{array}
\]

2. Positive Examples

The following are successful applications of Basic Categorial Semantics.

2. every woman who is virtuous is happy

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{every} & \text{woman} & \text{who} & \text{is} & \text{virtuous} & \text{is} & \text{happy} \\
\hline
\varnothing & \lambda x Vx & \varnothing & \lambda x Hx \\
\hline
\lambda x Wx & \lambda x Vx \\
\hline
\lambda P \lambda Q \forall x \{Px \rightarrow Qx\} & \lambda x (Wx \land Vx) & \lambda x Wx & \lambda x Vx \\
\hline
\lambda Q \forall x \{Wx \land Vx\} & \lambda x (Wx \land Vx) & \lambda x Hx \\
\hline
\forall x \{Wx \land Vx\} \rightarrow Hx \\
\hline
\end{array}
\]

3. not every virtuous woman respects Kay

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{not} & \text{every} & \text{happy} & \text{woman} & \text{respects} & \text{Kay} \\
\hline
\lambda P \lambda Q \forall x \{Px \rightarrow Qx\} & \lambda x Hx & \lambda x Wx & \lambda y \lambda x Rxy & k \\
\hline
\lambda Q \forall x \{Hx \land Wx\} & \lambda x Hx \land Wx & \lambda y \lambda x Rxy \\
\hline
\lambda P \sim P & \forall x \{Hx \land Wx\} \rightarrow Rxx \\
\hline
\sim \forall x \{Hx \land Wx\} \rightarrow Rxx \\
\hline
\end{array}
\]

3. Counter-Examples

In this context, by a counter-example we mean a string of English words that constitute a well-formed and meaningful phrase, but cannot be properly analyzed by our current account of semantic-composition. The following are just two of many counter-examples.
4. Jay does not respect Kay

\[ \lambda y \lambda x \neg R \, xy \]

Jay does not respect Kay

5. Jay respects every woman

\[ \lambda y \lambda x \neg R \, \neg K \]

Jay respects every woman

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

4. Expanded Grammatical Composition

In order to deal with these counter-examples, we propose to expand the rules of composition. For this purpose, we propose Categorial Logic,\(^1\) which is intended to be a calculus of composition.\(^2\)

We employ the term ‘logic’ because, in characterizing this system, we utilize (re-purpose) logical formalism, and we draw inspiration from existing 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

<table>
<thead>
<tr>
<th>a phrase</th>
<th>( \Lambda )</th>
<th>of type</th>
<th>( A \rightarrow B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>combines with a phrase</td>
<td>( \alpha )</td>
<td>of type</td>
<td>( A )</td>
</tr>
<tr>
<td>to produce the phrase</td>
<td>( \Lambda \langle \alpha \rangle )</td>
<td>of type</td>
<td>( B )</td>
</tr>
</tbody>
</table>

2. Conjunction

<table>
<thead>
<tr>
<th>a phrase</th>
<th>( \lambda v \Phi )</th>
<th>of type</th>
<th>( C \rightarrow D \rightarrow S )</th>
</tr>
</thead>
<tbody>
<tr>
<td>combines with a phrase</td>
<td>( \lambda v \Psi )</td>
<td>of type</td>
<td>( C \rightarrow D \rightarrow S )</td>
</tr>
<tr>
<td>to produce the phrase</td>
<td>( \lambda v (\Phi &amp; \Psi) )</td>
<td>of type</td>
<td>( C \rightarrow D \rightarrow S )</td>
</tr>
</tbody>
</table>

Let us concentrate on the first rule, which is the primary rule. If we think of arrow as 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-principle known as modus ponens.

The idea we propose is that, whereas Basic Categorial-Grammar admits only one mode of
composition,\(^3\) corresponding to *modus ponens*, Expanded Categorial-Grammar admits **infinitely-many** modes of composition, each one corresponding to a valid-inference of categorial logic. This principle is officially presented as follows.

<table>
<thead>
<tr>
<th>Categorial Logic Principle (for composing types)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where (A_0), (\ldots), (A_k) are types.</td>
</tr>
<tr>
<td>(A_1), (\ldots), (A_k) combine to form</td>
</tr>
<tr>
<td>IF (^4) the argument form (A_1), (\ldots), (A_k) / (A_0)</td>
</tr>
<tr>
<td>is valid according to Categorial Logic</td>
</tr>
</tbody>
</table>

5. **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.

6. **Classical Logic**

Since it is the strongest logic of the group,\(^5\) Classical Logic (CL) authorizes the maximum number of compositions, but unfortunately it also authorizes compositions that are grammatically implausible. For example, the following is a principle of Classical Logic.\(^6\)

\[(c1) \quad A \; \& \; B \vdash A \rightarrow B\]

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

the argument \(\{A \; \& \; B\} \rightarrow \) therefore \(A \rightarrow B\)\(^7\) 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.

\[(c2) \quad B \vdash A \rightarrow B\]

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

\(S \vdash S \rightarrow S\)

\(^3\) Plus Conjunction.

\(^4\) The connective is ‘if’, not ‘if and only if’, since we contemplate that other modes of composition may be available. For example, we also admit the Conjunction-Rule.

\(^5\) If a weaker logic validates an argument form \(A\), then a stronger logic also validates \(A\), granting the two logics both pass judgment on \(A\).

\(^6\) Although nothing hinges on this, we use one arrow-symbol ‘\(\rightarrow\)’ for logic, and another arrow-symbol ‘\(\Rightarrow\)’ for category theory.
Hardegree, Compositional Semantics, Chapter 3: Expanded Categorial Grammar

7. Intuitionistic Logic

Intuitionist Logic (IL) is weaker than 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, so we conclude that Intuitionistic Logic also does not properly model grammatical composition.

8. 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. 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.

\[ B \vdash C \Rightarrow A ; B \vdash C \]

In other words, if \( C \) follows from \( B \), then \( C \) also follows from \( A \) and \( B \).

Next, suppose we also grant the following identity principle,

\[ B \vdash B \]

which 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.

\[ A ; B \vdash B \]

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

\[ D ; S \vdash S \]

In the proposed composition, we compose a sentence out of 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 what follows, we examine two prominent non-monotonic logics – Relevance Logic, and Linear Logic.

9. 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

---

7 A transformation corresponds to a single-premise argument.
8 These are prominent examples of valid argument forms whose proof (say, in an intro logic class) cannot be accomplished using only rules pertaining to →.
9 This usage comes from the definition of monotonic (increasing) functions. Specifically, a function \( \phi \) is said to be monotonic precisely if \( \phi(x) \leq \phi(y) \) whenever \( x \leq y \). In the context of logical systems, the relevant order-relation is set-inclusion, and the relevant function is the consequence function \( \Gamma \), defined so that \( \Gamma(\Gamma) = \{ \alpha | \Gamma \vdash \alpha \} \). Then a logical system is monotonic precisely if its associated consequence-function is monotonic.
10 Here, we use the symbol ‘⇒’ as the meta-language’s if-then connective.
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 the premises and the conclusion. Most well-known systems of logic do not satisfy either of these desiderata.

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

1. **Contraction**

   \[ A \to (A \to B) \vdash A \to B \]

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

   \[ D \to (D \to S) \vdash D \to S \]

   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 \( \to \),\(^\text{11}\) 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. **Law of Assertion**

   \[ (A \to A) \to B \vdash B \]

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

   \[ (C \to C) \to (C \to C) \vdash C \to C \]

   From this, we obtain the following composition principle.

   \[ (C \to C) \to (C \to C) ; C \vdash C \]

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

4. **Tautology**

   A tautology is a formula that is logically-true, alternatively a formula that follows from nothing.\(^\text{12}\) 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’.

   \[ \vdash A \to A \]

---

\(^{11}\) We later add \( \times \) as a further type-forming operator; see Section 2.

\(^{12}\) When no formulas are in front of \( \vdash \), it is understood that the premise set is empty.
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”.

10. Linear Logic Without Identity

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. In particular, whereas Relevance Logic requires that every supposition be used at least once, Linear Logic requires every supposition to 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 also ban arguments with no premises. This restriction makes sense from a grammatical point of view, since we do not want phrases entering a grammatical-construction "out of thin air". The resulting system is sometimes called Linear Logic without Identity.

By moving to Linear Logic without Identity, 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.

\[
\begin{align*}
(1) & \quad A \rightarrow B ; A \rightarrow C \vdash A \rightarrow (B \times C) \\
(2) & \quad A \rightarrow (B \rightarrow C) ; A \rightarrow B \vdash A \rightarrow C \\
(3) & \quad A \rightarrow B ; A \rightarrow C ; (B \times C) \rightarrow D \vdash A \rightarrow D
\end{align*}
\]

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

11. Multi-Linear Logic

Based on the considerations of the previous sections, we propose that the logic of categorial-composition is:

Linear Logic without Identity

[Conditional Multiplication]

+ Conditional-Multiplication

for which we propose the name Multi-Linear Logic. It is formally presented in an appendix.

12. Summary of Logical Systems Considered

Note: the following systems form a chain; each system contains the systems below it in the list.

<table>
<thead>
<tr>
<th>KEY</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Classical Logic</td>
</tr>
<tr>
<td>I</td>
<td>Intuitionistic Logic</td>
</tr>
<tr>
<td>R</td>
<td>Relevance Logic</td>
</tr>
<tr>
<td>M</td>
<td>Multi-Linear Logic</td>
</tr>
<tr>
<td>L</td>
<td>Linear Logic (without identity)</td>
</tr>
</tbody>
</table>

13 Linear Logic originates with Jean-Yves Girard (1987).

14 See, for example, Restall (2000).

15 One obtains Relevance Logic from Multi-Linear Logic by adding Identity.
1. Arguments Valid in all Systems

1. Arguments Valid in all Systems

(1) \( A \rightarrow B ; A \vdash B \) [Modus Ponens]
(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) \( (A \rightarrow C) \rightarrow D ; A \rightarrow (B \rightarrow C) \vdash B \rightarrow D \) [Permutfivity]
(8) \( B \rightarrow C ; A \times B \vdash A \times C \) [Addition]
(9) \( (A \times B) \rightarrow C \vdash A \rightarrow (B \rightarrow C) \) [Schönfinkel's Law]

2. C-Valid Arguments Rejected by I

(1) \( (A \rightarrow B) \rightarrow B \vdash (B \rightarrow A) \rightarrow A \) [Łukasiewicz's Law\(^{16}\)]
(2) \( (A \rightarrow B) \rightarrow A \vdash A \) [Peirce's Law\(^{17}\)]

3. I-Valid Arguments Rejected by R

(1) \( A \vdash B \rightarrow A \) [Positive Paradox]
(2) \( A \rightarrow B \vdash A \rightarrow (A \rightarrow B) \) [Expansion]
(3) \( A \vdash B \rightarrow B \) [Irrelevance]
(4) \( A \times B \vdash A \) [Simplification]
(5) \( A \times B \vdash B \) [Simplification]

4. R-Valid Arguments Rejected by M

(1) \( A \rightarrow (A \rightarrow B) \vdash A \rightarrow B \) [Contraction]
(2) \( A \vdash A \times A \) [Duplication]
(3) \( \vdash A \rightarrow A \) [Tautology]
(4) \( (A \rightarrow A) \rightarrow B \vdash B \) [Assertion]

5. M-Valid Arguments Rejected by L

(1) \( A \rightarrow B ; A \rightarrow C \vdash A \rightarrow (B \times C) \) [Conditional-Multiplication]

B. Expanded Categorial Semantics

1. Introduction

So far, we have a method by which to compose types, but we do not have a method by which
to compose semantic-values.\(^{18}\) In what follows, we develop such a method.

2. Expanded Loglish – Type-Theory With Multiplication

Sub-structural Logic includes a multiplication connective, \( \times \), which corresponds to logical-
conjunction for Intuitionistic Logic and Classical Logic, and logical-fusion for Relevance Logic. This logical-connective gives rise to a corresponding type-operator in Type Theory, characterized as follows.

\[
\begin{align*}
(1) \text{if} \quad A & \quad \text{is a type} \\
\text{and} \quad B & \quad \text{is a type} \\
\text{then} \quad [A \times B] & \quad \text{is a type}
\end{align*}
\]

\[
\begin{align*}
(2) \text{if} \quad \alpha & \quad \text{is an expression of type} \quad A \\
\text{and} \quad \beta & \quad \text{is an expression of type} \quad B \\
\text{then} \quad [\alpha \times \beta] & \quad \text{is an expression of type} \quad [A \times B]
\end{align*}
\]

\(^{16}\) Named after Jan Łukasiewicz (1878-1956), who invented multi-valued logic. In his system, one can define
disjunction in terms of conditional via: \( A \lor B \equiv (A \rightarrow B) \rightarrow B \). See Łukasiewicz (1920).

\(^{17}\) Named after Charles Sanders Peirce (1839-1914); first presented in (1985).

\(^{18}\) 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.
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.

<table>
<thead>
<tr>
<th>Categorial Logic Principle (for compositing phrases)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where ( \Phi_0, \ldots, \Phi_k ) are phrases of Loglish.</td>
</tr>
<tr>
<td>( \Phi_1, \ldots, \Phi_k ) combine to form ( \Phi_0 )</td>
</tr>
<tr>
<td>IF</td>
</tr>
<tr>
<td>the argument form ( \Phi_1, \ldots, \Phi_k / \Phi )</td>
</tr>
<tr>
<td>is valid according to Categorial Logic</td>
</tr>
<tr>
<td>(expanded to include phrases of Loglish)</td>
</tr>
</tbody>
</table>


Just as we characterize validity in Multi-Linear Logic using a derivation system, presented in the appendices, we characterize validity in Categorial Logic using a derivation system, also presented in the appendices. The major difference is that, whereas the type-derivation system posits rules pertaining to \( \rightarrow \), the semantic-derivation system posits rules pertaining to \( \lambda \). The correspondence is straightforward.

<table>
<thead>
<tr>
<th>( \rightarrow ) Rules</th>
<th>( \lambda ) Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rightarrow ) Elimination</td>
<td>modus ponens</td>
</tr>
<tr>
<td>( \rightarrow ) Introduction</td>
<td>conditional-generation</td>
</tr>
<tr>
<td>( \lambda ) Elimination</td>
<td>Function-Application</td>
</tr>
<tr>
<td>( \lambda ) Introduction</td>
<td>Function-Generation</td>
</tr>
</tbody>
</table>

5. Examples of Semantic Derivations

1. Simple Function-Composition (Transitivity)

In set theory, if one has two functions

\[ F : B \rightarrow C \quad \text{i.e., } F \text{ is a function from } B \text{ into } C \]

\[ G : A \rightarrow B \quad \text{i.e., } G \text{ is a function from } A \text{ into } B \]

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.

\[ \begin{array}{l|l|l|l|l}
1 & \lambda x : F(x) & B & 1 & \text{Pr} \\
2 & \lambda x : G(x) & A & 2 & \text{Pr} \\
3 & \lambda x & A & 3 & \text{As} \\
4 & G(x) & \lambda x : F(x)(x) & B & 23 & 2,3,\lambda O \\
5 & F(G(x)) & \lambda x : F(x)(F(x)) & C & 123 & 1,4,\lambda O \\
6 & \lambda x & F(G(x)) & A & 12 & 3,5,\lambda I \\
\end{array} \]

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

2. Schönfinkel's Transform

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

\[ F : (A \times B) \rightarrow C \]

which is a function from the Cartesian-product \( A \times B \) 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.\(^{19}\)

\[ (A \times B) \rightarrow C \iff A \rightarrow (B \rightarrow C) \]

The following derivations reconstruct these two transformations.

<table>
<thead>
<tr>
<th>(1)</th>
<th>( F )</th>
<th>((A \times B) \rightarrow C)</th>
<th>1</th>
<th>\Pr</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2)</td>
<td>( x )</td>
<td>( A )</td>
<td>2</td>
<td>As</td>
</tr>
<tr>
<td>(3)</td>
<td>( y )</td>
<td>( B )</td>
<td>3</td>
<td>As</td>
</tr>
<tr>
<td>(4)</td>
<td>( x \times y )</td>
<td>( A \times B )</td>
<td>23</td>
<td>2,3,×I</td>
</tr>
<tr>
<td>(5)</td>
<td>( f(x \times y) )</td>
<td>( C )</td>
<td>123</td>
<td>1,4,λO</td>
</tr>
<tr>
<td>(6)</td>
<td>( λ.y \ f(x \times y) )</td>
<td>( B \rightarrow C )</td>
<td>12</td>
<td>3,5,λI</td>
</tr>
<tr>
<td>(7)</td>
<td>( λ.x \ λ.y \ f(x \times y) )</td>
<td>( A \rightarrow (B \rightarrow C) )</td>
<td>12</td>
<td>2,6,λI</td>
</tr>
</tbody>
</table>

Note that if \( α \) and \( β \) are type-theoretic items, then the product \( α \times β \) is just their un-ordered pair

<table>
<thead>
<tr>
<th>(1)</th>
<th>( F )</th>
<th>( A \rightarrow (B \rightarrow C) )</th>
<th>1</th>
<th>\Pr</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2)</td>
<td>( x \times y )</td>
<td>( A \times B )</td>
<td>2</td>
<td>As</td>
</tr>
<tr>
<td>(3)</td>
<td>( x )</td>
<td>( A )</td>
<td>3</td>
<td>As (2a)</td>
</tr>
<tr>
<td>(4)</td>
<td>( y )</td>
<td>( B )</td>
<td>4</td>
<td>As (2b)</td>
</tr>
<tr>
<td>(5)</td>
<td>( f(x) )</td>
<td>( B \rightarrow C )</td>
<td>13</td>
<td>1,3,λO</td>
</tr>
<tr>
<td>(6)</td>
<td>( <a href="y">f(x)</a> )</td>
<td>( C )</td>
<td>134</td>
<td>4,5,λO</td>
</tr>
<tr>
<td>(7)</td>
<td>( <a href="y">f(x)</a> )</td>
<td>( C )</td>
<td>12</td>
<td>2,2-6,×O</td>
</tr>
<tr>
<td>(8)</td>
<td>( λ(x \times y) \ <a href="y">f(x)</a> )</td>
<td>( (A \times B) \rightarrow C )</td>
<td>12</td>
<td>2,7,λI</td>
</tr>
</tbody>
</table>

Note that the resulting item \( λ(x \times y) \ φ(x)(y) \) 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.

\[ (D \rightarrow S) \rightarrow S \]

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 \( D \rightarrow S \) into an item of type \( (D \rightarrow S) \rightarrow S \), 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.

<table>
<thead>
<tr>
<th>(1)</th>
<th>( \mathbb{K} )</th>
<th>( D )</th>
<th>1</th>
<th>\Pr</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2)</td>
<td>( P )</td>
<td>( D \rightarrow S )</td>
<td>2</td>
<td>As</td>
</tr>
<tr>
<td>(3)</td>
<td>( P \mathbb{K} )</td>
<td>( S )</td>
<td>12</td>
<td>1,2,λO</td>
</tr>
</tbody>
</table>

\(^{19}\) 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).
Notice that this transforms the individual \( K \) into the function \( \lambda P : P(K) \), 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

\[[S \times S] \rightarrow S\]

can be transformed into an operator of type,

\[[(A \rightarrow S) \times (A \rightarrow S)] \rightarrow (A \rightarrow S)\]

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

\[[(B \rightarrow (A \rightarrow S)) \times (B \rightarrow (A \rightarrow S))] \rightarrow (B \rightarrow (A \rightarrow S))\]

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.

\[(F + G)(x) =_a F(x) + G(x)\]

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

\[\lambda x : F(x) + \lambda x : G(x) =_a \lambda x \{F(x) + G(x)\}\]

\[F + G =_a \lambda x \{F(x) + G(x)\}\]

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 \land Y\} \vdash \lambda P : \lambda Q : \lambda x \{P \land Q\} \}

For the sake of simplifying what rules we need, we render both conjunction-operators in their Schönfinkel forms.
6. Out of the Frying Pan

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

6. Jay does not respect Kay

\[
\begin{align*}
\lambda x \sim R x \kappa \\
\lambda y \lambda x \sim R x y \kappa \\
\lambda P \sim P < 1, \Pr \>
\end{align*}
\]

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.

(1) \( \lambda x \sim R x \kappa \)  \( D \rightarrow S \)  1  \( \Pr \)
(2) \( \lambda P \sim P \)  \( S \rightarrow S \)  2  \( \Pr \)
(3) \( x \)  \( D \)  3  \( \As \)
(4) \( R x x \)  \( S \)  13  \( 1,3,\lambda O \)
(5) \( \sim R x x \)  \( S \)  123  \( 2,4,\lambda O \)
(6) \( \lambda x \sim R x x \)  \( D \rightarrow S \)  123  \( 3,5,\lambda I \)

7. Jay respects every woman

\[
\begin{align*}
\lambda y \lambda x \sim R x y \\
\lambda P \lambda Q \forall x \{P x \rightarrow Q x\} \\
\lambda y \forall x \{W x \rightarrow Q x\} \\
\forall x \{W x \rightarrow R y x\}
\end{align*}
\]

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) \( \lambda Q \forall x \{W x \rightarrow Q x\\} \)  \( D \rightarrow S \rightarrow S \)  1  \( \Pr \)
(2) \( \lambda y \lambda x \sim R x y \lambda x \sim R x y \)  \( D \rightarrow (D \rightarrow S) \)  2  \( \Pr \)
(3) \( y \)  \( D \)  3  \( \As \)
(4) \( x \)  \( D \)  4  \( \As \)
(5) \( \lambda y R y x \)  \( D \rightarrow S \)  24  \( 2,4,\lambda O \)
(6) \( R y x \)  \( S \)  234  \( 3,5,\lambda O \)
(7) \( \lambda x R y x \)  \( D \rightarrow S \)  23  \( 4,6,\lambda I \)
(8) \( \forall x \{W x \rightarrow R y x\\} \)  \( S \)  123  \( 1,7,\lambda O \)
(9) \( \lambda y \forall x \{W x \rightarrow R y x\\} \)  \( D \rightarrow S \)  12  \( 3,8,\lambda I \)

---

20 In honor of one-time lexicographer J.R.R Tolkien, who uses these titles for adjacent chapters of *The Hobbit.*
7. Into the Fire

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

\[
\begin{align*}
(1) & \quad \lambda Q \forall x \{ W x \rightarrow Q x \} & | & (D \rightarrow S) \rightarrow S & | & 1 & & \text{Pr} \\
(2) & \quad \lambda y \lambda x \, R x y & & D \rightarrow (D \rightarrow S) & & 2 & & \text{Pr} \\
(3) & \quad y & & D & & 3 & & \text{As} \\
(4) & \quad \lambda x \, R x y & & D \rightarrow S & & 24 & & 2, 4, \lambda O \\
(5) & \quad \forall x \{ W x \rightarrow R x y \} & & S & & 123 & & 1, 7, \lambda O \\
(6) & \quad \lambda y \forall x \{ W x \rightarrow R x y \} & & D \rightarrow S & & 12 & & 3, 8, \lambda I
\end{align*}
\]

So the following semantic derivation is equally admissible.

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.

The following are the respective logical derivations.

\[
\begin{align*}
(1) & \quad \lambda y \lambda x \, R x y & & D \rightarrow (D \rightarrow S) & & 1 & & \text{Pr} \\
(2) & \quad \lambda y \lambda x \, R x y & & D \rightarrow (D \rightarrow S) & & 2 & & \text{Pr} \\
(3) & \quad y & & D & & 3 & & \text{As} \\
(4) & \quad \lambda x \, R x y & & D \rightarrow S & & 13 & & 1, 3, \lambda O \\
(5) & \quad R k y & & S & & 123 & & 3, 4, \lambda O \\
(6) & \quad \lambda y k y & & D \rightarrow S & & 12, 3, 5, \lambda I
\end{align*}
\]

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

---

\(21\) See note 20.
A similar problem afflicts accusative relative pronouns. Recall that a relative-pronoun – e.g., ‘who’ – is type-categorized as follows.

\[
\text{type(who)} = (D\to S)\to C
\]

Also recall that, since we currently treat \(C\) as a variant of \(D\to S\), 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

\[
\begin{array}{cccc}
\text{woman} & \text{who} & \text{respects} & \text{Kay} \\
\emptyset & \lambda y\lambda x Rx y & K & \\
\lambda x W x & \lambda x R x k \\
\lambda x (W x \& R x k)
\end{array}
\]

But what happens if ‘who’ serves as the object of the verb, in which case it is often pronounced ‘whom’, as in the following analysis.

9. woman whom Kay respects

\[
\begin{array}{cccc}
\text{woman} & \text{whom} & \text{Kay} & \text{respects} \\
\emptyset & K & \lambda y\lambda x Rx y & \\
\lambda x W x & \lambda x R x k \\
\lambda x (W x \& R x k)
\end{array}
\]

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!

8. 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.
C. Appendices

1. Expanded Grammatical Types

Although we don't make extensive use of product-types until Chapter 7 [Pronouns], we go ahead and introduce them, since they figure in our discussion of Categorial Logic. The expanded account of grammatical-types goes as follows.

| (1) S is a type. | sentences |
| (2) D is a type. | definite noun phrases |
| (3) C is a type. | common noun phrases |
| (4) if A and B are types, then so is (A→B). | monadic functors |
| (5) if A and B are types, then so is (A×B). | products |
| (6) nothing else is a type. |

The use of product-types is quite limited at first. One use is for type-rendering polyadic functors in the style of logicians, as illustrated in the following.\(^\text{22}\)

is [\(\Pi\)]

\[ (D×D)\rightarrow S \]

\[ (S×S)\rightarrow S \]

2. Multi-Linear Logic – Formal Presentation

1. Arrow-Fragment

We begin with the arrow-fragment of System \(M\), which is to say the fragment of \(M\) that pertains exclusively to the conditional operator '\(→\)', postponing the official introduction of '\(×\)'.

1. Syntax

Since it involves only one connective, the syntax is very easy to describe.

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

2. Derivation System

By way of systematically accounting for all the valid argument forms of System \(M\), we propose a natural-deduction system, formally presented as follows.

1. Preliminaries

For us, a derivation is a structure of the following form.

<table>
<thead>
<tr>
<th>line-number</th>
<th>formula</th>
<th>index</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(L_1)</td>
<td>(Φ_1)</td>
<td>(I_1)</td>
<td>(A_1)</td>
</tr>
<tr>
<td>(L_2)</td>
<td>(Φ_2)</td>
<td>(I_2)</td>
<td>(A_2)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(L_k)</td>
<td>(Φ_k)</td>
<td>(I_k)</td>
<td>(A_k)</td>
</tr>
</tbody>
</table>

A line's number indicates its location in the derivation, which is used to reference the formula in the annotation-column. A line's annotation indicates how that line is justified according to the rules of

\(^{22}\) The notation is reminiscent of set-theory notation, according to which \(A×B\) is the Cartesian product of set \(A\) and set \(B\), which consists of ordered-pairs \((α,β)\) such that \(α∈A\) and \(β∈B\). Our use of \(×\) is more abstract (purely algebraic); in particular, our \(×\) is both associative and commutative. This means that composition involving \(×\) does not depend upon word-order; this presents a problem, as we later see.
inference. A line's index basically lists the suppositions (premises, assumptions) on which the line depends. So, for example, the following derivation line,

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>1,2</th>
<th>3,5, →O</th>
</tr>
</thead>
</table>

indicates that formula ‘P’, which is line 7, follows from lines 3 and 5 by the inference-rule →O (arrow-out; modus ponens), and depends upon suppositions 1 and 2.

2. Indices

Indices, which keep track of relevance, are officially defined as follows.

1. every non-empty sequence of numerals is an index;
2. nothing else is an index.

Indices form an algebra under the operation of sequence-addition, +, which satisfies various algebraic principles depending upon the specific logical system. In System M, the principles are as follows, where i, j, k are indices.

1. \( i + (j + k) = (i + j) + k \) + is associative
2. \( i + j = j + i \) + is commutative

3. Definition of Derivation

Where \( \Phi_0, \ldots, \Phi_k \) are formulas, a derivation of \( \Phi_0 \) from \{ \( \Phi_1, \ldots, \Phi_k \) \} is a sequence of lines as follows.

<table>
<thead>
<tr>
<th>line number</th>
<th>formula</th>
<th>index</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L_1 )</td>
<td>( \Phi_1 )</td>
<td>1</td>
<td>Pr</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>( L_k )</td>
<td>( \Phi_k )</td>
<td>k</td>
<td>Pr</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>( L_0 )</td>
<td>( \Phi_0 )</td>
<td>( 1 + \ldots + k )</td>
<td></td>
</tr>
</tbody>
</table>

In particular:

1. the first \( k \)-many lines are the premises – \( \Phi_1, \ldots, \Phi_k \), each one indexed by the line-number \( k \).
2. every remaining line is either
   1. an assumption, or
   2. follows from previous lines by an inference-rule.
3. the last line is \( \Phi_0 \), which depends upon all and only the premises.

4. Assumptions

There are two kinds of suppositions allowed in derivations – premises, and assumptions. Whereas premises correspond to the input expressions of a grammatical-composition, assumptions arise in sub-derivations in connection with various inference-rules (see below). The following schematically exhibits the assumption-insertion rule.

<table>
<thead>
<tr>
<th>line-number</th>
<th>formula</th>
<th>index</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L )</td>
<td>( \Phi )</td>
<td>( m ) (new)</td>
<td>As</td>
</tr>
</tbody>
</table>

23 There are two reasonable ways of indexing suppositions; (1) one can simply use the supposition's line-number as its index. (2) one can number suppositions independently of formulas. We employ the latter approach.
24 Alternatively expressed, indices encode "sub-structural" information. See Greg Restall (2000) for a thorough presentation of sub-structural logic.
25 We obtain Linear Logic with Identity by adding the empty-index \( \emptyset \). We obtain Relevance Logic by adding contraction \([i+i = i]\). We obtain Intuitionistic Logic by getting rid of the relevance restriction – that the conclusion depends upon the premises.
Here, \( \Phi \) is any formula, and \( m \) is any numeral that is new, which is to say it does not occur earlier in the derivation.

5. Inference-Rules

We follow a derivation scheme according to which every logical-operator is characterized by two rules – a construction-rule and a deconstruction-rule. In the literature, these are generally known as introduction-rules and an elimination-rules; we prefer the more succinct terms ‘in’ and ‘out’.

1. Arrow-Out (\( \rightarrow O \))

<table>
<thead>
<tr>
<th>( L_1 )</th>
<th>( \mathcal{A} \rightarrow B )</th>
<th>( i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L_2 )</td>
<td>( \mathcal{A} )</td>
<td>( j )</td>
</tr>
<tr>
<td>( L_3 )</td>
<td>( B )</td>
<td>( i+j )</td>
</tr>
</tbody>
</table>

2. Arrow-In (\( \rightarrow I \))

<table>
<thead>
<tr>
<th>( L_1 )</th>
<th>( \mathcal{A} )</th>
<th>( j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L_2 )</td>
<td>( B )</td>
<td>( i+j )</td>
</tr>
<tr>
<td>( L_3 )</td>
<td>( \mathcal{A} \rightarrow B )</td>
<td>( i )</td>
</tr>
</tbody>
</table>

Here, \( i \) and \( j \) are indices (sequences of numerals), and \( i+j \) is the sequential-sum of \( i \) and \( j \). Whereas arrow-out is just modus ponens, arrow-in corresponds to the following key principle about conditionals.

\[ \mathcal{A}_1; \ldots; \mathcal{A}_k; \mathcal{A} \vdash B \Rightarrow \mathcal{A}_1; \ldots; \mathcal{A}_k \vdash \mathcal{A} \rightarrow B \]

Note that, in arrow-in, most derivations introduce \( \mathcal{A} \) by way of the assumption-rule.

6. Cross-Arrow-Fragment of System M

A conditional connective \( \rightarrow \) is said to be residuated\(^{26} \) precisely if there is an associated connective \( \times \) satisfying the following residual law.\(^{27} \)

\[ (A \times B) \rightarrow C \dashv \vdash A \rightarrow (B \rightarrow C) \]

For example, in Classical and Intuitionistic Logic, \( \times \) is simply logical-and.\(^{26} \)

\[ A \times B = A \& B \]

On the other hand, in Relevance Logic\(^{28} \) and Quantum Logic\(^{29} \), multiplication corresponds to "compossibility", which is defined as follows.\(^{30} \)

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

System M has a cross-operator with respect to which its arrow-operator is residuated.

7. Syntax

(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.

---

\(^{26}\) See T.S. Blyth and M.F. Janowitz (1972).

\(^{27}\) One can also call this a division principle; to see why, rewrite \( A \rightarrow B \) as \( B/A \). Then the residual law amounts to the following.

\[ A/(B \times C) = (A/B)/C \]

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

However, the word ‘residual’ pertains to subtraction, in which case the residual law is written thus.

\[ A-(B+C) = (A-B)-C \]

\(^{28}\) Dunn (1966).

\(^{29}\) Hardegree (1981).

\(^{30}\) The order doesn’t matter for relevance logic, in which \( \times \) is commutative, but it does matter for quantum logic, in which \( \times \) is not commutative. Also note that, in relevance logic, \( \times \) is referred to as "fusion", and is usually written using ‘\( \circ \)’.
8. **Derivation System** [Added Rules]

1. **Cross-In (×I)**

<table>
<thead>
<tr>
<th>L₁</th>
<th>A</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>L₂</td>
<td>B</td>
<td>j</td>
</tr>
<tr>
<td>L₃</td>
<td>A × B</td>
<td>i+j</td>
</tr>
</tbody>
</table>

L₁, L₂, ×I

2. **Cross-Out (×O)**

<table>
<thead>
<tr>
<th>L₀</th>
<th>A × B</th>
<th>i</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>L₁-L₂</td>
<td>A, B ↪ C</td>
<td>j</td>
<td>...</td>
</tr>
</tbody>
</table>

| L₃  | C     | i+j | L₀, L₁-L₂, ×O |

Here, A, B ↪ C is a sub-derivation of C from A, B, dependent on j, which is a sequence schematically presented as follows.

<table>
<thead>
<tr>
<th>L₁</th>
<th>A</th>
<th>m (new)</th>
<th>As</th>
</tr>
</thead>
<tbody>
<tr>
<td>L₂</td>
<td>B</td>
<td>n (new)</td>
<td>As</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>L₃</td>
<td>C</td>
<td>m+n+j</td>
<td>...</td>
</tr>
</tbody>
</table>

9. **Examples of Derivations**

In what follows, we write indices without commas. We also take associativity and commutativity for granted, which allows us to write all indices in simple numerical order.

1. **Transitivity**

B → C ; A → B ⊢ A → C

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Pr</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>B → C</td>
<td>1</td>
</tr>
<tr>
<td>(2)</td>
<td>A → B</td>
<td>2</td>
</tr>
<tr>
<td>(3)</td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>(4)</td>
<td>B</td>
<td>23</td>
</tr>
<tr>
<td>(5)</td>
<td>C</td>
<td>123</td>
</tr>
<tr>
<td>(6)</td>
<td>A → C</td>
<td>12</td>
</tr>
</tbody>
</table>

2. **Permutation**

A → (B → C) ⊢ B → (A → C)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>A → (B → C)</td>
<td>1</td>
</tr>
<tr>
<td>(2)</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>(3)</td>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>(4)</td>
<td>B → C</td>
<td>13</td>
</tr>
<tr>
<td>(5)</td>
<td>C</td>
<td>123</td>
</tr>
<tr>
<td>(6)</td>
<td>A → C</td>
<td>12</td>
</tr>
<tr>
<td>(7)</td>
<td>B → (A → C)</td>
<td>1</td>
</tr>
</tbody>
</table>

---

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

32 Except when this practice conflicts with the usual numeral morphology; for example, we may need to distinguish the numeral ‘23’ (twenty-three) from the sequence ‘2,3’ (two, three).
3. **Secondary Modus Ponens**

\[ A \rightarrow (B \rightarrow C) ; B \vdash A \rightarrow C \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>( A \rightarrow (B \rightarrow C) )</td>
<td>1</td>
</tr>
<tr>
<td>(2)</td>
<td>( B )</td>
<td>2</td>
</tr>
<tr>
<td>(3)</td>
<td>( A )</td>
<td>3</td>
</tr>
<tr>
<td>(4)</td>
<td>( B \rightarrow C )</td>
<td>13</td>
</tr>
<tr>
<td>(5)</td>
<td>( C )</td>
<td>123</td>
</tr>
<tr>
<td>(6)</td>
<td>( A \rightarrow C )</td>
<td>12</td>
</tr>
</tbody>
</table>

4. **Lifting [Montague's Law]^{33}**

\[ A \vdash (A \rightarrow B) \rightarrow B \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>( A )</td>
<td>1</td>
</tr>
<tr>
<td>(2)</td>
<td>( A \rightarrow B )</td>
<td>2</td>
</tr>
<tr>
<td>(3)</td>
<td>( B )</td>
<td>12</td>
</tr>
<tr>
<td>(4)</td>
<td>( (A \rightarrow B) \rightarrow B )</td>
<td>1</td>
</tr>
</tbody>
</table>

5. **Inflection^{34}**

\[ (A \rightarrow C) \rightarrow D ; A \rightarrow B \vdash (B \rightarrow C) \rightarrow D \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>( (A \rightarrow C) \rightarrow D )</td>
<td>1</td>
</tr>
<tr>
<td>(2)</td>
<td>( A \rightarrow B )</td>
<td>2</td>
</tr>
<tr>
<td>(3)</td>
<td>( B \rightarrow C )</td>
<td>3</td>
</tr>
<tr>
<td>(4)</td>
<td>( A )</td>
<td>4</td>
</tr>
<tr>
<td>(5)</td>
<td>( B )</td>
<td>24</td>
</tr>
<tr>
<td>(6)</td>
<td>( C )</td>
<td>234</td>
</tr>
<tr>
<td>(7)</td>
<td>( A \rightarrow C )</td>
<td>23</td>
</tr>
<tr>
<td>(8)</td>
<td>( D )</td>
<td>123</td>
</tr>
<tr>
<td>(9)</td>
<td>( (B \rightarrow C) \rightarrow D )</td>
<td>12</td>
</tr>
</tbody>
</table>

6. **Permutivity**

\[ (A \rightarrow C) \rightarrow D ; A \rightarrow (B \rightarrow C) \vdash B \rightarrow D \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>( (A \rightarrow C) \rightarrow D )</td>
<td>1</td>
</tr>
<tr>
<td>(2)</td>
<td>( A \rightarrow (B \rightarrow C) )</td>
<td>2</td>
</tr>
<tr>
<td>(3)</td>
<td>( B )</td>
<td>3</td>
</tr>
<tr>
<td>(4)</td>
<td>( A )</td>
<td>4</td>
</tr>
<tr>
<td>(5)</td>
<td>( B \rightarrow C )</td>
<td>24</td>
</tr>
<tr>
<td>(6)</td>
<td>( C )</td>
<td>234</td>
</tr>
<tr>
<td>(7)</td>
<td>( A \rightarrow C )</td>
<td>23</td>
</tr>
<tr>
<td>(8)</td>
<td>( D )</td>
<td>123</td>
</tr>
<tr>
<td>(9)</td>
<td>( B \rightarrow D )</td>
<td>12</td>
</tr>
</tbody>
</table>

---

^{33} 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 DNs, including proper-nouns, as second-order predicates.

^{34} We call this principle ‘inflection’ because it authorizes numerous compositions involving case-inflection, which we introduce in a later chapter.
7. Addition

\[ B \rightarrow C ; A \times B \vdash A \times C \]

<table>
<thead>
<tr>
<th>line</th>
<th>expression</th>
<th>type</th>
<th>index</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( B \rightarrow C )</td>
<td>Pr</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( A \times B )</td>
<td>Pr</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( A )</td>
<td>As</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( B )</td>
<td>As</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( C )</td>
<td>O</td>
<td>14</td>
<td>1,4,( \rightarrow O )</td>
</tr>
<tr>
<td>6</td>
<td>( A \times C )</td>
<td>I</td>
<td>134</td>
<td>3,5,( \times I )</td>
</tr>
<tr>
<td>7</td>
<td>( A \times C )</td>
<td>O</td>
<td>12</td>
<td>2,3-6,( \times O )</td>
</tr>
</tbody>
</table>

8. Residual Law [Schönfinkel’s Law 35]

\[ (A \times B) \rightarrow C \vdash \sqsubset A \rightarrow (B \rightarrow C) \]

<table>
<thead>
<tr>
<th>line</th>
<th>expression</th>
<th>type</th>
<th>index</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((A \times B) \rightarrow C)</td>
<td>Pr</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( A )</td>
<td>As</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( B )</td>
<td>As</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( A \times B )</td>
<td>I</td>
<td>23</td>
<td>2,3,( \times I )</td>
</tr>
<tr>
<td>5</td>
<td>( C )</td>
<td>O</td>
<td>123</td>
<td>1,4,( \rightarrow O )</td>
</tr>
<tr>
<td>6</td>
<td>( B \rightarrow C )</td>
<td>I</td>
<td>12</td>
<td>3,5,( \rightarrow I )</td>
</tr>
<tr>
<td>7</td>
<td>( A \rightarrow (B \rightarrow C) )</td>
<td>I</td>
<td>1</td>
<td>2,6,( \rightarrow I )</td>
</tr>
</tbody>
</table>

3. Derivation System for Lambda-Composition

1. Definition of Derivation

A derivation of \( \phi_0 \) from \( \{\phi_1, \ldots, \phi_k\} \) is a sequence of lines organized as follows. By an expression, we mean an expression of Loglish.

<table>
<thead>
<tr>
<th>line no.</th>
<th>expression</th>
<th>type</th>
<th>index</th>
<th>annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \phi_1 )</td>
<td>( T_1 )</td>
<td>1</td>
<td>Pr</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>( k )</td>
<td>( \phi_k )</td>
<td>( T_k )</td>
<td>( k )</td>
<td>Pr</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>( \phi_0 )</td>
<td>T_0</td>
<td>( 1+\ldots+k )</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

In particular:

35 Named after Moses Schönfinkel ["Uber die Bausteine der mathematischen Logik", *Math. Ann.* 92 (1924), 305-316], who first proposed that one can treat a two-place function as a family of one-place functions.
2. 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.36

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>(i + j) + k = i + (j + k)</td>
<td>associative</td>
<td></td>
</tr>
<tr>
<td>(2)</td>
<td>i + j = j + i</td>
<td>commutative</td>
<td></td>
</tr>
</tbody>
</table>

Key to a valid derivation is the requirement that every assumption is used exactly once, which is the fundamental principle of Linear Logic.

3. 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.

<table>
<thead>
<tr>
<th>L</th>
<th>α (new)</th>
<th>A</th>
<th>m (new)</th>
<th>Pr</th>
</tr>
</thead>
</table>

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.

4. Assumption-Rule

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

<table>
<thead>
<tr>
<th>L</th>
<th>α (new)</th>
<th>A</th>
<th>m (new)</th>
<th>As</th>
</tr>
</thead>
</table>

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.

5. Difference Between Premises and Assumptions

Whereas every provisional assumption must be discharged via lambda-introduction, no premise is discharged.

---

36 One obtains "identity" [tautologies] by permitting an empty index $∅ \models [∅+ i = i]$. One obtains Relevance Logic, by adding contraction: $i+i = i$. 
6. Inference-Rules for $\lambda$ and $\times$

1. **Lambda-Out ($\lambda O$) [function-application]**

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$\Lambda$</th>
<th>$A \rightarrow C$</th>
<th>$i$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_2$</td>
<td>$\alpha$</td>
<td>$A$</td>
<td>$j$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$L_3$</td>
<td>$[\Lambda]\langle \alpha \rangle$</td>
<td>$C$</td>
<td>$i+j$</td>
<td>$L_1, L_2, \lambda O$</td>
</tr>
</tbody>
</table>

Here, $\Lambda$ is any function-expression [usually a lambda-abstract], $\alpha$ is any expression, and $[\Lambda]\langle \alpha \rangle$ is the result of applying $\Lambda$ to $\alpha$. See section below on lambda-conversion.

2. **Lambda-In ($\lambda I$) [function-generation]**

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$\alpha$</th>
<th>$A$</th>
<th>$i$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_2$</td>
<td>$\beta$</td>
<td>$B$</td>
<td>$i+j$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$L_3$</td>
<td>$\lambda: \alpha: \beta$</td>
<td>$A \rightarrow B$</td>
<td>$j$</td>
<td>$L_1, L_2, \lambda I$</td>
</tr>
</tbody>
</table>

3. **Cross-Out ($\times O$)**

<table>
<thead>
<tr>
<th>$L_0$</th>
<th>$\alpha \times \beta$</th>
<th>$A \times B$</th>
<th>$i$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_1 L_2$</td>
<td>$\alpha, \beta \leftarrow \gamma$</td>
<td>$\ldots$</td>
<td>$j$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$L_3$</td>
<td>$\gamma$</td>
<td>$C$</td>
<td>$i+j$</td>
<td>$L_0, L_1 L_2, \times O$</td>
</tr>
</tbody>
</table>

Here, $\alpha, \beta \leftarrow \gamma$ is a sub-derivation of $\gamma$ from $\alpha, \beta$, dependent on $j$, which is a sequence schematically presented as follows.

4. **Sub-Derivations**

A sub-derivation of $\beta$ from $\alpha_1, \ldots, \alpha_k$, dependent on $j$ is a sequence arranged as follows.

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$\alpha_1$</th>
<th>$A_1$</th>
<th>$m_1$ (new)</th>
<th>As</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$L_k$</td>
<td>$\alpha_k$</td>
<td>$A_k$</td>
<td>$m_k$ (new)</td>
<td>As</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$L_0$</td>
<td>$\beta$</td>
<td>$B$</td>
<td>$m_1 + m_k + j$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

5. **Alternative Account of Lambda-In**

<table>
<thead>
<tr>
<th>$L_1 L_2$</th>
<th>$\alpha \leftrightarrow \beta$</th>
<th>$\ldots$</th>
<th>$j$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_0$</td>
<td>$\lambda: \alpha: \beta$</td>
<td>$A \rightarrow B$</td>
<td>$j$</td>
<td>$L_1 L_2, \lambda I$</td>
</tr>
</tbody>
</table>

Here, $\alpha \leftrightarrow \beta$ is a sub-derivation of $\beta$ from $\alpha$, dependent on $j$.

---

37 Note that $j$ may be empty, although no actual line will ever have an empty-index.
6. **Cross-In (×I)**

<table>
<thead>
<tr>
<th></th>
<th>L₁</th>
<th>α</th>
<th>A</th>
<th>i</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L₂</td>
<td>β</td>
<td>B</td>
<td>j</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>L₃</td>
<td>α × β</td>
<td>A×B</td>
<td>i+j</td>
<td>L₁, L₂, ×I</td>
</tr>
</tbody>
</table>

7. **Function-Multiplication**

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

<table>
<thead>
<tr>
<th></th>
<th>L₁</th>
<th>λα: β</th>
<th>A→B</th>
<th>i</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L₂</td>
<td>λα: γ</td>
<td>A→C</td>
<td>j</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>L₃</td>
<td>λα: β×γ</td>
<td>A→(B×C)</td>
<td>i+j</td>
<td>L₁, L₂, FM</td>
</tr>
</tbody>
</table>

4. **Lambda-Calculus**

1. **Lambda-Conversion**

\[
\left[\lambda v \varepsilon\right]\langle \sigma \rangle \quad // \quad \varepsilon[\sigma/v]
\]

(1) \(v\) is any variable;
(2) \(\sigma\) is any expression of the same type as \(v\);
(3) \(\varepsilon[\sigma/v]\) results from substituting \(\sigma\) for every occurrence of \(v\) that is free in \(\varepsilon\) for \(\sigma\).

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 \(\varepsilon\) is an abstractor, \(v\) is a variable, and \(\varepsilon\) is an expression, every occurrence of \(v\) in \(\varepsilon v\) is bound by \(\varepsilon\).\(^{38}\)
(2) An occurrence \(o\) of a variable \(v\) is free in expression \(\varepsilon\) if and only if \(o\) is not bound by an abstractor.
(3) A variable \(v\) is free in \(\varepsilon\) if and only if at least one occurrence of \(v\) is free.
(4) A variable \(v\) is free for \(\Sigma\) in \(\varepsilon\) if and only if every variable that is free in \(\Sigma\) is also free in \(\varepsilon[\Sigma/v]\).
(5) An expression \(\varepsilon\) is closed if and only if no variable is free in \(\varepsilon\).
(6) An expression \(\varepsilon\) is open if and only if it is not closed.

---

\(^{38}\) The letter ‘\(\varepsilon\)’ is the Hebrew aleph, which is a cognate of Greek alpha, both deriving from the Phoenician letter ‘Aleph’ (alep). It is most famously used by the mathematician Georg Cantor (1845-1918) in his theory of transfinite numbers (1874).
2. **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) \( \pi \) on the class \( V \) of variables, whose inverse is \( \pi^{-1} \), such that, for any variable \( \nu \), \( \mathcal{E}_1 \) results by substituting \( \pi(\nu) \) for every bound occurrence of \( \nu \) in \( \mathcal{E}_2 \), and \( \mathcal{E}_2 \) results by substituting \( \pi^{-1}(\nu) \) for every bound occurrence of \( \nu \) in \( \mathcal{E}_1 \).

An occurrence of a variable \( \nu \) is bound precisely if that occurrence lies within the scope of an operator binding \( \nu \) – i.e., \( \forall \nu \), \( \exists \nu \), \( \lambda \nu \). Otherwise, that occurrence is free.

A variable \( \nu \) is free in an expression \( \mathcal{E} \) precisely at least one occurrence of \( \nu \) in \( \mathcal{E} \) is free in \( \mathcal{E} \).

5. **Further Composition Rules**

1. **Conjunction [Fundamental Form]** \[ 39 \]

\[
\begin{array}{ccccl}
L_1 & \Phi & S & i & \ldots \\
L_2 & \Psi & S & j & \ldots \\
L_3 & \Phi \& \Psi & S & i+j & L_1, L_2, \text{ Conj.}
\end{array}
\]

\( \Phi \) and \( \Psi \) are formulas.

2. **Conjunction [Derived Form]** \[ 40 \]

\[
\begin{array}{ccccl}
L_1 & \lambda \nu \Phi & A \rightarrow S & i & \ldots \\
L_2 & \lambda \nu \Psi & A \rightarrow S & j & \ldots \\
L_3 & \lambda \nu \{ \Phi \& \Psi \} & A \rightarrow S & i+j & L_1, L_2, \text{ Conj.}
\end{array}
\]

\( \nu \) is a variable of any type; \( \Phi \) and \( \Psi \) are formulas.

3. **Conjunction [Original Form]** \[ 41 \]

\[
\begin{array}{ccccl}
L_1 & \lambda \nu \Phi & D \rightarrow S & i & \ldots \\
L_2 & \lambda \nu \Psi & D \rightarrow S & j & \ldots \\
L_3 & \lambda \nu \{ \Phi \& \Psi \} & D \rightarrow S & i+j & L_1, L_2, \text{ Conj.}
\end{array}
\]

\( \nu \) is a type-D variable; \( \Phi \) and \( \Psi \) are formulas.

---

39 Our nickname for this rule is ‘VVV’, which is short for ‘Veni Vidi Vici’ [I came, I saw, I conquered] – reputedly written by Julius Caesar after his swift victory against Pharnaces II, King of Pontus and the Kingdom of Cimmerian Bosporus.

40 Follows from function-multiplication and fundamental form of Conjunction.

41 Special case of 2.