Title of Your Dissertation

A Suitable Subtitle

on Multiple Lines

Published by

LOT
Trans 10
3512 JK Utrecht
The Netherlands

phone: +31 30 253 6111
e-mail: lot@uu.nl
http://www.lotschool.nl

Cover illustration: The Tower of Babel, by Pieter Bruegel

ISBN: 000-11-22222-33-4
NUR: 000

Title of Your Dissertation

A Suitable Subtitle

on Multiple Lines

Nederlandstalige Titel

De Tweede Regel van Je Titel,

ook Vrij Lang

(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit Utrecht
op gezag van de rector magnificus, prof. dr. Henk Kummeling,
ingevolge het besluit van het college voor promoties
in het openbaar te verdedigen op

door

Konstantinos Kogkalidis

geboren 19 Juli 1991
te Thessaloniki, Greece

Promotores: Prof. Dr. M. J. Moortgat
Prof. Dr. R. Moot

The research reported here was supported by the Netherlands Organization for Scientific Research under the scope of the project "A composition calculus for vector-based semantic modelling with a localization for Dutch" (project number 360-89-070).

todo

Contents

Preface

Greetings, reader. Out of coincidence, or some weird turn of events, I have written this dissertation to-be and you have stumbled upon it. Introductions would normally be in order, but since this communication channel is asynchronous and unidirectional I will be doing double duty for the both of us.

So, let's start with you. A few scenarios are plausible as to why you are browsing these pages. Most likely you are an acquaintance of mine, either social – in which case you are wondering what it is I spent 5 years in Utrecht for, or academic – which means you are probably trying to figure out whether I am worthy of the title of doctor. In the latter case, I hope you won't be disappointed (especially so if you happen to be a member of the examination committee, for both our sakes). Or, perhaps, you are lazilly scrolling through the opening pages to evaluate my fitness for a potential job in an organization you are representing? If so, you should definitely go for me – unless this happens to be any of the big five[1], in which case: shoo, and shame on you, future me! Otherwise, could it even be that you are genuinely interested in the subject matter of this thesis? That would be mostly awesome, but also slightly alarming: I feel a bit conscious knowing that you might be putting my words under a critical lens – I'll do my best not to fail your expectations. In the unlikely event that you do not fall in any of the above categories, excuse my lack of foresight and know that you are still very welcome, and I am happy to have you around. In the more likely event that nobody ever reads this (far), let this transmission be forever lost to the void.

But enough with you, what about me? At the time of writing, I am in my early thirties and I call myself Kokos. I had the enormous luck of crossing paths with my supervisor, Michael, five years ago, during the first weeks of my graduate studies in Utrecht. The repercussions of this encounter were (and still are) unforeseeable. Coming from an engineering background that basked in practicality, with an obsessive repulsion to anything formal, his course offered me a glimpse of a whole new world. I got to see that proofs are not irrel-

[1] Preemptive apologies for not making sense to readers in the year 2053.

evant bureaucracies to avoid, but objects of interest in themselves, hidden in plain sight from the working hacker under common programming patterns. If this naive revelation came as shock, you can imagine my almost mystical awe when I was shown how proof & type theories also offer suitable tools and vocabulary for the analysis of human languages. Despite my prior ignorance, the "holy trinity" between constructive logics, programming languages and natural languages has been (with its ups and downs) at the forefronts of theoretical research for well over a century. This dissertation aims to be my tiny contribution to this line of work, conducted from the angle of a late convert, a theory-conscious hacker.

If all this sounds enticing and you plan on sticking around, at least for a bit longer, I think it would be beneficial if we set down the terms and conditions of what is to follow. It is no secret that dissertations are often boring to read, and it can be easy to lose track of context in seemingly unending walls of text. Striking a balance between being pedantic and making too many assumptions on background knowledge is no easy task: the only way to spare you unecessary headaches requires a mutual contract. On my part, I will try to clearly communicate my intentions, both about the thesis in full, and its parts in isolation: the idea is to make this manuscript as self-contained as possible, but without nitpicking on details or taking detours unnecessary for the presentation of the few novelties I have to contribute. Of you, I ask to remain conscious of what you are reading and aware of my own biases and limitations. The absence of feedback means that I can only model you in my imagination; I will inadvertently skip things that to me seem self-evident, and rant at length about others that you take for granted. So, feel free to skip ahead when something reads trivial, and do not judge too harshly when you encounter an explanation you find insufficient.

What this thesis is about

todo
contributions
sectioning

CHAPTER I

Introduction

"In the beginning, there were types."

Our story begins with the (over-ambitious, in hindsight) ravings of one of the world's most well-renowned mathematicians, David Hilbert. Unhappy with the numerous paradoxes and inconsistencies of mathematics at the end of the 19th century, Hilbert would postulate the existence and advocate the formulation of a finite set of axiomatic rules, which, when put together, would give rise to the most well-behaved system known to [wo]mankind, capable of acting as a universal meta-theory for all mathematics, in the process absolving all mathematicians of their sins. The idea was of course appealing and gained traction, not the least due to Hilbert's influence over the field (and his will to exercise it). As with all ideas that generate traction, however, it was not long before a cultural counter-movement would develop. Intuitionism, with Luitzen Egbertus Jan Brouwer as its forefather, would challenge Hilbert's program by questioning the objective validity of (any) mathematical logic. What it would claim, instead, is that mathematics is but a subjective process of construction that abides by some rules of inference, which, internally consistent as they may be, hold no reflection of deeper truth or meaning. In practice, intuitionists would reject the law of the excluded middle (an essential tool for Hilbert's school of formalists) and argue that for a proof to be considered valid, it has to provide concrete instructions for the construction of the object it claims to prove. The dispute went on for a couple of decades, its flame carried on by the respective students of the two rivals. Logic, intrigue, conflict, fame, no LaTeX errors... these truly were the years to be an active mathematician. Eventually, in a critical moment of clarity and inspiration, and tired by the

ongoing drama, Kurt Gödel, with his famous incompleteness theorem, would declare Hilbert's program unattainable, thus putting a violent end to the line of formalist heathens, and paving the way for the true revolution that was to come. This is in reference, of course, to the biggest discovery of the last century[1], made independently (using wildly different words every time) by various mathematicians and logicians spanning different timelines. Put plainly, what is now known as the Curry-Howard correspondence establishes a syntactic equivalence between deductive systems in intuitionistic brands of logic and corresponding computational systems, called λ-calculi. Put even more plainly, it suggests that valid proofs in such logics constitute in fact compilable code for functional progams, bridging in essence the seemingly disparate fields of mathematical logic and computer science. The repercussions of this discovery were enormous, and are more tangible today than ever before; type systems comprised of higher-order λ-calculi and their logics provide the theoretical foundations for modern programming languages and proof assistants (this last fact is both important and interesting, but won't concern us much presently).

In a more niche (but equally beautiful) fragment of the academic world, and in parallel to the above developments, applied logicians and formally inclined linguists have been demonstrating a stunning perserverance in their self-imposed quest of modeling natural language syntax and semantics, making do only with the vocabulary provided by formal logics. This noble endeavour traces its origins back to Aristotle, but its modern incarnation is due to Jim Lambek, who was the first to point out that the grammaticality of a natural language utterance can be equated to provability in a certain logic (or type inhabitation, if one is to borrow the terminology of constructive type theories), if the grammar (a collection of empirical linguistic rules) were to be treated as a substructural logic (a collection of formal mathematical rules). Funnily enough, the kind of logics Lambek would employ for his purposes would be exactly those at the interesection of intuitionistic and linear logic, the latter only made formally explicit in a breakthrough paper by Jean-Yves Girard almost three decades later. By that time, Richard Montague had already come up with the fantastically novel idea of seeing no distinction between formal and natural languages, single-handedly birthing and popularizing the field of formal semantics (which would chiefly invole semantic computations using λ-calculus notation). With this, he fulfilled Gottlob Frege's long-prophesized principle of compositionality, which would once and for all put the Chomskian tradition to rest[2], ushering linguistics into a new era. With the benefit of posterity, it would be tempting for us to act smart and exclaim that Lambek and Montague's ideas were remarkably aligned. In reality, it took another couple of decades for someone to notice. The credit is due to Johan van Benthem, who basically pointed out that Lambek's calculi make for the perfect syntactic

[1] In proof theory, at least.

[2] In some corners of the world, this part of the prophecy is yet to transpire.

machinery for Montague's program, seeing as they admit the Curry-Howard correspondence, and are therefore able to drive semantic composition virtually for free (in fact one could go as far as to say that they are the only kind of machinery that can accomplish such a feat without being riddled with ad-hoc transformations). This revelation, combined with the contemporary bloom of substructural logics, was the spark that ignited a renewed interest in Lambek's work. The culmination point for this interest was type-logical grammars (or categorial type logics): families of closely related type theories extending the original calculi of Lambek with unary operators lent from modal logic, intended to implement a stricter but more linguistically faithful modeling of the composition of natural language form and meaning.

In this chapter, we will isolate some key concepts from this frantic timeline and expound a bit on their details. Other than reinvented notation or perhaps some fresh example, no novel contributions are to be found here; the intention is merely to establish some common grounds before we get to proceed. If confident in your knowledge of the subject matter, `goto` Chapter II, but at your own risk.

1 The Simple Theory of Types

Simple type theory is the computational formalization of intuitionistic logic. It is in essence an adornment of the rules of intuitionistic logic with the computational manipulations they dictate upon mathematical terms. Dually, it provides a decision procedure that allows one to infer the type of a given program by inspecting the operations that led up to its construction. It is a staple of almost folkloric standing for computer scientists across the globe, tracing its origins to the seminal works of Russel and Church [Russell, 1908; Church, 1940]. The adjective "simple" is not intended as either a diminutive nor a condescending remark pertaining to the difficulty of the subject matter, but rather to distinguish it from the broader class of intuitionistic type theories, which attempt to systematize the notions of quantification (universal and existential), stratification of propositional hierarchies, and more recently equivalence (neither of which we will concern ourselves with).

Our presentation will begin with intuitionistic logic. Once that is done, we will give a brief account of the the Curry-Howard correspondence, which shall allow us to give a computational account of the logic, that being the simply typed λ-calculus.

1.1 Intuitionistic Logic

Intuitionistic logic is due to Arend Heyting [Heyting, 1930], who was the first to formalize Bouwer's intuitionism. It is a restricted version of classical logic, where the laws of the excluded middle (*tertium non datur*) and the elimination of the double negation no longer hold universally. The first states that one must choose between a proposition A and its negation $\neg\mathrm{A}$ ($\mathrm{A} \vee \neg\mathrm{A}$), whereas the second that a double negation is equivalent to an identity ($\neg\neg\mathrm{A} \equiv \mathrm{A}$). The absence of these two laws implies that several theorems of classical logic are no longer derivable in intuitionistic logic, meaning that the logic is weaker in terms of expressivity. On the bright side, it has the pleasant effect that proofs of intuitionistic logic are constructive, i.e. they explicitly demonstrate the formation of a concrete instance of whatever proposition they claim to be proving.

Focusing on the disjunction-free fragment of the logic, we have a tiny recursive language that allows us to define the various shapes of logical *propositions* (or *formulas*).[1] Given some finite set of *propositional constants* (or *atomic formulas*) Prop_0, and A, B, C arbitrary well-formed propositions, the language of propositions in Backus-Naur form is inductively defined as:

$$\mathrm{A}, \mathrm{B}, \mathrm{C} := p \mid \mathrm{A} \to \mathrm{B} \mid \mathrm{A} \times \mathrm{B} \tag{I.1}$$

where $p \in \mathsf{Prop}_0$. Propositions are therefore closed under the two binary *logical*

[1] The full logic also includes disjunctive formulas, but we will skip them from this presentation as they are of little interest to us. For brevity, we will from now on use intuitionistic logic to refer to its disjunction-free fragment.

connectives $\to$ and $\times$; we call the first an *implication*, and the second a *conjunction*. A *complex* proposition is any proposition that is not a member of Prop_0, and its *primary* (or main) connective is the last logical connective used when writing it down according to the grammar (I.1).

Besides propositions, we have *structures*. Structures are built from propositions with the aid of a single binary operation, the notation and properties of which can varj between different presentations of the logic. In our case, we will indicate valid structures with Greek uppercase letters Γ, Δ, Θ, and define structures inductively as

$$\Gamma, \Delta, \Theta := 1 \mid \mathrm{A} \mid \Gamma, \Delta \tag{I.2}$$

In other words, structures are an inductive set closed under the operator $\_,\_$ which satisfies associativity and is equipped with an identity element 1 (the *empty* structure), i.e. a monoid. A perhaps more down-to-earth way of looking at a structure is as a *list* or *sequence* of propositions.

Given propositions and structures, we can next define *judgements*, statements of the form $\Gamma \vdash \mathrm{A}$. We read such a statement as a suggestion that from *assumptions* Γ (i.e. a structure of *hypotheses*) one can derive a proposition A. Formulas occurring within Γ are said to occur in *antecedent* position, whereas A is in *succedent* position.

A *rule* is a two-line statement separated by a horizontal line. Above the line, we have a (possibly empty) sequence of judgements, which we call the *premises* of the rule. Below the line, we have a single judgement, which we call the rule's *conclusion*. The rule can be thought of as a formal guarantee that if all of its premises are deliverable, then so is the conclusion. Each rule has an identifying name, written directly to the right of the horizontal line.

Rules may be split in two conceptual categories. *Logical* rules, on the one hand, provide instructions for eliminating and introducing logical connectives. Figure I.1a presents the logical rules of intuitionistic logic. The first rule, the axiom of identity id, contains no premises and asserts the reflexivity of the provability operator $\vdash$. It states that from a proposition A one can infer, guess what, that very proposition. The remaining logical rules come in pairs, one per logical connective. The elimination of the implication (or modus ponens) states that, given a proof of a proposition $\mathrm{A}\to\mathrm{B}$ from assumptions Γ and a proof of proposition A from assumptions Δ, one can join the two to derive a proposition B. Dually, the introduction of the implication (or *deduction theorem*) states that from a proof of a proposition B given assumptions Γ, A, one can use Γ alone to derive an implicational proposition $\mathrm{A}\to\mathrm{B}$. In a similar manner, the elimination of the conjunction $\times E$ states that, given a proof of a proposition $\mathrm{A} \times \mathrm{B}$ from assumptions Γ, and a proof that the triplet Δ, A, B allows us to derive a proposition C, one could well use Γ together with Δ to derive C directly. And dually again, the introduction of the conjunction $\times I$ permits us to join two unrelated proofs, one of A from Γ and one of B from Δ into a single proof, that of their product $\mathrm{A} \times \mathrm{B}$, from Γ joined with Δ.

$$\frac{}{A \vdash A}\ \mathrm{id}$$

$$\frac{\Gamma \vdash A \to B \quad \Delta \vdash B}{\Gamma, \Delta \vdash B}\ \to E \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to I$$

$$\frac{\Gamma \vdash A \times B \quad \Delta, A, B \vdash C}{\Gamma, \Delta \vdash C}\ \times E \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \times B}\ \times I$$

(a) Logical rules.

$$\frac{\Gamma, \Delta \vdash A}{\Delta, \Gamma \vdash A}\ \mathrm{ex}$$

$$\frac{\Gamma \vdash B}{\Gamma, A \vdash B}\ \mathrm{weak} \qquad \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\ \mathrm{contr}$$

(b) Structural rules.

Figure I.1: Intuitionistic Logic $\mathbf{IL}_{\to,\times}$.

Structural rules, on the other hand, allow us to manipulate structures (who would have thought); they are presented in Figure I.1b. Structural rules have a two-fold role. First, they explicate an extra propery of our structure binding operator, namely commutativity. One could also make do with an implicit exchange rule by treating structures as *multisets* rather than lists – having it explicit, however, will keep us conscious of its presence and strengthen our emotional bond to it, in turn making us really notice its absence when it will no longer be there (it also keeps the presentation tidier). Second, they give an account of the status of propositions as permanent and reusable facts. The weakening rule weak states that if we were able to derive a proposition B from some assumptions Γ, we will also be able to do so if the assumptions were to contain some arbitrary extra proposition A. Conversely, the contraction rule contr states that if we needed some assumption structure containing two instances of a proposition A to derive a proposition B, we could also make do with just one instance of it, discarding the other without remorse.

A *proof*, finally, is a heterogeneous variadic tree. At its root, it has a judgement, guaranteed to be derivable (provided we did not mess up somewhere), called its *endsequent*. Its branches are themselves proofs, fused together by a rule – the number of premises being the local tree's arity. At its leaves, it has identity axioms – the smallest kind of proof.

1.1.1 Proof Equivalences

The same judgement may be provable in more than one ways. The difference between two proofs of the same judgement can be substantial, when they in-

deed describe distinct derivation procedures, or trivial. Trivial variations come in two kinds: syntactic equivalences (i.e. sequences of rule applications that can safely be rearranged) and redundant detours (i.e. sequences of rule applications that can altogether removed).

The first kind is not particularly noteworthy. In essence, we say that two proofs are syntactically equivalent if they differ only in the positioning of structural rule applications. This notion can be formally captured by establishing an equivalence relation between proofs on the basis of commuting conversions.

The second kind is more interesting and slightly more involved. A proof pattern in which a logical connective is introduced, only to be immediately eliminated, is called a *detour* (or β redex). Detours can be locally resolved via proof rewrites – the fix-point of performing all applicable resolutions is called *proof normalization* and yields a canonical proof form. The strong normalisation property guarantees that a canonical form exists for any proof in the logic, and in fact the choice of available rewrites to apply at each step is irrelevant, as all paths have the same end point [de Groote, 1999]. Figure I.2 presents rewrite instructions for the two detour patterns we may encounter (one per logical connective). Read bottom-up[1], the first one suggests that if one were to hypothesize a proposition A, use it within an (arbitrarily deep) proof s together with extra assumptions Γ to derive a proposition B, before finally redacting the hypothesis and composing with a proof t that derives A from assumptions Δ, it would have been smarter (and more concise!) to just plug in t directly when previously hypothesizing A, since then no redaction or composition would have been necessary. In a similar vein, the second suggests that if one were to derive and merge proofs s and t (of propositions A and B, respectively), only to eliminate their product against hypothetical instances of A and B that were used in proof u to derive C (together with assumptions Θ), the proof can be reduced by just plugging s and t in place of the axiom leaves of u. Note the use of horizontal dots at the axiom leaves, denoting simultaneous substitutions of *all* occurrences of redundant hypotheses, and the use of unnamed vertical dots, denoting (invertible) sequences of contr and/or ex rules.

1.2 The Curry-Howard Correspondence

The Curry-Howard correspondence asserts an equivalence between the above presentation of the logic in natural deduction, and a system of computation known as the λ-calculus. It was first formulated by Haskell Curry in the 30s before being independently rediscovered by William Alvin Howard and Nicolas Goveert de Bruijn in the 60s [Curry, 1934; de Bruijn, 1983; Howard, 1980]. The entry point for such an approach is to interpret propositions as *types* of a minimal functional programming language (a perhaps more aptly named al-

[1] In the small-to-big rather than literal sense! If confused: start from the proof leaves and go down.

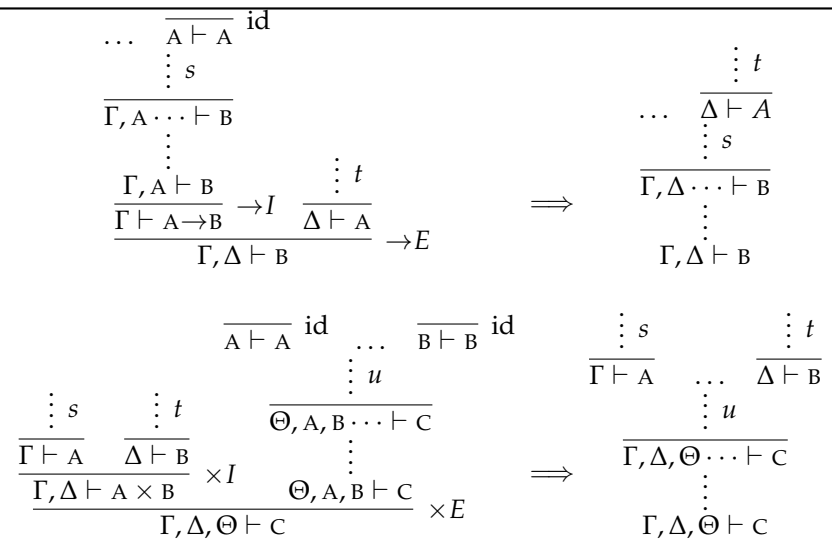


Figure I.2: Intuitionistic β redexes.

ternative to the Curry-Howard correspondence is the *propositions-as-types interpretation*). In that sense, the set of propositional constants Prop_0 becomes the programming language's basic set of *primitive* types (think of them as built-ins). Implicational formulas $\mathrm{A}\to\mathrm{B}$ are read as *function* types, and conjunction formulas are read as *tuple* (or cartesian product) types. From now we will use formulas, propositions and types interchangeably. Following along the correspondence allows us to selectively speak about individual, named instances of propositions – we call these *terms*. The simplest kind of term is a *variable*, corresponding to a hypothesis in the proof tree. Each logical rule is identified with a programming pattern: the axiom rule is variable *instantiation*, introduction rules are *constructors* of complex types, and elimination rules are their *destructors*. The question of whether a logical proposition is provable translates to the question of whether the corresponding type is inhabited; i.e. whether an object of such a type can be created – we will refer to the latter as a *well-formed* term.

Rather than present a grammar of terms and later ground it in the logic, we will instead simply revisit the rules we established just above, now adorning each with a term rewrite instruction – the result is a tiny yet still elegant and expressive type theory, presented in Figure I.3. Given an infinite but enumerable set Vars consisting of (unique names for) indexed variables with elements $\{\mathsf{x}_i, \mathsf{x}_j, \mathsf{x}_k, \mathsf{x}_l, \ldots\}$, and denoting arbitrary but well-formed terms with s, t, u, we will use $\mathsf{s} : \mathrm{A}$ (or s^{A}) to indicate that term s is of type A. Assumptions Γ, Δ will

$$\frac{}{x_i : A \vdash x_i : A}\ \text{id}$$

$$\frac{\Gamma \vdash s : A{\to}B \quad \Delta \vdash t : B}{\Gamma, \Delta \vdash s\ t : B}\ {\to}E \qquad \frac{\Gamma, x_i : A \vdash s : B}{\Gamma \vdash \lambda x_i.s : A{\to}B}\ {\to}I$$

$$\frac{\Gamma \vdash s : A \times B \quad \Delta, x_i : A, x_j : B \vdash t : C}{\Gamma, \Delta \vdash \text{case } s \text{ of } (x_i, x_j) \text{ in } t : C}\ {\times}E \qquad \frac{\Gamma \vdash s : A \quad \Delta \vdash t : B}{\Gamma, \Delta \vdash (s, t) : A \times B}\ {\times}I$$

$$\frac{\Gamma, \Delta \vdash s : A}{\Delta, \Gamma \vdash s : A}\ \text{ex}$$

$$\frac{\Gamma \vdash s : B}{\Gamma, x_i : A \vdash s : B}\ \text{weak} \qquad \frac{\Gamma, x_i : A, x_j : A \vdash s : B}{\Gamma, x_k : A \vdash s_{[x_i \mapsto x_k, x_j \mapsto x_k]} : B}\ \text{contr}$$

Figure I.3: Simple type theory.

now denote a *typing environment*:

$$x_1 : A_1,\ x_2 : A_2 \ldots\ x_n : A_n \tag{I.3}$$

i.e. rather than a sequence of formulas, we have a sequence of distinct variables, each of a specific type, and a judgement $\Gamma \vdash s : B$ will now denote the derivation of a term s of type B out of such an environment.

Inspecting Figure I.3, things for the most part look good. The implication elimination rule ${\to}E$ provides us with a composite term $s\ t$ that denotes the function application of *functor* s on *argument* t. Function application is left-associative: $s\ t\ u$ is the bracket-economic presentation of $(s\ t)\ u$ – we have no choice but to use brackets if want to instead denote $s\ (t\ u)$. The dual rule, ${\to}I$, allows us to create (so-called anonymous) functions by deriving a result s dependent on some hypothesized argument x_i which is then *abstracted* over as $\lambda x_i.s$. Any occurrence of x_i within s is then *bound* by the abstraction; variables that do not have a binding abstraction are called *free*. The conjunction introduction ${\times}I$ allows us to create tuple objects (s, t) through their parts s and t. Its dual, ${\to}E$, gives us the option to identify the two coordinates of a tuple s with variables x_i and x_j, when the latter are hypothesized assumptions for deriving some program t. If our assumptions are not in order, blocking the applicability of some rule, we can put them back where they belong with ex. With contr we can pretend to be using two different instances x_i and x_j of the same type before identifying the two as a single object x_i in term s with term t"); note here the meta-notation for *variable substitution*, $s_{[x_i \mapsto t]}$, which reads as "replace any occurrence of variable x_i. And finally, we can introduce throwaway variables

into our typing environment with weak (arguably useful for creating things like constant functions).

There's just a few catches to beware of. The first has to do with tracing variables in a proof; the concatenation of structures Γ, Δ is only valid if Γ and Δ contain no variables of the same name; if that were to be the case, we would be dealing with variable shadowing, a situation where the same name could ambiguously refer to two distinct objects (a horrible thing). The second has to do with do with the ex rule. The careful reader might notice that the rule leaves no imprint on the term level, meaning we cannot distinguish between a program where variables were a priori provided in the correct order, and one where they were shuffled into position later on. This is justifiable if one is to treat the rule as a syntactic bureaucracy that has no real semantic effect, i.e. if we consider the two proofs as equivalent, following along the commuting conversions mentioned earlier (supporting the idea that in this type theory, asssumptions are multisets rather than sequences). A slightly more perverse problem arises out of the product elimination rule $\times E$. The rule posits that two assumptions $x_i : A$ and $x_j : B$ can be substituted by a single (derived) term of their product type $s : A \times B$. Choosing different depths within the proof tree upon which to perform this substitution will yield distinct terms (because indeed they represent distinct sequences of computation); whether there's any merit in distinguishing between the two is, however, debatable. Finally, whereas other rules can be read as syntactic operations on terms, (this presentation of) the contr rule contains meta-notation that is not part of the term syntax itself. That is to say, $s_{[x_i \mapsto t]}$ is *not* a valid term – even if the result of the operation it denotes is. Generally speaking, substitution of objects for others of the same type is (modulo variable shadowing) an admissible property of the type system. Mixing syntax and meta-syntax in the same system is a dirty trick we will sporadically employ; this surely invites some trouble, but conscious use of it can be worth it, since it significantly simplifies presentation.

1.2.1 Term Equivalences

There exist three kinds of equivalence relations between terms, each given an identifying Greek letter.[1]

α conversion is a semantically null rewrite obtained by renaming variables according to the substitution meta-notation $s_{[x_i \mapsto x_j]}$ described above. Despite seeming innocuous at a first glance, α conversion is an evil and dangerous operation that needs to be applied with extreme caution so as to avoid variable capture, i.e. substituting a variable's name with one that is already in use. Two terms are α equivalent if we can rewrite one into the other using just α

[1] The denotational significance of these letters I have yet to understand – legend has it that it only starts making sense after having written your 10th compiler from scratch.

conversions, e.g.

$$\lambda \mathsf{x}_i.\mathsf{x}_i^{\mathsf{A}} \overset{\alpha}{\equiv} \lambda \mathsf{x}_j.\mathsf{x}_j^{\mathsf{A}} \tag{I.4}$$

Standardizing variable naming, e.g. according to the distance between variables and their respective binders, alleviates the effort required to check for α equivalence by casting it to simple syntactic equality (string matching).

β reduction The term rewrites we have so far inspected were either provided by specific rules, or were notational overhead due to the denominational ambiguity of variables. Aside from the above, our type system provides two minimal computation steps that tell us how to reduce expressions that involve the deconstruction of a just-constructed type:

$$(\lambda \mathsf{x}_i.\mathsf{s})\ \mathsf{t} \overset{\beta}{\rightsquigarrow} \mathsf{s}_{[\mathsf{x}_i \mapsto \mathsf{t}]} \tag{I.5}$$

$$\mathsf{case}\ (\mathsf{s}, \mathsf{t})\ \mathsf{of}\ (\mathsf{x}_i, \mathsf{x}_j)\ \mathsf{in}\ \mathsf{u} \overset{\beta}{\rightsquigarrow} \mathsf{u}_{[\mathsf{s} \mapsto \mathsf{x}_i, \mathsf{t} \mapsto \mathsf{x}_j]} \tag{I.6}$$

A term on which no β reductions can be applied is said to be in β-normal form. The Church-Rosser theorem asserts first that one such form exists for all well-formed terms, and second, that this form is inevitable and inescapable – any reduction strategy followed to the end will bring us to it [Barendregt et al., 1984]. Two terms are β equivalent to one another if they both reduce to the same β-normal form.

If you are at this point getting a feeling of deja vu, rest assured this is not on you; we have indeed gone through this before, last time around with proofs rather than terms. If one were to replicate the above term reductions with their corresponding proofs, they would end up exactly with the proof reduction patterns of Figure I.2. I will spare you the theatrics of faking surprise at this fact, but if this not something you were exposed to previously, take a moment here to marvel at the realization that proof normalization is in reality "just" computation. This ground-shattering discovery lies at the essence of the Curry-Howard correspondence.

η conversion In contrast to β conversion, which tells us how to simplify an introduce-then-eliminate pattern, η conversion tells us how to modify an eliminate-then-introduce pattern. An η long (or normal) form of a term is one in which the arguments to type operators are made explicit (i.e. all introductions of a connective are preceded by its elimination), whereas an η contracted (or pointfree) form is one where arguments are kept hidden [Prawitz, 1965]. We refer to the simplification of an expanded form as η reduction, which is the computational dual of β reduction. The reverse process is an η expansion: η long forms witness the completeness of atomic propositions, which posits that any derivable statement $\Gamma \vdash \mathrm{A}$ can be converted to an alternative form $\Gamma' \vdash \mathrm{A}$, where Γ' consists only of atomic propositions [Troelstra and Schwicht-

Logic	Computer Science
Propositional Constant	Base Type
Logical Connectives	Type Constructors
Implication	Function Type
Conjunction	Product Type
Axiom	Variable
Introduction Rule	Constructor Pattern
Elimination Rule	Destructor Pattern
Proof Normalization	Computation
Provability	Type Inhabitation

Table I.1: The Curry-Howard correspondence in tabular form.

enberg, 2000]. Both expansion and reduction are facets of η conversion – the equivalence relation enacted by this conversion is called η equivalence.

$$\lambda \mathsf{x}_i.\mathsf{s}\ \mathsf{x}_i \overset{\eta}{\equiv} \mathsf{s} \tag{I.7}$$

$$(\mathsf{case}\ \mathsf{s}\ \mathsf{of}\ (\mathsf{x}_i,\mathsf{x}_j)\ \mathsf{in}\ \mathsf{x}_i,\ \mathsf{case}\ \mathsf{s}\ \mathsf{of}\ (\mathsf{x}_k,\mathsf{x}_l)\ \mathsf{in}\ \mathsf{x}_l) \overset{\eta}{\equiv} \mathsf{s} \tag{I.8}$$

1.2.2 In Place of a Summary

Table I.1 summarizes the subsection.

1.3 Intermezzo

We now know how to prove things (or compute with types). Before moving along with this chapter's agenda, we will take a brief pause to provide some auxiliary definitions and notations that should prove relevant later on. This is also a chance to do a bit of warming up with some baby examples before some real world proofs start coming our way.

Formula Polarity Each unique occurrence of (part of) a formula within a judgement can be assigned a polarity value, positive or negative. All antecedent formulas are positive, and the lone succdent formula right is negative. Complex formulas propagate polarities to their constituents depending on their own polarity and primary connective – this way, all subformulas down to propositional constants are polarized. Conjunctive formulas propagate their polarity unchanged to both their coordinates, whereas implicative formulas flip their polarity for the constituent left of the arrow; see Table I.2. Intuitively, we can think of negative formulas as being in argument position (conditions for the proof to proceed), and positive formulas as being in result position (conditionally provable statements).

Complex formula / of polarity		Constituent polarity A	Constituent polarity B
$A \times B$	$+$	$+$	$+$
	$-$	$-$	$-$
$A \to B$	$+$	$-$	$+$
	$-$	$+$	$-$

Table I.2: Polarity induction.

$$\dfrac{\dfrac{\dfrac{}{x_0 : A \to B \vdash x_0 : A \to B}\ \text{id} \quad \dfrac{}{x_1 : A \vdash x_1 : A}\ \text{id}}{x_0 : A \to B, x_1 : A \vdash x_0\ x_1 : B}\ {\to}E}{x_1 : A \vdash \lambda x_0.x_0\ x_1 : (A \to B) \to B}\ {\to}I$$

Figure I.4: Type raising.

Type Raising Type raising $A \vdash (A \to B) \to B$ is a derivable theorem of intuitionistic logic presented in Figure I.4. It states that for A, B arbitrary propositions, from A one can derive its raised form $A \to B$. The converse, i.e. type lowering, is not true: $(A \to B) \to B \nvdash A$.

Function Order The implication-only fragment of the logic includes $\to$ as its sole logical connective. The resulting type theory is one that deals only with functions; for its types, we can define their order $\mathcal{O}$ as follows:

$$\begin{aligned} \mathcal{O}(p) &:= 0 && p \in \mathsf{Prop}_0 \\ \mathcal{O}(A \to B) &:= \max(\mathcal{O}(A) + 1, \mathcal{O}(B)) \end{aligned} \tag{I.9}$$

Types whose order is above 1 are called *higher-order* types; they denote functions that accept functions as their arguments.

Notational Shorthands The verbosity of term-decorated proofs can get cumbersome in the long run, and does not play well with the unforgiving horizontal margins enforced by the template imposed on writer and reader alike. It is probably inevitable that at some point proofs will need a smaller font size to fit in a page (or, worse yet, some neck-breaking rotations of the orientation plane), but in a futile attempt to postpone such emergency measures, we will occasionally make use of a shorthand notation for natural deduction proofs that avoids repetition, at the cost of maybe requiring some extra time to visually parse. In this notation, axioms will be rewritten as follows:

$$\dfrac{}{x_i : A \vdash x_i : A}\ \text{id} \quad \equiv \quad \dfrac{}{x_i : A}\ \text{id}$$

$$\dfrac{\dfrac{}{\mathrm{x}_0 : (\mathrm{A} \times \mathrm{B}) \to \mathrm{C}}\ \mathrm{id} \quad \dfrac{\dfrac{}{\mathrm{x}_1 : \mathrm{A}}\ \mathrm{id} \quad \dfrac{}{\mathrm{x}_2 : \mathrm{B}}\ \mathrm{id}}{\mathrm{x}_1, \mathrm{x}_2 \vdash (\mathrm{x}_1, \mathrm{x}_2) : \mathrm{A} \times \mathrm{B}}\ \times I}{\dfrac{\mathrm{x}_0, \mathrm{x}_1, \mathrm{x}_2 \vdash \mathrm{x}_0\ (\mathrm{x}_1, \mathrm{x}_2) : \mathrm{C}}{\mathrm{x}_0 \vdash \lambda \mathrm{x}_1 \mathrm{x}_2 . \mathrm{x}_0\ (\mathrm{x}_1, \mathrm{x}_2) : \mathrm{A} \to \mathrm{B} \to \mathrm{C}}\ \to I(x2)}\ \to E}$$

(a) Currying

$$\dfrac{\dfrac{\dfrac{}{\mathrm{x}_0 : \mathrm{A} \times \mathrm{B}}\ \mathrm{id} \quad \dfrac{\dfrac{\dfrac{}{\mathrm{x}_1 : \mathrm{A} \to \mathrm{B} \to \mathrm{C}}\ \mathrm{id} \quad \dfrac{}{\mathrm{x}_2 : \mathrm{A}}\ \mathrm{id}}{\mathrm{x}_1, \mathrm{x}_2 \vdash \mathrm{x}_1\ \mathrm{x}_2 : \mathrm{B} \to \mathrm{C}}\ \to E \quad \dfrac{}{\mathrm{x}_3 : \mathrm{B}}\ \mathrm{id}}{\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3 \vdash \mathrm{x}_1\ \mathrm{x}_2\ \mathrm{x}_3 : \mathrm{C}}\ \to E}{\mathrm{x}_0, \mathrm{x}_1 \vdash \mathrm{case}\ \mathrm{x}_0\ \mathrm{of}\ (\mathrm{x}_2, \mathrm{x}_3)\ \mathrm{in}\ \mathrm{x}_1\ \mathrm{x}_2\ \mathrm{x}_3 : \mathrm{C}}\ \times E}{\mathrm{x}_1, \mathrm{x}_0 \vdash \mathrm{case}\ \mathrm{x}_0\ \mathrm{of}\ (\mathrm{x}_2, \mathrm{x}_3)\ \mathrm{in}\ \mathrm{x}_1\ \mathrm{x}_2\ \mathrm{x}_3 : \mathrm{C}}\ \mathrm{ex}}{\mathrm{x}_1 \vdash \lambda \mathrm{x}_0 . \mathrm{case}\ \mathrm{x}_0\ \mathrm{of}\ (\mathrm{x}_2, \mathrm{x}_3)\ \mathrm{in}\ \mathrm{x}_1\ \mathrm{x}_2\ \mathrm{x}_3 : (\mathrm{A} \times \mathrm{B}) \to \mathrm{C}}\ \to I$$

(b) Uncurrying

Figure I.5: Interderivability of product and arrow.

And assumptions will appear without type assignments (if uncertain of what some variable's type is, just trace it back to its axiom). We will always provide type declarations for derived terms (right of the turnstile). The examples of the next paragraph (and many of the ones to follow) will use this alternative notation.

Currying A product type occurring in the argument position of an implication is interderivable with a longer implication where its coordinates are sequentialized: $(\mathrm{A} \times \mathrm{B}) \to \mathrm{C} \dashv\vdash \mathrm{A} \to \mathrm{B} \to \mathrm{C}$. The forward direction is called currying, and the backward uncurrying; you can find a proof for each in Figure I.5. Having proven that once, we can reuse that proof for deriving implicational equivalents from conjunctions (including nested ones, provided they occur as arguments to an implication). Combined with type raising, this trick is interesting, as it permits us to indirectly argue about product types as higher-order implications, even in presentations of the theory that do not include an explicit product (and thus avoid the issues related to its elimination), e.g. we have:

$$\mathrm{A} \times \mathrm{B} \vdash ((\mathrm{A} \times \mathrm{B}) \to \mathrm{C}) \to \mathrm{C} \dashv\vdash (\mathrm{A} \to \mathrm{B} \to \mathrm{C}) \to \mathrm{C} \tag{I.10}$$

Keep a mental note.

Proof Search Attempting to derive a judgement of the form $\Gamma \vdash \mathrm{A}$ amounts to searching for a suitable proof of that statement, a process called *proof search*.

We distinguish two directions of proof search: the *backward chaining* (or top-down) approach starts from the goal judgement and iteratively expands it into judgements with smaller assumptions – one judgement per premise generated by the rule of inference applied – with the intention being the eventual deconstruction of all branches into axioms of identity. The other direction is called *forward chaining* (or bottom-up), and starts from a collection of hypothesized propositions (axioms) that are glued together to form progressively more complex structures, until the goal judgement is reached. Without digressing further, it is important to realize that both directions suffer the same issue, albeit from different angles, namely hypothetical reasoning. Forward chaining requires a perfect guess of any and all propositions reqired in deriving A from Γ, even those that will be redacted and thus never occur in Γ. Dually, backward chaining might require introduction of substructures and subformulas that are nowhere to be found in either the antecedents or the succedent of the current judgement due to the modus ponens-like behavior of implication elimination. Long story short, proof search is hard.

2 Going Linear

We are now ready to start charting grounds in substructural territories: we will gradually impoverish our logic by removing structural rules one by one, and see where that gets us. The weakest links are the contr and weak rules. These two rules are a cultural and ideological remnant of a long-gone age infested by delusions of prosperity and abundance. In their presence, propositions are proof objects that can be freely replicated and discarded. Removing them (or controlling their applicability via other means) directs us towards a more eco-conscious regime by turning propositions into finite resources, the production and/or consumption of which is not to be taken for granted. Removing contr yields Affine Logic, a logic in which resources can be used no more than once. Removing weak yields Relevant Logic, a logic in which resources can be used no less than once. Removing both yields Linear Logic, a logic in which resources can be used *exactly* once. The intuitionistic formulations of the above give rise to corresponding type theories [Pierce, 2004]. For the purposes of this manuscript, we will focus our presentation on linear type theory.

2.1 Linear Types

Linear logic is due to Jean-Yves Girard [Girard, 1987], and its computational interpretation due to Samson Ambramsky [Abramsky, 1993]. The full logic includes two disjunctive connectives as well as a modality that allows one to incorporate non-linear propositions into the presentation, but we will happily forget about those. Note that with these missing connectives included, the logic is not impoverished but rather enhanced – full linear logic in fact subsumes intuitionistic logic; we have no use of this much expressivity here

though. Insights from the previous section carry over to this one; we will no longer seperate the presentation between the logic and the type theory, but instead do both in one go.

For the fragment of interest to us, the type grammar becomes:

$$\mathrm{A}, \mathrm{B}, \mathrm{C} := p \mid \mathrm{A} \multimap \mathrm{B} \mid \mathrm{A} \otimes \mathrm{B} \mid \mathrm{A} \& \mathrm{B} \tag{I.11}$$

There is not really much we have to do to manipulate these new types, other than a slight cognitive rewiring. We will note first that the meaning of the implication arrow changes from material implication to transformation process; i.e. where we previously had $\mathrm{A} \to \mathrm{B}$ to denote that B logically follows from A, we will now have $\mathrm{A} \multimap \mathrm{B}$ to denote an irreversible process that transforms a single A into a single B, consuming the former in the process (we can think of this as a perfect chemical reaction). The new, weird-looking arrow of linear implication is read as lolli(pop) due to its suggestive appearance.[1] Conjunction $\times$ is now separated into two distinct operators, the multiplicative $\otimes$ and the additive &. The first denotes a linear tuple, and $\mathrm{A} \otimes \mathrm{B}$ is read as *both* A *and* B. A linear tuple offers no possibility of projection: we will need to use both coordinates going forward. The second denotes a choice, and A&B is read as A *with* B, or *choose one of* A *or* B. This choice is *external*, as the freedom of applying it lies with the operator rather than the proof, and is manifested by the presence of two eliminators for our new connective: a left projection $\&E_1$ and a right projection $\&E_2$; choosing one means we lose the possibility of obtaining the other. Unique to the $\&I$ rule is the fact that two proof branches used to derive each coordinate of the A&B conclusion share the same assumptions Γ. The subset of linear logic concerning the connectives discussed is presented in Figure I.6, together with its term rewrites (assumptions, judgements, rules and proofs look just like before). For the sake of homogeneity and explicitness, ex still makes an appearance as the sole structural rule.

Notationally, the absence of non-linear intuitionistic terms allows us to freely reuse our prior term notation without fear of ambiguity. We have three new term patterns: $\langle \mathsf{s}, \mathsf{t} \rangle$ to denote the choice between proof terms s and t (contrast to the linear tuple (s, t)), and $\mathrm{fst}(\_)$ and $\mathrm{snd}(\_)$ to denote the first and second projections. Similarly for the implication introduction rule $\multimap I$, no redundant variables means that x_i must now appear free once in the abstraction body s – in other words, we have no way of syntactically instantiating constant functions.

2.1.1 Proof & Term Reductions

The notions of proof and term equivalence discussed in the previous section hold also for linear logic. Proof normalization looks almost identical to before in the case of $\to$ and $\times$ (substituting of course for $\multimap$ and $\otimes$). The key differ-

[1] If trying to typeset it yourself, DO NOT duckduckgo for "lolli in latex". It can be found as `\multimap`. You are welcome.

$$\frac{}{x_i : A \vdash x_i : A}\ \mathrm{id}$$

$$\frac{\Gamma \vdash s : A \multimap B \quad \Delta \vdash t : A}{\Gamma, \Delta \vdash s\ t : B}\ \multimap E \qquad \frac{\Gamma, x_i : A \vdash s : B}{\Gamma \vdash \lambda x_i.s : A \multimap B}\ \multimap I$$

$$\frac{\Gamma \vdash s : A \otimes B \quad \Delta, x_i : A, x_j : B \vdash t : C}{\Gamma, \Delta \vdash \mathsf{case}\ s\ \mathsf{of}\ (x_i, x_j)\ \mathsf{in}\ t : C}\ \otimes E \qquad \frac{\Gamma \vdash s : A \quad \Delta \vdash t : B}{\Gamma, \Delta \vdash (s, t) : A \otimes B}\ \otimes I$$

$$\frac{\Gamma \vdash s : A \& B}{\Gamma \vdash \mathsf{fst}(s) : A}\ \&E_1 \qquad \frac{\Gamma \vdash s : A \& B}{\Gamma \vdash \mathsf{snd}(s) : B}\ \&E_2 \qquad \frac{\Gamma \vdash s : A \quad \Gamma \vdash t : B}{\Gamma \vdash \langle s, t \rangle : A \& B}\ \& I$$

(a) Logical rules.

$$\frac{\Gamma, \Delta \vdash s : A}{\Delta, \Gamma \vdash s : A}\ \mathrm{ex}$$

(b) Structural rule.

Figure I.6: Linear Logic $\mathbf{ILL}_{\multimap,\otimes,\&}$ and its type theory.

ence lies in the absence of horizontal dots and unnamed vertical dots (since the contr rule is no more, meaning that there can only be a single occurrence of each axiom replaced with a proof). The extra connective & introduces its own two redexes (one per eliminator); the reduction of the first projection is shown in Figure I.7. Its reading is straightforward: if one were to use Γ to independently derive A and B along two parallel proofs, proceed by constructing a choice between the two, and then also make that choice (favoring either), there was never any need for the other in the first place. The equivalent term reduction steps this time around are:

$$(\lambda x_i.s)\ t \overset{\beta}{\rightsquigarrow} s_{[x_i \mapsto t]} \tag{I.12}$$

$$\mathsf{case}\ (s, t)\ \mathsf{of}\ (x_i, x_j)\ \mathsf{in}\ u \overset{\beta}{\rightsquigarrow} u_{[s \mapsto x_i, t \mapsto x_j]} \tag{I.13}$$

$$\mathsf{fst}(\langle s, t \rangle) \overset{\beta}{\rightsquigarrow} s \tag{I.14}$$

$$\mathsf{snd}(\langle s, t \rangle) \overset{\beta}{\rightsquigarrow} t \tag{I.15}$$

2.2 Proof Nets

We have basked in the beauty of the natural deduction presentation adopted so far, and seen how it gives rise to a straightforward computational interpretation. We have also seen how it is at times overly bureaucratic in its ex-

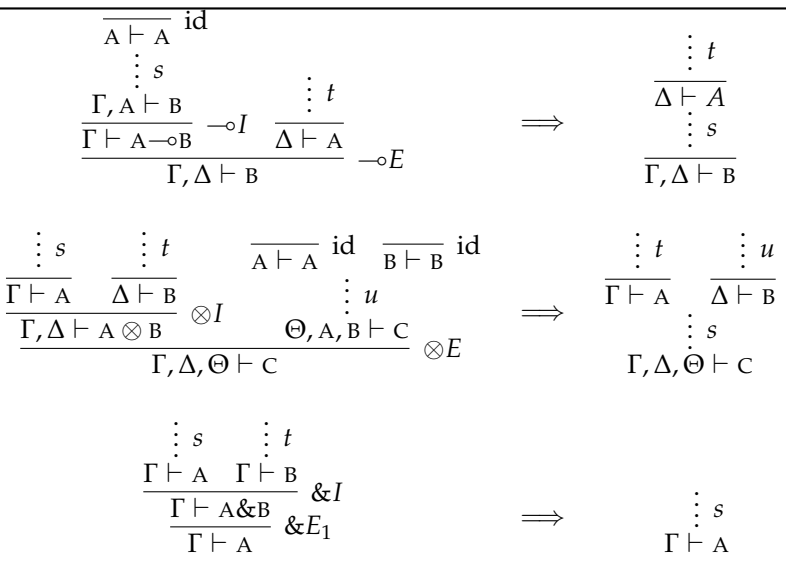


Figure I.7: Linear β redexes.

plication of structural rules that are null from a computational perspective. To our luck, an additional representation makes itself available as soon as we step into linear grounds, namely *proof nets*, also due to Girard [Girard, 1987]. Proof nets are best suited for the multiplicative fragment of the logic (they are amenable to extensions with additive connectives, but things get uglier there). In our case, we have foregone the disjunctive connectives, and we have already suggested that the multiplicative conjunction $\otimes$ is (to an extent) interchangeable with the implication arrow $\multimap$ (we actually did this for intuitionistic connectives, but there is no need to repeat ourselves – the story looks identical with their linear variants). This gives us the much needed excuse to justify limiting our presenting of proof nets to the implication-only fragment of linear logic, $\mathbf{ILL}_{\multimap}$, where things are easy and intuitive.

In natural deduction, our proofs are built sequentially. We start with some hypothesized variables and combine them via rules to derive more complex terms, which then serve as premises for a next iteration of rule applications. As long as we are careful not to get stuck in some detour loop hell, we rinse and repeat, and eventually, after a finite number of steps, we end up with our conclusion, at which point we can call it a day. Proof nets offer an appealing alternative: they are parallel, in the sense that they allow multiple conclusions to be derived simultaneously. They also have no notion of temporal precedence: everything happens in a single instant, meaning that positive subformulas are good to use without having to wait for their conditions to be met.

To see this in practice, a simple but concrete example will prove helpful. We will consider the natural deduction proof of linear function composition

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x_1 : B \multimap C}\,\mathrm{id} \quad \dfrac{\dfrac{}{x_0 : A \multimap B}\,\mathrm{id} \quad \dfrac{}{x_2 : A}\,\mathrm{id}}{x_0, x_2 \vdash x_0\ x_2 : B}\,\multimap E}{x_1, x_0, x_2 \vdash x_2\ (x_1\ x_0) : A \multimap C}\,\multimap E}{x_1, x_0 \vdash \lambda x_2.x_1\ (x_0\ x_2) : A \multimap C}\,\multimap I}{x_0, x_1 \vdash \lambda x_2.x_1\ (x_0\ x_2) : A \multimap C}\,\mathrm{ex}$$

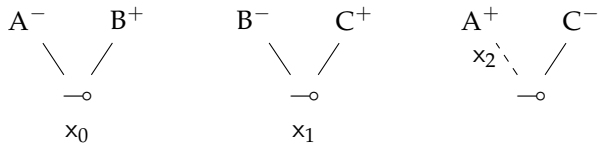
Figure I.8: Linear function composition.

of Figure I.8 and translate it into its proof net equivalent of Figure I.9.

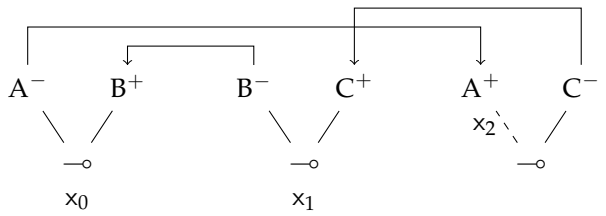
The first thing we need to do is write formulas as their *decomposition trees*; it sounds fancy, but in reality this is just recompiling formulas according to their underlying grammar, but now in tree form: propositional constants become leaves, and logical connectives become branch nodes. To make this a tiny bit more interesting, we will decorate atomic subformulas with a superscript denoting their polarity, according to the schema of Subsection 1.3. For formula trees with positive roots (i.e. trees occuring left of the turnstile), we will put a label beneath them to identify them with the variable that provides them. We will also distinguish tree edges originating from a negative implication and pointing to a positive tree by marking them with dashed lines – these denote positively rooted trees that are either nested within a higher-order positive implication, or in the argument position of an implication right of the turnstile.[1] Such positive formulas play the role of hypotheses that have been abstracted over, therefore we need to also give these a name; we can put it right next to the dashed line. An arrangement of the decomposition trees of all formulas as they occur within a judgement is called a *proof frame*; the one for our running example can be seen in Figure I.9a.

A proof frame must satisfy an invariance property before the eligibility of the judgement it prescribes can even be considered. Namely, it must contain an equal number of positive and negative occurrences of each unique atomic formula that appears within it. This perfectly fits the linear logic paradigm: everything produced has to be consumed, and vice versa. In our case, we have three unique formulas, A, B and C, each one of which has a single positive and a single negative occurrence: check. The next thing to do in building a proof net is to establish a bijection between atomic formulas of opposite polarity, and draw it as an extra set of edges, pointed negative atoms to their positive matches: we call those *axiom links*. Axiom links essentially specify the elimination of function types: they identify function arguments with their concrete inputs in a geometric fashion. We do not need to put much thought for our running example: since all atoms have a single occurrence of each polarity, there is just one possible bijection to consider, presented in Figure I.9b. A

[1] An alternative notation employs two distinct symbols for the positive and negative versions of an implication. In the literature, these can be encountered as tensor ($\otimes$) and par ($\parr$) links.



(a) Proof frame.



(b) Proof structure.

Figure I.9: Proof net construction for the proof of Figure I.8.

proof frame with axiom links on top is called a *proof structure*, and this representation provides all of the information contained within a proof.

Alas, the relation from proofs structures is not one-to-one: there exist many more proof structures than proofs. A proof structure is a *proof net* if and only if it satisfies a correctness criterion. There have been various formulations of this criterion, each with a different time complexity, ranging from exponential to linear [Girard, 1987; Danos and Regnier, 1989; Murawski and Ong, 2000; Guerrini, 2011]. We will adopt an (informally rephrased) version of the acyclicity and connectedness criterion of Danos and Regnier [1989]. We are going to treat a proof structure as a heterogeneous graph, consisting of two node types, logical connectives and atomic formulas, and three edge types: tree-structure edges, further subcategorized according to their polarity, and axiom links. We are then going to attempt a traversal of that graph using an algorithm defined on the basis of the above parameterization, that further utilizes the ancillary tree labels we have assigned earlier. At each step of the traversal, the algorithm will write down a (partial) term or term instruction. We will expect the traversal to be terminating (i.e. to not get stuck in a loop) and complete without repeats (i.e. to have passed through all nodes and edges exactly once), and the term it has transcribed at its last step to be well-formed, at which point we will happily claim the proof structure to indeed be a proof net.

Traversing ILL$_{\multimap}$ Nets Now, let's get our crayons out and sketch the outline of the traversal algorithm. We will have two traversal modes: negative (or upward) mode and positive (or downward) mode. In negative mode, we move

upwards along negative nodes. When encountering a negative implication, we will create a λ abstraction over the variable specified by the dashed edge of that implication. When encountering a negative leaf, we will traverse across the axiom link to its positive counterpart and switch to positive mode. In positive mode, we move downwards along positive nodes. When we encounter a positive implication, we will add its negative (upward) branch to our mental stack and proceed downwards. Upon running out of positive nodes to visit, we will write down the variable label assigned to the positive tree's root, and then perform a negative traversal of each of the negative branches that live in our stack (i.e. we have encountered going down), in reverse order. We will start from the root of the formula tree occurring right of the turnstile (we know which one that is by the fact it has no variable label underneath it) in negative mode.

In our case, this is a negative implication, the dashed line of which reads x_2, so we start by writing down $\lambda x_2.(\dots)$. We move on to C^-, which is a leaf, so we cross over to C^+ and switch to positive mode. Going down, we encounter an implication as the positive root, and write down the positive tree's name, getting $\lambda x_2.x_1(\dots)$. We proceed in negatative mode to B^-, cross over to B^+ and repeat the above, getting $\lambda x_2.x_1\ (x_0\ \dots)$ before going into negative mode again. The final axiom link points us to A^+, which is its own root, named x_2. At this point, our traversal has transcribed the term $\lambda x_2.x_1\ (x_0\ x_2)$ and we have ran out of paths to explore. By now, all our nodes and edges have been visited, and our final term is both well-formed and identical to the one prescribed by the natural deduction proof of Figure I.8: a joyful outcome! If we consider ourselves bound to the permutation of assumptions dictated by the variable indices, the only thing we would need to do going backwards is guess the presence (and position) of the ex rule.

Axiom Links and Where to Find Them It would be understandable if at this point we allowed ourselves a feeling of complacency; navigating a proof net is no small feat, after all. But upon careful inspection, you might realize that you have been tricked. There never was any room for error in transitioning from the proof frame of Figure I.9a to the proof structure of Figure I.9b. A rare and lucky coindidence, or perhaps the result of carefully planned concealment? Regardless of what it might have been, we cannot reasonably anticipate there to always be a single possible set of axiom links, so we need a decision procedure that tells us how to actually extract them from a less forgiving proof.

Let's revisit the natural deduction proof of Figure I.8. This time around, we will explicate polarity information, and put an identifying index to each atomic formula occurrence at its axiom leaves; the turnstile mirrors indices faithfully, but inverts atomic polarities. Going bottom up through Figure I.10, every encounter of an implication elimination allows us (i) to identify the set of indices coming from the functor's negative part with the set of indices provided by the counterpart positive argument and (ii) to propagate the negative

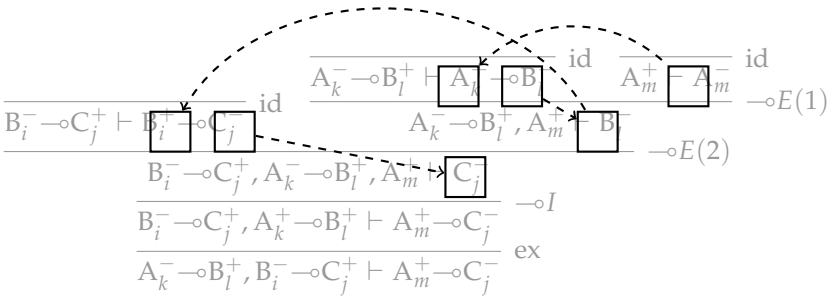


Figure I.10: Extracting axiom links from the proof of Figure I.8.

indices of the functor's remainder to the succedent of the next conclusion. That is, the first elimination $\multimap E(1)$ creates the identification $\{k \leftrightarrow m\}$, and propagates the positive leftover B_l^- to the next proof step, whereas the next elimination $\multimap E(2)$ identifies $\{i \leftrightarrow l\}$ and propagates C_j^- downwards. Upon reaching the proof's conclusion, we get to merge all identifications established along the proof[1], which can then be applied as a mapping (e.g. to the lexicographically first element) yielding a link-decorate judgement, in our case:

$$A_k^- \multimap B_i^+, B_i^- \multimap C_j^+ \vdash A_k^+ \multimap C_j^-$$

Matching indices correspond exactly to the axiom links of Figure I.9b – the two representations are in fact equivalent. Now, we really do know how to freely move back and forth between the proof net and natural deduction presentation of proofs in $\mathbf{ILL}_{\multimap}$.

The question then is, when should we use which? The original intention of proof nets was to provide a compact, bureacracy-free representation of proofs that abstracts away from structural rules. In that sense, their strength is also their weakness; same as the λ-terms they prescribe, they encode the semantically essential part of a proof, but hide structural subtleties that can prove hard to guess or recover. At the same time, performing search over proof nets is a horrible idea; the number of possible links we need to consider scales factorially with respect to the number of atoms in the proof frame, and checking

[1] Let's avoid any misteps here. The joining described is *not* set-theoretic union. Rather, we first take the set-theoretic union, and then iteratively reduce the set by conflating all identifications that agree in at least one element, up to a fixpoint. For instance, joining $\{i \leftrightarrow k\}$ and $\{l \leftrightarrow i, j \leftrightarrow m\}$ yields $\{i \leftrightarrow k \leftrightarrow l, j \leftrightarrow m\}$. Such a situation could arise for example when attempting to find the axiom links of non β normal proofs, like

$$\left(\lambda x_0.x_0^{A_i \multimap B_j}\ x_1^{A_k}\right)\ x_2^{A_l \multimap B_m}$$

The added burden has the enormous benefit of yielding "β normalized" links and resolving potential future headaches a priori.

whether a set of links is valid is in the best case linear. Due to these limitations, proof nets were envisaged as a compiled form of an existing proof, rather than a canvas to find that proof on. We will not see proof nets again for a while, but we will keep their memory warm in our hearts. Because when we do, we will challenge this perception and see how their parallel nature can actually be very convenient for proof search. Until then, we can temporarily store them in our mental backlog.

3 Lambek Calculi

3.1 Dropping Commutativity

There is only one structural rule left[1]: it is time for ex to go. Dropping ex makes the structures of our logic non-commutative. The transition, however, requires some care. If we were to naively go about our business using the inherited **ILL** connectives, we would soon stumble upon a pitfall. Recalling the shape of the $\multimap E$ rule, we come to the realization that functions carried over to this new logic are suddenly picky; they can only be applied to arguments to their right. This should raise some flags: a directionally flavored version of the implication is not bad in itself, but the presence of just such one such version is – where shall we look for the left-biased one? The answer is simple: the conflation between the two directions was natural, up until a moment ago; having them both would not amount to much, since by ex they would be interderivable. With ex removed, the veil is lifted and we can now see this clearly: there were always two implications, except disguised by the same symbol! Let us do our newfound friend justice, and make this distinction explicit.

The logic that provides us with the tools to accomplish this is due to Jim Lambek [Lambek, 1958], and has come to by known as the Lambek Calculus **L**. At this point, the careful reader will notice a chronological inconsistency in our presentational tour: the Lambek Calculus predates Linear Logic! Despite the fact, it is in essence a *refinement* of its purely linear part – a substructural logic within a substructural logic – and our previous exposition makes us better equiped to appreciate it. With commutativity gone, the Lambek Calculus brings *order* to Linear Logic – in the literal sense – assumptions must now be used exactly in the order they were instantiated. It also brings forth the notion of *adjacency*: structures joined by a rule are now immobile, and therefore obliged to remain adjacent from then on, unless broken apart by abstractions.

Formulas in the Lambek Calculus are generated by the grammar:

$$\mathrm{A}, \mathrm{B}, \mathrm{C} := p \mid \mathrm{A} \backslash \mathrm{B} \mid \mathrm{A} / \mathrm{B} \mid \mathrm{A} \otimes \mathrm{B} \tag{I.16}$$

The rules of this fragment are presented in Figure I.11. Alternative presentations can include additive conjunction and/or either of the disjunctions, but

[1] Or is there?

$$\frac{}{x_i : A \vdash x_i : A}\ \mathrm{id}$$

$$\frac{\Gamma \vdash s : B/A \quad \Delta \vdash t : A}{\Gamma, \Delta \vdash s \triangleleft t : B}\ /E \qquad \frac{\Gamma, x_i : A \vdash s : B}{\Gamma \vdash \lambda x_i.s : B/A}\ /I$$

$$\frac{\Gamma \vdash s : A \quad \Delta \vdash t : A\backslash B}{\Gamma, \Delta \vdash s \triangleright t : B}\ \backslash E \qquad \frac{x_i : A, \Gamma \vdash s : B}{\Gamma \vdash \lambda x_i.s : A\backslash B}\ \backslash I$$

$$\frac{\Gamma \vdash s : A \otimes B \quad \Delta[\![x_i : A, x_j : B]\!] \vdash t : C}{\Delta[\![\Gamma]\!] \vdash \mathrm{case}\ s\ \mathrm{of}\ (x_i, x_j)\ \mathrm{in}\ t : C}\ \otimes E \qquad \frac{\Gamma \vdash s : A \quad \Delta \vdash t : B}{\Gamma, \Delta \vdash (s, t) : A \otimes B}\ \otimes I$$

Figure I.11: Lambek Calculus **L**.

the key feature of interest lies in the two implications, / and \. The intuitive way of reading those is as directed fractionals, the formula hidden under the cover of the slash being the divisor, and the formula lying on it the dividend. The elimination rule $/E$ (resp. $\backslash E$) can then be read as fractional simplifications, whereby right (resp. left) multiplication by the divisor cancels out the division as a whole. An analogus reading can be attributed to the introduction rules, them now being the instantiation of a division by withdrawing items from the left or right of the assumption sequence (it might be helpful to think of $/I$ as dequeuing and $\backslash I$ as popping from the assumptions in the premise). The division paradigm is of pedagogical utility only, and we will not take it any further for fear of (incorrectly) hinting at other properties of fractionals being applicable in the logic. A noteworthy change of notation appears in the elimination of the product: with $\Delta[\![\Gamma]\!]$ we denote a structure Δ containing substructure Γ: $\Delta[\![\_]\!]$ now serves as a *context*, i.e. a structure of assumptions with a hole. The rule now claims it is acceptable to *replace* substructure A, B in Δ by Γ, if $\Gamma \vdash A \otimes B$ holds. The notions of structure and substructure depend of course on the logic used – in the current setting, Δ is a sequence, to which Γ is a subsequence. The reformulation of the rule is necessary to arbitrate elimination of nested products, since their extraction to the right or left edge of an assumption sequence is no longer possible. This also serves to better illustrate a remark made earlier: the rule can be applied at arbitrary nesting depths, each position corresponding to a supposedly different proof (consider for instance that if $\Delta[\![\Gamma]\!]$, and $\Gamma[\![A, B]\!]$, then it is also the case that $\Delta[\![A, B]\!]$).

The Lambek Calculus hails from an intuitionistic tradition, and is thus amenable to a propositions as types interpretation [Wansing, 1990]. Addorning its rules with faithful term rewrites translates into a type system that is

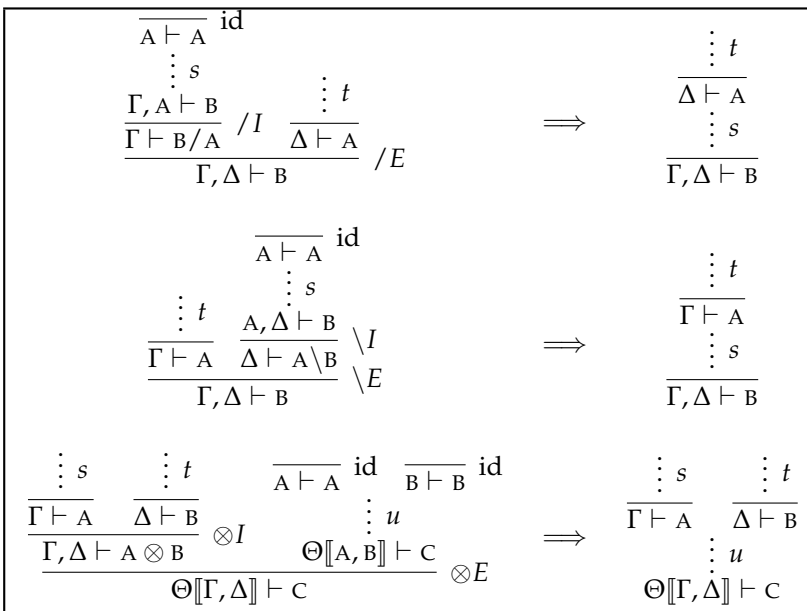


Figure I.12: Lambek β redexes.

both linear and ordered [Pierce, 2004]. Things get funky there: we now have two distinct modes of function application and λ abstraction, each pair with its own reduction. We use $\triangleleft$ and $\triangleright$ to denote right and left application, respectively – the mnemonic is that the triangle points to the function – and λ and $\curlywedge$ to denote the two kinds of anonymous functions.

3.1.1 Proof & Term Reductions

The proof reductions of Figure I.12 should be at this point straightforward to decode. The only addition is the symmetric version of the familiar implicational redex. For the redex of the product, the substituted A and B hypotheses are now wrapped on both sides by a context Θ, following the formulation of $\otimes E$. The corresponding term reductions are:

$$(\lambda x_i.s)\ t \overset{\beta}{\rightsquigarrow} s_{[x_i \mapsto t]} \tag{I.17}$$

$$t\ (\curlywedge x_i.s) \overset{\beta}{\rightsquigarrow} s_{[x_i \mapsto t]} \tag{I.18}$$

$$\text{case } (s, t) \text{ of } (x_i, x_j) \text{ in } u \overset{\beta}{\rightsquigarrow} u_{[s \mapsto x_i, t \mapsto x_j]} \tag{I.19}$$

3.2 Dropping Associativity

Judging by the apparent absence of any more structural rules to remove, someone eager to be done with the whole story could at this point proclaim our substructural tour finished. We are not quite done yet, however, for one last structural equivalence still remains unchecked (one we have made extensive use of, for that matter). The culprit can be found by going back to our original definition of structures in the long and distant past of Subsection 1.1 – by treating them as sequences, we have mindlessly equipped them with *associativity* for free, the use of which we never made explicit. The one to notice was Lambek once more [Lambek, 1961]. In the new logic (pragmatically named the non-associative Lambek Calculus **NL**) the definition of a structure changes to:

$$\Gamma, \Delta, \Theta := \ \mathrm{A} \mid (\Gamma \cdot \Delta) \tag{I.20}$$

i.e. the structural unit of the empty sequence is no more, and the scope of the binary structural binder is made explicit with brackets (we use the distinct symbol $\cdot$ to tell this new structural binder apart from its associative sibling). On top of adjacency and order, the non-associative Lambek Calculus further considers *constituency*; structures are now binary trees, with atomic propositions as their leaves and $\cdot$ as branching nodes, and judgements are differentiated on the basis of the binary branching form their assumptions take. Formulas remain as they were, but the presentation of the rules changes to that of Figure I.13 in order to accommodate the new, stricter structures. Merging structures Γ and Δ via $\backslash E$, $/E$ or $\otimes I$ is translated to building up a tree with the two as branches. Decomposing a structure via an abstraction $\backslash I$ or $/I$ now requires that the formula abstracted over occurs not just at the edge of the tree's linear projection, but also at its top-most branching level. The notation $\Gamma[\![\Delta]\!]$ now denotes that Δ is a *subtree* of Γ – for the product elimination $\otimes E$ to be applicable, A and B need not just be adjacent, but also commonly rooted.

The syntax of the isomorphic λ-calculus is identical to before, except this time we use $\blacktriangleleft$ and $\blacktriangleright$ to notationally differentiate with the non-associative application (not unlike how we replaced the intuitionistic implication $\rightarrow$ with its linear counterpart $\multimap$ earlier). The new structural constraint on the introduction of a directed implication can be intuititively translated to a constraint on the applicability of an abstraction. Namely, the variable to abstract over needs to occur at the top-most level of a function application in the term's inductive body.[1]

3.2.1 Proof & Term Reductions

Proof & term reductions are notationally identical to those of the previous subsection, modulo bracketing, and substituting white for black triangles. I

[1] Abusing terminology, here by inductive body we mean the term itself (if it's an implicative one), the term's inner body (if it's a sequence of abstractions), the left or right coordinate (if it's a product), or the nested body on which substitution is performed (if it's a deconstructed product).

$$\frac{}{x_i : A \vdash x_i : A}\ \mathrm{id}$$

$$\frac{\Gamma \vdash s : B/A \quad \Delta \vdash t : A}{(\Gamma \cdot \Delta) \vdash s \blacktriangleleft t : B}\ /E \qquad \frac{(\Gamma \cdot x_i : A) \vdash s : B}{\Gamma \vdash \lambda x_i.s : B/A}\ /I$$

$$\frac{\Gamma \vdash s : A \quad \Delta \vdash t : A\backslash B}{(\Gamma \cdot \Delta) \vdash s \blacktriangleright t : B}\ \backslash E \qquad \frac{(x_i : A \cdot \Gamma) \vdash s : B}{\Gamma \vdash \curlywedge x_i.s : A\backslash B}\ \backslash I$$

$$\frac{\Gamma \vdash s : A \otimes B \quad \Delta[\![(x_i : A, x_j : B)]\!] \vdash t : C}{\Delta[\![\Gamma]\!] \vdash \mathsf{case}\ s\ \mathsf{of}\ (x_i, x_j)\ \mathsf{in}\ t : C}\ \otimes E \qquad \frac{\Gamma \vdash s : A \quad \Delta \vdash t : B}{\Gamma, \Delta \vdash (s, t) : A \otimes B}\ \otimes I$$

Figure I.13: Non-associative Lambek Calculus **NL**.

trust the missing picture is easy enough to create mentally.

3.3 The Full Landscape

We have seen **NL** as a refinement of **L**, and **L**, in turn, as a refinement of **ILL**. The three can be perceived as points in a lattice of substructural logics, upon which we can move by adding or removing structural rules at a global level; this view lends **ILL** its alternative name **LP**, for the Lambek Calculus with permutation. At the top of the diamond we have **ILL**, where (linearity aside), anything goes, and at the bottom we have **NL**, where neither associativity nor commutativity hold. At the center, there's **L**, where only associativity holds. Next to it, an unexpected curiosity pops up: **NLP** (for the non-associative Lambek with permutation), an offbeat logic where associativity holds but commutativity doesn't – its structures are *mobiles*: orderless, binary branching trees that make no distinction between left and right daughters. Unlike its relatives, **NLP** has received limited attention from theorists and practitioners alike. This will still remain the case even after (if?) this manuscript sees the light of day, but its peculiar structures will reemerge and have their moment to shine later on.

4 Restoring Control

With every step we have taken further into substructuraland, we have been paying a price in expressivity; it is now time for us to acknowledge the accumulated bill. Dropping contr and weak made us resource conscious, but theorems of **IL** that required resource duplication or erasure became underivable. Dropping ex forced us to pay attention to the order of assumptions, but

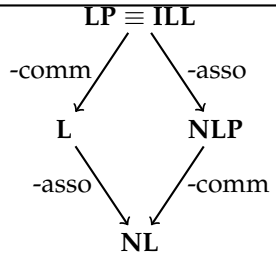


Figure I.14: **(N)L(P)**: **ILL** and substructural friends.

costed us access to theorems that required permutation to derive. Substituting the structural comma $\_, \_$ with the non-associative $(\_ \cdot \_)$ casted our sequences to trees, this time at the expense of theorems that required rebracketing. Woe is us – is there even anything left we can derive?

Perhaps this is painting an overly dramatic picture, considering that none of this is necessarily bad. From an epistemic perspective, the less structural equivalences we take for granted, the better our mental grasp of structural difference becomes. In the best case, if it just so happens that the kind of structures we want to investigate overlaps *fully* with the kind of structures our logic can explicitly reason about, the distinction between theorem and non-theorem becomes a refinement rather than a loss of expressivity. From a more pragmatic perspective, more structural constraints means easier proof search, and less theorems means faster exhaustion of possibilities. To make the scale of the combinatorics tangible, reflect for a second on this. A single judgement of n hypotheses in **NL** is but one of the Catalan number of bracketings $C(n)$ it would be syntactically undistinguishable from in **L**, each one of which in turn is but one of the factorially many permutations $n!$ it would be equivalent to in **LP**.[1] The point to take home is that proof search becomes decidedly easier in the absence of syntactic equivalences, so perhaps a double-edged sword would have made for a better analogy than a bill.

The defeatist attitude here would be to just accept the trade-off between expressivity and complexity, weep for the theorems forever lost, take our victory and walk away. The problem lies however in the common occasion where the structure of objects under scrutiny overlaps only *partially* with a specific substructural flavor, modulo some exceptional but real cases that require added expressivity. In such a scenario, taking a step up in the hierarchy would cause an undesirable combinatorial explosion, whereas staying put would sacrifice our ability to argue about these exceptional cases. By contrast, the maximalist attitude makes no concessions and seeks both for the cake to be whole and the dog to be fed.[2] What if there was a way to keep our logic computationally

[1] Boom, goes the combinatorial explosion.

[2] Direct translation of a silly but fitting Greek aphorism that won by a small margin over the Italian equivalent (wine barrel full and wife drunk). In any case, cake is bad for dogs.

$$\frac{\Gamma \vdash s : \Box A}{\langle \Gamma \rangle \vdash \blacktriangledown s : A}\ \Box E \qquad \frac{\langle \Gamma \rangle \vdash s : A}{\Gamma \vdash \blacktriangle s : \Box A}\ \Box I$$

$$\frac{\Gamma[\![\langle x_i : A \rangle]\!] \vdash s : B \quad \Delta \vdash t : \Diamond A}{\Gamma[\![\Delta]\!] \vdash \text{case } \triangledown t \text{ of } x_i \text{ in } s : B}\ \Diamond E \qquad \frac{\Gamma \vdash s : A}{\langle \Gamma \rangle \vdash \triangle s : \Diamond A}\ \Diamond I$$

Figure I.15: Logical rules of modal inference.

tractable but with temporary and on-demand access to normally excluded reasoning tools?

4.1 The Logic of Modalities

The answer comes in the form of unary *modalities*, type-forming operators lent from modal logics, that allow navigation between logics of different structural properties. Unary modalities hold a key role in the presentation of full linear logic; there, a single operator ! (called *bang*) would allow an embedding of intuitionistic (non-linear) propositions into the linear regime, essentially acting as a licensor of contr and weak. In our case, we will make do with two modalities from temporal logic, the diamond $\Diamond$ and the box $\Box$.

The two form a residuated pair, the properties of which can be formulated either (i) in the form of a type-level biconditional derivability relation:

$$\Diamond A \vdash B \text{ iff } A \vdash \Box B \tag{I.21}$$

or (ii) the monotonic behavior of its parts:

$$A \vdash B \implies \Diamond A \vdash \Diamond B \tag{I.22}$$
$$A \vdash B \implies \Box A \vdash \Box B \tag{I.23}$$

and the adjointness of their compositions, where $\Diamond\Box(\_)$ is an *interior* and $\Box\Diamond(\_)$ a *closure* operator:

$$\Gamma \vdash A \implies \Gamma \vdash \Box\Diamond A \tag{I.24}$$
$$\Gamma \vdash \Diamond\Box A \implies \Gamma \vdash A \tag{I.25}$$

The logical manipulation of these modalities is handled by corresponding elimination and introduction rules, presented in Figure I.15. The presentation is intentionally detached from a specific subtructural strand – modalities are plug-and-play to any member of the **(N)L(P)** family. Their incorporation adds a new kind of structure to the ones provided by the underlying logic, altering judgements accordingly:

$$\Gamma, \Delta, \Theta := \ldots \mid \langle \Gamma \rangle \tag{I.26}$$

Angular brackets denote unary tree branches that behave slightly differ-

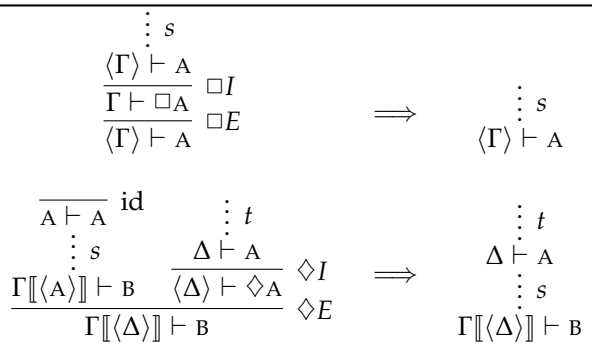


Figure I.16: Modal β redexes.

ent to the rest; they act as an impenetrable barrier that permits or hinders the introduction or elimination of modal connectives in a judgement. The box elimination rule $\Box E$ grants us the option of removing a logical box from the succedent of the premise (as long as it is its main connective), but encloses the premises in angular brackets in the process. Its introduction counterpart $\Box I$ does the exact opposite: it frees a judgement's assumptions from their brackets, but puts the succedent proposition under the scope of a box. The diamond behaves just the other way around. Its introduction rule $\Diamond I$ is straightforward: it offers the possibility of putting the succedent under the scope of a diamond, in exchange wrapping the antecedents with brackets. The elimination rule $\Diamond E$ is more of a problem child, behaving akin to a unary product. Without locality restrictions, it inspects a proof of B, the assumptions of which contain a substructure $\langle \mathrm{A} \rangle$ within context $\Gamma[\![\_]\!]$, and allows the post-hoc substitution of the hypothesis together with its brackets by a structure Δ, if from it one can derive $\Diamond \mathrm{A}$.

Rules are adorned with term rewrite instructions in the propositions as types style, similar to how temporal logic can be operationalized in the λ-calculus [Wansing, 2002]. The mnemonic is now two-dimensional: upward triangles denote introduction and downward ones elimination, whereas black triangles are for the box, white ones for the diamond. Term constructions for the single-premise rules are uncomplicated: each type operation just leaves the corresponding term footprint. This is not the case for the $\Diamond E$ rule, which requires some attention: the structural substitution of $\langle \mathrm{A} \rangle$ for Δ necessitates a case construct that calls for a term substitution of the variable x_i for $\triangledown \mathsf{t}$. Note that the free variables of the resulting expression (case $\triangledown \mathsf{t}$ of x_i in s) are the union of the free variables of t and those of s except for x_i, which becomes *bound* by the case construct.

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, s\\ \Gamma \vdash \Box \mathrm{A}\end{array}}{\langle \Gamma \rangle \vdash \mathrm{A}}\,\Box E}{\Gamma \vdash \Box \mathrm{A}}\,\Box I \quad \equiv \quad \begin{array}{c}\vdots\, s\\ \Gamma \vdash \Box \mathrm{A}\end{array}$$

$$\dfrac{\dfrac{\dfrac{}{\mathrm{A} \vdash \mathrm{A}}\,\mathrm{id}}{\langle \mathrm{A} \rangle \vdash \Diamond A}\,\Diamond I \qquad \begin{array}{c}\vdots\, s\\ \Delta \vdash \Diamond \mathrm{A}\end{array}}{\Delta \vdash \Diamond \mathrm{A}}\,\Diamond E \quad \equiv \quad \begin{array}{c}\vdots\, s\\ \Delta \vdash \Diamond \mathrm{A}\end{array}$$

Figure I.17: Modal η redexes.

4.1.1 Proof & Term Reductions

The proof patterns of Figure I.16 exhibit introduction elimination chains of modal operators, and thus constitute β redexes subject to normalization. The first one is trivial: it just says that a sequential application of $\Box I$ followed by $\Box E$ can be safely excised. The second one proposes that if a $\Diamond I$ is the last rule to have been applied on the substitution branch t of the $\Diamond E$ rule, it would make sense to simply plug proof t in place of the proposition A hypothesized in the other branch s. On the term level, these rules correspond to computations:

$$\blacktriangledown\blacktriangle \mathsf{s} \overset{\beta}{\rightsquigarrow} \mathsf{s} \tag{I.27}$$

$$\mathsf{case}\ \triangledown\triangle \mathsf{t}\ \mathsf{of}\ \mathsf{x}\ \mathsf{in}\ \mathsf{s} \overset{\beta}{\rightsquigarrow} \mathsf{s}_{[\mathsf{x} \mapsto \mathsf{t}]} \tag{I.28}$$

The dual direction of η equivalences also holds – since these are not as frequently encountered as the more pedestrian implication and product equivalences, we explictly present them in Figure I.17. The term equivalences they materialize are:

$$\blacktriangle\blacktriangledown \mathsf{s} \overset{\eta}{\equiv} \mathsf{s} \tag{I.29}$$

$$\mathsf{case}\ \triangledown \mathsf{s}\ \mathsf{of}\ \mathsf{x}\ \mathsf{in}\ \triangle \mathsf{x} \overset{\eta}{\equiv} \mathsf{s} \tag{I.30}$$

4.1.2 A Digression on Modal Terms

For the modally savvy, the term rewrites attributed to the modal rules might seem unorthodox. A more common presentation employs the simpler metasyntax notation of term substitution. For instance, $\Diamond E$ can often be spotted in the wild as:

$$\dfrac{\Gamma[\![\langle \mathsf{x}_i : \mathrm{A} \rangle]\!] \vdash \mathsf{s} : \mathrm{B} \quad \Delta \vdash \mathsf{t} : \Diamond \mathrm{A}}{\Gamma[\![\Delta]\!] \vdash \mathsf{s}_{[\mathsf{x}_i \mapsto \triangledown \mathsf{t}]} : \mathrm{B}}\,\Diamond E$$

In this disguise, the rule is again seen as realizing a retroactive substitution of x_i with $\triangledown t$, except this time around the substitution is *actually* performed, resulting in less cumbersome terms being carried around.

Opting for this alternative notation has, however, a number of negative consequences. The more superficial one is that the main term connective does not take scope at the outermost layer of rule's yield, but rather nested arbitrarily deeply within it, unlike its better behaved version. From a proof-theoretic perspective, normalization is now baked directly into the theory, as the term yield of the rule exactly coincides with its β reduced form. At the same time, all rule permutations boil down to having the exact same reduction, i.e. multiple previously distinct terms are conflated into a single representation. This establishes an impicit syntactic equivalence on proofs that claims that the exact position of the $\Diamond E$ rule is *syntactically irrelevant* (so long of course as the same variable x_i is substituted by the same term $\triangledown t$). Finally, the shorthand version hides variables; hypotheses that would be bound by the case construct are instead erased and forgotten, obfuscating the term-to-proof correspondence. All these are perhaps minor points not worth taking too seriously, but for one concerned with concrete implementation the extra merit of notational simplicity comes at the cost of equality checking become way more tedious. With this in mind (and in a rare moment of excessive formal zeal), we will exercise some self restraint and avoid indulging in the convenience of this version.

4.1.3 Properties

Situating our unary operators within the modal logic zoo is no trivial endeavour (especially if you, like me, have had limited exposure to it before). They are best characterized by the properties they satisfy, so inspecting them should shed some light on their proof-theoretic behavior (as a bonus, it will also help us get better acquainted with the kind of term rewrites their rules prescribe). Figure I.18 presents the proof transformations equivalent to the properties foretold: (a) and (b) for monotonicity, (c) and (d) for composition, and (e) and (f) for the two directions of the residuation law.

Worth a special mention are also the so-called triple laws:

$$\Diamond A \dashv\vdash \Diamond\Box\Diamond A \tag{I.31}$$

$$\Box A \dashv\vdash \Box\Diamond\Box A \tag{I.32}$$

which can be intuitively read as claiming that prepending an already modal type with (one or more) diamond-box pairs in alteration has no real effect, as these can unconditionally cancel out or be expanded into. Figure I.19 presents proofs of the above in both directions.

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, s\\ \mathsf{x}_i : \mathrm{A} \vdash \mathsf{s} : \mathrm{B}\end{array}}{\langle \mathsf{x}_i : \mathrm{A}\rangle \vdash \triangle \mathsf{s} : \Diamond \mathrm{B}}\,\Diamond I \qquad \dfrac{}{\mathsf{x}_j : \Diamond \mathrm{A} \vdash \mathsf{x}_j : \Diamond \mathrm{A}}\,\mathrm{id}}{\mathsf{x}_j : \Diamond \mathrm{A} \vdash \mathrm{case}\ \triangledown \mathsf{x}_j\ \mathrm{of}\ \mathsf{x}_i\ \mathrm{in}\ \mathsf{s} : \Diamond \mathrm{B}}\,\Diamond E$$

(a) Monotonicity of the diamond.

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\, s\\ \mathsf{x}_i : \mathrm{A} \vdash \mathsf{s} : \mathrm{B}\end{array}}{\vdash \lambda \mathsf{x}_i.\mathsf{s} : \mathrm{A} \multimap \mathrm{B}}\,\multimap I \qquad \dfrac{\dfrac{}{\mathsf{x}_j : \Box \mathrm{A} \vdash \mathsf{x}_j : \Box \mathrm{A}}\,\mathrm{id}}{\langle \mathsf{x}_j : \Box \mathrm{A}\rangle \vdash \blacktriangledown \mathsf{x}_j : \mathrm{A}}\,\Box E}{\langle \mathsf{x}_j : \Box \mathrm{A}\rangle \vdash (\lambda \mathsf{x}_i.\mathsf{s})\ \blacktriangledown \mathsf{x}_j : \mathrm{B}}\,\multimap E}{\mathsf{x}_j : \Box \mathrm{A} \vdash \blacktriangle((\lambda \mathsf{x}_i.\mathsf{s})\ \blacktriangledown \mathsf{x}_j : \Box \mathrm{B}}\,\Box I$$

(b) Monotonicity of the box.

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, s\\ \Gamma \vdash \mathsf{s} : \mathrm{A}\end{array}}{\langle \Gamma\rangle \vdash \triangle \mathsf{s} : \Diamond \mathrm{A}}\,\Diamond I}{\Gamma \vdash \blacktriangle \triangle \mathsf{s} : \Box \Diamond \mathrm{A}}\,\Box I$$

(c) The closure $\Diamond\Box(\_)$.

$$\dfrac{\dfrac{\dfrac{}{\mathsf{x}_i : \Box \mathrm{A} \vdash \mathsf{x}_i : \Box \mathrm{A}}\,\mathrm{id}}{\langle \mathsf{x}_i : \Box \mathrm{A}\rangle \vdash \blacktriangledown \mathsf{x}_i : \mathrm{A}}\,\Box E \qquad \begin{array}{c}\vdots\, s\\ \Gamma \vdash \mathsf{s} : \Diamond \Box \mathrm{A}\end{array}}{\Gamma \vdash \mathrm{case}\ \triangledown \mathsf{s}\ \mathrm{of}\ \mathsf{x}_i\ \mathrm{in}\ \blacktriangledown \mathsf{x}_i : \mathrm{A}}\,\Diamond E$$

(d) And the interior $\Box\Diamond(\_)$.

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, s\\ \mathsf{x}_i : \mathrm{A} \vdash \mathsf{s} : \Box \mathrm{B}\end{array}}{\langle \mathsf{x}_i : \mathrm{A}\rangle \vdash \blacktriangledown \mathsf{s} : \mathrm{B}}\,\Box E \qquad \dfrac{}{\mathsf{x}_j : \Diamond \mathrm{A} \vdash \mathsf{x}_j : \Diamond \mathrm{A}}\,\mathrm{id}}{\mathsf{x}_j : \Diamond \mathrm{A} \vdash \mathrm{case}\ \triangledown \mathsf{x}_j\ \mathrm{of}\ \mathsf{x}_i\ \mathrm{in}\ \mathsf{s} : \mathrm{B}}\,\Diamond E$$

(e) Residuation law: from $\mathrm{A} \vdash \Box \mathrm{B}$ to $\Diamond \mathrm{A} \vdash \mathrm{B}$.

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\, s\\ \mathsf{x}_i : \Diamond \mathrm{A} \vdash \mathsf{s} : \mathrm{B}\end{array}}{\vdash \lambda \mathsf{x}_i.\mathsf{s} : \Diamond \mathrm{A} \multimap \mathrm{B}}\,\multimap I \qquad \dfrac{\dfrac{}{\mathsf{x}_j : \mathrm{A} \vdash \mathsf{x}_j : \mathrm{A}}\,\mathrm{id}}{\langle \mathsf{x}_j : \mathrm{A}\rangle \vdash \triangle \mathsf{x}_j : \Diamond \mathrm{A}}\,\Diamond I}{\langle \mathsf{x}_j : \mathrm{A}\rangle \vdash \mathrm{case}\ \triangledown \triangle \mathsf{x}_j\ \mathrm{of}\ \mathsf{x}_i\ \mathrm{in}\ \lambda \mathsf{x}_i.\mathsf{s} : \mathrm{B}}\,\Diamond E}{\mathsf{x}_j : \Diamond \mathrm{A} \vdash \blacktriangle(\mathrm{case}\ \triangledown \triangle \mathsf{x}_j\ \mathrm{of}\ \mathsf{x}_i\ \mathrm{in}\ \lambda \mathsf{x}_i.\mathsf{s}) : \mathrm{B}}\,\Box I$$

(f) Ditto, the other way around.

Figure I.18: Derivations for the various aspects of residuation.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x_i : \Box A \vdash x_i : \Box A}\,\mathrm{id}}{\langle x_i : \Box A\rangle \vdash \blacktriangledown x_i : A}\,\Box E \quad \dfrac{\dfrac{}{x_j : \Box\Diamond\Box A \vdash x_j : \Box\Diamond\Box A}\,\mathrm{id}}{\langle x_j : \Box\Diamond\Box A\rangle \vdash \blacktriangledown x_j : \Diamond\Box A}\,\Box E}{\langle x_j : \Box\Diamond A\rangle \vdash \text{case } \triangledown\blacktriangledown x_j \text{ of } x_i \text{ in } \blacktriangledown x_i : A}\,\Diamond E}{x_j : \Box\Diamond\Box A \vdash \blacktriangle(\text{case } \triangledown\blacktriangledown x_j \text{ of } x_i \text{ in } \blacktriangledown x_i) : \Box A}\,\Box I$$

(a) Contraction of $\Box\Diamond\Box(\_)$ to $\Box(\_)$.

$$\dfrac{\dfrac{\dfrac{}{x_i : \Box A \vdash x_i : \Box A}\,\mathrm{id}}{\langle x_i : \Box A\rangle \vdash \triangle x_i : \Diamond\Box A}\,\Diamond I}{x_i : \Box A \vdash \blacktriangle\triangle x_i : \Box\Diamond\Box A}\,\Box I$$

(b) Expansion of $\Box(\_)$ to $\Box\Diamond\Box(\_)$.

$$\dfrac{\dfrac{\dfrac{}{x_i : \Box\Diamond A \vdash x_i : \Box\Diamond A}\,\mathrm{id}}{\langle x_i : \Box\Diamond A\rangle \vdash \blacktriangledown x_i : \Diamond A}\,\Box E \quad \dfrac{}{x_j : \Diamond\Box\Diamond A \vdash x_j : \Diamond\Box\Diamond A}\,\mathrm{id}}{x_j : \Diamond\Box\Diamond A \vdash \text{case } \triangledown x_j \text{ of } x_i \text{ in } \blacktriangledown x_i : \Diamond A}\,\Diamond E$$

(c) Contraction of $\Diamond\Box\Diamond(\_)$ to $\Diamond(\_)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x_i : A \vdash \blacktriangle\triangle x_i : \Box\Diamond A}\,(\mathrm{I.18c})}{\langle x_i : A\rangle \vdash \triangle\blacktriangle\triangle x_i : \Diamond\Box\Diamond A}\,\Diamond I \quad \dfrac{}{x_j : \Diamond A \vdash x_j : \Diamond A}\,\mathrm{id}}{x_j : \Diamond A \vdash \text{case } \triangledown x_j \text{ of } x_i \text{ in } \triangle\blacktriangle\triangle x_i : \Diamond\Box\Diamond A}\,\Diamond E$$

(d) Expansion of $\Diamond(\_)$ to $\Diamond\Box\Diamond(\_)$.

Figure I.19: The triple laws for the two modalities in both directions.

4.2 Structural Reasoning

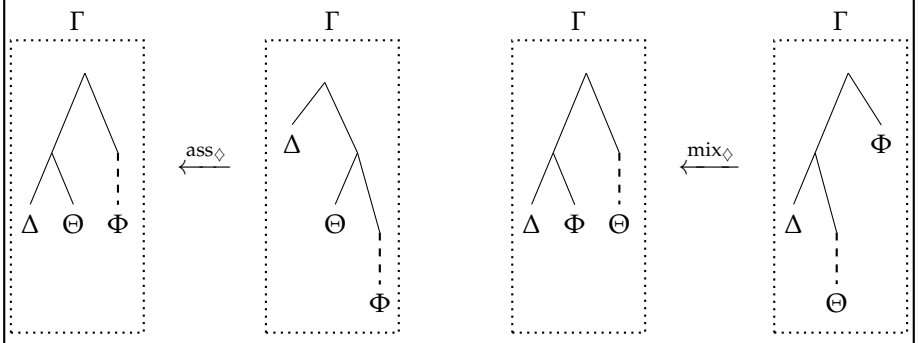
This detour may have proven lengthy, but has hopefully helped us acquire a first taste for modalities. We now know how to introduce and eliminate them and what the effect of doing so is on the antecedent structure, and got a first glimpse of their properties, the term rewrites they prescribe and the type inequalities (in the form of unidirectional derivations) they give rise to. The question then becomes how to actually use them for the task at hand, namely disciplined traversal between substructural logics. Structural reasoning is accomplished via structural postulates, rules of inference that enact commutativity and associativity (or combinations thereof), except in a controlled fashion. These are permissible only under strict conditions on the substructures constituent to the antecedent structure – this is exactly where the new kind of structures will prove useful. There is no fixed vocabulary of structural rules, as they are intended for application-specific finetuning of a universal logical core, so we are free to design and populate it according to our own needs. Prime examples and standard items for consideration include the controlled associativity and mixed associativity-commutativity rules of Figure I.20a (and the corresponding tree transformations of Figure I.20b, if you have a disdain for brackets). The first rule $\text{ass}_\Diamond$ allows a unary branch $\langle\Phi\rangle$ to escape its bind to its neighbour Θ, forcing it to associate to the structure Δ to its left instead. The second one $\text{mix}_\Diamond$ allows a unary $\langle\Theta\rangle$ to swap position with its right-adjacent neighbor Φ, disassociating from its left neighbour Δ in the process. In domains where even finer control is needed, one can consider indexed families of (possibly interacting) modalities, each with their own structural brackets and rulesets.

5 The Linguistic Perspective

Despite their presentation having intentionally been left vague and abstract, the ideas explored so far have been a keystone element of computer science, from its inception until recent modernity. Beyond that, they form the common theoretical underpinnings for the formal treatment of natural languages and their various aspects, where they manifest as so-called *Categorial Grammars*. Categorial grammars is a heavily overloaded term that refers to a wide and diverse family of related formalisms, each with its own ambitions, goals, strengths and weaknesses. The most encompassing way of defining a categorial grammar is thus best accomplished through a high-level intersection of their common points. A categorial grammar is tied to a (usually substructural) logic, commonly a choice from the ones reviewed so far. The choice of logic is part personal preference, but is usually motivated by the degree of alignment between the options under consideration and the characteristics of the target language – a factor that also comes into play is also the trade-off between expressivity and complexity. On the basis of the chosen logic, a categorial gram-

$$\frac{\Gamma[\![\Delta,(\Theta,\langle\Phi\rangle)]\!]\vdash A}{\Gamma[\![(\Delta,\Theta),\langle\Phi\rangle]\!]\vdash A}\ \mathrm{ass}_\Diamond \qquad \frac{\Gamma[\![(\Delta,\langle\Theta\rangle),\Phi]\!]\vdash A}{\Gamma[\![(\Delta,\Phi),\langle\Theta\rangle]\!]\vdash A}\ \mathrm{mix}_\Diamond$$

(a) In rule format.



(b) Corresponding tree transformations.

Figure I.20: Controlled associativity/commutativity.

mar has a *lexicon*; a mapping from primitive linguistic entries (i.e. words) to formulas of that logic. Their dependence on a lexicon grants categorial grammars their *strongly lexicalized* title – as the slogan goes, words carry their combinatorics on their sleeves. With these two components in hand, compiling composite structures for complex linguistic entries (i.e. parsing) becomes a process of formal deduction dictated by the interplay between the types of the participating atomic elements, and the rules of inference the logic is equipped with. Categorial grammars are a staple of the linguistic tradition and a focal point for practitioners, logicians and linguists alike. In this section we will examine some of their main strands, with a special emphasis on two spiritual progenitors of the unique flavor that is to be developed and presented later in this thesis.

5.1 Type-Logical Grammars

The earliest take at a categorial grammar are the AB grammars attributed to Kazimierz Adjukiewicz [Ajdukiewicz, 1935] and Yehoshua Bar-Hillel [Bar-Hillel, 1953], but it was Jim Lambek that raised the existing notation and operations into the glory of a fully-fledged type theory. In their original purpose as envisaged by Lambek, his calculi would find use as *grammar logics*, i.e. universal systems of *grammatical* computation – a perspective adopted and advanced into what has presently come to be known as type-logical grammars [Morrill, 1994; Moortgat, 1997, 2014]. In a natural language setting, the linear base of the Lambek calculi is naturally equated to the resource sensitivity of grammar:

Logic	Computer Science	Linguistics
Propositional Constant	Base Type	Syntactic Category
Inference Rule	Term Rewrite	Phrase Formation
Axiom	Variable	Word (or Empty Category)
Provability	Type Inhabitation	Grammaticality

Table I.3: The Curry-Howard correspondence applied in linguistics.

words play a single grammatical role in the phrases they help form – there's no ignoring or reusing items at will. There, the original Lambek calculus **L** would be the logic of *strings*; it can faithfully portray the generation of natural language utterances, where arbitrary reordering is a destructive process that ruins coherence. Its stricter version **NL** would instead be the logic of *constituency trees*; on top of word order, it further specifies constituency structure, allowing a distinction between different syntactic analyses of the same surface form. Type-logical grammars extend the Curry-Howard correspondence with a new axis, that of natural language; the transference of points of interest across that axis is presented in Table I.3.

To see this in action, let's consider first an instantiation of a Lambek Calculus **NL** with the set of primitive types Prop_0 populated with signs characterizing the grammatical role of a piece of text that can independently stand on its own (i.e. phrasal categories or, more crudely, parts of speech). In a toy fragment and for illustration purposes, this could look like:

$$\mathsf{Prop}_0 := \{\mathrm{N}, \mathrm{NP}, \mathrm{S}, \mathrm{PP}\}$$

for a grammar able to reason about nouns N, noun phrases and bare nouns NP, sentential clauses S, prepositional phrases PP and functions thereof in English. One might wonder: what happened to the remaining kinds of phrasal categories like verbs, adjectives and adverbs? These would indicate grammatical functions, and in fact should be represented as such. An intransitive phrase, for instance, is a grammatical function that would consume a left-adjacent noun phrase to produce a sentence, therefore it would materialize as $\mathrm{NP}\backslash\mathrm{S}$. It follows that a transitive phrase or copula would then be of type $(\mathrm{NP}\backslash\mathrm{S})/\mathrm{NP}$, a function that requires a right-adjacent noun phrase to produce an intransitive phrase, whereas a bitransitive, requiring two, would be $((\mathrm{NP}\backslash\mathrm{S})/\mathrm{NP})/\mathrm{NP}$, etc. In the same vein, determiner phrases consume right-adjacent nouns and lift them to noun phrases NP/N, whereas prenominal adjectives are noun phrase (or noun) endomorphisms modifying them but keeping their type intact, NP/NP (and the other way around for postnominal use). Adverbs would also be endomorphisms, except this time higher-order – $(\mathrm{NP}/\mathrm{NP})/(\mathrm{NP}/\mathrm{NP})$ for adjectival and $(\mathrm{NP}\backslash\mathrm{S})\backslash(\mathrm{NP}\backslash\mathrm{S})$ for verbal modification, respectively.

Linguistic reasoning is not done *ex nihilo* – formulas like the above are

eye	::	N
oceans, suns, deeps , dolphins		
sea-nymphs, whirlpools	::	NP
the	::	NP/N
opiate, strange, unrememberable, their	::	NP/NP
poured	::	ITV := NP\S
behold	::	TV := (NP\S)/NP
there	::	$ADV_{\backslash}$:= (NP\S)\(NP\S)
never	::	$ADV_{/}$:= (NP\S)/(NP\S))
litten	::	(NP\NP)/PP
may	::	AUX := (NP\S)/(NP\S))

Table I.4: Toy lovecraftian lexicon of pure Lambek types.

$$\dfrac{\dfrac{}{\text{strange} \vdash \text{NP/NP}}\ \text{lex} \quad \dfrac{}{\text{dolphins} \vdash \text{NP}}\ \text{lex}}{\text{strange} \cdot \text{dolphins} \vdash \text{NP}}\ /E$$

(a) Derivation for "*strange dolphins*".

$$\dfrac{\dfrac{}{\text{the} \vdash \text{NP/N}}\ \text{lex} \quad \dfrac{}{\text{eye} \vdash \text{N}}\ \text{lex}}{\text{the} \cdot \text{eye} \vdash \text{NP}}\ /E$$

(b) Derivation for "*the eye*".

$$\dfrac{\dfrac{}{\text{litten} : (\text{NP}\backslash\text{NP})/\text{PP}}\ \text{lex} \quad \dfrac{\dfrac{}{\text{by} : \text{PP/NP}}\ \text{lex} \quad \dfrac{}{\text{suns} : \text{NP}}\ \text{lex}}{\text{by} \cdot \text{suns} \vdash \text{PP}}\ /E}{\text{litten} \cdot (\text{by} \cdot \text{suns}) \vdash \text{NP}\backslash\text{NP}}\ /E$$

(c) Derivation for "*litten by suns*".

$$\dfrac{\dfrac{}{\text{sea-nymphs} : \text{NP}}\ \text{lex} \quad \dfrac{\dfrac{}{\text{of} : (\text{NP}\backslash\text{NP})/\text{NP}}\ \text{lex} \quad \dfrac{\dfrac{}{\text{unremeberable} : \text{NP/NP}}\ \text{lex} \quad \dfrac{}{\text{deeps} : \text{NP}}\ \text{lex}}{\text{unrememberable} \cdot \text{deeps} \vdash \text{NP}}\ /E}{\text{of} \cdot (\text{unrememberable} \cdot \text{deeps}) \vdash \text{NP}\backslash\text{NP}}\ /E}{\text{sea-nymphs} \cdot (\text{of} \cdot (\text{unrememberable} \cdot \text{deeps})) \vdash \text{NP}}\ \backslash E$$

(d) Derivation for "*sea-nymphs of unrememberable deeps*".

$$\dfrac{\dfrac{\dfrac{}{\text{opiate} : \text{NP/NP}}\ \text{lex} \quad \dfrac{}{\text{oceans} : \text{NP}}\ \text{lex}}{\text{opiate} \cdot \text{oceans} \vdash \text{NP}}\ /E \quad \dfrac{\dfrac{}{\text{poured} : \text{NP}\backslash\text{S}}\ \text{lex} \quad \dfrac{}{\text{there} : \text{ADV}_{\backslash}}\ \text{lex}}{\text{poured} \cdot \text{there} \vdash \text{NP}\backslash\text{S}}\ \backslash E}{(\text{opiate} \cdot \text{oceans}) \cdot (\text{poured} \cdot \text{there}) \vdash \text{S}}\ \backslash E$$

(e) Derivation for "*Opiate oceans poured there*".

Figure I.21: Deriving simple multiplicative phrases in **NL**.

supplied by and grounded in the lexicon. This does not exclude the option of utilizing hypotheticals instantiated by the axiom rule id – hypothetical reasoning lives, in fact, at the core of the type-logical inferential process, as we will soon see. It means, rather, that our building blocks will for the most part be *lexical constants*, proof objects that behave just like variables, except they are neither wantonly typed nor amenable to abstraction. To convey the difference between the two, we will instantiate the latter with a seemingly new rule of inference, lex, which simply performs lexical lookup, i.e. pulls a word's type from the lexicon.

The internet guide *how to write a dissertation* I am consulting insists it is important to set clear goals and stick to them. It seems like sound advice, so we are going to do just that, and attempt to demonstrate the analysis of a non-contrived example in the type-logical framework. The following looks like a fitting match:

> "*Opiate oceans poured there, litten by suns that the eye may never behold, and having in their whirlpools strange dolphins and sea-nymphs of unrememberable deeps.*"
>
> H.P. Lovecraft, *Azathoth* (1938). In *Leaves (2).*

Let's pave the way towards this ambitious goal with the miniature mock-up lexicon of Table I.4, and see just how far it can get us.

Figure I.21 presents derivations for parts of the goal phrase, and our very first linguistic examples (!) – the absence of abstractions should make them straightforward to decipher. The two proofs of I.21e and I.21c can readily be combined to yield a derivation for the phrase "*Opiate oceans litten by suns poured there*". Close, but not quite there... The participial "*litten*", which acts here as a postnominal modifier, has the special property of being able to position itself either immediately after the noun phrase "*opiate oceans*" it modifies, or deferred until after the matrix head "*poured*" has made an appearance (with any adverbials attached to it). Attempting to produce a derivation for the original version seems like a dead-end enterprise, though. We are not to blame for this incompetence: the problem lies with the grammar – we could never hope to capture this behavior with our current machinery. Despite their elegance and formal appeal, grammars relying purely on Lambek calculi suffer from an aversion to anomalies like discontinuities and long-distance dependencies, which natural languages tend to exhibit at an unfortunately striking degree.

One could of course attept to cop out of the problem by just introducing ad-hoc raised forms for movable parts, one per distinct position they can be found at. The repercussions of such a move would soon, however, prove catastrophic. On the one hand, the once reliably concise lexicon would become overpopulated by endless variations on the same theme: each expansion point of a lexical type would percolate into all other lexical items it interacts with (either as consumers or producers thereof), the effect cascading at progressively larger lexical neighborhoods, until (if ever) an eventual equilibrium is

reached. On the other hand, raised types obfuscate the functional relations and constituency structures we have worked so hard to reveal and incorporate, virtually beating the very purpose of the logic. Relaxing the structural constraints of the logic to globally allow movement and/or rebracketing is no good either. Spurious ambiguity would be the least of our concerns as we would be faced with *overgeneration*, i.e. the unwelcome ability to derive proofs that have no correspondence to correct linguistic structures whatsoever, leading us back to square zero. If you have not skipped any parts yet, your reward should now manifest as an unwavering faith for a solution, and a premonition of what is to come: modalities to the rescue!

5.1.1 The Role of Modalities

Ever since their original integration with the vanilla multiplicative toolkit, (the early pioneer being none other than my #1 supervisor!) modalities have played an indispensable role in the history and development of type-logical grammars [Hendriks, 1995; Moortgat, 1996; Kurtonina and Moortgat; Moortgat, 1997; Vermaat, 1999]. They find use as either licensors or licensees of structural rewrites, now in the form of movement and rebracketing of words and phrases. Figure I.23 progresses our agenda by accounting for the presence of a (hypothetical) movable postnominal modifier via the rules of Figure I.20. To make the hypothesis movable, we need to instantiate it as a box – for the pure function contained therein to be applicable, the box needs to be removed, enclosing the hypothesis in angular brackets, which in turn license its structural extraction to the rightmost edge of the assumptions via the $\text{mix}_{\Diamond}$ rule. At that point, we need to eliminate the bracketed variable with a term of the corresponding type, plus a diamond. For this to work, we need to make the tiniest of modifications to our lexicon so as to get access to the saught-after diamond:

$$\text{litten} \quad :: \quad \Diamond\Box(\text{NP}\backslash\text{NP})/\text{PP} \tag{I.33}$$

Intuitively, the new type requests a prepositional phrase complement to the right, after the consumption of which it produces a movable postnominal modifier that can penetrate constituent phrase boundaries to the left. Equipped with it, we can derive both the local versions hinted at earlier, and their discontinuous variations; see Figure I.22 for a proof of concept.[1]

This methodology is in fact adopted from Moortgat [1999], where it finds similar use in dealing with the grammatical ambivalence of relativizers like "*that*" or "*which*". Bound relative clauses headed by complementizers like the above contain a subordinate sentence with a *gap*, which can varj in its position. Let's make things unnecessarily convoluted for the sake of clichéd self-referentialism by considering the relative clause "*which can very in its position*" of the previous sentence. There, the subordinate clause "*_ can varj in its position*" contains a gap in the subject position, which the head "*a gap*" occupies

[1] Get it? It's an actual *proof*.

$$\dfrac{\dfrac{\vdots}{\text{opiate} \cdot \text{oceans} \vdash \text{NP}} \quad \dfrac{\dfrac{}{x_i : \Box(\text{NP}\backslash\text{NP})}\,\text{id}}{\langle x_i \rangle \vdash \text{NP}\backslash\text{NP}}\,\Box E}{(\text{opiate} \cdot \text{oceans}) \cdot \langle x_i \rangle \vdash \text{NP}}\,\backslash E \qquad \vdots \text{ poured} \cdot \text{there} \vdash \text{NP}\backslash\text{S}}{((\text{opiate} \cdot \text{oceans}) \cdot \langle x_i \rangle) \cdot (\text{poured} \cdot \text{there}) \vdash \text{S}}\,\backslash E$$
$$\dfrac{((\text{opiate} \cdot \text{oceans}) \cdot (\text{poured} \cdot \text{there})) \cdot \langle x_i \rangle \vdash \text{S}}{}\,\text{mix}_\Diamond$$

(a) Extracting a hypothetical postnominal modifier...

$$\dfrac{\dfrac{\dfrac{\text{(I.22a)}}{((\dots) \cdot (\dots)) \cdot \langle x_i \rangle \vdash \text{S}} \quad \dfrac{\dfrac{}{\text{litten} : \Diamond\Box(\text{NP}\backslash\text{NP})/\text{PP}}\,\text{lex} \quad \dfrac{\dfrac{}{\text{by} : \text{PP}/\text{NP}}\,\text{lex} \quad \dfrac{}{\text{suns} : \text{NP}}\,\text{lex}}{\text{by} \cdot \text{suns} \vdash \text{PP}}\,/E}{\text{litten} \cdot (\text{by} \cdot \text{suns}) \vdash \Diamond\Box(\text{NP}\backslash\text{NP})}\,/E}{((\text{opiate} \cdot \text{oceans}) \cdot (\text{poured} \cdot \text{there})) \cdot (\text{litten} \cdot (\text{by} \cdot \text{suns})) \vdash \text{S}}\,\Diamond E$$

(b) ...before substituting the hypothesis for its material instance.

Figure I.22: Deriving long-distance postnominal modification with the aid of type assignment (I.33).

implicitly. This is not the case in the last relative clause "*which the head "gap" occupies implicitly*", whose subordinate clause "*the head "gap" occupies _ implicitly*" contains a non-peripheral (nested) gap in direct object position. What a mess! The subject-relative case can easily be dealt with in a pure Lambek grammar, as the gap hypothesis occurs adjacent to the verb phrase, but the same cannot be said for the object-relative case, whose structurally free gap seems to pose a challenge. The solution comes in the form of two distinct type assignments for the relativizer, one per grammatical role fulfilled:

$$\text{that} :: \text{REL}_\text{S} := (\text{NP}\backslash\text{NP})/(\text{NP}\backslash\text{S}) \tag{I.34}$$
$$\text{that} :: \text{REL}_\text{O} := (\text{NP}\backslash\text{NP})/(\text{S}/\Diamond\Box\text{NP}) \tag{I.35}$$

The second version launches a mobile NP hypothesis via the same diamond-box pattern showcased earlier. The proof of Figure I.23 employs this typing in combination with the $\text{ass}_\Diamond$ rule to derive the object-relative clause "*that the eye may never behold*", which applied to "*suns*" and combined with the proof of Figure I.22 yields the correct form of the postnominal modifier "*opiate oceans poured there, litten by suns that the eye may never behold*", bringing us one step closer to success.

5.1.2 Intricacies of the Lexicon

The analysis just performed illustrated the necessity of (at least) two distinct types for the same string "*that*", hinting at the fact that the lexicon is *not a function* from words to types, but rather a *relation* between them. One, more

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\text{that} : \text{REL}_o}\ \text{lex} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\text{the} \cdot \text{eye} \vdash \text{NP}}\ (\text{I.21b}) \quad \dfrac{}{\text{may} : \text{AUX}}\ \text{lex} \quad \dfrac{\dfrac{}{\text{never} : \text{ADV}_/}\ \text{lex} \quad \dfrac{\dfrac{}{\text{behold} : \text{TV}}\ \text{lex} \quad \dfrac{}{\langle \text{x}_i \rangle \vdash \text{NP}}\ \dfrac{}{\text{x}_i : \Box\text{NP}}\ \text{id}\ \Box E}{\text{behold} \cdot \langle \text{x}_i \rangle \vdash \text{NP}\backslash\text{S}}\ /E}{\text{never} \cdot (\text{behold} \cdot \langle \text{x}_i \rangle) \vdash \text{NP}\backslash\text{S}}\ /E}{\text{may} \cdot (\text{never} \cdot (\text{behold} \cdot \langle \text{x}_i \rangle)) \vdash \text{NP}\backslash\text{S}}\ \backslash E}{(\text{the} \cdot \text{eye}) \cdot (\text{may} \cdot (\text{never} \cdot (\text{behold} \cdot \langle \text{x}_i \rangle))) \vdash \text{S}}\ \text{ass}_\Diamond}{(\text{the} \cdot \text{eye}) \cdot (\text{may} \cdot ((\text{never} \cdot \text{behold}) \cdot \langle \text{x}_i \rangle)) \vdash \text{S}}\ \text{ass}_\Diamond}{(\text{the} \cdot \text{eye}) \cdot ((\text{may} \cdot (\text{never} \cdot \text{behold})) \cdot \langle \text{x}_i \rangle) \vdash \text{S}}\ \text{ass}_\Diamond}{((\text{the} \cdot \text{eye}) \cdot (\text{may} \cdot (\text{never} \cdot \text{behold}))) \cdot \langle \text{x}_i \rangle \vdash \text{S}} \quad \dfrac{}{\text{x}_j : \Diamond\Box\text{NP}}\ \text{id}}{((\text{the} \cdot \text{eye}) \cdot (\text{may} \cdot (\text{never} \cdot \text{behold}))) \cdot \text{x}_j \vdash \text{S}}\ \Diamond E}{(\text{the} \cdot \text{eye}) \cdot (\text{may} \cdot (\text{never} \cdot \text{behold})) \vdash \text{S}/\Diamond\Box\text{NP}}\ /I}{\gamma := \text{that} \cdot ((\text{the} \cdot \text{eye}) \cdot (\text{may} \cdot (\text{never} \cdot \text{behold}))) \vdash \text{NP}\backslash\text{NP}}\ /E$$

(a) Deriving an object-relative clause...

$$\dfrac{\dfrac{}{\text{litten} : \Diamond\Box(\text{NP}\backslash\text{NP})/\text{PP}}\ \text{lex} \quad \dfrac{\dfrac{\dfrac{}{\text{by} : \text{PP}/\text{NP}}\ \text{lex} \quad \dfrac{\dfrac{}{\text{suns} : \text{NP}}\ \text{lex} \quad \dfrac{}{\gamma \vdash \text{NP}\backslash\text{NP}}\ (\text{I.23a})}{\text{suns} \cdot \gamma \vdash \text{NP}}\ \backslash E}{\text{by} \cdot (\text{suns} \cdot \gamma) \vdash \text{PP}}\ /E}{\text{litten} \cdot (\text{by} \cdot (\text{suns} \cdot (\text{that} \cdot ((\text{the} \cdot \text{eye}) \cdot (\text{may} \cdot (\text{never} \cdot \text{behold})))))) \vdash \Diamond\Box(\text{NP}\backslash\text{NP})}\ /E$$

(b) ...and using it to derive the full long-distance postnominal modifier.

Figure I.23: An object-relative clause in action, prompted by type assignment (I.35).

opinionated than I, might argue that each type is mapped to a distinct lexical item (one per relativization type), and that the identification between their strings is a mere coincidence, an idiosyncracy of the language, or anyway irrelevant; even if a string is multi-typed, each type is a witness to a unique latent word hiding behind it. Of different effect but similar flavor would be the line of defense that appeals to null syntax, a covert process that can conditionally nominalize infinitives, determine plural nouns, relativize gerunds or do any sort of thing, really; a word is never multi-typed, but ad-hoc type conversions can take place out of the blue. Even under premises as radical as the above, occassions of type undeterminism are all but rare. Consider for instance the verb "*to have*", whose argument structure for the possessive meaning alone) is specified (according to its FrameNet entry [Baker et al., 1998]) as having mandatory owner and possession semantic arguments (corresponding to syntactic subject and direct object), but also any combination of depictive, duration, explanation, manner and temporal optional complements, in various orders – each variation necessarily expressed with a distinct type. In our

case, we need the type:

$$\text{having} :: (\Diamond\Box(\text{NP}\backslash\text{NP})/\text{NP})/\text{PP} \tag{I.36}$$

for a gerund that requisits first a prepositional complement phrase and then an object noun phrase (i.e. *having somewhere something*) to act as a movable post-nominal modifier (an argument permutation that FrameNet does not even contain an example of!).

The reality of optional arguments and non-trivial argument order variations alone should suffice to convince us of the issue at hand: *lexical type ambiguity* is a real phenomenon, and one that is here to stay. Having acknowledged that, the question shifts to how we deal with it. From a theoretical perspective, we can incorporate the question of type choice into our proof-machinery via the additive conjunction & of **ILL**, which is essentially recovering the functional nature of our lexicon, with type assignments reformulated as nested choices:

$$\text{A}_1\&(\text{A}_2\&(\text{A}_3 \ldots (\text{A}_{n-1}\&\text{A}_n))) \tag{I.37}$$

and the subscript enumerating each of the possible instantiations in the context of a single sentence. Under this regime, the lexical assignment rule lex would need to be followed by a sequence of projections to isolate the desired type, contributing little other than excessive verbosity.[1] Given the limited use we would have for all this "proof waste", we will stick with the current formulation of the lex rule – if it helps us sleep better at night, we can imagine it as a shorthand notation for the correct sequence of projections requested by the current analysis, the construction of which we have delegated to a silent and omnipotent oracle. Be at rest knowing that this oracle will be temporary and for presentation purposes only; we will address its demystification later on.

The ambiguity problem is exacerbated and pushed to the limit by function words enacting context-dependent chameleon roles. Coordinators are the main culprit; they can bind together pairs of the same (almost) arbitrary type to produce an instance of the conjoined pair, a complex phrase of the same type. We will write:

$$(\chi\backslash\chi)/\chi \tag{I.38}$$

to denote the coordinator type pattern parameterized over the *type variable* χ, which can be instantiated as any type of our type grammar.[2]

Armed with this last trick, we are now in possession of all the knowl-

[1] A more ambitious usecase could allow the *simultaneous* derivation of multiple unique analyses, and the incorporation of derivational ambiguity arising out of lexical choice as a first class citizen of the proof theory – a proof object that resides *within* it rather than a notion in the metatheory *above* it. The repercussions of this would be magnificent for semantic applications, but no concrete results that I am aware of were ever produced in that direction.

[2] This is in fact an exemplar of *parametric polymorphism*, which is properly formalized in second-order intuitionistic logic and its type-equivalent System F [Girard, 1972; Reynolds, 1974]. There, we write:

$$\dfrac{\dfrac{\text{(I.21a)}}{\text{strange} \cdot \text{dolphins} \vdash \text{NP}} \quad \dfrac{\dfrac{}{\text{and} : (\text{NP}\backslash\text{NP})/\text{NP}}\ \text{lex} \quad \dfrac{\text{(I.21d)}}{\text{sea-nymphs} \cdot (\text{of} \cdot (\text{unrememberable} \cdot \text{deeps})) \vdash \text{NP}}}{\text{and} \cdot (\text{sea-nymphs} \cdot (\text{of} \cdot (\text{unrememberable} \cdot \text{deeps}))) \vdash \text{NP}\backslash\text{NP}}\ /E}{\delta := (\text{strange} \cdot \text{dolphins}) \cdot (\text{and} \cdot (\text{sea-nymphs} \cdot (\text{of} \cdot (\text{unrememberable} \cdot \text{deeps})))) \vdash \text{NP}}\ \backslash E$$

(a) Deriving noun-phrase coordination...

$$\dfrac{\dfrac{\dfrac{}{\text{having} : (\Diamond\Box(\text{NP}\backslash\text{NP})/\text{NP})/\text{PP}}\ \text{lex} \quad \dfrac{\dfrac{}{\text{in} : \text{PP}/\text{NP}}\ \text{lex} \quad \dfrac{\dfrac{}{\text{their} : \text{NP}/\text{NP}}\ \text{lex} \quad \dfrac{}{\text{whirlpools} : \text{NP}}\ \text{lex}}{\text{their} \cdot \text{whirlpools} \vdash \text{NP}}\ /E}{\text{in} \cdot (\text{their} \cdot \text{whirlpools}) \vdash \text{PP}}\ /E}{\text{having} \cdot (\text{in} \cdot (\text{their} \cdot \text{whirlpools})) \vdash \Diamond\Box(\text{NP}\backslash\text{NP})/\text{NP}} \quad \dfrac{\text{(I.24a)}}{\delta \vdash \text{NP}}}{(\text{having} \cdot (\text{in} \cdot (\text{their} \cdot \text{whirlpools}))) \cdot \delta) \vdash \Diamond\Box(\text{NP}\backslash\text{NP})}\ /E$$

(b) ...and using it to construct yet another postnominal modifier.

Figure I.24: Filling in the missing bits using the polymorphic type (I.38).

edge necessary to finally tackle the full derivation. First, we must instantiate the polymorphic coordinator once by substituting χ for NP to derive the noun phrase conjunction "*strange dolphins and sea-nymphs of unrememmemberable deeps*", as portrayed in Figure I.24a. This, together with our freshly typed "*having*", allows the derivation of the mobile postnominal modifier "*having in their whirlpools strange dolphins and sea-nymphs of unrememberable deeps*", as in Figure I.24b. At this point, we must employ another instance of the polymorphic coordinator, this time substituting χ for $\Diamond\Box(\text{NP}\backslash\text{NP})$ – this opens the door to the derivation of the structurally free complex postnominal modifier "*litten by suns that the eye may never behold and having in their whirlpools strange dolphins and sea-nymphs of unrememberable deeps*", which can apply to the nested "*opiate oceans*" in the same fashion as the proof of Figure I.22. At long last, we are rewarded with a type-checking and syntactically faithful analysis of the full sentence (and a check mark on *how to write a dissertation*). Collaging these last bits together is left as an exercise to the motivated reader, for fear of repetition sterilizing the quotation of its beauty.

$$\dfrac{\Gamma, \alpha : \text{TYPE} \vdash \mathsf{M} : \sigma}{\Gamma \vdash \lambda\mathsf{ff}.\mathsf{M} : \Pi\alpha.\sigma}\ \Pi I \qquad \dfrac{\Gamma \vdash \mathsf{M} : \Pi\alpha.\sigma \quad \Delta \vdash \mathsf{B} : \text{TYPE}}{\Gamma, \Delta \vdash \mathsf{M}\ \mathsf{B} : \sigma_{[\alpha \mapsto \mathsf{B}]}}\ \Pi E$$

to denote terms abstracting over types $\lambda\alpha.M$ and types quantified over types $\Pi\alpha.\sigma$. In this notation, a coordinator would be a quantification of type $\Pi\chi.(\chi\backslash\chi)/\chi$, that when reduced against arbitrary type A would yield $(\text{A}\backslash\text{A})/\text{A}$. Other than this unique occurrence of polymorphism, second order term and type constructions are an overkill to our purposes here, relegating this comment to footnote status.

5.1.3 Subtleties of Proof Search

The last sentence was merely a test to weed out the uncommited. Of those that passed it and attempted to really proceed with the derivation, the observant ones should have found themselves at multiple crossroads regarding the order of applying the numerous modifiers in the sentence – a matter carefully concealed in the derivations presented so far. The choice of **NL** over **L** implies that scope assigned to competing modifiers should reflect in a corresponding judgement that differs to the rest in the bracketing structure of its antecedents (and of course the proof justifying it). The following endsequents are all valid alternatives provable with the lexical types of Figure I.24a:

i. strange · (dolphins · (and · (sea-nymphs · (of · (unrememberable · deeps)))))

ii. strange · ((dolphins · (and · sea-nymphs)) · (of · (unrememberable · deeps)))

iii. (strange · (dolphins · (and · sea-nymphs))) · (of · (unrememberable · deeps))

iv. ((strange · dolphins) · (and · sea-nymphs)) · (of · (unrememberable · deeps))

v. (strange · dolphins) · (and · (sea-nymphs · (of · (unrememberable · deeps)))))

This is an admittedly stretched case of *derivational ambiguity*, a situation where from the same lexical assignments one can obtain multiple syntactic analyses, which may correspond to equinumerous subtly or drastically diverging semantic interpretations (more on that in a bit). If the underlying proofs were to be produced in the associative calculus **L**, the ambiguity would have been *spurious*, as all would lead to a structurally identical (bracketless) judgement. Derivational ambiguity is not necessarily bad, provided the divergence in the proofs constructed is linguistically meaningful.[1] This is not the case for spurious ambiguity, as it introduces undesirable (and uninterpretable) redundancy, at the dismay of parsers (or mostly their designers). What is, however, worth noting is the structural discrepancy between what we see (a flat sequence) and what we want to parse into (a binary branching tree). Even though constituency structure is de facto acknowledged by linguistic theory, it is a latent mental construct revealed through (or assigned by) the parsing process, rather than an observable feature of text that we can assume as a given. The connotation of this is that even though backwards proof search in **NL** may find use in *verifying* the plausibility of a type-assigned, prebracketed phrase, forward search is necessary in *eliciting* a type and a bracketing structure from a phrase.

5.1.4 Syntax-Semantics Interface

The game played so far, challenging as it may be, might prove dull to someone indifferent to syntax or its type-theoretic formulation; we will attempt to fix that by expanding our target crowd to semanticists and Montagovian grammarians, who are said to recite daily before beditme:

[1] Just think of all the different things you could do with pijamas, elephants, telescopes, etc.

> I fail to see any interest in syntax except as a preliminary to semantics. [Montague, 1970]

Montague's Insights A full exposition to Montague grammar is beyond the scope of this thesis, but a brief introduction to some of its foundations will go a long way in helping us perceive its relevance to the type-logical approach. Richard Montague was disillusioned with the tackling of natural language semantics at the time, which he found formally inadequate and lacking the elegance of contemporary approaches to mathematical syntax. He saught to fill this gap by arguing that formal and natural languages are morally indistinguishable – different instantiations of the same theory – and advocating their treatment in just the same way. Influenced by his own background on modal logic and the highly influential work of Saul Kripke on possible world semantics [Kripke, 1963], the machinery he thought was best fit for the task at hand was a model theoretic semantics axiomatized on the basis of set theory and higher-order logic; his work is marked with heavy use of λ notation, the adoption of which by today's working linguist is largely attributed to him.

Revolutionary as it may have been at the time, this semantic machinery and its antiquated details are largely irrelevant to this work. What is of prime interest, though, is Montague's treatment of the passage between syntax and semantics. In his view, if syntax is an algebra describing the process of synthesizing a grammatically passable sentence, semantics is another algebra providing a logical recipe for evaluating that sentence's truth-validity. The two systems are viewed as distinct, but not independent: they are connected by a unidirectional transformation that preserves and transports (certain aspects of) the structure of the former into the latter, in other words a *homomorphism*. The slogan "syntax is an algebra, semantics is an algebra and meaning is a homomorphism between them" summarizes this notion [Janssen, 2014]. The gracefulness of this statement is easy to miss. It proclaims that the semantic expression assigned to complex linguistic entries mimics (or is at least informed by) the structural form of their syntactic analyses. This perspective actuates the ideal of compositionality, a concept passed down by Gottlob Frege and summarized as stating that the meaning of a complex expression is computable on the basis of its primitive expressions and the rules that dictate their combination [Partee et al., 1984].

The Type-Logical View Let's appropriate this view and translate it to the type-logical setup, as done by van Benthem [1988]. Here, syntax is a type theory: a logic whose rules are equated to term rewrite instructions, and proofs to programs. Semantics can also be a type theory; one with its own types and terms, potentially more expressive and certainly unriddled by (some of) the structural constraints of grammar. The meaning interpretation would then be a homomorphism that translates syntactic proofs and programs to corresponding semantic ones – a translation from one constructive logic to another.

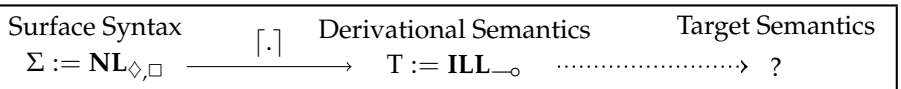


Figure I.25: The syntax-semantics interface in the type-logical setting.

Its design would need to follow the rule-to-rule approach, according to which every syntactic construction will have its homomorphic image in the target system [Bach, 1976]. This viewpoint is quite open-ended and admits a whole lot of creative liberty with respect to the the nature of the target system and the details of the translations. The only constraint imposed is the only one that matters: the high-level principle of compositionality needs to hold, i.e. the function-argument structure specified by syntax need to be carried through to the semantics.

Interestingly, the approach permits a division of labour between syntax, semantics and everything in between: the end-to-end translation can be decomposed into a sequence of homomorphisms, each intermediate step explicating an additional layer of added expressivity (or fortfeited structure) and singling out a subset of the desiderata towards the end-target. A natural first stop would be that of $\mathbf{ILL}_{\multimap}$ as a *derivational semantics* logic: it captures the function-argument structures prescribed by the syntactic proof and respects its no-reuse principle, but without the semantically void headaches of order and bracketing structures, or that of the rules manipulating them.

To make things concrete, let's consider this in the context of the source logic Σ being identified with the instantiation of $\mathbf{NL}_{\diamond,\Box}$ of the previous section, and the intermediate logic T being its $\mathbf{ILL}_{\multimap}$ mirror image. Using the superscript $X := \Sigma \mid \mathrm{T}$ to distinguish between the two logics, we will denote with Prop_0^X the set of atomic types of X, and $\mathcal{U}^X$ its type universe, i.e. the inductive closure of types under type operators. Similarly, we will denote with Cons^X its set of constants, Vars^X its set of variable names, and Terms^X its well-formed terms, i.e. the inductive closure of terms under term operators.

The homomorphism $\lceil . \rceil$ operates on proofs, i.e. typed terms, and thus does double duty: it transforms both terms and types of Σ to corresponding terms and types of T. It is handy, then, to define it on the basis of two components $\langle \eta, \theta \rangle$, where $\eta : \mathcal{U}^\Sigma \to \mathcal{U}^\mathrm{T}$ and $\theta : \mathsf{Terms}^\Sigma \to \mathsf{Terms}^\mathrm{T}$, such that $\lceil \mathsf{s} : \mathrm{A} \rceil = \theta(\mathsf{s}) : \eta(\mathrm{A})$, where the typing relation at the right-hand side of the equation must hold (i.e. the two maps mutually respect derivability). On the type level, η must specify a pointwise mapping η_0 from the propositional constants Prop_0^Σ of the source logic to types $\mathcal{U}^\mathrm{T}$ of the intermediate logic. In our case, we will consider this a bijection from Prop_0^Σ to $\mathsf{Prop}_0^\mathrm{T}$, such that $\eta_0(p) \mapsto p$ (i.e. instantiating $\mathsf{Prop}_0^\mathrm{T}$ as a literal copy of Prop_0^Σ). Then, to extend η_0 to η we need to specify its action on complex types, where it essentially forgets the unary modalities and removes the directionality of the implications, as shown in Table I.5. In the exact same vein, θ pointwise sends constants and variables

to their copycat images, and is then inductively defined on complex terms, where it casts directional applications and abstractions to undirectional ones, drops modal decorations and performs the simplified substitution prescribed by the $\Diamond E$ rule, as shown in Table I.6. As an example, applying $\lceil . \rceil$ to the proof of Figure I.23 should yield the derivational term:

$$\mathsf{litten}\ (\mathsf{by}\ (\mathsf{that}\ (\lambda \mathsf{x}_i.(\mathsf{may}\ (\mathsf{never}\ (\mathsf{behold}\ \mathsf{x}_i)))\ (\mathsf{the\ eye})))\ (\mathsf{suns}))^{\mathrm{NP} \multimap \mathrm{NP}} \quad \text{(I.39)}$$

$\mathcal{U}^{\Sigma}$		$\mathcal{U}^{\mathrm{T}}$
$p \in \mathsf{Prop}_0^{\Sigma}$	$\mapsto$	$\eta_0(p) := p \in \mathsf{Prop}_0^{\mathrm{T}}$
$\mathrm{A}\backslash\mathrm{B},\ \mathrm{B}/\mathrm{A}$	$\mapsto$	$\eta(\mathrm{A}) \multimap \eta(\mathrm{B})$
$\Diamond\mathrm{A},\ \Box\mathrm{A}$	$\mapsto$	$\eta(\mathrm{A})$

Table I.5: Translating $\mathbf{NL}_{\Diamond,\Box}$ types to $\mathbf{ILL}_{\multimap}$.

Terms^{Σ}		$\mathsf{Terms}^{\mathrm{T}}$
$\mathsf{c} \in \mathsf{Cons}^{\Sigma}$	$\mapsto$	$\theta_0(\mathsf{c}) := \mathsf{c} \in \mathsf{Cons}^{\mathrm{T}}$
$\mathsf{x}_i \in \mathsf{Vars}^{\Sigma}$	$\mapsto$	$\theta_0(\mathsf{x}_i) := \mathsf{x}_i \in \mathsf{Vars}^{\mathrm{T}}$
$\mathsf{s} \blacktriangleleft \mathsf{t},\ \mathsf{t} \blacktriangleright \mathsf{s}$	$\mapsto$	$\theta(\mathsf{s})\ \theta(\mathsf{t})$
$\lambda \mathsf{x}_i.s,\ \curlywedge \mathsf{x}_i.s$	$\mapsto$	$\lambda \theta(\mathsf{x}_i).\theta(\mathsf{s})$
$\triangle\mathsf{s},\ \blacktriangle\mathsf{s},\ \blacktriangledown\mathsf{s}$	$\mapsto$	$\theta(\mathsf{s})$
$\mathsf{case}\ \triangledown\mathsf{t}\ \mathsf{of}\ \mathsf{x}_i\ \mathsf{in}\ \mathsf{s}$	$\mapsto$	$\theta(\mathsf{s})_{[\theta(\mathsf{x}_i) \mapsto \theta(\mathsf{t})]}$

Table I.6: Translating $\mathbf{NL}_{\Diamond,\Box}$ terms to $\mathbf{ILL}_{\multimap}$.

In this navigation between syntactic and semantic theories, the Curry-Howard isomorphism serves as our North Star. Syntactic proofs are equated to syntactic terms, on which our homomorphism can be applied to yield derivational semantics terms, in turn equatable to derivational semantics proofs. This might seem like a lot of work to simply "forget" syntax, but it showcases how one can step up the computational hierarchy of substructural logics in order to attain access to more expressive semantics. Note also that such a path is merely a suggestion and not an imperative; a more ambitious line of though could maintain that word order variations (and the structural rules licensing them) can carry semantic cues which, albeit subtle, need to be upheld in the compositional meaning translation (see for instance the contemporary work of Correia [2022] for some exotic interpretations of the control modalities).

The Role of the Lexicon The sentiments of the previous paragraph could be met with some skepticism. A critical eye might argue that semantic interac-

tions not already manifested in the syntax may never be born of this process, and thus wonder whether this added expressivity can serve any real purpose or offer any tangible benefits. To dispel such doubts, we need to keep in mind that derivational terms refrain from specifying *lexical meaning*, i.e. they treat lexical items as black boxes, from a semantic perspective. Opening these black boxes would reveal flat entries (i.e. term constants) in the case of words providing meaning *ingredients*, as opposed to structurally rich entries (i.e. complex terms with internal structure) in the case of words providing meaning *recipes*.[1] Structurally rich lexical entries can utilize *any* term constructor made available by the semantic logic; crucially, this includes constructors that escape the narrow borders of the homomorphic codomain (i.e. do not have a syntactic origin). Of course, such terms are still bound by the promise to obay the type dictated by the homomorphic translation of their original syntactic type, and must also be derivable theorems of the semantic logic they live in. Increasing expressivity therefore may indeed not in itself add to the function-argument structures inherited by syntax, but provides the tools necessary for complex lexical semantic actions to take effect as needed.

A case in point is the coordinator "*and*" conjoining the two modifiers of the previous section: "*litten by . . .* ***and*** *having in . . .*". Each individual conjunct fulfills a descriptive filter that intersects the properties of its argument with the properties attributed by its internal meaning. That is, of all objects of type $*$ (where $*$ an arbitrary type, denoting the interpretation target of NP), the first modifier withdraws all but those lit by unseeable suns, whereas the second one withdraws all but those with weird entities in their whirlpools. For the full conjunction to have the intended meaning, i.e. evoke the image of exclusively this subset of oceans characterized by both the above properties, the coordinator would need to enact the role of a portable implementation of function composition[2] as in Figure I.8, so as to allow the iteration of the intersective modifiers:

$$\lambda x_0 x_1 x_2 . x_0\ (x_1\ x_2) :: (*\multimap *)\multimap(*\multimap *)\multimap * \multimap * \tag{I.40}$$

Even though no non-standard term constructors are to be found in this recipe, it is nontheless *not* a theorem of the source logic, as function composition is not derivable in **NL**. In a set-theoretic semantics domain unbound by linearity constraints, another (perhaps more reasonable) translation might make use of an added operator $\wedge : * \to * \to *$ for set-theoretic intersection ($*$ now an

[1] This distinction is usually paralleled with the linguistic distinction between *content* and *function* words, but commiting to this being the case is an unecessary restriction. Depending on the end-target semantics logic and the granularity of the semantic lexicon, content words might still be assigned complex term structure – a common trick, for instance, in delivering dependent type semantics; see the book of Chatzikyriakidis and Luo [2020] for an overview of recent developments.

[2] Before anyone gets angry: I am not pitching some provocative theory of conjunction semantics here – just trying to make a point.

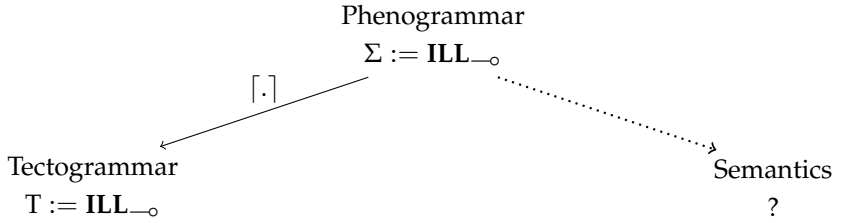


Figure I.26: The syntax-semantics interface in the abstract categorial setting.

arbitrary set), to deliver the recipe:

$$\lambda x_0 x_1 x_2.(x_0\ x_2) \wedge (x_1\ x_2) :: (*\to *)\to(*\to *)\to *\to * \tag{I.41}$$

5.2 Abstract Categorial Grammars

So far, we have been predisposed to treating syntax as the hidden process that forms grammatically correct sentences. It is insightful to contrast this treatment with the view of Curry [1961], who thought of syntax as a two-layered hierarchy of grammaticality criteria. The deep layer, called *tectogrammar*, would be concerned solely with the well-typedness of grammatical function domains and the validity of their interpretations. The shallow layer, called *phenogrammar*, would be where tectogrammatical proofs are transformed and cast to surface forms that abide by the linear order and constituency restrictions imposed by the language. Type-logical grammars pose no challenge to the legitimacy of this distinction: it should be clear that phenogrammar, in Curry's terms, is our syntactic logic, and tectogrammar is what we earlier referred to as derivational semantics. In being tectogrammar-first, however, they diverge in its operationalization. The computational pipeline they propose is sequential in nature, and follows the Aristotelian path from observable evidence to latent variables: the surface string is perceived as the yield of a (shallow) syntactic proof, from which a deep semantic proof is extracted. The operationalization closer to Curry would be inverted, placing phenogrammar at the top of the generative process, and following the Platonic information flow from deep and abstract to shallow and concrete. This perspective is embodied by abstract categorial grammars [de Groote, 2001] and their contemporary and closely related lambda grammars [Muskens, 2001]. Both are tectogrammar-first formalisms that make use of **ILL**$_{\multimap}$ and the Curry-Howard isomoprhism to obtain phenogrammatic realizations via homomorphic translations of the tectogrammatic parse.

5.2.1 Basic Definitions

The focus of our presentation will be on abstract categorial grammars, as they are closer in spirit to what is to come later. In its original definition, an abstract categoral grammar consists of two instantiations Σ, T of $\mathbf{ILL}_{-\circ}$, and a map between them. The source instantiation Σ provides a set of base types Prop_0^Σ, and the so-called *abstract vocabulary*: a set of *abstract constants* Cons^Σ, each assigned a type from $\mathcal{U}^\Sigma$. The target instantation T provides another set of base types $\mathsf{Prop}_0^\mathrm{T}$, and constants Cons^T with types from $\mathcal{U}^\mathrm{T}$, called the *object vocabulary*. The map between them is once again a homomomorphism $\lceil . \rceil$, defined on the basis of $\langle \eta_0, \theta_0 \rangle$. Not unlike before, η_0 is seen as implementing a mapping $\mathsf{Prop}_0^\Sigma \to \mathcal{U}^\mathrm{T}$, and θ_0 a mapping $\mathsf{Cons}^\Sigma \to \mathsf{Terms}^\mathrm{T}$, both pointwise defined. Their homomorphic extension is trivially obtained by recursively defining their actions on implicational types, function terms and λ abstractions, where they simply mimic the source type- and term- structure. This formulation lends itself nicely to the notion of *grammar composition*, if one is to use the object logic of a grammar as the abstract logic of another. Each grammar is accompanied by two *languages*; the abstract language, i.e. the set of terms (of some *distinguished* type $p_d \in \mathsf{Prop}_0^\Sigma$) derivable in the source logic, and the object language, i.e. the set of object terms the abstract language maps into. For the phenogrammar to tectogrammar picture to be made evident, the distinguished type needs to be mapped to the functional string type $p_d \mapsto \mathtt{str}$, forcing terms of the object language to evaluate to strings. Note that, despite appearances, $\mathtt{str}$ is a first-order type $* \multimap *$ (where $*$ some arbitrary primitive) so as to permit the view of string concatenation as function composition, identical to (I.40):

$$+ := \lambda \mathsf{x}_0^{\mathtt{str}} \mathsf{x}_1^{\mathtt{str}} \mathsf{x}_2^{*}.\mathsf{x}_0\ (\mathsf{x}_1\ \mathsf{x}_2) \tag{I.42}$$

5.2.2 Artificial Languages

Abstract categorial grammars are characterized by two measures of complexity: the maximal order of source constants' types, and the maximal order of the codomain of η_0. The two together constitute the grammar's *class*, which concisely describes the sort of languages the grammar can model. This can prove effective in revealing a more granular stratification underlying the Chomsky hierarchy of formal grammars, when the latter are embedded into abstract categorial equivalents; as such, the framework has found extensive use as a meta-language for the study and formalization of formal grammars (as done by de Groote and Pogodalla [2004], *inter alia*). [1]

To see this in practice, let's have some meta-fun pretty-printing the types of $\mathbf{(N)L}_{\Diamond,\Box}$ by modeling their type formation rules (which constitute a context-free grammar) using an abstract categorial grammar. First item on the agenda

[1] There is a certain irony in formal grammars requiring or benefiting from formalization. If you're having trouble parsing this, consider that formal languages are essentially ad-hoc rules on strings; by formalization we mean giving these rules the type-theoretic treatment they deserve.

$$\begin{aligned}\mathsf{Cons}^{\Sigma} := \{&\mathsf{n} :: \textsc{Type},\ \mathsf{np} :: \textsc{Type},\ \mathsf{pp} :: \textsc{Type},\ \mathsf{np} :: \textsc{Type},\\ &\mathsf{dia} :: \textsc{Type} \multimap \textsc{Type},\ \mathsf{box} :: \textsc{Type} \multimap \textsc{Type}\\ &\mathsf{ldiv} :: \textsc{Type} \multimap \textsc{Type} \multimap \textsc{Type}\\ &\mathsf{rdiv} :: \textsc{Type} \multimap \textsc{Type} \multimap \textsc{Type}\}\end{aligned}$$

Figure I.27: Abstract lexicon for the language of **(N)L**$_{\Diamond,\Box}$ types.

Abstract Constant	Object Term
n	$\underline{\mathrm{N}}$
np	$\underline{\mathrm{NP}}$
pp	$\underline{\mathrm{PP}}$
s	$\underline{\mathrm{S}}$
dia	$\lambda \mathsf{x}_i.\underline{\Diamond} + \mathsf{x}_j$
box	$\lambda \mathsf{x}_i.\underline{\Box} + \mathsf{x}_k$
ldiv	$\lambda \mathsf{x}_i \mathsf{x}_j.\underline{(} + \mathsf{x}_i + \underline{\backslash} + \mathsf{x}_j + \underline{)}$
rdiv	$\lambda \mathsf{x}_i \mathsf{x}_j.\underline{(} + \mathsf{x}_j + \underline{/} + \mathsf{x}_i + \underline{)}$

Table I.7: Object translation for the lexicon of Figure I.27.

is the specification of our two logics Σ and T. The source logic Σ will provide the abstract backbone of the type grammar, containing a single base type, that of a well-formed "type" $\mathsf{Prop}_0^{\Sigma} := \{\textsc{Type}\}$. The abstract vocabulary is then populated in Figure I.27 by all abstract constants denoting base "types"[1] and "type" constructors. The target logic T will be our phenogrammatic printer tasked with translating abstract terms ("types") to object terms (strings). We will need a single object base type $\mathsf{Prop}_0^{\mathrm{T}} := \{*\}$, such that η_0 Some auxiliary object constants are necessary before we proceed: opening and closing brackets, a diamond and a box the two implications, and a unique match for each unique *constant* abstract constant (i.e. each abstract constant whose type is of order zero). The above – all of type `str`, and underlined to distinguish from functional symbols – are used by the abstract constant translation θ_0 defined in Table I.7: base constructors are mapped to their corresponding string representations, the two unary modalities simply concatenate their symbol to their single argument, whereas the two implications infix their arguments with the a slash or backslash, and wrap the result under brackets.

Figure I.28 presents the construction of the type previously assigned to "*litten*", $\Diamond\Box(\mathrm{NP}\backslash\mathrm{NP})/\mathrm{PP}$ (contexts are intentionally left empty and axioms replaced by abstract constants for brevity). Applying the homomorphic translation to its abstract yields a printout in the form of the object term below (source

[1] In hindsight, that might have been an unfortunate choice of term[2] to overload.

$$\dfrac{\dfrac{\mathsf{rdiv}}{\textsc{Type}\multimap\textsc{Type}\multimap\textsc{Type}} \quad \dfrac{\dfrac{\mathsf{dia}}{\textsc{Type}\multimap\textsc{Type}} \quad \dfrac{\dfrac{\mathsf{box}}{\textsc{Type}\multimap\textsc{Type}} \quad \dfrac{\dfrac{\mathsf{ldiv}}{\textsc{Type}\multimap\textsc{Type}\multimap\textsc{Type}} \quad \dfrac{\mathsf{np}}{\textsc{Type}} \quad \dfrac{\mathsf{np}}{\textsc{Type}}}{\mathsf{ldiv\ np\ np} : \textsc{Type}}\multimap E}{\mathsf{box}\ (\mathsf{ldiv\ np\ np}) : \textsc{Type}}\multimap E}{\mathsf{dia}\ (\mathsf{box}\ (\mathsf{ldiv\ np\ np})) : \textsc{Type}}\multimap E \quad \dfrac{\mathsf{pp}}{\textsc{Type}}}{\mathsf{rdiv}\ (\mathsf{dia}\ (\mathsf{box}\ (\mathsf{ldiv\ np\ np}))\ \mathsf{pp} : \textsc{Type}}\multimap E$$

Figure I.28: Constructing the type assignment of (I.33).

function-argument brackets substituted with indendation levels for legibility):

$$\begin{aligned}
&\lceil \mathsf{rdiv}\ (\mathsf{dia}\ \mathsf{box}\ (\mathsf{ldiv\ np\ np}))\ \mathsf{pp} \rceil \\
&= \lambda \mathsf{x}_i \mathsf{x}_j . \underline{(} + \mathsf{x}_j + \underline{/} + \mathsf{x}_i + \underline{)} \\
&\quad \underline{\mathrm{PP}} \\
&\quad \lambda \mathsf{x}_k . \underline{\Diamond} + \mathsf{x}_k \\
&\quad\quad \lambda \mathsf{x}_l . \underline{\Box} + \mathsf{x}_l \\
&\quad\quad\quad \lambda \mathsf{x}_m \mathsf{x}_n . \underline{(} + \mathsf{x}_m + \underline{\backslash} + \mathsf{x}_n + \underline{)} \\
&\quad\quad\quad\quad \underline{\mathrm{NP}} \\
&\quad\quad\quad\quad \underline{\mathrm{NP}} \\
&\overset{\beta*}{\rightsquigarrow} \underline{(\Diamond\Box(\mathrm{NP}\backslash\mathrm{NP})/\mathrm{PP})}
\end{aligned} \tag{I.43}$$

Six reduction steps later and... voilà – our pretty printer works! The maximal order of the abstract constants is 1, and the maximal order of the translation is 2, making our grammar's complexity class (1, 2), a proper subset of the (2,2) that encapsulates context free grammars.

5.2.3 Human Languages

Elegant and successful as they might be in the meta-theoretical world, abstract categorial grammars have not fared as well with linguistic applications, in large part due to their computationally intractable nature. On the one hand, they stand out from the rest of the categorial family in not being lexicalized by default. The conceptual separation between lexicon and rules no longer holds: rules are fixed to the ones supplied by $\mathbf{ILL}_{\multimap\circ}$, but inference is largely guided by the abstract constants. Abstract constants may contain lexical items that make their way to the final (object) derivation, or simply compositional recipes that leave no imprint whatsoever. At the same time, the framework is overly reliant on the constant map θ_0 (defined on a per-item basis) for the translation into the object language to take effect. Even in the lexicalized setup where the abstract lexicon is populated by words and words only, every abstract constant needs to be assigned both an abstract type and a unique object

Abstract Constant	Abstract Type	Object Term
eye	N	$/\!/\mathsf{eye}/\!/$
suns	NP	$/\!/\mathsf{suns}/\!/$
oceans	NP	$/\!/\mathsf{ocreans}/\!/$
the	N$\multimap$NP	$\lambda \mathsf{x}_i./\!/\mathsf{the}/\!/ + \mathsf{x}_i$
opiate	NP$\multimap$NP	$\lambda \mathsf{x}_i./\!/\mathsf{opiate}/\!/ + \mathsf{x}_i$
poured	NP$\multimap$S	$\lambda \mathsf{x}_i.\mathsf{x}_i + /\!/\mathsf{poured}/\!/$
behold	NP$\multimap$NP$\multimap$S	$\lambda \mathsf{x}_i\mathsf{x}_j.\mathsf{x}_j + /\!/\mathsf{behold}/\!/ + \mathsf{x}_i$
there	(NP$\multimap$S)$\multimap$NP$\multimap$S	$\lambda \mathsf{x}_i\mathsf{x}_j.(\mathsf{x}_i\ \mathsf{x}_j) + /\!/\mathsf{there}/\!/$
never	(NP$\multimap$S)$\multimap$NP$\multimap$S	$\lambda \mathsf{x}_i\mathsf{x}_j./\!/\mathsf{never}/\!/ + (\mathsf{x}_i\ \mathsf{x}_j)$
may	(NP$\multimap$S)$\multimap$NP$\multimap$S	$\lambda \mathsf{x}_i\mathsf{x}_j./\!/\mathsf{may}/\!/ + (\mathsf{x}_i\ \mathsf{x}_j)$
by	NP$\multimap$PP	$\lambda \mathsf{x}_i./\!/\mathsf{by}/\!/ + \mathsf{x}_i$
litten	PP$\multimap$NP$\multimap$NP	$\lambda \mathsf{x}_i\mathsf{x}_j.\mathsf{x}_j + /\!/\mathsf{litten}/\!/ + \mathsf{x}_i$
that	(NP$\multimap$S)$\multimap$NP$\multimap$NP	$\lambda \mathsf{x}_i\mathsf{x}_j.\mathsf{x}_j + /\!/\mathsf{that}/\!/ + \mathsf{x}_i(/\!/\_/\!/)$

Table I.8: Abstract lovecraftian lexicon abiding to the types of Table I.4.

term for every phenogrammatic behavior it exhibits; two lexical dimensions, compared to the one of vanilla categorial grammars. Enforcing grammaticality while blocking overgeneration of the object language similarly requires a careful, parallel finetuning of both the abstract language and the translation – gone is the adage of words carrying their combinatorics on their sleeves. What's worse, words triggering higher-order tectogrammatic phenomena will then need object translations of an even higher order for their surface forms, making the design and population of a strict tectogrammatic translation $\lceil . \rceil$ practically unfeasible. This part could in principle be partially mitigated by flattening complex syntactic phenomena into lower-order (alias) types in the source domain, and outsourcing their expansion to a parallel grammar for concrete semantics – this is less of a solution and more of a deferral, though. Beyond issues of practicality, there are also foundational problems at stake, as resorting to a lexical enumeration of phenogrammatic forms evidences inability to perform linguistic generalization – what Moot [2014] calls a problem of *descriptive inadequacy* revolving around *any* abstract replacement to a Lambek higher-order type. Last but not least, it is hard to imagine an abstract categrial grammar in action: it is unclear how to procure an abstract proof object from the *evaluated* yield of its object translation (i.e. the string form we are most likely encounter in the open) using traditional proof-theoretic disciplines – the two layers of function-argument structures (abstract- and object- level) and their interacting reductions would unnerve even the sturdiest of parsers (or so it seems).

As an artificial yet illustrative and down-to-earth example, let's brave the design of an abstract categorial grammar tasked with the production of an end-to-end linguistic example. Once more, we start with the specification of

our two logics Σ and T. For efficacy and simplicity, we can have Σ coincide with the derivational semantics logic of Section 5.1.4, inheriting its terms and types for free. Doing so requires of course that we assume some high-level equivalence between the representations of what used to be abstract semantics before, and what now we call deep syntax – let's naively take this for granted. The object logic T, being responsible for the surface materialization of our derivational proofs, will again need to be a logic of strings and is thus populated by a single atomic type $\mathsf{Prop}_0^{\mathrm{T}} = \{*\}$; all abstract atoms atoms are then sent to `str`. For each word, we will need an abstract constant $\mathsf{c} \in \mathsf{Cons}^{\Sigma}$, and a corresponding object constant $/\!/\mathsf{c}/\!/ :: \mathtt{str} \in \mathsf{Cons}^{\mathrm{T}}$, denoting the word's string form. Abstract constants will be sent to object terms via θ_0, each of which must contain a single occurence of a term constant, in order to preserve lexicalism and respect lexical transparency. Worth a special mention is the fact that the tectogrammatic image of words assigns them syntactic recipes, as they carry their own λ terms. The story is summarized in Table I.8 (object types are ommitted for brevity – simply substitute all atoms of the corresponding abstract types with `str` to obtain them).

The equation below shows the computation of the homomorphic translation to the derivational term (I.39) inspected earlier.

$$
\begin{aligned}
&\lceil \mathsf{litten}\ (\mathsf{by}\ (\mathsf{that}\ (\lambda \mathsf{x}_i.(\mathsf{may}\ (\mathsf{never}\ (\mathsf{behold}\ \mathsf{x}_i)))\ (\mathsf{the}\ \mathsf{eye})))\ (\mathsf{suns}))\rceil \\
&= (\lambda \mathsf{x}_i \mathsf{x}_j.\mathsf{x}_j + /\!/\mathsf{litten}/\!/ + \mathsf{x}_i) \\
&\quad (\lambda \mathsf{x}_k./\!/\mathsf{by}/\!/ + \mathsf{x}_k) \\
&\quad\quad (\lambda \mathsf{x}_l \mathsf{x}_m.\mathsf{x}_m + /\!/\mathsf{that}/\!/ + \mathsf{x}_l(/\!/\_/\!/)) \\
&\quad\quad\quad (\lambda x_n. \\
&\quad\quad\quad\quad (\lambda x_o x_p./\!/\mathsf{may}/\!/ + (x_o\ x_p)) \\
&\quad\quad\quad\quad\quad (\lambda x_q x_r./\!/\mathsf{never}/\!/ + (x_q\ x_r)) \\
&\quad\quad\quad\quad\quad\quad (\lambda x_s x_t.x_t + /\!/\mathsf{behold}/\!/ + x_s) \\
&\quad\quad\quad\quad\quad\quad\quad x_n \\
&\quad\quad\quad ((\lambda x_u./\!/\mathsf{the}/\!/ + x_u)\ /\!/\mathsf{eye}/\!/) \\
&\quad\quad /\!/\mathsf{suns}/\!/ \\
&\overset{\beta*}{\rightsquigarrow} \lambda \mathsf{x}_i.\mathsf{x}_i + /\!/\mathsf{litten}/\!/\,/\!/\mathsf{by}/\!/\,/\!/\mathsf{suns}/\!/\,/\!/\mathsf{that}/\!/\,/\!/\mathsf{the}/\!/\,/\!/\mathsf{eye}/\!/\,/\!/\mathsf{may}/\!/\,/\!/\mathsf{never}/\!/\,/\!/\mathsf{behold}/\!/\,/\!/\_/\!/
\end{aligned}
\tag{I.44}
$$

This already proves quite an endeavour; twelve reduction steps later, we are presented with a seemingly reasonable output. Upon closer inspection, though, an issue pops up: the end-term s is of type `str`$\multimap$`str`, making it an ill-fit for the discontinuous application to the *substring* $/\!/\mathsf{opiate}/\!/\,/\!/\mathsf{oceans}/\!/$, which we expect to find nested within the main clause $/\!/\mathsf{opiate}/\!/\,/\!/\mathsf{oceans}/\!/\,/\!/\mathsf{poured}/\!/\,/\!/\mathsf{there}/\!/$. For the tectogrammatical form to be able surface correctly, some drastic adjustments are necessary. We could alter the derivational proof by including non-lexical abstract constants, or refine the source types to impose (or enable) structural variation, but either option would be undermining the cohesion between deep syntax and semantics we started from. The only alternative that does not resort to contaminating the original term – abstract and pure – with

vulgar restrictions of form would be to lift the complexity of the interpretation and design it anew.

It seems therefore that the appealing simplicity and elegance of the tectogrammatic logic is counterbalanced by an increasingy bulky and cumbersome transition to the (equally simple, yet far less elegant) phenogrammatic logic. The problem is of course more pronounced for natural languages, which overstep the strict confines of their formal counterparts [Moot, 2014]. If only we had a way to keep just the good part of a type-driven and semantically transparent deep syntax, without having to get involved with all the tedious labour of its surface materialization or the translation to it... Spoiler alert: we will in a bit.

5.3 Other Formalisms

Type-logical and abstract categorial grammars have monopolized our interest, yet are not the only members of the categorial grammar family. For the sake of completeness and impartiality, we will briefly discuss two other major flavors and contrast them to the ones so far presented.

5.3.1 Combinatory Categorial Grammars

A deviant from the categorial tradition are the broadly adopted combinatory categorial grammars [Ades and Steedman, 1982; Szabolcsi, 1989; Steedman, 2022]. These stray from the norm by rejecting the very idea of the syntactic variable (and with it, hypothetical reasoning), citing reasons of cognitive plausibility and parsing complexity. Obviously, a categorial grammar stripped of hypothetical reasoning would not amount to much on its own: it would only be able to resolve syntactically flat sentences. To regain some of the lost expressivity (ideally, exactly and only as much as needed), combinatory categorial grammars incorporate a collection of rules lent from the combinatory logic of Curry et al. [1958], albeit in restricted form. The first such rule is morpholexical in nature; it allows lexical items to raise their types once, before administering them to the syntactic derivation, forcing a flip in the local function-argument structure and allowing different semantic scopes to take effect as/when needed. The remaining rules are essentially four instances of function composition – one for each unique pair of directional implications considered. The absence of hypothetical reasoning means that these are no longer derivable theorems of some underlying type theory, but ad-hoc schemata, fixed a priori to fit their designated purpose. To counteract overgeneration, these rules are made available only to a pre-defined subset of the sum of lexical types, empirically specified. With respect to the interface, a combinatory derivaton can be cast into a semantic λ term the usual way; by assigning to each rule a corresponding term constructor. Note, however, that this procedure is a non-invertible *transformation* rather than an isomorphic correspondence; the purity of the Curry-Howard correspondence is lost, traded away

for the aforementioned decrease in parsing complexity.

In spite of their (non-minor) differences, the agendas of multimodal type-logical grammars and combinatory categorial grammars are quite aligned, at least at a high level: they both stipulate the presence of syntactic universals that guide structure formation, utilize them as a pathway to semantics a là Montague, and acknowledge the need for language-specific syntactic fine-tuning; one exercising proof-theoretic control via unary type operators and structural rules, the other controlling the applicability of the so-called combinatory rules via lexical adjustment. For better or worse, combinatory categorial grammars have taken the lion's share of the practitioners' focus: they boast a vast expanse of tools and annotated corpora across languages that far exceeds that of their less popular siblings – to the point where the term categorial grammars has become an almost synonym of combinatory categorial grammars.[1] I hope but have no expectation that, by its end, this thesis will have slightly adjusted the scales towards a healthier epistemological pluralism.

5.3.2 Hybrid Type-Logical Grammars

In the lands between type-logical and their abstract siblings, there lives the strangeness of hybrid type-logical grammars. Originally proposed by Kubota and Levine [2012], hybrid type-logical grammars utilize a combination of the two slashes of traditional Lambek Calculi with the non-directional linear implication, to give birth to a type grammar that combines the good aspects of both approaches while purportedly suffering the restrictions of neither. The types of a hybrid type-logical grammar are the result of two stages of induction. The first stage creates standard Lambek types, as in (I.16). The second stage is of the form

$$\mathrm{A}, \mathrm{B}, \mathrm{C} := \mathrm{L} \mid \mathrm{A} \multimap \mathrm{B} \tag{I.45}$$

where L a valid Lambek type. The term calculus of hybrid type-logical grammars requires a translation of logical types into so-called *prosodic* types ST (for structure or string), such that all stage 1 (Lambek) types are sent to ST, and stage 2 types are inductively translated as (higher-order) functions over ST. The term constructors assigned to the elimination (resp. introduction) of the Lambek connectives is that of structure concatenation (resp. separation), whereas the term constructors assigned to the linear connective are standard function application and variable abstraction. The marriage of these two layers of abstraction in the same term calculus might seem unorthodox or at least aesthetically displeasing, but is not without merit. Concetenative terms allow the framework to relax the lexical pressure of an abstract categorial grammar by preserving the canonical categorial grammar treatment of local syntactic phenomena. Applicative terms constitute localized and controlled bursts of

[1] At least if one is to consider the reviewers I get assigned a reliable statistical sample of the NLP population.

ad-hoc expressivity that can exceptionally allow the derivation of higher-order and non-local phenomena normally inacessible to the OG Lambek calculi. Hybrid type-logical grammars are a relatively new addition to the family, and are subject to ongoing research, both from the linguistic and the proof-theoretic perspective [Kubota and Levine, 2020; Moot and Stevens-Guille, 2022]. As to why one would choose this setup over alternatives, your guess is as good as mine; hybrid proponents proclaim the system less obscure and more fit for linguistic applications [Kubota and Levine, 2020] – external validation is still pending.

6 Key References & Further Reading

Key references for this chapter were the Stanford Encyclopedia of Phisolophy entry on type-logical grammars [Moortgat, 2014] and the tried-and-true extended introduction books on λ-calculi and type theories of Sørensen and Urzyczyn [2006] and Pierce [2004]. Moral credit is owed to my once faithful travel companion, the categorial grammar bible of Moot and Retoré [2012]; it provides an accessible yet detailed documentation of most of the concepts hinted at in this chapter. Sections 1 and 2 draw heavily, both in content and in style, from the excellent tutorial paper of Wadler [1993] on linear type theory – waning presentational influences might be discernible up to Section 4.

If unhappy about this chapter ending, or unsatisfied with the exposition provided, here's some extra reading material to keep you company. For a detailed inquiry on proof nets and their linguistic applications, or an exemplar of what an actual great dissertation looks like, take a look at my co-supervisor's one [Moot, 2002]. For a more mathematically eloquent presentation of modalities and their potential as tools of inferential and structural reasoning, refer to the (also superb) dissertation of Bernardi [2002]. For a slightly outdated but still very educative overview of abstract categorial grammars, the lecture notes of Kanazawa and Pogodalla [2009] should prove handy. If your eco-conscious side was moved by linear logic, but you find yourself lacking the bravery of facing the original manuscript of Girard [1987], the lecture notes of Troelstra [1991] would make for a good alternative. If on the other hand you were intrigued about the vast expanse of type theories beyond the tiny scope of this thesis, the entry point to the downwards descent into the rabbit hole should be the seminal work of Martin-Löf [1982]. A convincingly easy-to-swallow application of such type theories in the formal semantics world is extensively summarized by Chatzikyriakidis and Luo [2020]. If you do like formal semantics but big lambdas give you nausea, there's a broad selection of books to go for; I still find myself guiltily cross-checking definitions and examples with that of Winter [2016] at times. Finally, if what caught your attention was the historical drama at the beginning of the chapter, you will enjoy reading about the history of constructivism by Troelstra [2011].

Chapter Bibliography

S. Abramsky. Computational interpretations of linear logic. *Theoretical computer science*, 111(1-2):3–57, 1993.

A. E. Ades and M. J. Steedman. On the order of words. *Linguistics and philosophy*, 4(4):517–558, 1982.

K. Ajdukiewicz. Die syntaktische konnexitat. *Studia philosophica*, pages 1–27, 1935.

E. Bach. An extension of classical transformational grammar. 1976.

C. F. Baker, C. J. Fillmore, and J. B. Lowe. The berkeley framenet project. In *COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics*, 1998.

Y. Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29(1):47–58, 1953.

H. P. Barendregt et al. *The lambda calculus*, volume 3. North-Holland Amsterdam, 1984.

R. A. Bernardi. *Reasoning with polarity in categorial type logic*. PhD thesis, 2002.

S. Chatzikyriakidis and Z. Luo. *Formal Semantics in Modern Type Theories*. John Wiley & Sons, 2020.

A. Church. A formulation of the simple theory of types. *The journal of symbolic logic*, 5(2):56–68, 1940.

A. D. Correia. *Quantum distributional semantics: Quantum algorithms applied to natural language processing*. PhD thesis, Utrecht University, 2022.

H. B. Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Sciences*, 20(11):584–590, 1934.

H. B. Curry. Some logical aspects of grammatical structure. *Structure of language and its mathematical aspects*, 12:56–68, 1961.

H. B. Curry, R. Feys, W. Craig, J. R. Hindley, and J. P. Seldin. *Combinatory logic*, volume 1. North-Holland Amsterdam, 1958.

V. Danos and L. Regnier. The structure of multiplicatives. *Archive for Mathematical logic*, 28(3):181–203, 1989.

N. G. de Bruijn. Automath, a language for mathematics. In *Automation of Reasoning*, pages 159–200. Springer, 1983. Original manuscript from 1968.

P. de Groote. On the strong normalization of natural deduction with permutation-conversions. In *International Conference on Rewriting Techniques and Applications*, pages 45–59. Springer, 1999.

P. de Groote. Towards abstract categorial grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 252–259, 2001.

P. de Groote and S. Pogodalla. On the expressive power of abstract categorial grammars: Representing context-free formalisms. *Journal of Logic, Language and Information*, 13(4):421–438, 2004.

J.-Y. Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. PhD thesis, Éditeur inconnu, 1972.

J.-Y. Girard. Linear logic. *Theoretical computer science*, 50(1):1–101, 1987.

S. Guerrini. A linear algorithm for mll proof net correctness and sequentialization. *Theoretical Computer Science*, 412(20):1958–1978, 2011.

P. Hendriks. Ellipsis and multimodal categorial type logic. In *Proceedings of Formal Grammar*, pages 107–122. Citeseer, 1995.

A. Heyting. Die formalen regeln der intuitionistischen logik. *Sitzungsbericht PreuBische Akademie der Wissenschaften Berlin, physikalisch-mathematische Klasse II*, pages 42–56, 1930.

W. A. Howard. The formulae-as-types notion of construction. *To HB Curry: essays on combinatory logic, lambda calculus and formalism*, 44:479–490, 1980. Original manuscript from 1969.

T. Janssen. *Foundations and applications of Montague grammar*. PhD thesis, University of Amsterdam, 2014. Original publication date 1983.

M. Kanazawa and S. Pogodalla. Advances in abstract categorial grammars: Language theory and linguistic modeling. *Lecture notes, ESSLLI*, 9, 2009.

S. A. Kripke. Semantical analysis of modal logic i normal modal propositional calculi. *Mathematical Logic Quarterly*, 9(5-6):67–96, 1963.

Y. Kubota and R. Levine. Gapping as like-category coordination. In *International Conference on Logical Aspects of Computational Linguistics*, pages 135–150. Springer, 2012.

Y. Kubota and R. D. Levine. *Type-logical syntax*. MIT Press, 2020.

N. Kurtonina and M. Moortgat. Structural control.

J. Lambek. The mathematics of sentence structure. *The American Mathematical Monthly*, 65(3):154–170, 1958.

J. Lambek. On the calculus of syntactic types. *Structure of language and its mathematical aspects*, 12:166–178, 1961.

P. Martin-Löf. Constructive mathematics and computer programming. In *Studies in Logic and the Foundations of Mathematics*, volume 104, pages 153–175. Elsevier, 1982.

R. Montague. Universal grammar. *Theoria*, 36(3):373–398, 1970.

M. Moortgat. Multimodal linguistic inference. *Journal of Logic, Language and Information*, 5(3):349–385, 1996.

M. Moortgat. Categorial type logics. In *Handbook of logic and language*, pages 93–177. Elsevier, 1997.

M. Moortgat. Constants of grammatical reasoning. *Constraints and resources in natural language syntax and semantics*, pages 195–219, 1999.

M. Moortgat. Typelogical Grammar. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2014 edition, 2014.

R. Moot. Hybrid type-logical grammars, first-order linear logic and the descriptive inadequacy of lambda grammars. *arXiv preprint arXiv:1405.6678*, 2014.

R. Moot and C. Retoré. *The logic of categorial grammars: a deductive account of natural language syntax and semantics*, volume 6850. Springer, 2012.

R. Moot and S. J. Stevens-Guille. Logical foundations for hybrid type-logical grammars. *Journal of Logic, Language and Information*, 31(1):35–76, 2022.

R. C. A. Moot. *Proof nets for linguistic analysis*. PhD thesis, 2002.

G. Morrill. Type logical grammar: Categorial logic of signs. 1994.

A. S. Murawski and C.-H. Ong. Dominator trees and fast verification of proof nets. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No. 99CB36332)*, pages 181–191. IEEE, 2000.

R. Muskens. Lambda grammars and the syntax-semantics interface. 2001.

B. Partee et al. Compositionality. *Varieties of formal semantics*, 3:281–311, 1984.

B. C. Pierce. *Advanced topics in types and programming languages*. MIT press, 2004.

D. Prawitz. A proof-theoretical study. *Uppsala: Almqvist & Wilkselll*, 1965.

J. C. Reynolds. Towards a theory of type structure. In *Programming Symposium*, pages 408–425. Springer, 1974.

B. Russell. Mathematical logic as based on the theory of types. *American journal of mathematics*, 30(3):222–262, 1908.

M. H. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard isomorphism*. Elsevier, 2006.

M. Steedman. Combinatory categorial grammar. 2022.

A. Szabolcsi. Bound variables in syntax (are there any?). *Semantics and contextual expression*, 295:318, 1989.

A. S. Troelstra. Lectures on linear logic. 1991.

A. S. Troelstra. History of constructivism in the 20th century. *Set Theory, Arithmetic, and Foundations of Mathematics*, pages 150–179, 2011.

A. S. Troelstra and H. Schwichtenberg. *Basic proof theory*. Number 43. Cambridge University Press, 2000.

J. van Benthem. *The semantics of variety in categorial grammar*. John Benjamins, 1988.

W. Vermaat. *Controlling movement*. PhD thesis, 1999.

P. Wadler. A taste of linear logic. In *International Symposium on Mathematical Foundations of Computer Science*, pages 185–210. Springer, 1993.

H. Wansing. Formulas-as-types for a hierarchy of sublogics of intuitionistic propositional logic. In *Workshop on Nonclassical Logics and Information Processing*, pages 125–145. Springer, 1990.

H. Wansing. Sequent systems for modal logics. *Handbook of philosophical logic*, pages 61–145, 2002.

Y. Winter. *Elements of formal semantics: An introduction to the mathematical theory of meaning in natural language*. Edinburgh University Press, 2016.

CHAPTER II

Chapter 2: Title Pending