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).

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 n^1 , in which case: shoo, and shame on you, future me. Otherwise, could it even be that you are actually interested in the subject matter of this thesis? Wooo, exciting 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 of-

[1] Once 4, now 5, soon to be 3; left to range over $\mathbb{N}$ to still make sense after the imminent tech collapse.

fered me a glimpse of a whole new world. I got to see that proofs are not irrelevant 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

The quote below, received almost verbatim as a review, adequately summarizes the contents of this thesis (mean-spirited as it may be):

> [The thesis] starts with Lambek Calculus, some how uses dependency labels in some of its semantic types, provides a parsing algorithm for it; there are neural networks and vectors used and some accuracy results provided, but I am still unsure about the contributions [of the thesis] and their relevance.
>
> Unknown reviewer, 2019.

CHAPTER I

Introduction

In the beginning there was the word, and the word had a TYPE.

Our story begins with the (over-ambitious, in hindsight) musings 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 Earth, 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 $\neg A$ ($A \vee \neg A$), whereas the second that a double negation is equivalent to an identity ($\neg\neg A \equiv A$). 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:

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

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

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

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 $\Gamma \vdash A$. 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

[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.

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.

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 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 indeed describe distinct derivation procedures, or trivial. Trivial variations come

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

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

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

(a) Logical rules.

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

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

(b) Structural rules.

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

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,

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

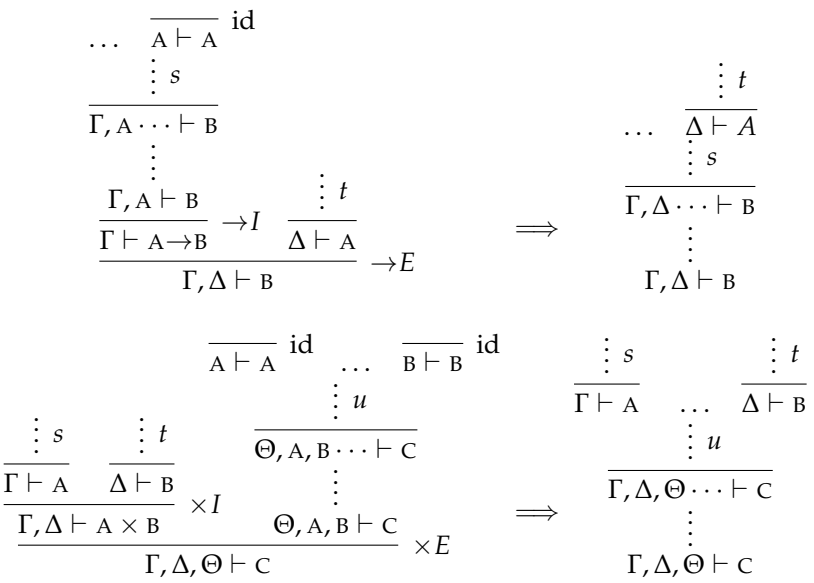


Figure I.2: Intuitionistic β redexes.

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 Nicolaas Govert 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 alternative 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 now denote a *typing environment*:

$$\mathsf{x}_j : \mathrm{A}_1,\ \mathsf{x}_k : \mathrm{A}_2 \ldots\ \mathsf{x}_n : \mathrm{A}_n \tag{I.3}$$

i.e. rather than a sequence of formulas, we have a sequence of distinct variables, each of a specific type, and a judgement $\Gamma \vdash \mathsf{s} : \mathrm{B}$ 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 $\mathsf{s}\,\mathsf{t}$ that denotes the function application of *functor* s on *argument* t. Function application is left-associative: $\mathsf{s}\,\mathsf{t}\,\mathsf{u}$ is the bracket-economic presentation of $(\mathsf{s}\,\mathsf{t})\,\mathsf{u}$ – we have no choice but to use brackets if want to instead denote $\mathsf{s}\,(\mathsf{t}\,\mathsf{u})$. 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 $\lambda\mathsf{x}_i.\mathsf{s}$. 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, $\times E$, 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*, $\mathsf{s}_{[\mathsf{x}_i \mapsto \mathsf{t}]}$, 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

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

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

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

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

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

Figure I.3: Simple type theory.

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 but useful trick people 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 $\mathsf{s}_{[\mathsf{x}_i \mapsto \mathsf{x}_j]}$ 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 conversions, e.g.

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

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:

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

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

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

[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.

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.

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. Both directions are facets of η conversion – the equivalence relation enacted by this conversion is called η equivalence.

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

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

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

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

Table I.2: Polarity induction.

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

Figure I.4: Type raising.

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). The two judgements of the next paragraph have their subformulas annotated with a superscript denoting their polarity, for illustrative purposes.

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:

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

Types whose order is above 1 are called *higher-order* types; they denote functions that accept functions as their arguments. For instance, for p and s atomic propositions of order 0, their respective identity functions $p{\to}p$ and $s{\to}s$ are of order 1, and the raised form of p into s, i.e. $(p{\to}s){\to}s$, is of order 2.

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:

$$\frac{}{\mathrm{x}_i : \mathrm{A} \vdash \mathrm{x}_i : \mathrm{A}}\ \mathrm{id} \quad =: \quad \frac{}{\mathrm{x}_i : \mathrm{A}}\ \mathrm{id}$$

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.

$$\dfrac{\dfrac{}{\mathsf{x}_i : (\mathrm{A} \times \mathrm{B}) \to \mathrm{C}}\ \mathrm{id} \quad \dfrac{\dfrac{}{\mathsf{x}_j : \mathrm{A}}\ \mathrm{id} \quad \dfrac{}{\mathsf{x}_k : \mathrm{B}}\ \mathrm{id}}{\mathsf{x}_j, \mathsf{x}_k \vdash (\mathsf{x}_j, \mathsf{x}_k) : \mathrm{A} \times \mathrm{B}}\ \times I}{\mathsf{x}_i, \mathsf{x}_j, \mathsf{x}_k \vdash \mathsf{x}_i\ (\mathsf{x}_j, \mathsf{x}_k) : \mathrm{C}}\ \to E}{\mathsf{x}_i \vdash \lambda \mathsf{x}_j \mathsf{x}_k . \mathsf{x}_i\ (\mathsf{x}_j, \mathsf{x}_k) : \mathrm{A} \to \mathrm{B} \to \mathrm{C}}\ \to I(x2)$$

(a) Currying

$$\dfrac{\dfrac{\dfrac{}{\mathsf{x}_i : \mathrm{A} \times \mathrm{B}}\ \mathrm{id} \quad \dfrac{\dfrac{\dfrac{}{\mathsf{x}_j : \mathrm{A} \to \mathrm{B} \to \mathrm{C}}\ \mathrm{id} \quad \dfrac{}{\mathsf{x}_k : \mathrm{A}}\ \mathrm{id}}{\mathsf{x}_j, \mathsf{x}_k \vdash \mathsf{x}_j\ \mathsf{x}_k : \mathrm{B} \to \mathrm{C}}\ \to E \quad \dfrac{}{\mathsf{x}_l : \mathrm{B}}\ \mathrm{id}}{\mathsf{x}_j, \mathsf{x}_k, \mathsf{x}_l \vdash \mathsf{x}_j\ \mathsf{x}_k\ \mathsf{x}_3 : \mathrm{C}}\ \to E}{\mathsf{x}_i, \mathsf{x}_j \vdash \mathsf{case}\ \mathsf{x}_i\ \mathsf{of}\ (\mathsf{x}_j, \mathsf{x}_k)\ \mathsf{in}\ \mathsf{x}_j\ \mathsf{x}_k\ \mathsf{x}_l : \mathrm{C}}\ \times E}{\mathsf{x}_j, \mathsf{x}_i \vdash \mathsf{case}\ \mathsf{x}_i\ \mathsf{of}\ (\mathsf{x}_j, \mathsf{x}_k)\ \mathsf{in}\ \mathsf{x}_j\ \mathsf{x}_k\ \mathsf{x}_l : \mathrm{C}}\ \mathrm{ex}}{\mathsf{x}_j \vdash \lambda \mathsf{x}_i . \mathsf{case}\ \mathsf{x}_i\ \mathsf{of}\ (\mathsf{x}_j, \mathsf{x}_k)\ \mathsf{in}\ \mathsf{x}_j\ \mathsf{x}_k\ \mathsf{x}_l : (\mathrm{A} \times \mathrm{B}) \to \mathrm{C}}\ \to I$$

(b) Uncurrying

Figure I.5: Interderivability of product and arrow.

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

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

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

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

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

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

(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.

the additive &. The first denotes a linear tuple, and $A \otimes B$ 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: $\langle s, t \rangle$ 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.

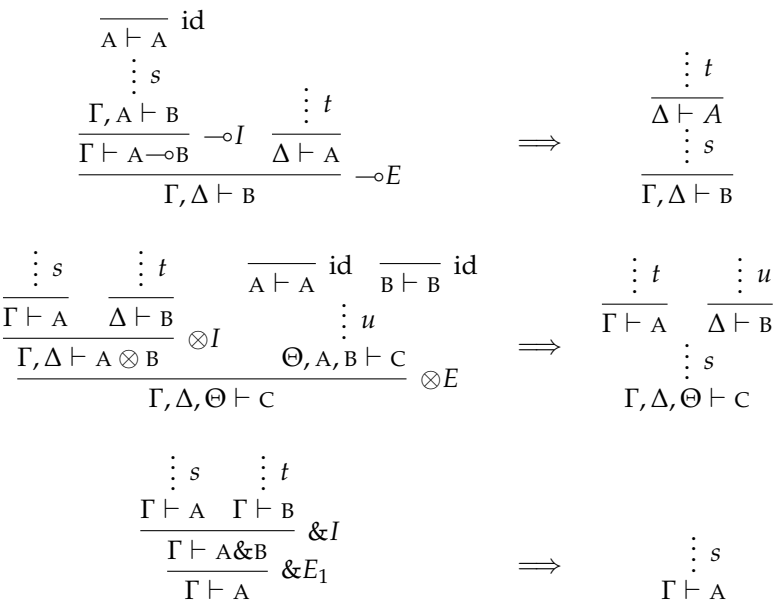


Figure I.7: Linear β redexes.

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 difference 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:

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

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

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

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

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x_j : B \multimap C}\ \text{id} \quad \dfrac{\dfrac{}{x_i : A \multimap B}\ \text{id} \quad \dfrac{}{x_k : A}\ \text{id}}{x_i, x_k \vdash x_i\ x_k : B}\ \multimap E}{x_j, x_i, x_k \vdash x_k\ (x_j\ x_i) : A \multimap C}\ \multimap E}{x_j, x_i \vdash \lambda x_k.x_j\ (x_i\ x_k) : A \multimap C}\ \multimap I}{x_i, x_j \vdash \lambda x_k.x_j\ (x_i\ x_k) : A \multimap C}\ \text{ex}$$

Figure I.8: Linear function composition.

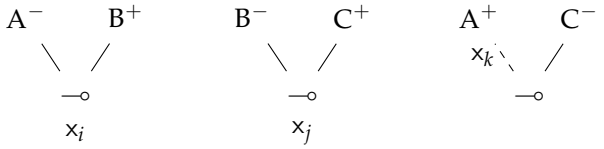
2.2 Proof Nets

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 explication 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.

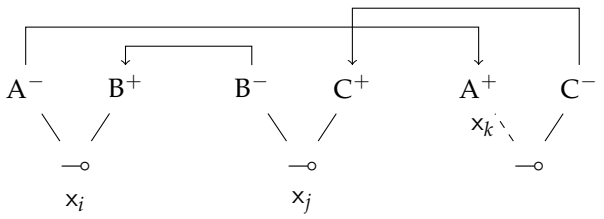
In natural deduction, our proofs are built sequentially. We start with some hypothesized variables and combine them via rules to derive more complex terms, which then serve as premises for a next iteration of rule applications. As long as we are careful not to get stuck in some detour loop hell, we rinse 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 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



(a) Proof frame.



(b) Proof structure.

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

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

[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.

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 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 will then 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 its dashed edge. 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_k, so we start by writing down $\lambda x_k.(\dots)$. 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,

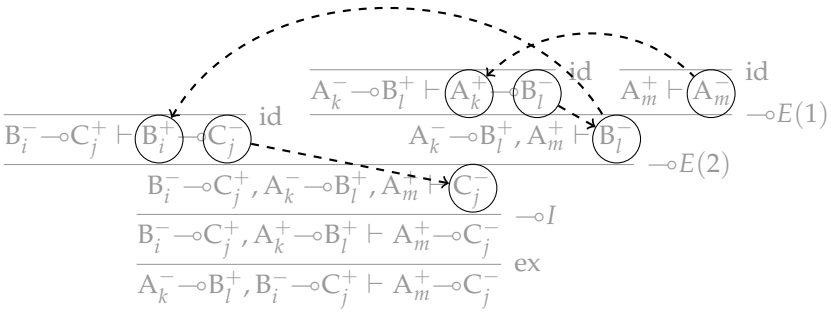


Figure I.10: Extracting axiom links from the proof of Figure I.8.

getting $\lambda x_k.x_j(\dots)$. We proceed in negatative mode to B^-, cross over to B^+ and repeat the above, getting $\lambda x_k.x_j\ (x_i\ \dots)$ before going into negative mode again. The final axiom link points us to A^+, which is its own root, named x_k. At this point, our traversal has transcribed the term $\lambda x_k.x_j\ (x_i\ x_k)$ 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 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:

$$\mathrm{A}_k^- \multimap \mathrm{B}_i^+, \mathrm{B}_i^- \multimap \mathrm{C}_j^+ \vdash \mathrm{A}_k^+ \multimap \mathrm{C}_j^-$$

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 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[2]: 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**

[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

$$\left(\lambda \mathrm{x}_i.\mathrm{x}_i^{\mathrm{A_i} \multimap \mathrm{B_j}}\ \mathrm{x}_j^{\mathrm{A_k}}\right)\ \mathrm{x}_k^{\mathrm{A_l} \multimap \mathrm{B_m}}$$

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

[2] Or is there?

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 flavoured 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 the key feature of interest lies in the two implications, $/$ and $\backslash$. 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 $\Gamma \vdash \mathrm{A} \otimes \mathrm{B}$ 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

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

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

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

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

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

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 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

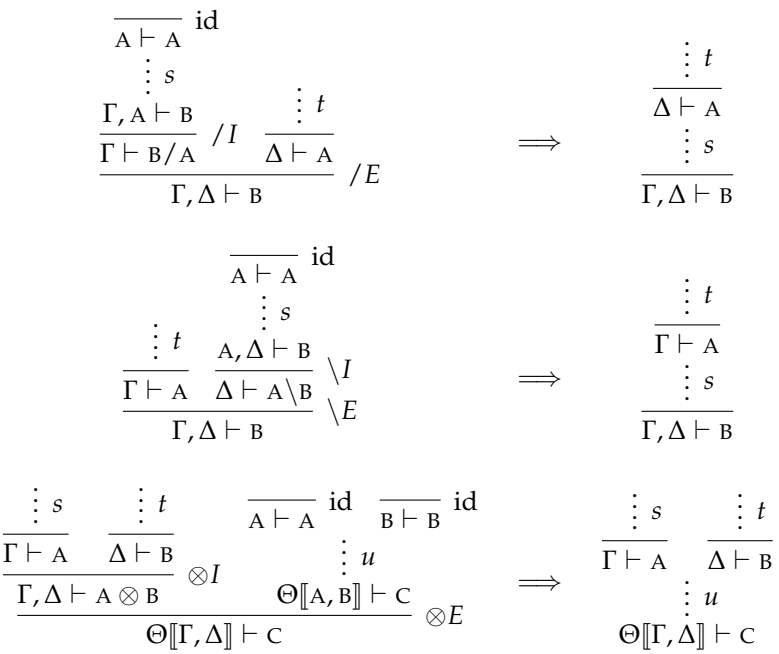


Figure I.12: Lambek β redexes.

$\otimes E$. The corresponding term reductions are:

$$(\lambda \mathsf{x}_i.\mathsf{s}) \triangleleft \mathsf{t} \overset{\beta}{\rightsquigarrow} \mathsf{s}_{[\mathsf{x}_i \mapsto \mathsf{t}]} \tag{I.17}$$

$$\mathsf{t} \triangleright (\curlywedge \mathsf{x}_i.\mathsf{s}) \overset{\beta}{\rightsquigarrow} \mathsf{s}_{[\mathsf{x}_i \mapsto \mathsf{t}]} \tag{I.18}$$

$$\text{case } (\mathsf{s}, \mathsf{t}) \text{ of } (\mathsf{x}_i, \mathsf{x}_j) \text{ in } \mathsf{u} \overset{\beta}{\rightsquigarrow} \mathsf{u}_{[\mathsf{s} \mapsto \mathsf{x}_i, \mathsf{t} \mapsto \mathsf{x}_j]} \tag{I.19}$$

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

$$\frac{}{\mathsf{x}_i : \mathrm{A} \vdash \mathsf{x}_i : \mathrm{A}}\ \mathrm{id}$$

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

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

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

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

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

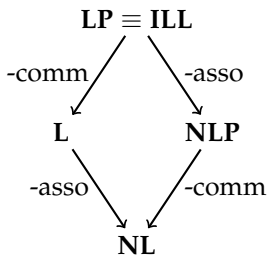


Figure I.14: **(N)L(P)**: **ILL** and substructural friends.

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 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 (also encountered as the Lambek-van Benthem Calculus [van Benthem, 1988]). 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.

[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).

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 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] In the case for checking the satisfiability of a judgement (i.e. searching for *any* valid proof), all the above would have made for potential proof candidates; in the case for attempting to enumerate the proofs of a judgement (i.e. searching for *all* valid proof), they would all have needed to be exhausted. 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 flavour, 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 different 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.

4.1.1 Proof & Term Reductions

The proof patterns of Figure I.16a 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:

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

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

The dual direction of η equivalences also holds – since these are discovered in the literature less frequently than the more pedestrian implication

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

$$\dfrac{\dfrac{\dfrac{}{\mathrm{A} \vdash \mathrm{A}}\,\mathrm{id}}{\vdots\, s}}{\Gamma[\![\langle \mathrm{A}\rangle]\!] \vdash \mathrm{B}} \quad \dfrac{\begin{array}{c}\vdots\, t\\ \Delta \vdash \mathrm{A}\end{array}}{\langle\Delta\rangle \vdash \Diamond\mathrm{A}}\,\Diamond I}{\Gamma[\![\langle\Delta\rangle]\!] \vdash \mathrm{B}}\,\Diamond E \quad\Longrightarrow\quad \begin{array}{c}\vdots\, t\\ \Delta \vdash \mathrm{A}\\ \vdots\, s\\ \Gamma[\![\langle\Delta\rangle]\!] \vdash \mathrm{B}\end{array}$$

(a) Modal β redexes.

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

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

(b) Modal η redexes.

Figure I.16: Modal redexes

and product equivalences, we explictly present them in Figure I.16b. The term equivalences they materialize are:

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

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

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 meta-syntax notation of term substitution. For instance, $\Diamond E$ can often be spotted in the wild as:

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

In this disguise, the rule is again seen as realizing a retroactive substitution of x_i with $\triangledown \mathsf{t}$, 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 $\triangledown \mathsf{t}$). 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.17 presents the proof transformations equivalent to the properties

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

(a) Monotonicity of the diamond.

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

(b) Monotonicity of the box.

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

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

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

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

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

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

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

(f) Ditto, the other way around.

Figure I.17: Derivations for the various aspects of residuation.

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.18 presents proofs of the above 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.19a (and the corresponding tree transformations of Figure I.19b, 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 (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

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

(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 $\Diamond(\_)$.

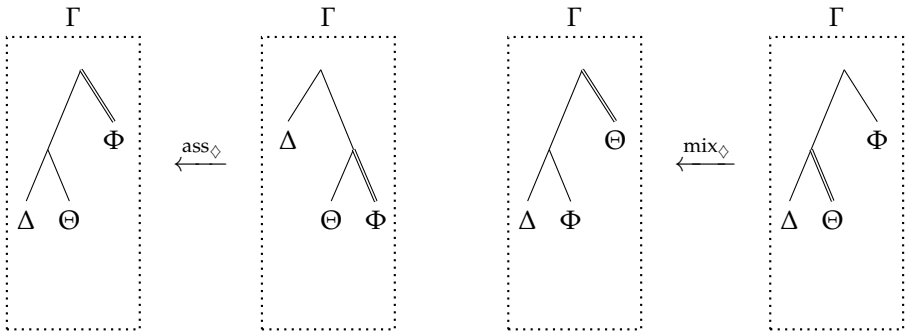
$$\dfrac{\dfrac{\dfrac{\dfrac{}{\text{(I.17c)}}}{x_i : A \vdash \blacktriangle\triangle x_i : \Box\Diamond A}}{\langle x_i : A\rangle \vdash \triangle\blacktriangle\triangle x_i : \Diamond\Box\Diamond A}\,\Diamond I \quad \dfrac{}{x_j : \Diamond A \vdash x_j : \Diamond A}\,\mathrm{id}}{x_j : \Diamond A \vdash \text{case } \triangledown x_j \text{ of } x_i \text{ in } \triangle\blacktriangle\triangle x_i : \Diamond\Box\Diamond A}\,\Diamond E$$

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

Figure I.18: The triple laws for the two modalities in both directions.

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

(a) In rule format.



(b) Corresponding tree transformations. Double edges denote bracketed substructures.

Figure I.19: Controlled associativity/mixed commutativity.

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 usually tied to a logic, commonly a choice from the ones reviewed so far (or at least loosely inspired by one). 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 grammar 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 point of attraction 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 flavour that is to be developed and presented later in this thesis.

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
Deduction	Program Synthesis	Parsing

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

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: 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; our motto shall from now on be *parsing as deduction*.

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 illustrative purposes, this could look like:

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

for a grammar able to reason about nouns N, noun phrases and bare nouns NP, sentential clauses $\mathrm{S}_{\mathrm{main}}$, 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 intran-

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.

sitive phrase, for instance, is a grammatical function that would consume a left-adjacent noun phrase to produce a sentence, therefore it would materialize as $NP\backslash S_{main}$. It follows that a transitive phrase or copula would then be of type $(NP\backslash S_{main})/NP$, a function that requires a right-adjacent noun phrase to produce an intransitive phrase, whereas a bitransitive, requiring two, would be $((NP\backslash S_{main})/NP)/NP$, 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 – (NP/NP)/(NP/NP) for adjectival and (NP\S)\(NP\S) for verbal modification, respectively.

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

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

(a) Derivation for *strange dolphins*.

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

(b) Derivation for *the eye*.

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

(c) Derivation for *litten by suns*.

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

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

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

(e) Derivation for *opiate oceans poured there*.

Figure I.20: Deriving simple multiplicative phrases in **NL**.

rememberable 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.20 presents derivations for parts of the goal phrase, and our very first linguistic examples (!) – their purely applicative nature should make them straightforward to decipher. The two proofs of I.20e and I.20c 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

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

(a) Extracting a hypothetical postnominal modifier...

$$\dfrac{\dfrac{\text{(I.21a)}}{((\dots)\cdot(\dots))\cdot\langle x_i\rangle\vdash \text{S}} \quad \dfrac{\dfrac{}{\text{litten}:\Diamond\Box(\text{NP}\backslash\text{NP})/\text{PP}}\,\text{lex} \quad \dfrac{\dfrac{}{\text{by}:\text{PP}/\text{NP}}\,\text{lex} \quad \dfrac{}{\text{suns}:\text{NP}}\,\text{lex}}{\text{by}\cdot\text{suns}\vdash \text{PP}}\,/E}{\text{litten}\cdot(\text{by}\cdot\text{suns})\vdash\Diamond\Box(\text{NP}\backslash\text{NP})}\,\Diamond E}{((\text{opiate}\cdot\text{oceans})\cdot(\text{poured}\cdot\text{there}))\cdot(\text{litten}\cdot(\text{by}\cdot\text{suns}))\vdash \text{S}}$$

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

Figure I.21: Deriving long-distance postnominal modification with the aid of type assignment (I.33).

grammars [Hendriks, 1995; Moortgat, 1996; Kurtonina and Moortgat, 1997; 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.22 progresses our agenda by accounting for the presence of a (hypothetical) movable postnominal modifier via the rules of Figure I.19. 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.21 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

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

contain a subordinate sentence with a *gap*, which can vary 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 vary in its position* contains a gap in the subject position, which the head *a gap* occupies 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:

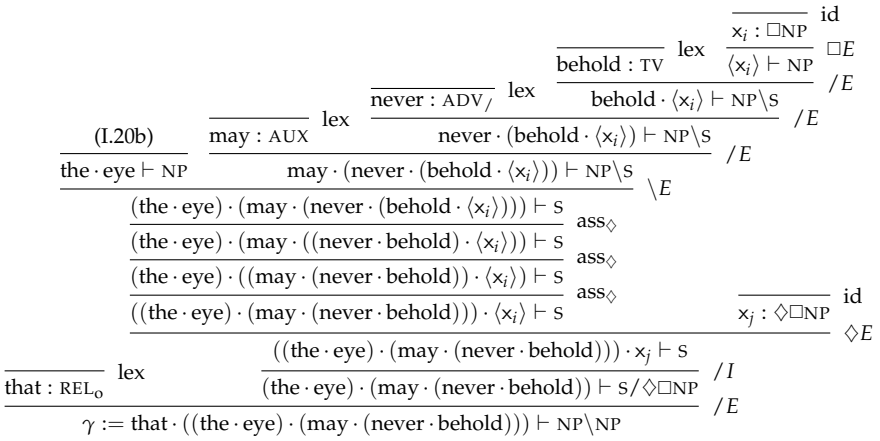
$$\text{that} :: \text{REL}_\text{S} := (\text{NP}\backslash\text{NP})/(\text{NP}\backslash\text{S}) \tag{I.34}$$

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

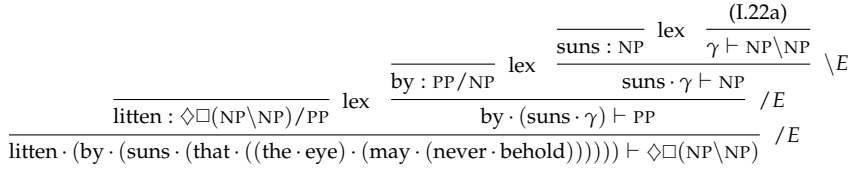
The second version launches a mobile NP hypothesis via the same diamond-box pattern showcased earlier. The proof of Figure I.22 employs this typing in combination with the $\text{ass}_\Diamond$ rule to derive the object-relative clause *that the eye may never behold*, which applied to *suns* and combined with the proof of Figure I.21 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 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 flavour 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 neces-



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



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

Figure I.22: An object-relative clause in action, prompted by type assignment (I.35).

sarily 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 knowledge necessary to finally tackle the full derivation. First, we must instanti-

[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 meta-theory *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].

$$\dfrac{\dfrac{\text{(I.20a)}}{\text{strange} \cdot \text{dolphins} \vdash \text{NP}} \quad \dfrac{\dfrac{}{\text{and} : (\text{NP}\backslash\text{NP})/\text{NP}}\ \text{lex} \quad \dfrac{\text{(I.20d)}}{\text{sea-nymphs} \cdot (\text{of} \cdot (\text{unrememberable} \cdot \text{deeps})) \vdash \text{NP}}}{\text{and} \cdot (\text{sea-nymphs} \cdot (\text{of} \cdot (\text{unrememberable} \cdot \text{deeps}))) \vdash \text{NP}\backslash\text{NP}}\ /E}{\delta := (\text{strange} \cdot \text{dolphins}) \cdot (\text{and} \cdot (\text{sea-nymphs} \cdot (\text{of} \cdot (\text{unrememberable} \cdot \text{deeps})))) \vdash \text{NP}}\ \backslash E$$

(a) Deriving noun-phrase coordination...

$$\dfrac{\dfrac{\dfrac{}{\text{having} : (\Diamond\Box(\text{NP}\backslash\text{NP})/\text{NP})/\text{PP}}\ \text{lex} \quad \dfrac{\dfrac{}{\text{in} : \text{PP}/\text{NP}}\ \text{lex} \quad \dfrac{\dfrac{}{\text{their} : \text{NP}/\text{NP}}\ \text{lex} \quad \dfrac{}{\text{whirlpools} : \text{NP}}\ \text{lex}}{\text{their} \cdot \text{whirlpools} \vdash \text{NP}}\ /E}{\text{in} \cdot (\text{their} \cdot \text{whirlpools}) \vdash \text{PP}}\ /E}{\text{having} \cdot (\text{in} \cdot (\text{their} \cdot \text{whirlpools})) \vdash \Diamond\Box(\text{NP}\backslash\text{NP})/\text{NP}} \quad \dfrac{\text{(I.23a)}}{\delta \vdash \text{NP}}}{(\text{having} \cdot (\text{in} \cdot (\text{their} \cdot \text{whirlpools}))) \cdot \delta) \vdash \Diamond\Box(\text{NP}\backslash\text{NP})}\ /E$$

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

Figure I.23: Filling in the missing bits using the polymorphic type (I.38).

ate the polymorphic coordinator once by substituting χ for NP to derive the noun phrase conjunction *strange dolphins and sea-nymphs of unrememememberable deeps*, as portrayed in Figure I.23a. 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.23b. 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.21. 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.

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

The rules above showcase the introduction and elimination of types quantified over types $\Pi\alpha.\sigma$, and the term analogue of abstracting over types $\lambda\alpha.M$. 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.23a:

i. $\text{strange} \cdot (\text{dolphins} \cdot (\text{and} \cdot (\text{sea-nymphs} \cdot (\text{of} \cdot (\text{unrememberable} \cdot \text{deeps})))))$
ii. $\text{strange} \cdot ((\text{dolphins} \cdot (\text{and} \cdot \text{sea-nymphs})) \cdot (\text{of} \cdot (\text{unrememberable} \cdot \text{deeps})))$
iii. $(\text{strange} \cdot (\text{dolphins} \cdot (\text{and} \cdot \text{sea-nymphs}))) \cdot (\text{of} \cdot (\text{unrememberable} \cdot \text{deeps}))$
iv. $((\text{strange} \cdot \text{dolphins}) \cdot (\text{and} \cdot \text{sea-nymphs})) \cdot (\text{of} \cdot (\text{unrememberable} \cdot \text{deeps}))$
v. $(\text{strange} \cdot \text{dolphins}) \cdot (\text{and} \cdot (\text{sea-nymphs} \cdot (\text{of} \cdot (\text{unrememberable} \cdot \text{deeps})))))$

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). Derivational ambiguity is not necessarily bad, provided the divergence in the proofs constructed is linguistically meaningful.[1] 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, pre-bracketed 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:

> 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.

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

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. 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

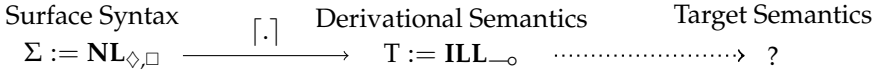


Figure I.24: The syntax-semantics interface in the type-logical setting.

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.22 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 $\mathbf{ILL}_{\multimap}$.

With the Curry-Howard isomorphism as our guiding star, there's no peril in navigating between syntactic and semantic theories. 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 interactions 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:

$$\lambda x_i x_j x_k . x_i \ (x_j \ x_k) :: (*\multimap *)\multimap(*\multimap *)\multimap * \multimap * \tag{I.40}$$

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 arbitrary set), to deliver the recipe:

$$\lambda x_i x_j x_k . (x_i \ x_k) \wedge (x_j \ x_k) :: (*\to *)\to(*\to *)\to * \to * \tag{I.41}$$

[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 neither pitching some provocative theory of conjunction semantics here, nor secretly advocating for the dot-combinator – just trying to make a point.

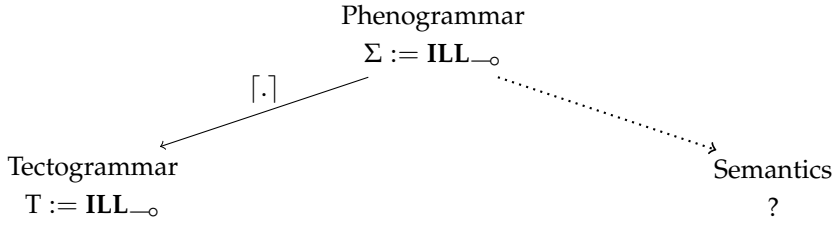


Figure I.25: The syntax-semantics interface in the abstract categorial setting.

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 $\mathbf{ILL}_{\multimap}$ 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 $\mathbf{ILL}_{\multimap}$, 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 $\mathsf{Cons}^{\mathrm{T}}$ 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):

$$+ := \lambda \mathsf{x}_i^{\mathtt{str}} \mathsf{x}_j^{\mathtt{str}} \mathsf{x}_k^{*}.\mathsf{x}_i\ (\mathsf{x}_j\ \mathsf{x}_k) \tag{I.42}$$

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 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.26 by all abstract constants denoting base "types"[2] and "type" constructors. The target logic T will be our phenogrammatic printer

[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.

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

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

Figure I.26: 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.26.

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 $\eta_0(\textsc{Type}) = \mathtt{str}$, the type alias of $* \multimap *$. 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.27 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 function/argument brackets substituted with indendation levels for

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

Figure I.27: Constructing the type assignment of (I.33).

legibility):

$$\begin{aligned}
&\lceil \mathsf{rdiv\ (dia\ box\ (ldiv\ np\ np)))\ pp} \rceil \\
&= \lambda x_i x_j.\underline{(} + x_j + \underline{/} + x_i + \underline{)} \\
&\quad \underline{\mathrm{PP}} \\
&\quad \lambda x_k.\underline{\Diamond} + x_k \\
&\quad\quad \lambda x_l.\underline{\Box} + x_l \\
&\quad\quad\quad \lambda x_m x_n.\underline{(} + x_m + \underline{\backslash} + x_n + \underline{)} \\
&\quad\quad\quad\quad \underline{\mathrm{NP}} \\
&\quad\quad\quad\quad \underline{\mathrm{NP}} \\
&\overset{\beta*}{\rightsquigarrow} \underline{(\Diamond\Box(\mathrm{NP}\backslash\mathrm{NP})/\mathrm{PP})}
\end{aligned} \tag{I.43}$$

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 their meta-theoretical enterprises, 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 $\mathbf{ILL}_{\multimap}$, 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 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 $\mathsf{Prop}_0^{\mathsf{T}} = \{*\}$; all abstract atoms atoms are then sent to $\mathtt{str}$. For each word, we will need an abstract constant $\mathsf{c} \in \mathsf{Cons}^{\Sigma}$, and a corresponding object constant $/\!/\mathsf{c}/\!/ :: \mathtt{str} \in \mathsf{Cons}^{\mathsf{T}}$, 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 $\mathtt{str}$ to obtain them).

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

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

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

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

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 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 to 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 flavours 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 *à la* 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 an assortment of tools and annotated corpora across languages, the size of which 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 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 grammars 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, Utrecht Institute of Linguistics OTS, Utrecht University, 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. *Specifying syntactic structures*, pages 75–113, 1997.

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, Utrecht Institute of Linguistics OTS, Utrecht University, 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.

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

W. Vermaat. *Controlling movement*. PhD thesis, Utrecht Institute of Linguistics OTS, Utrecht University, 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

Typing Dependency Structure

Predicates are functors,
complements – diamonds,
adjuncts – boxes;
Everything a type.

The previous chapter initiated us into the history-rich world of substructural logics in the intuitionistic tradition. Along the (artificially homogenized) story, we got to dip our toes into linguistic waters, where we saw these logics thrive and prosper, finding their place as the foundation for categorial grammars. The many flavours of categorial grammars all have a single common denominator: they treat syntax as a hierarchical structure that puts phrases together from small to big, starting from words and reaching up to the sentence, the imprint being a natural deduction tree (and perhaps a phrasal bracketing structure). This emphasis on the phrase, combined with the distinctive shape of the categorial parse, allows for a partial parallel to be drawn between categorial grammars and phrase structure grammars, despite their stark methodological and theoretical contrasts. Phrase structure grammars are rule-based systems that assign categories to phrases according to their syntactic function, and manipulate phrasal formation by specifying how their constituent parts combine – the produce being a bracketing structure, commonly visualized in tree format. A different approach to grammatical theory abandons the constituency relation, adopting the dependency relation in its stead. Dependency relations do not seem compatible with the categorial setup at a first glance: they are flat, and lack the notion of finite phrasal parts – in showing no attachment to iterative phrasal division, they are also not obviously compositional.

In this, chapter we will focus our efforts into bridging this gap between these two perspectives under a unified categorial grammar setup. We will motivate the incorporation of dependency relations into the categorial vocabulary by repurposing existing and well-studied tools that remain faithful to the type theory roots the previous chapter has established (hint: it's the modalities). We will finally discuss how their inclusion alters the structural paradigms of the previous chapter, and the opportunities and problems this change comes with.

7 Phrase vs. Dependency Structure

Before we get to theorycrafting, it would be useful to try and clarify what exactly is meant by constituency- and dependency- structure, and how the two differ.

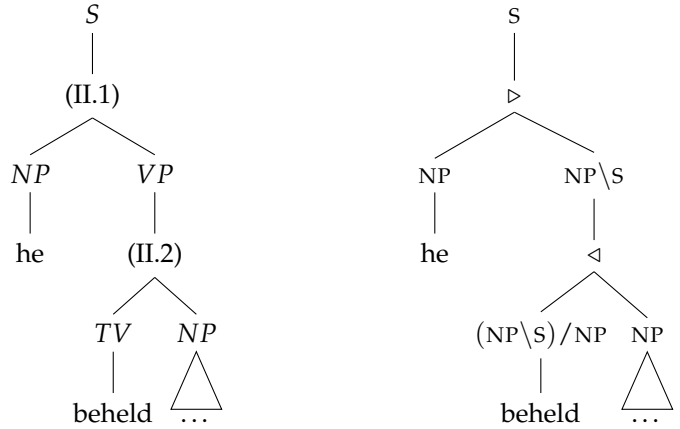
7.1 Phrase Structure Grammars

Phrase structure grammars build on the observation that certain phrases seem to act as rigid and independent chunks, sometimes referred to as *constituents*. Viewed from within, these phrases may be rich in internal structure, but keep it sealed off to the outside. Viewed externally (i.e. in the context of a wider phrase that contains them), they are indivisible units, or at least for the purposes of phrasal composition. Phrases are inventorized according to their syntactic *categories*. If one so wishes, they can for the most part replace a phrase for another of the same category, with no effect to grammaticality or local structure, which suggests they are functionally indiscernible. The examples below testify to this[1]; the underlined phrases can be freely interchanged – despite their wildly different internal structures, substituting one for another has no effect on the outer sentential structure:

i. $\text{he} \cdot (\text{beheld} \cdot \underline{(\text{the} \cdot \text{city})})$
ii. $\text{he} \cdot (\text{beheld} \cdot \underline{(\text{the} \cdot ((\text{glittering} \cdot \text{minarets}) \cdot (\text{of} \cdot (\text{the} \cdot \text{city}))))})$
iii. $\text{he} \cdot (\text{beheld} \cdot \underline{((\text{such} \cdot \text{beauty}) \cdot (\text{of} \cdot ((\text{red} \cdot (\text{and} \cdot \text{white})) \cdot \text{flowers})))})$
iv. $\text{he} \cdot (\text{beheld} \cdot \underline{((\text{some} \cdot (\text{feature} \cdot (\text{or} \cdot \text{arrangement})) \cdot (\text{which} \cdot (\text{he} \cdot ((\text{had} \cdot \text{known}) \cdot \text{before}))))})$

These so-called constituents interact, then, with one another depending not on their contents, but rather their categories. This perspective promotes a disciplined approach to grammar modeling, dating back to the formal grammars of Chomsky [1956], the archetypical example being context-free grammars [Chomsky, 1956; Backus, 1959]. There, the construction of complex expressions is guided by *production rules*, grammatical recipes that dictate what categories can sequentially combine, in what order, and what the category of their combination is. The above example would correspond, for instance, to

[1] Sourced from H.P. Lovecraft, *Celephaïs* (1922). In *The Rainbow, vol. 2.*



(a) Context-free parse tree. (b) Tree-formatted Lambek derivation.

Figure II.1: A phrase-structure grammar parse tree (a) contrasted with a Lambek abstract syntax tree (b).

production rules of the form:

$$S \rightarrow NP\ VP \tag{II.1}$$
$$VP \rightarrow TV\ NP \tag{II.2}$$

claiming that one of the ways to make a verb phrase VP involves concatenating a transitive verb TV with a noun phrase NP, which in turn can be plugged to the right of another NP to produce a declarative sentence S – each rule leaving a bracketing structure (or binary tree) in its wake (see Figure II.1a).

Even though context-free grammars are no longer seriously considered in the linguistic world, they have directly influenced most early attempts at grammar design – and by extension, their later successors and refinements. Notational evidence of this past are more than noticeable today, ranging from the wide adoption of tree-style notation for syntactic analyses to the conceptual syncretism of constituency grammars and phrase structure ones. More up-to-date frameworks expand upon the barebones context-free backend with niceties like a separation of functional dominance and linear precedence, added rules that manipulate movement and discontuinity, the proclamation of a single category as the *head* of a production rule (or subtree), feature markings that carry semantic, morphological or phonological information, incorporation of dependency information, etc. [Gazdar et al., 1985; Jacobson, 1987; Pollard and Sag, 1994; Dalrymple, 2001, *inter alia*].

To our ill fortune, the field of formal syntax is actually an informal mess, more akin to a mine field rather than an academic one; it's best if I tread carefully and refrain from overextending myself here, in order to avoid setting off

unseen traps or causing easily avoidable confusion. The point I want to make is that the focal center of the above formalisms is the *phrase* and its structure – as such, they are all referred to as phrase structure grammars, regardless of whatever extra fluff they carry or what their expressive capacity is. In that broader sense of the term, and removing any implicit connotations of intellectual lineage, categorial grammars can also be conceived as phrase-centric. Pure Lambek systems, for one, also explicate how phrases are combined, adhere to hierarchical forms similar to those of context-free grammars (except beautifully, see Figure II.1b) and in fact have the same expressive capacity with respect to string formation (i.e. the two are *weakly* equivalent) [Pentus, 1993]. Categorial grammars abstract away from the rule inventory by utilizing the smallest and purest set of rules possible – those of function application and variable abstraction – and internalize what used to be rule-imposed structure within the lexical categories themselves. Bracketing structure is now the footprint of function application, and the interface with semantics is naturalized by virtue of the Curry-Howard correspondence, as we saw earlier. Rather than a VP category and rule (II.1), we have the *type* $\text{NP}\backslash\text{S}$ – transparent with respect to both its syntactic combinatorics and semantic function. Performing the function application on a left-adjacent NP will then result to a local tree structure, not unlike the corresponding production rule – see Figure II.1 for a comparison. Note that in reality, categorial derivations in the type-theoretic tradition resemble trees only locally, since in high-order phenomena involving abstractions the unary, non-terminal λ nodes will either need to be uniquely named with the variable they are binding, or otherwise point to it with an additional edge – hence a directed acyclic graph could make for a more accurate representation format. Long story short, even without extensions to the logical core for managing discontuinity, calling deductive parsing constituency parsing would obviously not be doing the former justice; yet despite their methodological and theoretical divergences, their end yield is comparable – the antecedent structures of Figures I.20 to I.23 testify that the former may in fact be seen as subsuming the latter.

7.2 Dependency Grammars

The constituency tradition has co-evolved along the opposing view of dependency grammars [Tesnière, 2015; Gaifman, 1965; Sgall et al., 1986; Mel'cuk, 1988; Sleator and Temperley, 1995, *inter alia*]. Dependency grammars reject the binary phrasal division that constituency grammars abide by, and instead adopt a flatter structural form, the only unit of which is the word. Words are connected with one another by dependency arcs, i.e. directed edges between word pairs. Each word can have arbitrarily many outgoing edges (dependents), but only a single incoming edge (head) – the exception is the root word which has no head of its own (i.e. the head of the matrix clause). A word is said to directy dominate its dependents, and indirectly dominate all words its dependents dominate (directly or otherwise) – e.g. the root indirectly dom-

inates every other word in the sentence. This distinction between head and dependent is central to dependency grammars; broadly speaking, heads can be thought of as the words that decide the syntactic functionality of the collection of words (for fear of calling it a phrase) they indirectly dominate. The dependency structure of a sentence is once more a tree, with words now as both terminal and non-terminal nodes, glued together with dependency relations. A dependency tree is unconstrained by adjacency and word order: edges can fly over other edges; planarity is optionaly respected: an edge penetrating another edge to enter a nested domain is called *non projective*. This perspective is computationally appealing due to its simplicity and uniformity, as it allows a dependency grammar to argue about languages with wildly diverging syntactic and typological properties while remaining virtually unchanged. For the exact same reasons, it can also be seen as concealing – it sacrifices any potential of targeted analysis in the pedestal of universality. Finally, the semantically inclined might find a two-directional extension of dependency arcs enticing. In that setup, the added direction (which needs not agree with that of syntactic dominance) is devoted to semantic information flow, pointing from semantic predicates[1], to semantic arguments [Mel'cuk, 2003].

A dependency grammar that has gained significant traction over the last decade is the framework of universal dependencies [de Marneffe et al., 2021], claiming a broad collection of multi-lingual treebanks [Nivre et al., 2020] and tools. In universal dependencies, words are usually assigned a label pulled from a rudimentary set of part of speech tags and lexical identifiers and more importantly, dependency relations are also labeled according to their grammatical function, allowing the distinction of a words' dependents according to the grammatical role they fulfill. Grammatical roles are typologically and thematically informed, and are inventorized with language universality as the prime goal. This inventorization upholds no semantic promises, but is not inconsistent with the aforementioned semantic view either. To obtain a semantic transcription of the dependency tree, one needs only specify whether the semantic flow of each grammatical role is co- or contra- directional to the edge's syntactic flow, i.e. whether the arc marks its dependent as a *complement* (where syntactic head and semantic predicate coincide) or an *adjunct* (where the syntactic head is the semantic argument to its syntactic dependent).[2]

Figure II.2 shows an example dependency parse. Unlike before, we can not claim any semblance to the proofs that have occupied us thus far. At a first glance, dependency grammars have little in common with categorial grammars – structures are no longer binary nor made out of phrases, the axis of grammatical functions is competely new, and there seems to be little there reminiscent of the notions of induction and composition. Though on closer in-

[1] Apparently also an overloaded term. I will only ever use this in the strictly logical sense.

[2] Universal dependencies, a staunch pacifist, carefully and vocally refuses to make this claim, as complements and adjuncts are largely language-particular syntactic constructs and a notorious point of debate [Haspelmath, 2014]. I would like to believe I am not trespassing here, either – as will be made clearer in a bit, I employ the two terms in a purely semantic fashion.

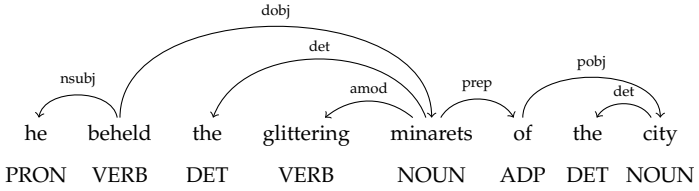


Figure II.2: A sample dependency parse in the universal dependencies format.

spection and armed with some goodwill, we can recover from some of these divergences if we make a few concessions from both sides. We can start by treating any collection of words rooted in the same ancestor in the dependency tree as a constituent phrase, albeit possibly discontinuous – the result will give us at least some partial overlap with the categorial directive. Grammatical functions can then be thought of as being implicit, having been internalized in their positioning within a functor. For instance we do intuitively know that the Lambek transitive verb $(\text{NP}\backslash\text{S})/\text{NP}$ requires an *object* NP to the right and a *subject* NP to the left – marking them as such is perhaps redundant, since the verbal meaning recipe places each syntactic argument into a distinct semantic slot. The binary bracketing structure is irrevocably lost, but this loss can be deemed as inconsequential if we "flatten" functor-induced phrasal boundaries by considering them only at these intermediary points where all of their arguments (however many) have been applied. It is not much further we can get with this mediatory role, though. No concession from the categorial side would be able to justify the underspecification of higher-order phenomena in a dependency tree, and no concession from the dependency side could make peace with the omission of the concept of headedness in an applicative natural deduction proof.

8 Modalities for Dependency Demarcation

In our new quest, we will seek to design a type logic that subsumes and rises above both phrase structure grammars and dependency grammars. As we saw in the previous section, the bar is not set particularly high for the first kind; the Lambek calculus can already do more than well enough. The challenge then is to integrate the added values of a dependency grammar in a type-theoretic framework. There's two elements we are missing; distinguishing between syntactic heads and syntactic/semantic predicates, and marking words and phrases according to their grammatical roles within some wider context.

8.1 Two Dimensional Predicates

Categorial grammars are inherently and by design biased towards predicate structures, primarily syntactic, but simultaneously also semantic (if one is to believe the story of Section 5.1.4, no distinction can be made between the two). Each phrase can be iteratively split apart into two subphrases (not necessarily contiguous), where one provides a functor, and the other the argument thereof. Note that this distinction does not preclude the possibility that the argument itself has a functional type – no assumption is made on the form of either subphrase's type, other than the two being compatible. But what assumes the role of the functor in a local domain needs not always be the syntactic head of that domain. There's a plethora of example cases. Quantifiers, for one, are inarguably predicates over the objects they quantify, yet they exactly obey the morphosyntactic characteristics prescribed by these objects (i.e. grammatical gender, case, number, etc.), evidencing that the latter are in fact the heads – a clear violation of any alignment between syntactic predicate and syntactic head we could ever hypothesize. A similar argument can be made for determiners and, more broadly speaking, any phrasal element that takes functional precedence without being the syntactically prominent part of its phrase, e.g. adjectival and adverbial modifiers.

This observation gives rise to a binary subcategorization of a binary predicate structure; to establish some risky terminology, it is either:

i. an application of a head to its *complement*, or
ii. an application of an *adjunct* to its head

where the distinction between complement and adjunct is made solely on the basis of their functional relation to the head.

The vanilla categorial vocabulary does not suffice to capture this extra dimension of function application – a problem also noticed by the intellectuals of proto-categorial civilizations, as archeological excavations reveal. In an unpublished manuscript, Moortgat and Morrill [1991] propose a two-dimensional implicational type operator and corresponding residuation laws: the first binary dimension is reserved for the usual left- vs. right- application distinction, whereas the second binary dimension specifies whether the head occurs to the left or to the right; the result is four unique ways of building up an implication. Congruent with the substructural trend of revealing structure that was once hidden, this division brings forth a two-valued structural binder, allowing the corresponding logic **DNL** to reason about *headed* binary trees. The authors refrain from commiting to a specific linguistic application, but, translated into our terminology, their proposal can be schematically summarized by Figure II.3. Further away from syntax, Hendriks [1997] employs **DNL** to account for prosodic structures, where the head is assigned to intonationally prominent elements.

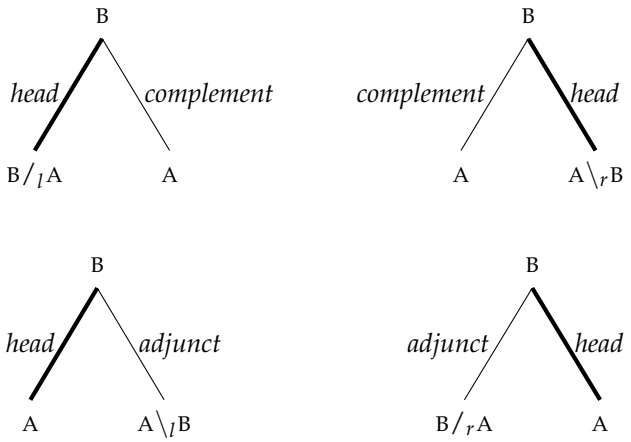


Figure II.3: The four implications of **DNL**.

8.2 Modal Dependents

As an alternative to introducing implicational (and by residuation, product) variants, we can instead opt for the more fashionable modal decomposition approach [Kurtonina and Moortgat, 1997]. The allows us to view a specialized (here: head-aware) implicational variant as a composition of its uniform base with a unary modality. The standard route would have use the modality to mark the head – we will instead mark the dependent. More than a petty act of rejection to establishment, this shall provide us with the means to further differentiate dependents according to the exact grammatical slots they occupy – after all, there's quite a few different dependency labels, but only the one head.

8.2.1 Complements vs. Adjuncts

Sticking to our risky agenda, the first distinction we need to make is that of complements versus adjuncts. In the complement case, such a decomposition would look as follows:

$$B/_l A \equiv B/\Diamond A \tag{II.3}$$

$$A\backslash_r B \equiv \Diamond A\backslash B \tag{II.4}$$

The translation is straightforward: predicates in head position are functors requiring the same arguments as they would before, except now under a diamond. In that sense, they *assign* diamonds to their complements by necessitating an application of the $\Diamond I$ rule of Figure I.15 prior to the function application. Recalling the structural imprint of the rule, this results in an extra layer of

bracketing structure the delimits complement phrases and isolates them from their surroundings.

The adjunct case may at first glance seem slightly more obscure. Following the directives of the previous paragraph, we need to mark the dependent – this time a predicate in non head position – in a way such that its application on its argument leaves a bracketing imprint on the structure of the former rather than the latter. The solution manifests in the form of a box:

$$\mathrm{B}/_r\mathrm{A} \equiv \Box(\mathrm{B}/\mathrm{A}) \tag{II.5}$$

$$\mathrm{A}\backslash_l\mathrm{B} \equiv \Box(\mathrm{A}\backslash\mathrm{B}) \tag{II.6}$$

The translation is not much different: adjuncts are predicates wrapped by a box. To reveal the pure function contained therein and allow a proof to progress, we need to invoke the $\Box E$ rule of Figure I.15, the effect being a bracketing structure that now delimits adjunct phrases.

There is symmetry between the above two cases. The task of imposing dependency structure is always upon the functional predicate and its type. Head predicates mark their complements, whereas adjunct predicates mark themselves; in either case, it is the dependent structure that gets the brackets. The duality of predicate structure is thus mirrored in the innate distinction between function and argument of the applicative categorial backend, whereas the duality of syntactic headedness is captured by the unary modalities; typechecking in both dimensions.

8.2.2 Grammatical Functions

Let's take this a bit further. Universal dependencies may make no adjunct vs. complement distinction, but they go the extra mile of subspecifying dependents according to the exact grammatical roles they play. Extending our grammar logic accordingly is trivial. Rather than have a single diamond and box, we can consider a usecase where modalities are a *family* of unary residuals, i.e. a set of pairs, each labeled according to a single, unique dependency label. The generalization is a multimodal type system consisting of modal pairs:

$$\{(\Diamond_d, \Box_d) \mid d \in \mathsf{Deps}\} \tag{II.7}$$

where Deps the full set of dependency labels made available to each specific instantiation of the theory.[1] The edge case of Deps being a singleton set collapses to the previous exposition (whereby explicit labeling is redundant).

Each instance of a labeled modality will now come with its own introduction and elimination rules. Concomitantly, both the term calculus and the

[1] Note that the label set can vary depending on the designer's end goal; grammatical functions is just one of the possibilities. The setup is also more than compatible with frame semantics, where event-specific semantic structures (*frames*) are evoked by lexicalized syntactic heads to assign semantic/thematic roles to their dependents (*frame elements*) [Fillmore et al., 1976].

bracketing structures are extended with multiple labels; the modal rewrites and unary brackets of Section 4.1 are now differentiated on the basis of the dependency label that induced them. Unlike before, the structural effect is not a means to the end of structural reasoning, but the very purpose of the dependency modalities – as such, they are not necessarily associated with any structural rules (even though nothing precludes the possibility – it might even be reasonable to condition each dependency or combinations thereof to a unique set of structural irregularities, as we will see in a bit). Note, also, that the residuation properties and normalization routines apply only between diamonds and boxes of the *same label* – no interaction between mismatched types and terms is stipulated.

8.3 Inference with Dependency-Enhanced Types

Logical inference in the setup envisaged here is not dissimilar conceptually to the standard type-logical pipeline of Section 5.1, but there's some crucial differences that require explication, plus a few critical gotchas to beware of.

8.3.1 Initial Lexical Adjustments

For starters, functors previously involved with simple applicative phenomena will now need to abide to either of the type patterns below:

$$\mathrm{A}, \mathrm{B} := \Diamond_d \mathrm{A} \backslash \mathrm{B} \mid \mathrm{B} / \Diamond_d \mathrm{A} \mid \Box_d (\mathrm{A} \backslash \mathrm{B}) \mid \Box_d (\mathrm{B} / \mathrm{A}) \tag{II.8}$$

For these dependency-enhanced types to appear and take effect, the lexicon needs to be adjusted accordingly.

It is the lexicon's first duty then to discriminate between head and non-head functors by decorating them or their arguments with the appropriate modalities. An intransitive, for instance, would now be typed as $\Diamond_{su}\mathrm{NP} \backslash \mathrm{S}$ – to produce a sentence, the type demands to its left not just any noun phrase, but rather one marked as a subject. The story is no different with more than one complements – i.e. a transitive would be $(\Diamond_{su}\mathrm{NP} \backslash \mathrm{S}) / \Diamond_{obj}\mathrm{NP}$, and so on. A determiner, however, would be typed as $\Box_{det}(\mathrm{NP}/\mathrm{N})$ – it recognizes the right-adjacent noun as its head, but still takes functional precedence over it, licensing the function application by dropping its determiner box. Similarly, a prenominal modifier would be typed as $\Box_{mod}(\mathrm{NP}/\mathrm{NP})$ – to apply to its unmarked head, the type would need first liberate itself of its box, being a modifying adjunct.

Atomic type assignments will remain for the most part unchanged, as propositional constants are necessarily complements (or heads of a singleton phrase, to be pedantic) and their grammatical role cannot be decided *a priori*, anticipating a phrasal head to enforce it instead. Plural nouns like the *dolphins* and *whirlpools* of Table I.4, for instance, would still be typed as NP – there's no telling in advance whether they will occur as subjects, direct ob-

jects or something else. Exceptionally for words whose morphological characteristics already confine them to a single possible grammatical role, we can consider an alternative typing that restricts them to exclusively that grammatical role. The straightforward thing to do would be to lexically mark them with a diamond – e.g. for the nominative version of the third person singular personal pronoun, *he*, we might assign the type $\Diamond_{su}\mathrm{NP}$, denoting it must necessarily occur in subject position. However, this would create a structural assymetry between a nominal *assigned* the subject role via the $\Diamond_{su}I$ rule (inducing corresponding brackets) versus the pronoun *carrying* the subject role (and thus remaining bracket-free). To break this asymmetry, a better alternative would be to use a lexical assignment that rests on the *closure* operator of (I.24) instead, i.e. $\Box_{su}\Diamond_{su}\mathrm{NP}$. Now for the verbal head to find its subject-marked argument, the pronoun would need to reveal its diamond via the $\Box_{su}E$ rule, independently bracketing *itself* in the process, while excluding any potential for grammatical misuse.

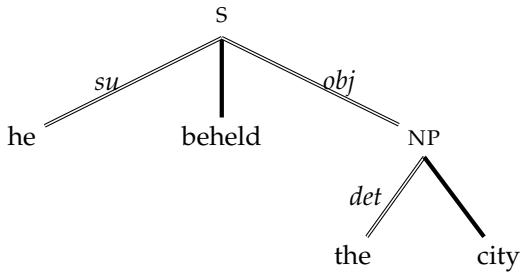
8.3.2 Dependencies & Structural Reasoning

Lambek vs. Lambek (vs. van Benthem) The next thing to consider is how the inclusion of dependency modalities alters the structural core of each base logic. In any case, modalities induce a multi-labeled unary bracketing structure. For the non-associative **NL**, the result is trees of mixed but consistent arity – our subclassing of functors in (II.8) means that each binary branch (imposed by a function/argument structure) will contain a normal branch that corresponds to the local head *phrase* and a distinguished unary branch that labels the non-head phrase – brackets and parentheses galore. Opting for the more traditional associative base of **L** changes the scenery to one of shorter, wider and less exucerbant trees. Since the vanilla structure is now just a sequence, treeness is imposed solely by the modal brackets; the result is a variadic but uniform tree structure, where each subtree contains a single local head *word* and multiple dependent phrases, each of them in turn wrapped under a unary branch (or simply just having its edge labeled, to make things easier to the eye). This visual paradigm also applies to the even laxer **NLP** – the difference being that the yield of each local tree is recursively equivalent under bracket-preserving permutation, i.e. commutativity now holds between constituent phrases (subtrees) rather than words (terminal nodes).

The above points hint to the fact that dependency modalities introduce structural constraints that may (to some extent) obviate the need for a strict structural binder. From the linguistic perspective, **L** seems to hit the sweet spot – it has constituents live happily together in horizontal, non-binary clusters set upon lush trees, with each constitutent given a role to fulfill. The systematic ordering of arguments according to their obliqueness order is made redundant by their labeling; the positional explication of **NL** is replaced by the denominational explication of the modal brackets. What's more, headedness is not proliferated among functionally incomplete constituents, i.e. each

$$\dfrac{\dfrac{\text{he} : \Box_{su}\Diamond_{su}\text{NP}}{\langle\text{he}\rangle^{su} \vdash \Diamond_{su}\text{NP}}\,\Box_{su}E \qquad \dfrac{\text{beheld} : (\Diamond_{su}\text{NP}\backslash\text{S})/\Diamond_{obj}\text{NP}}{\dfrac{\dfrac{\dfrac{\text{the} : \Box_{det}(\text{NP}/\text{N})}{\langle\text{the}\rangle^{det} \vdash \text{NP}/\text{N}}\,\Box_{det}E \qquad \dfrac{}{\text{city} : \text{N}}}{\langle\text{the}\rangle^{det}, \text{city} \vdash \text{NP}}\,/E}{\langle\langle\text{the}\rangle^{det}, \text{city}\rangle^{obj} \vdash \Diamond_{obj}\text{NP}}\,\Diamond_{obj}I}{\text{beheld}, \langle\langle\text{the}\rangle^{det}, \text{city}\rangle^{obj} \vdash \Diamond_{su}\text{NP}\backslash\text{S}}\,/E}}{\langle\text{he}\rangle^{su}, \text{beheld}, \langle\langle\text{the}\rangle^{det}, \text{city}\rangle^{obj} \vdash \text{S}}\,\backslash E$$

(a) Simple applicative derivation in dependency-enhanced **L**.



(b) Corresponding tree structure, read off the antecedent. As before, heavy edges denote heads, and double edges telescope unary branching (now labeled).

Figure II.4: The structural effect of dependency-enhanced functors.

complete phrase (read: one typed as a propositional constant) is flat among its arguments, requiring only a single head and implicitly disallowing heads from being phrases in themselves (in line with the mandates of dependency grammars). The effect could be paralleled to a single argument functor that takes the n-ary product of all its arguments in at once, giving rise to a corresponding n-ary structural binder. Figure II.4 presents a simple first example to illustrate the point.

How about **NLP** though? We have so far dismissed the syntactic utility of the logic as being overly permissive, presenting it as meaningful only from a semantic perspective. With dependency brackets in the picture, the explosive combinatorics of global commutatitivity are somewhat tamed; it is now permitted only within the context of subtrees. In programming language terms, this can be paralleled to a tree-shaped variable scoping strategy, where scopes are identified by their names (unary labels) and those of their ancestors, and the order of variable declaration is irrelevant, but the nestedness of embedded trees is not. From a linguistic perspective, this would be akin to a natural language that exhibits quasi-free local word order but makes heavy use of overt morphological case marking to disambiguate.

This is still not very realistic, but presents an interesting opportunity. Rather than commit to commutativity in general, we are invited to step into the cross-

roads between the two logics, employing **L** as the global base while inventorizing commutative scrambling- and topicalization- like behaviors on the basis of structural postulates informed by dependency roles. Utilizing the now explicit boundaries of dependency domains, we can repurpose the notion of context to denote subtrees (despite being in an associative calculus!), obtaining the means to formulate rules like:

$$\frac{\Gamma[\![\langle \Delta, \langle \Phi \rangle^{obj} \rangle^{d}]\!] \vdash \mathrm{A}}{\Gamma[\![\langle \langle \Phi \rangle^{obj}, \Delta \rangle^{d}]\!] \vdash \mathrm{A}}\ \text{obj-top} \tag{II.9}$$

which can be read as saying that an object can be preposed within its local d-labeled clause, if one such is nested within Γ.[1] In principle, this could simplify the categorial treatment of such phenomena: one can always start with a canonical derivation, e.g. one where all arguments are in their expected positions, and proceed by shuffling them around given the structural rule inventory. As a bonus, adorning these rules with non-void term rewrites would allow them to upkeep their relevance for pragmatics. Exciting (or not) as this might sound, it was only ever meant to incentivize the use of dependency modalities; it won't be something we will be pursuing presently, for we have another kind of beast to face.

Crossing Boundaries The structural rule format hinted at would be capable of dealing with the movement of phrases *within* a dependency domain, conditionally relaxing the word order constraints of **L** under certain dependency configurations. Yet it fails to provide any insights on how this could work in the case of structures that are misplaced not in terms of linear order, but of nestedness level. That is, beyond the standard question of word order, dependency modalities import previously invisible structural brackets that can pose a challenge when it comes to traversing *along* dependency domains – a challenge external to the grammar's implicational core. To make this clearer, let us revisit the keystone achievement of the previous chapter, namely the derivation of the object-relative clause of Figure I.22a (the phrase in question is *that the eye may never behold*, in case you were confident enough to skip the chapter). Conforming to our routine, let's first adapt some of the lexical assignments of Table II.1 (and add a few new ones for good measure).

The propositional constants for nouns N and noun phrases NP are unmarked, plain and boring; let's not speak of them any further. The first-order types of the determiner DET, adjectival modifier $\mathrm{ADJ}_{/}$ and verbal types, ITV and TV, should by now also be familiar. The higher-order types of the adverb and the modal auxiliary, ADV and AUX, might, however, require some elucidation. Note, first, that despite seemingly divergent, the two types are identical

[1] In reality, we would need to mark complete clauses to remove the possibility of arbitrary shuffling and accidental overgeneration, but this can be trivially accomplished by boxing the phrasal end-result, e.g. $\Diamond_{su}\mathrm{NP}\backslash\Box_{cl}\mathrm{S}$ for an intransitive, where *cl* would mark a complete clause and assume the role of *d* in (II.9).

eye, city	::	N	
walls, twilight	::	NP	
high, sterile	::	$\text{ADJ}_/$	$:= \Box_{mod}(\text{NP}/\text{NP})$
a, the	::	DET	$:= \Box_{det}(\text{NP}/\text{N})$
reigned	::	ITV	$:= \Diamond_{su}\text{NP}\backslash\text{S}$
behold	::	TV	$:= \text{ITV}/\Diamond_{obj}\text{NP}$
never	::	$\text{ADV}_/$	$:= \Box_{mod}(\text{ITV}/\text{ITV})$
may	::	AUX	$:= \Diamond_{vc}\text{ITV}/\text{ITV}$

Table II.1: Dependency-enhanced lovecraftian lexicon.

when stripped of their modalities. This is perfectly in line with our agenda of revealing previously coalescent diversity. The negation *never* functions like an adverbial adjunct: it is an endomorphism of an intransitive phrase that marks itself as a modifier in the process. The modal auxiliary *may*, on the other hand, heads its local phrase by assigning to an intransitive dependent the role of a verbal complement, *vc*.

Next, we need to turn our attention to the relativizer – let's first address the missing dependencies:

$$\Box_{mod}(\text{NP}\backslash\text{NP})/\Diamond_{body}(\text{S}/\Diamond_{obj}\text{NP}) \tag{II.10}$$

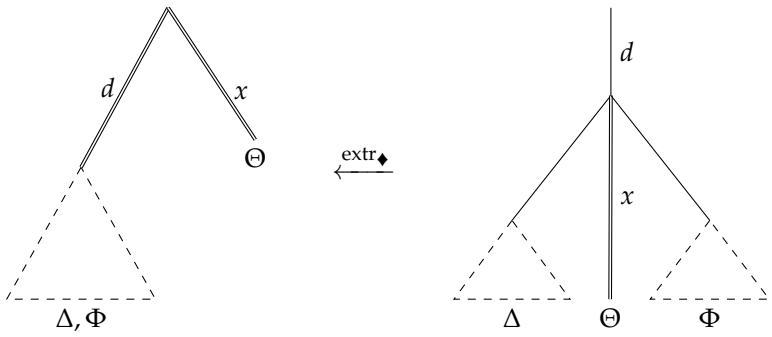
The functor now states the following: it requires first a relative clause body, namely a sentence missing an object-marked noun phrase in its rightmost border, in order to produce a postnominal adjectival phrase, i.e. a modifying adjunct.

Naively, we would assume that the absence of binary bracketing structure in **L** would allow us direct access to the gap within the relative clause body, counteracting the need for the control modalities of (I.35). But the gap is still enclosed, except this time under layers of impenetrable dependency domains! Seems like we need to reinstate control – both kinds of modalities must be employed in tandem; although in truth, the two do not really constitute distinct kinds per se: they only differ insofar as their linguistic purposes do. Nevertheless, it might be handy to make a notational distinction between them, just for the sake of reading comprehension. From now on, we will use filled symbols to denote control modalities (i.e. ones whose purposes are confined to rebracketing and movement), and white symbols to denote dependency modalities (i.e. ones whose purpose is linguistic annotation, and the brackets of which we expect to see in the antecedent of the proof's end yield). Even when having a single control pair, assigning it an explicit label x is still useful, since it will allow us to tell its brackets apart from the rest. In this regime, our relativizer's type assignment becomes:

$$\text{that} :: \text{REL}_{\text{O}} := \Box_{mod}(\text{NP}\backslash\text{NP})/\Diamond_{body}(\text{S}/\blacklozenge_x\blacksquare_x\Diamond_{obj}\text{NP}) \tag{II.11}$$

$$\frac{\Gamma[\![\langle \Delta, \langle \Theta \rangle^{x}, \Phi \rangle^{d}]\!] \vdash A}{\Gamma[\![\langle \Delta, \Phi \rangle^{d}, \langle \Theta \rangle^{x}]\!] \vdash A}\ \text{extr}_{\blacklozenge}$$

(a) Controlled extraction rule.



(b) Corresponding tree transformation.

Figure II.5: Controlled extraction in the dependency-bracketed setting.

which is faithful to both (II.10) and (I.35), as it conveys that the missing object is now *movable*.

Mobility, however, also means something different now. Taking inspiration from the established structural vocabulary (see Figure I.19 if you need to jog your memory), we need to concoct a novel structural rule: one that allows the *extraction* of a nested substructure under the appropriate bracketing conditions. The magical conconction is presented in Figure II.5a. Within arbitrary context Γ, it looks for a d-labelled unary tree enclosing a x-labelled unary tree Θ wrapped by sequences Δ to the left and Φ to the right. There, it allows us to pull the Δ out, casting the outermost unary tree into a binary sequence and assigning d to the concatenation of Δ and Φ alone. If this makes little sense, see Figure II.5b for a visual rendition. If still unclear, move your mental cursor four sentences back (this one included) and try again.

Equipped with this missing bit of alchemical understanding, we're at long last able to produce analyses for some less contrived linguistic examples; Figure II.6 presents the dependency-enhanced derivation we set out to deliver. Before we move on, though, some important observations are in order. First, the presentation makes an implicit quantification over the outer label, d – if we were to be really pedantic, we'd need a unique instantiation of that rule for each dependency (but not control!) modal label in the logic. Beyond a bookkeeping obligation, this parameterization can also be to our advantage: it allows us to directly control which of the dependency domains allow extraction, and which do not. On a less bureaucratic note, the contextual formulation of

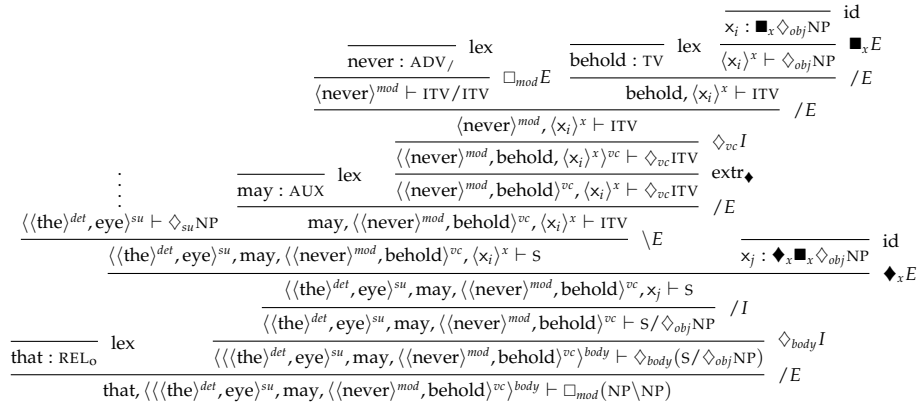


Figure II.6: Dependency-enhanced adaptation of the proof of Figure I.22a, exemplifying the interaction between dependency and control modalities.

the rule carries the usual problem of complicating syntactic equality checking. In practice, we can employ a localized version, i.e. one where Γ is a flat context (i.e. a sequence with a hole), which means that extractions need to be preemptively applied before every bracketing operation, rather than deferred and done in bulk in the future. Finally, it is important to remember that the rule is supplementary to (and *not* a substitute for) controlled associacitivity and/or commutativity: altering the linear order of substructures in **L**, or their binary brackets in **NL**, still calls for different structural rules with possibly different control modalities that will need to coexist with the $\text{extr}_\blacklozenge$ rule for structures to find their intended positions.

Ever higher order The example just inspected hides a crucial wisdom: variable abstraction applies to variables – *not* complex structures thereof. Control modalities may impose transient bracketing structure upon hypotheses, temporarily hindering their abstraction, but it is always retroactively redacted with the $\Diamond E$ rule (after all the necessary structural operations have taken effect). The bracketing imposed by dependency modalities, however, is built to last, posing a potential roadblock to hypothetical reasoning and higher-order types.

Hypothetical complements are relatively easy to tackle: they just come packed with their diamonds at variable instantiation time. Excluding the presence of the irrelevant control box, this is exactly the strategy followed in the example under scrutiny (see the id rule intantating x_i in Figure II.6). Upon closer inspection, we can verify that this is in fact just the η normalized version of hypothesizing a plain type, assigning it the desired dependency brackets via the $\Diamond I$ rule, and then performing a substitution of the bracketed variable for a

$$\dfrac{\dfrac{\dfrac{}{\mathsf{x}_k : \mathrm{NP}}\,\mathrm{id}}{\langle \mathsf{x}_k \rangle^{obj} \vdash \Diamond_{obj}\mathrm{NP}}\,\Diamond_{obj}I \qquad \dfrac{\dfrac{}{\mathsf{x}_i : \blacksquare_x \Diamond_{obj}\mathrm{NP}}\,\mathrm{id}}{\langle \mathsf{x}_i \rangle^{x} \vdash \Diamond_{obj}\mathrm{NP}}\,\blacksquare_x E}{\langle \mathsf{x}_i \rangle^{x} \vdash \Diamond_{obj}\mathrm{NP}}\,\Diamond_{obj}E \quad \overset{\eta}{\equiv} \quad \dfrac{\dfrac{}{\mathsf{x}_i : \blacksquare_x \Diamond_{obj}\mathrm{NP}}\,\mathrm{id}}{\langle \mathsf{x}_i \rangle^{x} \vdash \Diamond_{obj}\mathrm{NP}}\,\blacksquare_x E$$

Figure II.7: η long form of the $\Diamond_{obj}$ connective of hypothesis x_i in Figure II.6.

logical equivalent (wrapped under the necessary control brackets) via the $\Diamond E$ rule – consult Figure II.7 and contrast with Figure I.16b if unconvinced.

Hypothetical adjuncts are less forgiving. A hypothesized adjunct will seek to apply itself to some phrasal head, which is impossible unless it first drops its box. But in dropping its box, it becomes enclosed in structural dependency brackets that prohibit its eventual abstraction. We will need to once more resort to the $\Diamond E$ rule to remove , except this time it will be the interior combination of Figure I.17d that we are invoking, not subject to η contraction. To see this in action, let us once more consider an excerpt from our go-to source: *a city of high walls where sterile twilight reigned*[1]. The relative adverb *where* heads yet another a relative clause, *where sterile twilight reigned*, acting as a postnominal modifier to the noun phrase *a city of high walls*. The relative clause differs to the ones so far inspected, in missing from its embedded subordinate *sterile twilight reigned* _ not a complement, but an adjunct: a gap-equivalent to the lexical *there* of Figure I.20e, except dependency-enhanced, i.e. $\Box_{mod}(\mathrm{ITV}\backslash\mathrm{ITV})$ in shorthand. As Figure II.8a illustrates, the need for bracket erasure necessitates prefixing the gap type with a residual diamond, steering us towards our first ever fourth order type:

$$\text{where} :: \mathrm{REL}_{\mathrm{loc}} := \Box_{mod}(\mathrm{NP}\backslash\mathrm{NP})/\Diamond_{body}(\mathrm{S}/(\Diamond_{mod}\Box_{mod}(\mathrm{ITV}\backslash\mathrm{ITV})) \qquad \text{(II.12)}$$

This beast of a type plays the starring role in the derivation of Figure II.8; it promises to provide a postnominal modifier, if presented with a (third order) relative clause body, that being a sentence missing to its right a diamond-marked (second order) postverbal modifier.

But why go through all this trouble of producing the dependency brackets only to then immediately cancel them out? The alternative of hypothesizing a plain functor without any dependency markings would be logically valid but grammatically suboptimal and contrary to our agenda, as it would give us no insights on what the dependency function of the gap is; the modalities stay. Another, more pressing question is that of the compatibility between hypothetical adjuncts and the control modality of the previous paragraph, i.e. how could we deal with the nested gap in e.g. *where sterile twilight **may** reign* _. Fortunately, we need not worry: our previous treatment still holds with only the most minor of adjustments. Same as before, we have to alter the typing of the gap by prepending a boundary crossing permit, the interior pair $\blacklozenge_x\blacksquare_x$. As

[1] H.P. Lovecraft, *Azathoth* (1938). In *Leaves* (2).

$$\dfrac{\dfrac{}{\text{reigned} : \text{ITV}}\ \text{lex} \qquad \dfrac{\dfrac{\dfrac{}{\mathsf{x}_i : \Box_{mod}(\text{ITV}\backslash\text{ITV})}\ \text{id}}{\langle \mathsf{x}_i \rangle^{mod} \vdash \text{ITV}\backslash\text{ITV}}\ \Box_{mod}E \qquad \dfrac{}{\mathsf{x}_j : \Diamond_{mod}\Box_{mod}(\text{ITV}\backslash\text{ITV})}\ \text{id}}{\mathsf{x}_j \vdash \text{ITV}\backslash\text{ITV}}\ \Diamond_{mod}E}{\text{reigned}, \mathsf{x}_j \vdash \text{ITV}}\ \backslash E$$

(a) Structurally freeing a hypothesized adjunct...

$$\dfrac{\dfrac{}{\text{where} : \text{REL}_{\text{loc}}}\ \text{lex} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\text{sterile} : \text{ADJ}_{/}}\ \text{lex}}{\langle \text{sterile} \rangle^{mod} \vdash \text{NP}/\text{NP}}\ \Box_{mod}E \qquad \dfrac{}{\text{twilight} : \text{NP}}\ \text{lex}}{\langle \text{sterile} \rangle^{mod}, \text{twilight} \vdash \text{NP}}\ /E}{\langle \langle \text{sterile} \rangle^{mod}, \text{twilight} \rangle^{su} \vdash \Diamond_{su}\text{NP}}\ \Diamond_{su}I \qquad \dfrac{\text{(II.8a)}}{\text{reigned}, \mathsf{x}_j \vdash \text{ITV}}}{\langle \langle \text{sterile} \rangle^{mod}, \text{twilight} \rangle^{su}, \text{reigned}, \mathsf{x}_j \vdash \text{S}}\ \backslash E}{\langle \langle \text{sterile} \rangle^{mod}, \text{twilight} \rangle^{su}, \text{reigned} \vdash \text{S}/(\Diamond_{mod}\Box_{mod}(\text{ITV}\backslash\text{ITV}))}\ /I}{\langle \langle \langle \text{sterile} \rangle^{mod}, \text{twilight} \rangle^{su}, \text{reigned} \rangle^{body} \vdash \Diamond_{body}(\text{S}/(\Diamond_{mod}\Box_{mod}(\text{ITV}\backslash\text{ITV})))}\ \Diamond_{body}I}{\text{where}, \langle \langle \langle \text{sterile} \rangle^{mod}, \text{twilight} \rangle^{su}, \text{reigned} \rangle^{body} \vdash \Box_{mod}(\text{NP}\backslash\text{NP})}\ /E$$

(b) ...to provide a gap in the higher-order argument of the relative adverb *where*.

Figure II.8: Deriving a locative relative clause.

the modal chains are no longer η contractable, the gap will manifest as three distinct variables, sequentially substituting one for another via two $\Diamond E$ rules, as shown in Figure II.9. The first substitution of x_i for x_j is responsible for removing the *mod* brackets for *x* ones, allowing any applications of the extr$_\blacklozenge$ rule in the telescoped subproof *s*. The bracketed variable will eventually be positioned at the outermost branch of the antecedent tree, at which point it can be substituted for x_k, the "true" variable of the gap. Don't give in to despair; this is peak complexity – with these type patterns at our disposal we should be able to tackle any linguistic phenomenon we might encounter from this point on. Note, finally, that complement and adjunct gaps are not that different to one another after all: they both mimic their corresponding lexical assignment (a plain type or a boxed functor, respectively), except prepended by a diamond of whichever dependency role they will assume and (if needed) a structural extraction licensor.

8.4 Interfaces

Having completed our tour of dependency-decorated proofs, it is time for respite and reflection. Let us take a moment to internalize where we started from and where we are now.

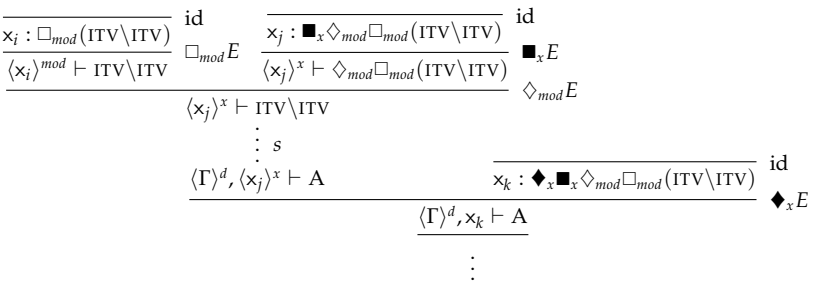


Figure II.9: Schematic pattern for extracting a nested hypothetical adjunct.

8.4.1 Dependency Trees

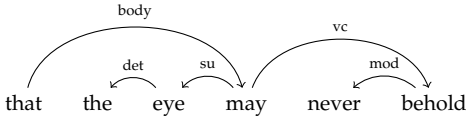
Our first ambition was to provide categorial type logics with the means necessary to argue about dependency relations, ideally subsuming dependency trees within the logic's antecedent structures. To claim our endeavour a success, we need to actually *show* how these dependency trees can be extracted.

Let's begin with some trivial observations. In the domain of dependency annotated proofs, endsequents will contain neither variables nor any control brackets. Collapsing any notion of structure imposed by the functional core (i.e. drop constituency and word order from the builtin structural binder, since dependency trees are agnostic to those), we end up with structures made exclusively of constants (terminal nodes), multisets of structures (unordered variadic branches) and dependency enclosed structures (unary branches). This typological description can be further refined by noticing that the non-zeroary tree operators alternate in turns: there's no chaining of unary brackets (each phrase is only assigned at most one dependency role), nor a multiset of multisets (as the simplified structural binder is flat). Finally, consider that each multiset will coincide with a dependency domain, containing exactly one (unmarked) head constant (necessarily a word), some n adjuncts and some m complements.

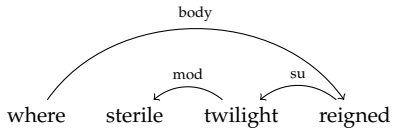
With these in mind, we arrive at the following (modulo permutation) inductive definition of a dependency induced structure:

$$\Gamma := \underbrace{\langle \Gamma_0 \rangle^{d_0} \dots \langle \Gamma_{n-1} \rangle^{d_{n-1}}}_{\text{adjuncts}} \underbrace{\langle \Gamma_n \rangle^{d_n} \dots \langle \Gamma_{n+m-1} \rangle^{d_{n+m-1}}}_{\text{complements}} \underbrace{\kappa}_{\text{head}} \tag{II.13}$$

Converting such a structure to a dependency tree akin to the one of Figure II.2 is pleasantly easy: just apply the function below to it (plug an invisible "root"



(a) Dependency tree of $[\![\text{that}, \langle\langle\langle\text{the}\rangle^{det}, \text{eye}\rangle^{su}, \text{may}, \langle\langle\text{never}\rangle^{mod}, \text{behold}\rangle^{vc}\rangle^{body}]\!]$.



(b) Dependency tree of $[\![\text{where}, \langle\langle\langle\text{sterile}\rangle^{mod}, \text{twilight}\rangle^{su}, \text{reigned}\rangle^{body}]\!]$.

Figure II.10: Trees extracted from the derivations of Figures II.6 (a) and II.8 (b).

node and label to get things going).[1]

$$\begin{aligned}&\mathsf{deptree} :: \mathsf{Struct} \to \mathsf{Head} \to \mathsf{Deps} \to \mathsf{Set}[\mathsf{Arc}]\\&\mathsf{deptree}\ \left(\langle\Gamma_0\rangle^{d_0} \ldots \langle\Gamma_{n+m-1}\rangle^{d_{n+m-1}}\ \kappa\right)\ \mathsf{root}\ \mathsf{label} =\\&\quad \{\mathsf{root} \xrightarrow{\mathsf{label}} \kappa\} \cup \bigcup_{i=0}^{n+m-1} \{\mathsf{deptree}\ \Gamma_i\ \ \kappa\ \ d_i\}\end{aligned}$$

Figure II.10 shows the function's yield applied to two of the section's example derivations. Given a partition of Deps into adjuncts and complements, the conversion can trivially be extended with the bidirectional dependency arcs described in Section 7.2. Word order is also straightforward to capture, if we just keep the structural order provided by a non-commutative calculus intact. In any case, our efforts go to show that the general structure of a dependency tree can easily be captured by a dependency-enhanced type logic using the type assignment patterns discussed. Aligning the type logic to a specific flavour of a dependency grammar was never our main intention, but should be fairly easy to accomplish by reverse-engineering the annotational specifics of the target: a task for future generations.

8.4.2 Semantics

The trees of Figure II.10 leave something to be desired: by backpedaling towards dependency grammars, higher-order phenomena have been completely dismissed – wasted are all our efforts to tame and manage the bracketing structures of hypotheses. Perhaps importing residuated modalities just for the sake of cracking some dull and flat dependency relations was an overkill after all? The answer is no. Antecedent bracketing structures is but the most super-

[1] For the more verbally inclined, we need to simply enter every dependency domain and establish an arc from the local head to (the head of) each of its dependents.

ficial aspect of our grammar logic's proofs – the function argument relations of the implicational core are retained, and in fact coexist with the dependency annotations (including higher-order ones) in the logic's term calculus, which we have been unjustly ignoring.

The far richer type and term structure of the syntactic calculus creates, in turn, ample opportunity for the passage to semantics. We are of course still presented with the cautious option of simply forgetting about dependencies, retracting to the same dependency agnostic **ILL** we did earlier. But the radical path would have us preserve dependency operators in the intermediate station of derivational semantics[1], in hope of them findind downstream applications later on. A first merit would be the availability of richer type assignments in the lexical semantics domain, where previously identical function types become distinguishable by virtue of their modal decorations. There, syntactic modalities can be translated to semantic operators, lifting the target signature accordingly – *monads* make for a natural choice [Kobayashi, 1997]. Furthermore, dependents are delineated and identified by their syntactic roles, allowing distinct semantic treatments, if so desired. Even in the modest setting of a non-inflated translation, modalities can help tell which of the (possibly many) combinations of semantic slots are occupied in a given construction, allowing the correct compositional recipe to be retrieved from the semantic lexicon. In the edge case, they open the possibility for a homomorphic translation that "forgets" parts of the implicational directives of the logic, relying instead on the source dependency markings for the construction of its semantic terms.

9 Key References & Further Reading

The chapter has presented novel work, assembled and expanded from streamlined bits and pieces of earlier published drafts. The good news is you're reading the most extensive overview available at the time of writing. The bad news is I don't have much references for you – if it's any consolation, there's still two more chapters to go. Of work not included in this chapter or the ones to come, the one by Moortgat et al. [2022] has dependency modalities play a key role in the annotation and generation of complex verb clusters with subtle semantic dependencies, the recursive nature of which seems to break the omnipotence attributed to neural language models. For a less avant-garde reading, I'd like to draw your attention to this chapter's historical roots, namely the manuscript of Moortgat and Morrill [1991]. If unsated you crave for more modalities-as-dependencies, you have the option of either writing something yourself or waiting until someone else does.[2]

More broadly and outside their use as dependency markers, unary modalities have long now been a tempting way to encode linguistic features – see Heylen [1997] and Johnson [1999a,b], for instance. Further away from our

[1] And thus necessarily also the extraction pair $\blacklozenge_x, \blacksquare_x$.

[2] Calling it now, next big development in Moortgat et al. [2051].

niche territories, the relation between dependency grammars and lexicalism is extensively analyzed by Kuhlmann [2010], who asks (and answers) the question of which formal grammars induce what kind of dependency structures. On a relevant note, the slightly disheveled categorial dependency grammars make for an attempt to provide a categorial treatment of dependency structure, where the arcs and their labels are themselves the grammar's propositional constants [Dekhtyar et al., 2015]. For even more general literature on constituency grammars, dependency grammars or instantiations thereof, I will unfortunately have to direct you towards your favorite web search engine.

Chapter Bibliography

J. W. Backus. The syntax and semantics of the proposed international algebraic language of the zurich acm-gamm conference. In *Proceedings of the International Conference of Information Processing UNESCO Paris June*, 1959.

N. Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

M. Dalrymple. *Lexical functional grammar*. Brill, 2001.

M.-C. de Marneffe, C. D. Manning, J. Nivre, and D. Zeman. Universal Dependencies. *Computational Linguistics*, 47(2):255–308, 07 2021. ISSN 0891-2017.

M. Dekhtyar, A. Dikovsky, and B. Karlov. Categorial dependency grammars. *Theoretical Computer Science*, 579:33–63, 2015.

C. J. Fillmore et al. Frame semantics and the nature of language. In *Annals of the New York Academy of Sciences: Conference on the origin and development of language and speech*, volume 280, pages 20–32. New York, 1976.

H. Gaifman. Dependency systems and phrase-structure systems. *Information and control*, 8(3):304–337, 1965.

G. Gazdar, E. Klein, G. K. Pullum, and I. A. Sag. *Generalized phrase structure grammar*. Harvard University Press, 1985.

M. Haspelmath. Arguments and adjuncts as language-particular syntactic categories and as comparative concepts. *Linguistic Discovery*, 12(2):3–11, 2014.

H. Hendriks. The logic of tune a proof-theoretic analysis of intonation. In *International Conference on Logical Aspects of Computational Linguistics*, pages 132–159. Springer, 1997.

D. Heylen. Underspecification in type-logical grammars. In *Selected Papers from the Second International Conference on Logical Aspects of Computational Linguistics*, LACL '97, page 180–199, Berlin, Heidelberg, 1997. Springer-Verlag. ISBN 3540657517.

P. Jacobson. Phrase structure, grammatical relations, and discontinuous constituents. In *Discontinuous constituency*, pages 27–69. Brill, 1987.

M. Johnson. Type-driven Semantic Interpretation and Feature Dependencies in R-LFG. In *Semantics and Syntax in Lexical Functional Grammar: The Resource Logic Approach*. The MIT Press, 03 1999a. ISBN 9780262271172. doi: 10.7551/mitpress/6169.003.0012.

M. Johnson. A resource sensitive interpretation of lexical functional grammar. *Journal of Logic, Language and Information*, 8(1):45–81, 1999b.

S. Kobayashi. Monad as modality. *Theoretical Computer Science*, 175(1):29–74, 1997.

M. Kuhlmann. *Dependency Structures and Lexicalized Grammars: An Algebraic Approach*, volume 6270. Springer, 2010.

N. Kurtonina and M. Moortgat. Structural control. *Specifying syntactic structures*, pages 75–113, 1997.

I. A. Mel'cuk. *Dependency syntax: theory and practice*. State University of New York Press, 1988.

I. A. Mel'cuk. Levels of dependency in linguistic description: Concepts and problems. *Dependency and Valency. An International Handbook of Contemporary Research*, 1:188–229, 2003.

M. Moortgat and G. Morrill. Heads and phrases: Type calculus for dependency and constituent structure. *Manuscript, Universiteit Utrecht*, pages 429–450, 1991.

M. Moortgat, K. Kogkalidis, and G. Wijnholds. *Diamonds are Forever: Theoretical and Empirical Support for a Dependency-Enhanced Type Logic*. Studies in Computational Intelligence (SCI). Springer, 2022.

J. Nivre, M.-C. de Marneffe, F. Ginter, J. Hajič, C. D. Manning, S. Pyysalo, S. Schuster, F. Tyers, and D. Zeman. Universal dependencies v2: An evergrowing multilingual treebank collection. *arXiv preprint arXiv:2004.10643*, 2020.

M. Pentus. Lambek grammars are context free. In *[1993] Proceedings Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 429–433. IEEE, 1993.

C. Pollard and I. A. Sag. *Head-driven phrase structure grammar*. University of Chicago Press, 1994.

P. Sgall, E. Hajicová, E. Hajicová, J. Panevová, and J. Panevova. *The meaning of the sentence in its semantic and pragmatic aspects*. Springer Science & Business Media, 1986.

D. D. Sleator and D. Temperley. Parsing english with a link grammar. *arXiv preprint cmp-lg/9508004*, 1995.

L. Tesnière. *Elements of structural syntax*. John Benjamins Publishing Company, 2015. Original publication date 1959.

CHAPTER III

Proof Extraction

A program that only lives on paper is not a program.

With our theorycrafting over, we have in our hands an uninstantiated descriptive model of syntactic and semantic composition, promising to capture dependency relations while keeping both its feet set firmly in type theory. Unfortunately, it is well attested by now that "all models are wrong..." [Box, 1976]. Any promise of theoretical universality, cognitive plausibility, linguistic intrinsicness or what have you, would require some degree of handwaving and conjencturing that I am not comfortable with. What is undisputable, however, is the nobility of our goals and the purity of our tools: the *omniversality* of the λ calculus [Wadler, 2015] asserts that our modeling approach is not some ad hoc machinery designed to tackle a highly localized problem, but the one and only programming language ever worth writing. Beyond purpose and methodology, and having *a priori* given up any ideation of truth, the only measure of success for our model is that of its utility ("...but some are useful", [Box, 1976]). This sets up a new research imperative: we must prove the model useful!

"Oof, that's a tricky one", you might say, and you wouldn't be wrong. Thankfully, there are two tried and tested ways to proceed. The first is the way of the scholar, which requires a rare combination of high intellectual capacity, strong persuasive skills and a pinch of luck. It starts off with some profound abstract thinking, gradually overtaken by agressive campaigning: bashing the competition at workshops and conferences, making bold claims and generating traction at any chance given (optionally over social media platforms, for the modernists), eventually building a cult of personality, and fi-

nally resting on your laurels as the hype becomes self-sustainining – at long last, utility affirmed by popular approval. This path, sometimes called the scientific method, is a rather involved and painstakingly slow process, a high stakes gamble that only starts yielding profits in the long run. More befitting the modern paradigm of the mobile, adaptive, multi-purpose researcher is the alternative path, the way of the engineer. A shorter term investment, it requires only the acquirable skills of endurance and hardheadedness, and offers a recipe that's easier to follow: simply swing at it until it cracks. After a lot of obsessive iteration and self-correction (interchanged with the occasional feeling of despair), (f)utility will sooner or later be affirmed by cold, hard numbers. We'll go for this one.

This choice has some methodological repercussions. Under more tranquil circumstances, we'd wait for the theory to be disseminated, criticized, adapted, error-corrected and returned to sender before finally moving on; it being a theory of language, this would entail a thorough qualitative analysis of several kinds of linguistic phenomena, coupled with a theoretical investigation of what it can or cannot adequately capture. We are however in a compressed timeframe, forcing our hand into putting it straight to the test; the pragmatic approach is then to try and directly align it with real-world linguistic data at scale, and hope for the best. The process, called *proof extraction*, revolves around "proving" some source corpus of syntactically annotated sentences via the design and application of an algorithm tasked with translating the existing annotation format into derivations of the target grammar – in our case, a grammar of dependency-enhanced compositional assembly. Proof extraction serves a ternary purpose. One, it gives us access to an uncompromisingly realistic testbed upon which we can immediately inspect and iteratively finetune the specifics of the grammar logic. Two, it fills in for a strict and impartial external critic in providing a quantitative evaluation regime – at each point in time, we are able to measure the proportion of source analyses (and corresponding linguistic phenomena) the algorithm provides a (reasonable) output for. And three, the end-yield of this process has merit of its own. As a derived dataset, it is first a building block necessary for populating the computational toolshed of the theory, and also a public resource for the world to with as they please.

Before we get started, let me apologize for the gloomy mood that might permeate this chapter here and there. Proof extraction has been the most gruesome and iterated aspect of my path to doctorhood, its maintance and upkeep having an almost annual recurrence (with nothing festive about it).[1] For the sake of scientific transparency, I have no choice other than to expose it in all its horrific details, but it might be that my fatigue of the topic occassionally reflects in a less-than-usually interesting prose.

[1] Fun(?) fact: at least one revision to the extraction was made after these lines were written.

10 Preliminaries

10.1 The Dutch Language

For our linguistic inquiries, the focus will be on Dutch. Other than being the language I was contractually obliged to conduct this research on, Dutch is an interesting specimen, the idiosyncracies of which have in the past proven quite a topic of debate for others, and a source of headaches for myself. I don't have any intention (or delusion of competence) to casually throw a detailed exposition of the Dutch grammar here, but a brief and superficial typological overview might help smooth the transition into the what is to come. If the section has the opposite effect to the one intended, know that I'll hold no grudge if you skip ahead.

But first things first. Dutch is a West Germanic language, spoken primarily in the Low Countries within Europe by some 25 million speakers. Owing to the Netherlands' nasty colonial history, Dutch has left a noticeable mark on the global linguistic atlas: it has played a primary role in the evolution of Afrikaans and, to a far lesser extent, Indonesian, while native Dutch speakers can be found as far as South America and the Dutch Carribean region. Demographics aside, the language is said to be one of the closest relatives of English and German, sharing many of their morphosyntactic characteristics – we'll go through some of those together. The abbrevations in the glosses to follow are industry standard, you can find their transcriptions in Table V.1 of Appendix A. Any internal inconsistencies are deliberate: I am trying to gloss only those features relevant to the local discussion.

10.1.1 The Noun Phrase

Nouns Dutch nouns have three grammatical genders: the masculine, the feminine and the neuter. The masculine and feminine genders are morphologically indistinguishable – telling them apart is done on the basis of the lexicon alone.

(III.1) a. *de aard*
the earth(M)

b. *de zee*
the sea(F)

c. *het bos*
the forest(N)

There's two numbers, the singular and the plural, the latter constructed with the aid of the suffix *-en* or rarely *-s*, depending on the noun.

(III.2) a. *de zeven zee-ën*
the seven sea-PL
'the seven seas'

b. *de trekvogel-s*
the migratory.bird-PL
'the migratory birds'

Case markings are not overtly realized, except for some mostly frozen leftovers from the distant past – their functionality has largely been replaced by word order constraints, with the indirect object (formerly in the dative) preceding the direct object (formerly in the accusative), and periphrastic constructions, with the preposition *aan* 'to' used to indicate indirect objects (see Gloss III.8 later on), and *van* 'of' to substitute the genitive.

(III.3) *het oog van de geest*
the eye of the mind
'the mind's eye'

A cute peculiarity of Dutch is the strikingly common use of a productive diminutive form, denoting either small size or an affectionate disposition – diminutives are formed with the suffix *-tje* or regionally *-ke*, and are always neuter (notice the gender change of Gloss III.4).

Determiners Determiners precede the noun and match its gender and number. There are two articles: the definite, and the indefinite. The definite has two singular forms *de*/*het* (non-neuter/neuter) and a single plural *de*. Non-neuter forms are used for both masculine and feminine (refer back to Gloss III.1).

(III.4) a. *de klein-e vogel*
the(NN.SG) small-DEF bird
'the small bird'

b. *het klein-e vogel-tje*
the(N.SG) small-DEF bird-DIM
'the small birdie'

The indefinite *een* is gender-agnostic, and has no plural form.

(III.5) a. *een donker lied*
a dark song
'a dark song'

b. *donker-e lied-eren*
dark-PL song-PL
'dark songs'

Indefinite pronouns can be used to convey universal or existential quantification and negation, materializing as substitutes for determiners (e.g. *alles* 'all', *sommige* 'some', *geen* 'no', etc.), or as stand-alones (e.g. *iets* 'something', *niets* 'nothing', etc.).

(III.6) *Alle mensen willen iets, sommige willen alles.*
all people want something some want everything
'All people want something, some want everything.'

Possessive and demonstrative pronouns can also enact determiners, both uninflected (except for the first person plural *ons* which inflects like an indefinite adjective, see next paragraph).

(III.7) *De eend is mijn favoriet vogel.*
the duck is my favorite bird
'The duck is my favorite bird.'

(III.8) *Het bos is onz-e tempel.*
the forest is our-DEF temple
'The forest is our temple.'

There's two types of demonstrative pronouns: the proximal *deze/dit* (plural *deze*) and the distal *die/dat* (plural *die*). The last two do double duty as relative pronouns – more on that later. When used to substitute (rather than determine) a noun in a copula, each variant reverts to its neuter singular form.

(III.9) a. *Deze bergen zijn hoog.*
these(PL) mountains are tall
'These mountains are tall.'

b. *Dit zijn mijn bergen.*
this(N) are my mountains
'These are my mountains.'

Adjectives Adjectives used as nominal modifiers find their place between the determiner and the head noun. They appear inflected with an *-e* affix in all cases except for the idefinite use with a neuter singular noun (refer back to Gloss III.5a, contrast with III.4). The same affix applies to nominalized adjectives, which can allow the ommission of a contextually implied noun, or be used independently as an abstract concept or a quantifying property.

(III.10) *dood aan het heilig-e*
death to the holy-NMLZ
'death to the holy'

(III.11) *De zachtmoedig-en zullen niets beërven.*
the meek-NMLZ.PL shall nothing inherit(INF)
'The meek shall inherit nothing.'

An alternative inflection with a *-s* affix marks the partitive use, when the adjective modifies an indefinite pronoun.

(III.12) *iets ernstigs-s*
something grave-PTV
'something grave'

The language provides acccess to a comparative and a superlative form, via the affixes *-er* and *-ste* respectively, or periphrastically with *meer* 'more' and *meest* 'most'.

(III.13) *het donker-ste lied*
the(N.SG) dark-SUP song
'the darkest song'

Personal Pronouns Noun phrases can be substituted by personal pronouns, which in Dutch are morphologically marked for case and gender. The nominative is used for the subject position, the genitive corresponds to the possessive determiners discussed earlier, and the accussative is used to denote objects.

(III.14) *Ik zie een vreemde man in de spiegel.*
I(NOM) see a strange man in the mirror
'I see a strange man in the mirror.'

(III.15) *Hij merkt mij op.*
he(NOM) perceives me(ACC) on
'He notices me.'

A dative form is sometimes exceptionally used for indirect objects in the third person plural. Depending on the regional variation, personal pronouns must match the grammatical gender of the noun they refer to, or simply default to the neuter (reserving the masculine and feminine forms for animates). Third person singulars are also interchangeable with the appropriate demonstratives. Personal pronouns come in two variants: the stressed (emphatic) and the unstressed (standard).

10.1.2 The Verb

Conjugation In their citation form, verbs match their infinitival versions, regularly consisting of the verbal stem plus *-en*. Verbal conjugation patterns distinguish between two grammatical tenses, the non-past and the past, and three moods, the indicative, the subjunctive and the imperative, of which the first two are morpohologically conflated. Each pattern is parameterized by person and number. Aspectual flavours, passivization and an explicit future are accessible as productive constructions with modals, the latter being the common culprits of irregular conjugation.

Participles Participles exist for both tenses, and have a multitude of uses. The present participle is formed by affixing *-de*, and is commonly employed as an duration-denoting adjective or adverb, always inflected in the first case, and optionally in the second.

(III.16) *de zweven-de vrouw*
the levitate-PRS.PTCP woman
'the levitating woman'

Rarely, it can be used as a complement to the auxiliary *zijn* 'to be' to produce a kind of present continuous. Present participles of transitives can attach to

the end of their object nouns, appearing as fused compounds. The past participle is more versatile. Regular past participles are formed by prefixing *ge-* to the verbal stem (morphophonological terms and conditions may apply) and substituting the infinitival suffix for either *-t*, *-d* or nothing, depending on the stem's last phoneme. Like the present participle, it can be used as an adjective, denoting now a completed event.

(III.17) *beelden met ge-sloten ogen*
images with PST.PTCP-close eyes
'closed eye visuals'

Combined with with the modal *hebben* 'to have' (or exceptionally *zijn* for unaccusatives and verbs of movement), it produces the perfect tense.

(III.18)

Combined with the modals *worden* 'to become' and *zijn* 'to be', it produces the passive voice and its perfect tense.

Infinitives Infinitival forms commonly occur as the verbal complements of a modal, auxiliary or perception verb.

(III.19) *Ik wil slapen.*
I want sleep(INF)
'I want to sleep.'

Depending on the modal, the preposition *te* 'to', or the discontinuous *om ... te* 'to', may be either necessary, optionally admissible or completely disallowed – if one does manifest, it precedes the infinitive.

(III.20) *Ik probeer te schrijven.*
I try to write(INF)
'I try to write.'

The discontinuous variant is only selected for by specific verbs – it can enclose linguistic material, like the infinitive's object or any adverbs modifying it.

(III.21) *Ik dwing mezelf om wat te schrijven.*
I force myself _ something to write(INF)
'I force myself to write something.'

An infinitive directly following *aan het* can combine with *zijn* to construct the continuous aspect, in either the present or the past tense.

(III.22) *Ik ben aan het schrijven.*
I am in the write(INF)
'I am writing.'

Infinitives are also often nominalized, the resulting nominals being singular neuters.

(III.23) *Het schrijven is pijnlijk.*
The write(INF) is painful
'The writing is painful.'

Separable Verbs The verbal lexicon contains several compound items comprised of a preposition and a verbal stem, usually with a compositional meaning. The stem is separated from its prepositional prefix when the verb heads a matrix clause, in which case the preposition is moved to the end of the clause – in all other cases, including the finite form in subordinate clauses, the two remain attached. When inflecting for the perfect, the prefix applies to the stem (i.e. after the preposition).

10.1.3 The Sentence

By far the most fun aspect of Dutch is its absolutely wild sentential word order.

Main Clauses The main clauses inspected so far emanate a false sense of safety, coming off as SVO at first glance, with participles used for the passive voice and the perfect tense pushed to the right edge of the clause. The truth is far more sinister – the verb is placed there only by exception, abiding by the V2 rule that has it appear second for matrix clauses only. The effect becomes apparent when employing a preverbal adverb – in Gloss III.24b, both the subject and its predicate complement follow the verb in a VSO pattern.

Questions and Imperatives When it comes to questions, things look familiar again. Line in English, wh-questions begin with an interrogative pronoun or adverb (e.g. *wie* 'who', *wat* 'what', *welk* 'which', *waarom* 'why' etc.), which is immediately followed by the conjugated verb (Gloss III.24d). In direct questions without an interrogative, as well as positive imperatives, the verb is placed first (Glosses III.24c and III.24e). In negative imperative sentences and impersonal commands, the infinitival is placed last.

(III.24) a. *Frans verkoop-t kaas.*
Frans sell-PRS.3SG cheese
'Frans sells cheese.'

b. *Morgen verkoop-t Frans kaas.*
tomorrow sell-PRS.3SG Frans cheese
'Frans will sell cheese tomorrow.'

c. *Verkoop-t Frans kaas?*
sell-PRS.3SG Frans cheese
'Does Frans sell cheese?'

d. *Wie verkoop-t kaas?*
who sell-PRS.3SG cheese
'Who sells cheese?'

e. *Verkoop kaas!*
sell(IMP) cheese
'Sell cheese!'

Subordinate Clauses Subordinate clauses is where things really get interesting. Unaffected by the V2 rule, the word order turns out to be SOV; this affects indirect questions, verbal complements and relative clauses alike. Indirect questions are straightforward – modulo the word order permutation, they match their direct counterparts.

(III.25) *Weet je wie kaas verkoopt?*
know you who cheese sells
'Do you know who sells cheese?'

Infinitives in non-nested verbal complements are likewise just pushed to the end of their clause.

(III.26) *Frans wil koopman worden.*
Frans wants merchant be(INF)
'Frans wants to be a merchant.'

Relative clauses are instigated by relative adverbs and pronouns. Interestingly, the language does not make an overt distinction between an object- and a subject- relative pronoun; combined with the SOV word order, and the absence of case markings, the effect is that the two relative clause types end up having the exact same surface form when the grammatical gender of the antecedent noun and the non-gap embedded argument are the same – contrast the two sentences below.

(III.27) a. *het geheim dat een bos verbergt*
the secret(N) that a forest(N) hides
i. 'the secret the hides a forest'
ii. 'the secret that a forest hides'

b. *de kerk die vuur opslokt*
the church(NN) that(NN) vuur(N) consumes
'the church that fire consumes'

The SOV order means that the chaining of verbs requiring non-finite complements inadvertently leads to *verb clusters*, i.e. collections of two or more verbs situated within the dependent clause and adjacent to one another. Verb clusters are marked by their inability to accommodate non-verbal material, and may follow a number of different word orders, which don't necessarily abide by the order of selectional dominance. The question of which factors influence the grammaticality of word order variations is a hot potato and a topic of active research for decades – to make matters worse, these factors tend to differ between regional variations of the language.[1] What follows are some

[1] As a fun trivia, out of the 6 possible orderings of 3-verb clusters, 4 to 5 were found admissible by Dutch speakers depending on the construction [Barbiers, 2005].

simplified common observations – the interested reader should find the thesis of Augustinus [2015] a good entry point towards more detailed discussions.

For starters, bare infinitives usually follow their governor – but: this is not necessarily the case for clusters of 2 verbs where the finite verb is a modal. The first two below depict the canonical and "inverted" word orders; the third example is ungrammaticaul due to the adjective *naamloos* interrupting the cluster.

(III.28) a. *... waar ik naamloos zal rusten*
.... where I nameless will rest(INF)

b. *... waar ik naamloos rusten zal*
... where I nameless rest(INF) will
'... where I will rest nameless'

c. * *... waar ik zal naamloos rusten*

Past participles used in the formation of the perfect or passive may occur either to the left or the right of the tense auxiliaries *hebben* and *zijn*, leading to either a German- or English- like construction.

(III.29) a. *... omdat ik de eend ge-zien heb*
... because I the duck PST.PTCP-see have

b. *... omdat ik de eend heb ge-zien*
... because I the duck have PST.PTCP-see
'... because I have seen the duck'

This gets complicated by the so-called IPP (*Infinitivus Pro Participio*) effect, where a participle that selects for an infinitive changes to an infinitive itself, creating a cluster in the process – once more, whether this substitution is mandatory, optional or altogether impossible is lexically decided.

(III.30) a. *Ik heb de eend ge-zien.*
I have the duck PST.PTCP-see
'I have seen the duck.'

b. *Ik heb de eend zien vliegen.*
I have the duck see(INF) fly(INF)
'I have seen the duck fly.'

c. *Ik heb de eend een nest zien maken.*
I have the duck a nest see(INF) make(INF)
'I have seen the duck make a nest.'

Next, the infinitival head of a dependent clause may be forced to occur directly after the verb dominating it, if the latter belongs to a closed set of so-called *raising* verbs (i.e. the raiser is infixed between the infinitive to the right, and the infinitive's non-verbal arguments to the left). These include modals like *willen* 'to want' and *moeten* 'must', perception verbs like *horen* 'to hear' and *zien* 'to see', and some nondescript verbs like *doen* 'to do' and *laten* 'to let'. As before, they can be subcategorized as obligatory raisers and optional ones.

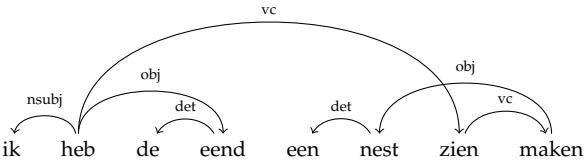


Figure III.1: Crossing dependencies in the 2-verb cluster of Gloss III.30c.

(III.31) *Ik denk dat ik iemand hoor naderen*
I think that I someone hear approach(INF)
'I think I hear someone approaching.'

Yet other verbs like *verplichten* 'to forbid' select not a bare infinitive but a *te*-marked infinitival phrase, which they leave intact at the end of the clause – a phenomenon known as *extraposition*. The twist is that the intersection of extraposition verbs and raising verbs is non-empty – *proberen* 'to try', for instance, can behave as either.

(III.32) a. *Ik denk dat hij probeert iets te zeggen.*
I think that he tries something to say(INF)
b. *Ik denk dat hij iets probeert te zeggen.*
I think that he something tries to say(INF)
'I think that he is trying to say something'

Typology aside, Dutch verb clusters have been a favorite topic of debate for formal grammarians for a while now, since their construction requires expressive capacity beyond what a context-free grammar can offer, and thus brinking an end to any delusion that human languages are context-free [Huybregts, 1984; Shieber, 1985, *inter alia*].[1] The impenetrable nature of verb clusters means that verbs that partake in their construction may often be forced to detach from their arguments, or worse yet become stranded from them by the infixation of another verb in between – in the dependency grammar paradigm, these discontinuities materialize as non-projective (or cross-serial) dependencies – see Figure III.1 for an example.

Adverbs Last item on the agenda are adverbs, commonly enacted by uninflected adjectives. When not appearing in the emphatic first position, adverbial phrases must occur after definite objects and before indefinite ones, their internal order following a time-manner-location pattern.

(III.33) *Turkse varkens doden momenteel laf burgers in Koerdistan.*
Turkish pigs kill currently coward(ADV) civilians in Kurdistan
'Turkish pigs are cowardly killing civilians in Kurdistan right now.'

[1] Or, depending on the reader, that Dutch is a human language.

Adverbial counterparts to pronouns make for locative adverbs (i.e. *dit* becomes *hier* 'here', *dat* becomes *daar* 'there', *wat* becomes *waar* 'where', etc.). Unique among them is *er* from the singular neuter personal pronoun *het*, which finds use as an unstressed locative or a subject-filler for an impersonal passive.

(III.34) *Er is geen zaken doen op een dode planeet.*
there is no business make(INF) on a dead planet
'There is no business on a dead planet.'

When a preposition would apply to a singular neuter pronoun, the adverbial form is commonly employed instead; their positions are inverted, and if they end up adjacent they become fused, giving rise to a so-called pronominal adverb. Pronominal adverbs are common in Dutch, and find use as interrogative adverbs or relative pronouns.

(III.35) a. * *Wat prat je over?*

b. *Waar praat je over?*
what talk you about

c. *Waarover praat je?*
what.about talk you
'What are you talking about?'

d. *We praten erover.*
we speak it.about
'We are talking about it.'

10.2 Parsing: Recognition vs. Discovery

With this lengthy detour over, we arrive at some painful truths: if there's any message to take home, it's the sheer amount of complexity inherent in a natural language grammar. Nominals come with a ton of morphosyntactic rules, exposing the fact that there's no syntactic entity as simple and unmarked as the idealized NP we have extensively utilized in earlier chapters. The erroneous ommission or addition of as much as an inflectional marking suffices to turn a phrase ungrammatical. The same issue pesters verbal morphology – combining a verb with its potential arguments requires solving a number of constraints relating to number and person (let aside any notions of semantic compatibility). To see these constraints solved on the type level is not out of the question [Pollard, 2004] (modalities might in fact make for the perfect tool for the job, see Heylen [1997]); it is, however, a problem in its own right. But all these concerns are trivial compared to the monumental intricacies laid down by word order constraints. For a language like Dutch, devising the type assignments and structural rules that exactly allow the word permutations admissible by the language is a tremendous undertaking that requires navigating a complex network of layered rules and exceptions subject to regional variety.

That is not to say that such endeavours are without merit; a formalism that presents itself as syntactic in nature yet fails to provide a general and transparent account of morphosyntactic and word order constraints wouldn't be particularly honest. The above remarks bring forth, however, a question of priorities, and by implication put us at a juncture point. We have on the one hand the option to pursue the Lambekian holy grail: the design of a substructural type system, the proof search over which should amount to a decision procedure capable of telling sentence and non-sentence apart. Put in practical terms, we'd settle for nothing less than absolute alignment with the Dutch grammar; just enough expressivity to ensure no overgeneration, no undergeneration and a perfect resolution of any and all syntactic ambiguities. The other option is perhaps more sober – depending on our end-goals, we can check our ambitions to a more realistic level. A focus on sentence formation, for instance, would justify some morphological concessions. But since the focus here is on compositionality, we're given the chance for a much more radical leap: we can also skip word order altogether.

Now this might be met with some scepticism, but put down your pitchforks and let me explain. In the type-logical setup, we'd follow the schema of Figure I.24: start from a strict syntactic logic, and then strip it down to its bare essentials to cast it into a logic of derivational semantics (the bare essentials in our case being linear implication and the dependency modalities). With derivational semantics being itself the point of interest, why take the hard route instead of just starting directly? This is somewhat akin to the abstract categorial grammar operationalization of Figure I.25, which would have us start from a tectogrammatic logic, and transition to phenogrammar via a morphism (here left unimplemented). A perhaps less stretched operationalization would involve a two-stage inferential setup, the first stage being the phase of logical meaning assembly, followed by a higher-level phase of structural reorganization and reordering (here we'll call it quits immediately after the first). Put bluntly, this option allows us to sneak our way out of having to reason about the non-compositional aspects of syntax.[1] Obviously this alters the scope of our endeavours. Before any regrettable accusations are thrown, consider that we need not be apologetic for the elephant in the room, namely overgeneration. $\mathbf{NLP}_{\Diamond,\Box}$ is *not* intended to be the logic you put in generation mode, and its proof theory is *not* Lambek's decision process. Rather, it is an adequately expressive formalism that can accommodate the duality of function-argument structures and dependency annotations we set out to capture. The problem is more practical than theoretical: who is going to hand us these deep syntactic proofs if not for the morphism that was promised? Suffice it to say this requires adopting a new notion of parsing (still well under the deductive paradigm): that of associating a well-formed input with the correct tectogrammatic analysis. How exactly this is to be done doesn't need to concern us for now – we'll cross that bridge when we get to it.

[1] An atonement to the ghost of Montague, returned to claim his dues.

10.3 Lassy

To facilitate the agenda just established, what we need next is a sizeable resource of syntactic annotations that are both high-quality and sufficiently compatible to our needs. Fortunately, the search is rather short – the only candidate is also the perfect one: Lassy [van Noord et al., 2013].

Lassy consists of two annotated corpora, containing sentences paired with a single analysis. Analyses are provisioned by Alpino [van Noord, 2006], a powerful parser that stands the test of time by combining a high quality handcrafted lexicon, a statistical feature disambiguation model and a sophisticated collection of phrase formation rules based on the HPSG framework [Pollard and Sag, 1994]. Alpino annotations are described as spanning three axes: a hierarchical one, answering which words form a phrase, a relational one, answering what the grammatical functions between words are, and a categorial one, answering what the syntactic labels of each word and phrase are. Unlike shallow dependency grammars, Alpino does not shy away from higher-order phenomena: it serves annotations as dependency *graphs* rather than trees by employing *secondary* edges to represent words that assume multiple syntactic functions. Expanded into the equivalent tree by node duplication, the sentential structure is visualized as a tree of nodes connected by named edges (one incoming edge per node, except for the root which has none). In what follows, I will occasionally abuse terminology and call an Alpino or Lassy graph a tree, in reference to this expanded representation; it is important to remember this is distinct from the shallow dependency trees inspected earlier in the context of dependency grammars.

Nodes can be either *material*, representing words and phrases, or *phantom*, representing elided constituents – every phantom node is indexically associated to a material one. Material nodes are assigned a syntactic category label, either a lexical part of speech tag (in the case of a terminal node representing a word) or a phrasal category (in the case of a non-terminal representing a phrase).[1] The label of a phantom node may be retrieved by inspecting its material counterpart. Nodes can be told apart by their unique identifier, which differs even among nodes sharing the same index (i.e. index $\neq$ identifier). Word order has no bearing on the tree structure – the span of each material node in the sentence is just an attribute of that node. A material phrasal node connects to its constituents (themselves nodes of any kind) by virtue of directed edges labeled with grammatical functions. Modulo some (supposedly) exceptional cases, each phrasal node emits exactly one head-labeled edge, the rest being a combination of complements and adjuncts. An example analysis as provided by Lassy is shown in Figure III.2.

[1] Terminal nodes are in fact assigned tags from two distinct sets: a simplified one (denoted **pos**) and an extended one (denoted **postag**). The latter in turns consist of a generic label (denoted **pt**) and a set of label-specific morphological values. Consistent with our dismissal of morphological constraints, we use **pt** in what follows, but this is by no means a hard constraint – a short discussion will follow later.

Of the two corpora only the smallest one is of real interest. Lassy Small includes approximately 65 000 sentences, amassing a total of almost 1 million words. Its annotations have been manually checked, corrected and externally validated, with a reported 97.8% of sentences correctly analyzed, and a 98.63% of tokens correctly tagged – neither perfect, but both more than good enough for our present needs.[1] Its larger sibling favors quantity over quality – it is more than 500 times the size, but an ill-fit for our endeavours, being the parser's unmodified output. Table III.1 and Table III.2 present aggregated summaries of the syntactic category tags and depedency labels found in Lassy Small, together with their relative frequencies. A breakdown of the corpus' contents is presented in Table III.3.[2]

11 Æthel

The stage is set. We need to devise an algorithm that accepts trees like the one of Figure III.2 and spits out proofs of $\mathbf{NLP}_{\Diamond,\Box}$. Following our prior discussions, some assumptions need to be met before we get to even contemplate our approach. First, we need a clear three-way partition of the set of dependency labels, so that each dependency relation marks either a head, a complement or an adjunct. Further, we require that each dependency domain has exactly one head. Finally, we must ensure that higher order phenomena reflected in secondary edges (or phantom nodes) are homogeneous so that they can be treated in a uniform way. Unfortunately, these requirements are not always met; our first step is therefore to massage any rough edges with a series of transformations aimed at (i) harmonizing the input trees with the target logic and (ii) fixing inconsistent formattings and underspecified or otherwise incompatible annotations.

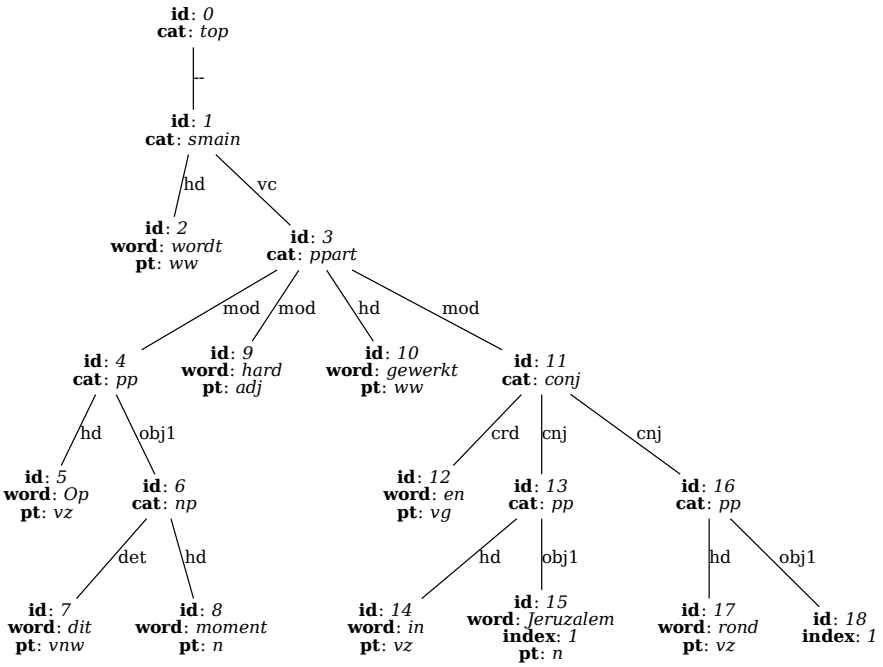
11.1 Taming Lassy

11.1.1 Edge Relabeling

A phrasal annotation canonically contains a single head, a collection of complements (no more than one of each), and a collection of adjuncts (without any restriction on their plurality), where verbal, nominal and sentential domains differ in the the labels they may contain. Deciding whether a dependency edge signifies a head, a complement or an adjunct requires little effort on our part, as the distinction has already been made. The Lassy annotation manual specifies labels {*hd*, *rhd*, *whd*, *cmp*, *crd*, *dlink*} as heads of various kinds, {*det*, *mod*, *app*, *predm*} as adjuncts, and {*body*, *cnj*, *hdf*, *ld*, *me*, *obcomp*, *obj1*, *obj2*, *pc*, *pobj1*, *predc*, *se*, *su*, *sup*, *svp*, *vc*} as complements. This is not a full partition of the set

[1] For a detailed exposition of Lassy annotation guidelines, refer to `http://www.let.rug.nl/vannoord/Lassy/sa-man_lassy.pdf` (in Dutch).

[2] Sourced from `http://nederbooms.ccl.kuleuven.be/eng/tags`.



WS-U-E-A-0000000013.p.37.s.1

Op dit moment wordt hard gewerkt in en rond Jeruzalem.
'At the moment there is hard work being done in and around Jerusalem.'

Figure III.2: Example Lassy graph. Note the identification of nodes 15 and 18 by a common index, marking the double use of *Jeruzalem* as the direct object of prepositioal phrases 13 and 16.

Tag	Description	Frequency (%)	Assigned Type
	Lassy Short POS Tags		
adj	Adjective	7.3	ADJ
bw	Adverb	4.5	BW
let	Punctuation	11.2	LET
lid	Article	10.7	LID
n	Noun	22.5	N
spec	Special Token	3.5	NP
tsw	Interjection	<0.1	TSW
tw	Numeral	2.4	TW
vg	Conjunction	4.2	VG
vnw	Pronoun	6.5	VNW
vz	Preposition	13.7	VZ
ww	Verb	13.2	WW
	Lassy Phrasal Categories		
advp	Adverbial Phrase	0.6	ADV
ahi	Aan-Het Infinitive	<0.1	AHI
ap	Adjectival Phrase	2.1	ADJP
cp	Complementizer Phrase	3.3	CP
detp	Determiner Phrase	0.2	DETP
inf	Bare Infinitival Phrase	4.7	INF
np	Noun Phrase	36.7	NP
oti	Om-Te Infinitive	0.8	OTI
pp	Prepositional Phrase	23.2	PP
ppart	Past Participial Phrase	4.2	PPART
ppres	Present Participial Phrase	0.1	PPRES
rel	Relative Clause	1.9	REL
smain	SVO Clause	4.7	S_{main}
ssub	SOV Clause	0.8	S_{sub}
sv1	VSO Clause	<0.1	S_{vi}
svan	Van Clause	<0.1	S_{van}
ti	Te Infinitive	1.8	TI
whq	Main WH-Q	0.1	WH_{q}
whrel	Free Relative	0.2	WH_{rel}
whsub	Subordinate WH-Q	0.2	WH_{sub}
du	Discourse Unit	2.6	n/a
mwu	Multiword Unit	5.9	–
conj	Conjunct	5.7	–

Table III.1: Lassy POS tags and phrasal category labels, and corresponding atomic types. The *du* category doesn't make its way to the extracted proofs, while *mwu* and *conj* don't have their own type.

Label	**Description**	**Frequency** (%)	**Modality**
app	Apposition	0.8	$\Box_{app}$
body	WH-question Body	0.1	$\Diamond_{whbody}$
body	Relative Clause Body	0.1	$\Diamond_{relcl}$
body	Complementizer body	2	$\Diamond_{cmpbody}$
cnj	Conjunct	4.3	$\Diamond_{cnj}$
crd	Coordinator	1.8	–
crd	Second Element of Correlative	<0.1	$\Diamond_{cor}$
det	Determiner	9.7	$\Box_{det}$
dlink	Discourse Link	0.2	n/a
dp	Discourse Part	0.8	n/a
hd	Phrasal Head	27.8	–
hdf	Final Part of Circumposition	<0.1	$\Diamond_{hdf}$
ld	Locative Complement	0.5	$\Diamond_{ld}$
me	Measure Complement	0.1	$\Diamond_{me}$
mod	Modifier	16.4	$\Box_{mod}$
mwu	Multiword Part	5.1	n/a
nucl	Nuclear Clause	0.5	n/a
obcomp	Comparison Complement	0.1	$\Diamond_{obcomp}$
obj1	Direct Object	10.8	$\Diamond_{obj1}$
obj2	Secondary Object	0.2	$\Diamond_{obj2}$
pc	Prepositional Complement	10.6	$\Diamond_{pc}$
pobj1	Preliminary Direct Object	<0.1	$\Diamond_{pobj1}$
predc	Predicative Complement	1.3	$\Diamond_{predc}$
predm	Predicative Modifier	0.1	$\Box_{predm}$
sat	Satellite	0.2	n/a
se	Obligatory Reflexive Object	0.7	$\Diamond_{se}$
su	Subject	6.9	$\Diamond_{su}$
sup	Preliminary Subject	<0.1	$\Diamond_{sup}$
svp	Separable Verbal Participle	0.7	$\Diamond_{svp}$
vc	Verbal Complement	2.8	$\Diamond_{vc}$
tag	Appendix	0.1	$\Diamond_{tag}$
whd	WH-question Head	0.1	–
rhd	Relative Clause Head	0.1	–

Table III.2: Lassy dependency labels, and corresponding modalities. Grayed out dependencies don't make their way to the extrated proofs. Heady dependencies don't get a modality.

Treebank	Contents	Acronym	# Sentences	# Words
DPC	Dutch Parallel Corpus	`dpc`	11 716	193 029
Wikipedia	Wikipedia Pages	`wiki`	7 341	83 360
WR-P-E	E-magazines	`WR-P-E-C`	14 420	232 631
	Newsletters	`WR-P-E-E`		
	Teletext Pages	`WR-P-E-H`		
	Web Sites	`WR-P-E-I`		
	Wikipedia	`WR-P-E-J`		
WR-P-P	Books	`WR-P-P-B`	17 691	281 424
	Brochures	`WR-P-P-C`		
	Guides & Manuals	`WR-P-P-E`		
	Legal Texts	`WR-P-P-F`		
	Newspapers	`WR-P-P-G`		
	Periodicals & Magazines	`WR-P-P-H`		
	Policy Documents	`WR-P-P-I`		
	Proceedings	`WR-P-P-J`		
	Reports	`WR-P-P-K`		
	Surveys	`WR-P-P-L`		
WS-U	Auto Cues	`WS-U-E-A`	14 032	184 611
	News Scripts	`WS-U-T-A`		
	Text for the Visually Impaired	`WS-U-T-B`		

Table III.3: Breakdown of Lassy Small contents.

of dependency labels of Table III.2, as several items fall in neither of the above bins – more on that in a second. First, we'll take on the less severe problem of standardizing the labels already categorized.

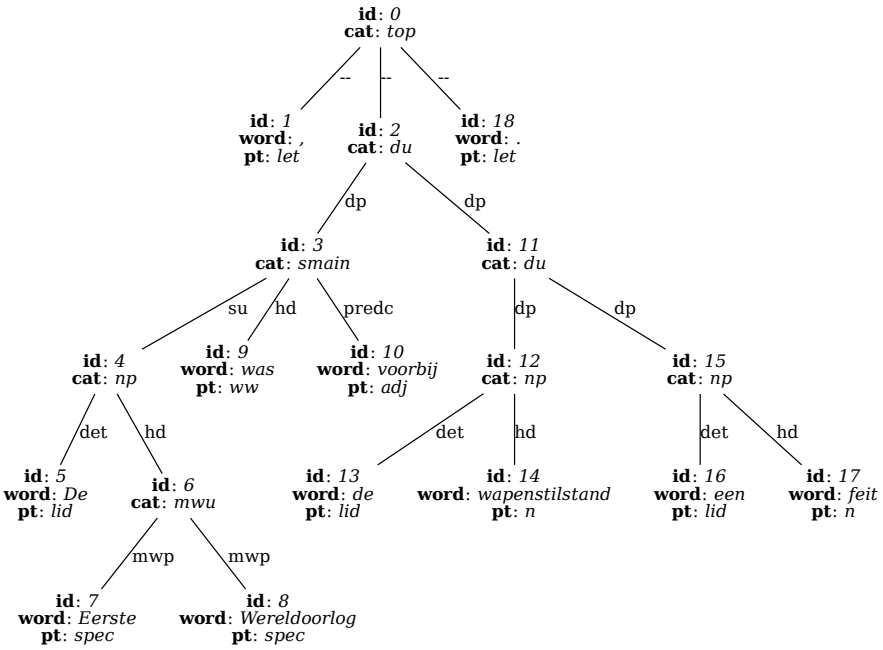
***body*, but what kind of?** A minor problem appears in the reuse of the *body* label in three different contexts: as the body of a wh-question, a relative clause, or a complementizer. This conflation is perfectly reasonable from Lassy's angle: in all three constructions, the head specifies the dependency (being either *whd*, *rhd* or *cmp*) and selects for a subordinate clause with a gap that is practically agnostic to its external context. This scheme backfires in our setup, due to heads not carrying their own annotation but rather imposing one on their complements – if we keep the *body* relation as is, the three different types of head would be indistinguishable. Counteracting this is easy; we simply subcategorize the *body* label according to the label of its head, giving rise to labels *whbody*, *relcl* and *cmpbody*. This doesn't have any intended consequences on the interal structure and typing of the complement, as its contents still has no premonition as to what diamond it will eventually be assigned.

***det* or *mod*?** Lassy is occassionally inconsistent with the use of the determiner *det* and modifier *mod* labels in the nominal domain, marking either as the other in various contexts. Examples include marking indefinite, demonstrative or possessive pronouns as modifiers, and, the other way around, marking numerals, names in genitive form, quantifiers and complex quantifying phrases as determiners – but neither direction is strictly followed. These annotations can at times result in a phrase with multiple determiners. Despite determiners not being heads, the presence of multiple of them makes it hard to decide on a compositional structure as it poses the challenge of choosing one as the primary between them. To impose the restriction of a single determiner per nominal domain and to standardize (some of the) inconsistencies, we uniformly cast the former to determiners and the latter to modifiers, using simple lexical filtering. This results in a unique determiner per phrase (resolving constructions like *geen enkel* 'no', *de beide* 'both', etc.), and an elimination of complex determiner phrases.

Nominal and Verbal Domains Lassy uses the label *hd* to refer to both the head of a matrix clause and to the head of a noun phrase. To distinguish between the two, we relabel heads co-occurring with a determiner to *np-head*. This has no effect on our extracted types and proofs but shall help us formulate the extraction algorithm in a more transparent way.

11.1.2 Non-Compositional Annotations

Despite its admittedly high quality, Lassy has not been built with an inherent focus towards compositionality. This reflects in some not so uncommon exceptions to the canonical phrasal annotation that *de facto* necessitate some global



wiki-4941.p.4.s.2

De Eerste Wereldoorlog was voorbij, de wapenstilstand een feit.
'The first world war was over, the armistice a fact.'

Figure III.3: Example Lassy graph showcasing non-compositional annotations.

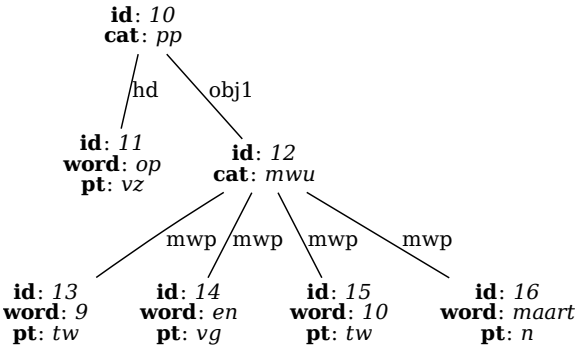
concessions and adaptations, and some local emergency measures, ranging from targeted transformations in the best case, to occassionally just giving up on a sample in the worst. The biggest problem that we are faced with right off the bat is the abundance of vague, general purpose annotation schemes to convey non-compositional structures. These come in two flavours – discourse level annotations, and multiword phrases.

Discourse Level Annotations Discourse level annotations are materialized by dependencies *dlink*, *dpart*, *nucl* and *sat*, used in a catch-all fashion in place of an actual syntactic analysis. In the example of Figure III.3 the two sentences are analyzed as "discourse parts" of a single "discourse unit" rather than matrix clauses conjoined by the comma – with some goodwill we could let that slide, but that same strategy is internally applied within the second sentence

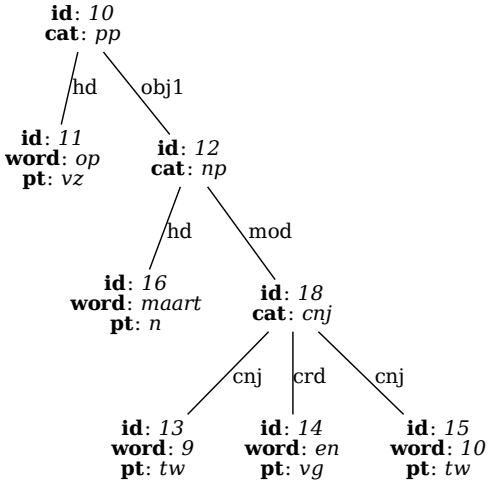
(apparently a "discourse unit" rather than a sentence), thus avoiding a proper syntactic justification for the elided verb. Unfortunatey, there is no algorithmic way to mend these pretend annotations, in part due to their wildly general use, but mostly due to the fact they give us nothing to work with. This phenomenon is unpleasantly common; discourse level annotations sum up to about 2% of the total dependency edges in the corpus, and are present in some 11 700 samples, polluting a hard to ignore 18% of the dataset. In order not to lose all the precious samples in their totality, we take the more conservative approach of simply pruning the problematic edges rather than discard the entire tree. The subtree underneath each cut is subsequently rooted as an independent sample, sprouting an array of smaller samples from the larger unusuable original; in the example under scrutiny, we end up with three samples rooted at nodes 3, 12 and 15. Albeit being a sensible solution in terms of data preservation, this has the unavoidable downside of *a priori* breaking the alignment between the source corpus and the collection of proofs to be. In order to facilitate some level of back-and-forth matching, the new samples inherit the sample name of their origin and are distinguished between one another by a suffix corresponding to the identifier of their root node.

The pruning might sound easy on paper but proves tricky in certain regards. Lassy by default provides no annotations for punctuation symbols, which are instead attached to a conventional "top" node with an unlabeled edge. By truncating trees naively, we'd be dropping punctuation that might be necessary for a phrase to remain grammatical or otherwise prove useful in the provision of a proper derivation. Internal commas, for instance, could be the key to constructing a conjunction, whereas final punctuation might be crucial in deciding the phrasal type, motivating their reinstation. Including internal and right-adjacent punctuation only, however, carries the risk of upholding only one end of circumfixing punctuations like parentheses or brackets, accidentally turning a phrase ungrammatical. The heuristic solution is to iteratively expand a truncated subtree by attaching any internal or peripheral punctuation marks, excluding opening brackets from the right edge and closing brackets and sentence-final punctuation from the left edge. To homogenize trees (truncated or otherwise), punctuations are displaced from the "top" node to the top-most node that carries an actual syntactic category label – the latter serves as the new tree's root.

The next issue to address is the occasional disconnect between a phantom node and its material counterpart as a result of pruning. To avoid trees with floating nodes, we check whether phantoms in the pruned tree can access their material counterparts. When that's not the case, the phantom node is substituted by a copy of the material one, and, in the event of it being phrasal, the entire tree that lies underneath it. The process is repeated (to circumvent the possibility of adding a new floating node when fixing the first) until a fixpoint is reached. The result is the possibile duplication of lingustic material among different prunings (i.e. a subtree that occurred once in the original Lassy sample can sometimes be found in more than one of the processed samples).



(a) Before.

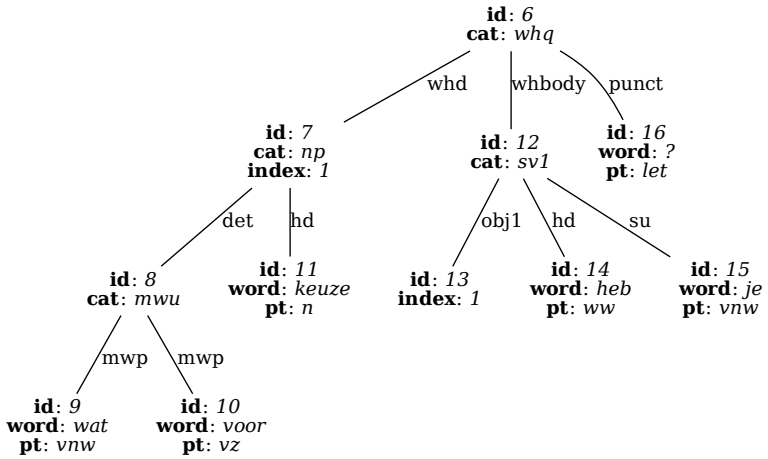


(b) After.

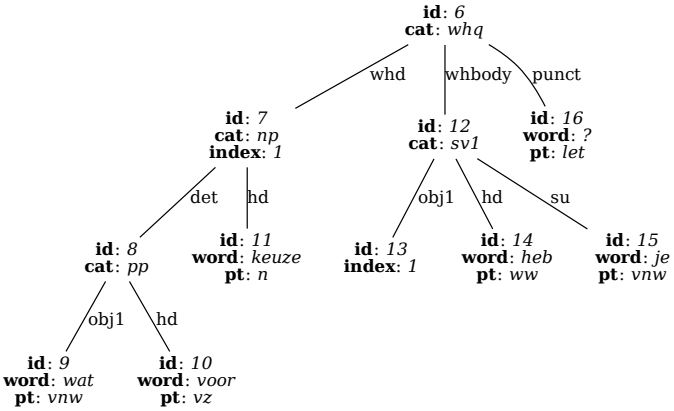
`WR-P-P-I-0000000242.p.16.s.2` (excerpt)

...op 9 en 10 maart...
'...on March 9 and 10...'

Figure III.4: Reannotating a date expression containing a conjunction.



(a) Before.



(b) Before.

WR-P-P-H-0000000039.p.8.s.3(6)

wat voor keuze heb je?
'what kind of choice do you have?'

Figure III.5: Rennotating an erroneously analyzed interrogative.

Multiword Phrases Another common pain point is the prominence of multiword phrase annotations, indicated in Lassy by the *mwu* dependency. Multiword expressions are a pervasive pest from the shadowy realms between lexicon and syntax. They can be categorized as being either (i) morphosyntactically fixed or (partially) productive expressions that deviate from the expected compositional meaning, or (ii) just compositional expressions that have an idiosyncratic frequency. In all but the first case and regardless of their semantic use, they are not necessarily without internal structure. The criteria of what constitutes a multiword phrase and what doesn't are somewhat muddy, subjective and not clearly motivated. The example of Figure III.3 claims that *Eerste Wereldoorlog* 'first world.war' is one, for instance – eliciting the questions of whether expressions like the *third/current/last/next world war* are also instances of a multiword phrase, and, if so, where the line is drawn (if at all). Anyway, multiword expressions are bad, but what's really bad is how overindulgent Lassy is with their use, which feels more like a free pass at disclaiming any responsibility of actually providing an analysis: a preposterous 5.9% of all composite phrases are labeled as being multiword expressions.

Multiword phrases are not an impassable roadblock; they can be tackled by relaxing the lexicalist word-to-type restriction, i.e. allowing entries in the lexical dictionary to be arbitrary strings rather than words. This is indeed the approach we'll follow, but only after having salvaged however many of the missing annotations as we can. Doing so is in our best interest: it will reduce the lexicon's load and provide us with a collection of annotations that are easier to generalize from. Generating structure out of thin air is of course impossible, but several existing patterns are amenable to an automatic reannotation.

A first filter can tell us whether a word is a numeral or measure by inspecting its part of speech assignment. Two numerals separated by a coordinator make for a complex numeral, in which case a new tree can be instantiated, with the coordinator and the two numerals as its daughters; the first marked as a coordinator, the other two as conjuncts. A numeral, complex or singleton, adjacent to a quantity denoting noun (like *paar* 'pair', *honderd* 'hundred', *duizend* 'thousand', etc.), a unit of measurement (like *kilo* 'kilogram', *eur* 'euro', etc.) is cast into a modifier, and the noun is cast into a head. In a similar vein, a tiny parser is employed to analyze expressions of time and date; it follows a binarization scheme that assigns headedness to the more general part of a datetime expression (i.e. year over month over day), and analyzes the rest as a modifier with internal structure; an example is presented in Figure III.4.

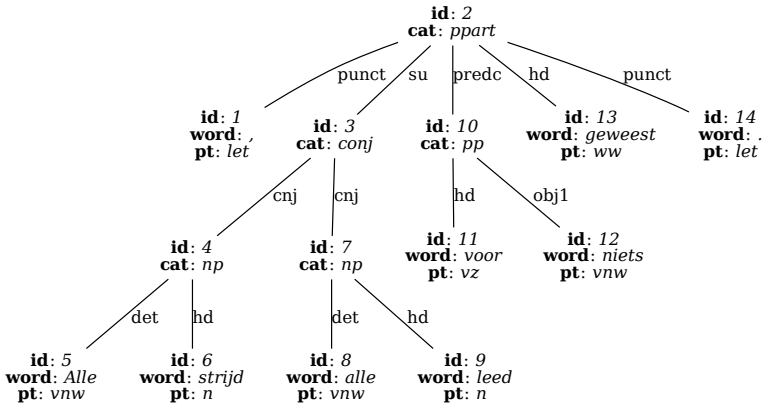
On the lexical side of things, some expressions tend to default to a multiword annotation despite not being one – clearly an artifact of Alpino's rule-based parser that was never corrected in the manual verification stage. These are for the most part prepositional phrases (like *ten noorden van* 'north of', *met uitzondering van* 'except for', etc.), which are caught and reanalyzed by having the genitive-substitute *van* attach to the modified noun (as consistently done otherwise throughout Lassy), which the remainder of the expression consumes as a direct object. More problematic is the occasional mischaracteri-

zation of the interrogative *wat voor* 'what kind of', where the preposition *voor* 'for' is analyzed as part of a multiword expression together with a phantom node coindexed with the pronoun *wat* 'what'. These are also caught and corrected, as in the example of Figure III.5; the preposition is reannotated as being the head of a prepositional phrase selecting for a direct object, that being the afforementioned phantom node. Other, less severe, cases include the mislabeling of nationality adjectives (like *afrikaans* 'Afrikan', *europees* 'european', etc.), which are recast as modifiers of the noun they were merged with. For consistency, punctuations originally analyzed as multiword parts (presumably so as not to break phrasal contiguity) are instead pushed to the topmost root.

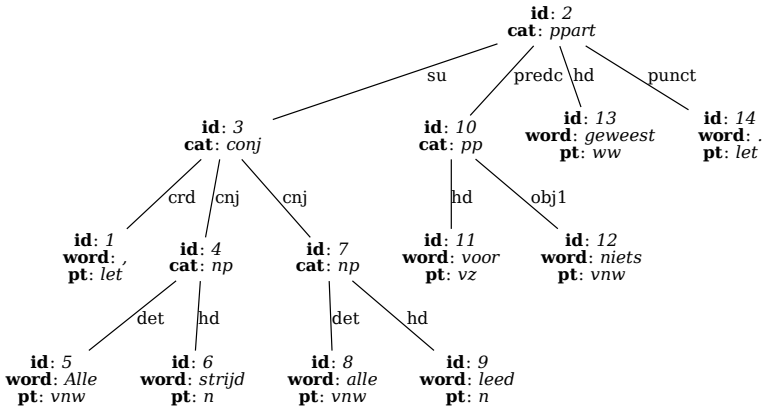
These minor changes suffice to cut down the frequency of multiword expressions to a more manageable 4.5% (an overall reduction of 25%). Unresolved expressions have their parts merged into a single node; the resulting node gets a new syntactic label, that being the most common part of speech tag of the merged units (with a bias towards *n* or *np*, if either is present in the parts). The goal is to contain the effect of multiword annotations within their own phrasal boundaries (i.e. to avoid functors higher in the tree from selecting for MWU-typed arguments). In the case of a multiword phrase consisting solely of phantom parts, the merged phrase also gets assigned an index and associating it to its material counterpart.

11.1.3 There Can Be Only One (Head)

Having dealt with structureless structures, the next thing to tackle are subtrees that fail to elect a single head, falling to civil war (when multiple nodes compete for the role) or rising to anarchy (when all relinquish it). Here, we'll need to assume the interventive role of a self-appointed stabilizing force, and take it upon ourselves to reinstate normalcy (read: we'll assign a single head of our own choice). The culprit behind both cases is always a conjunction, canonically containing a number of conjuncts (labeled *cnj*) and a single coordinator (labeled *crd*). Exceptionally, we may have an instance of a so-called *correlative conjunction*, where two words jointly perform the role of the coordinator (e.g. *zowel ... als* 'as much ... as', etc.). We resolve this by changing the label of the second coordinator (in terms of left-to-right sentential precedence) to *cor* (for correlative), which we will later treat as a complement. Otherwise in the second case we have an arrangement of conjuncts with no coordinator in between. In reality, the conjunction is licensed by a punctuation symbol (usually a comma, but sometimes a dash or an ampersand), which, being a punctuation, has gone under Lassy's radar. We heuristically resolve this by first locating any occurrence of a single punctuation infixed between headless conjuncts, relocating it to its rightful place, and assigning it a *crd* label; see Figure III.6 for an example.



(a) Before.



(b) After.

`WR-P-E-I-0000050381.p.1.s.704(2)`

Alle strijd, alle leed voor niets geweest.
'All the struggle, all the suffering were for nothing.'

Figure III.6: Reannotating a headless conjunction.

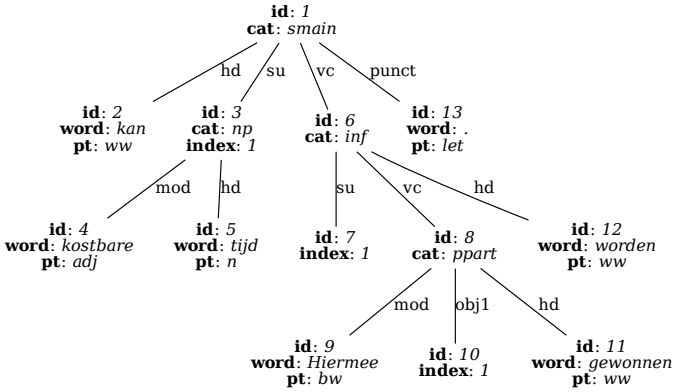
11.1.4 Phrasal Restructuring

Other than the incompatibilities detailed so far, some Lassy annotations are suboptimal for the target logic in specifying a phrasal structure that we want to treat differently than prescribed. Such cases are treated by automatically adjusting the phrasal structure to one we are happier with. Since these adjustments involve removing or establishing new subtrees and edges, we run the risk of accidentally removing the material tree that grounds a set of phantom nodes sharing the same index. As a safety measure, we first enforce the convention that the most shallow occurrence of a node should correspond to a material tree.

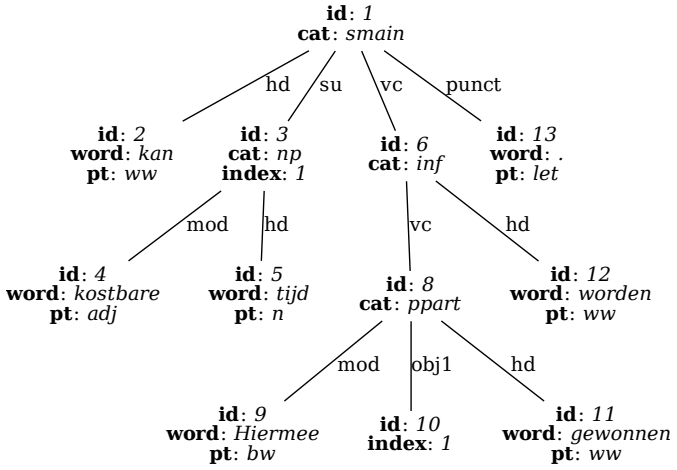
(Mis)understood Arguments Lassy treats the non-finite verbal forms (participles and infinitives) as verbal elements proper, selecting for all the arguments their finite counterparts would. Obviously, participles used for the passive or the perfect and infinitives in verbal complements cannot possibly find all these arguments, some being located in higher levels of the dependency graph. To resolve this, Lassy opts for establishing phantom nodes coindexed with the so-called understood argument; the more non-finites in the path between the one under scrutiny and the top level clause, the more phantom nodes are inserted. Some of these nodes are of a syntactic quality: non-finite forms of transitives used in passive constructions refer back to the auxiliary subject as their object. This makes past participles consistent with their use in the perfect, and infinitives consistent with their use in subordinate clauses. Others are purely semantic: both past participles and infinitives select for a subject (e.g. the "agent" of the action they denote), which can be any of the main clause's arguments. We are not happy with the latter, since no syntactic item allows for the duplication of material they demand, and they have no place in the compositional structure we seek to extract; we thus invoke our moral right to cast them away.[1] Concretely, we look for any edge with a *su* label that points to a phantom node, such that any of its non-immediate ancestors is a sentential clause with another outgoing *su* edge pointing to a node of the same index. Edges caught in our web are deleted, as are any nodes left floating; see Figure III.7 for an example. This transformation incurs a loss of semantic coindexing, which anyway is irrelevant to us: it's up to lexical semantics entries to decide what arguments they have, how their slots are filled, and how these are propagated and updated down the sentence. Other than this coindexing, it will soon be made apparent that no meaningful function-argument structures are actually lost by this erasure.

Modifier Scope Unlike complements, adjuncts in Lassy are not limited to one occurrence per unique label. In other words, they are attached in parallel

[1] Funnily, our recurring complaint with Lassy so far has been that it gives us too little. This time around, it gives us too much.



(a) Before.

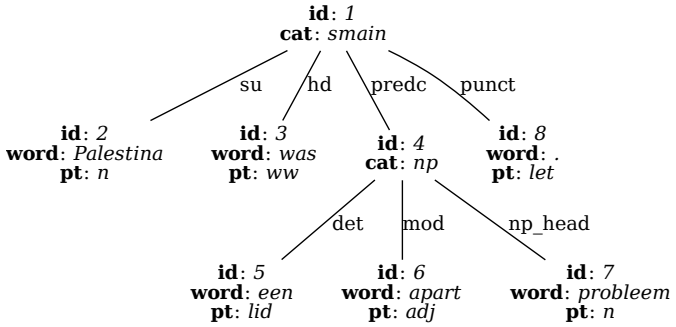


(b) After.

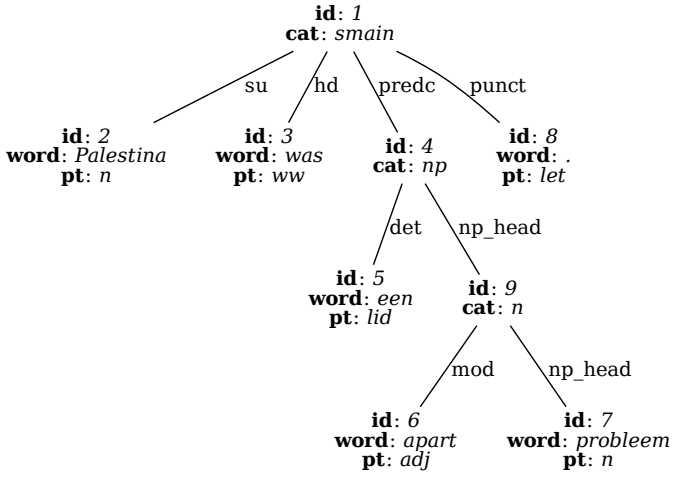
`dpc-gaz-001006-nl-sen.p.52.s.4(1)`

Hiermee kan kostbare tijd gewonnen worden.
'Precious time can be won with this.'

Figure III.7: Removing the understood subject from an infinitival verbal complement.



(a) Before.



(b) After.

WR-P-E-I-0000051928.p.1.s.140(1)

Palestina was een apart probleem.
'Palestine was a separate problem.'

Figure III.8: Inserting an intermediate layer for nominal modification.

to the phrasal domain they are part of, rather than recursive paired in a binary fashion to the node they modify or determine. We managed to cheat our way around multiple determiners so as to avoid any conflicts of priority, but the same lexical strategy does not apply to modifiers. Since Lassy abstains from taking a stance on what the order of modifier attachment is, and sees no distinction between modifying a phrase or its head, we are forced to by and large adopt the same strategy. Exceptionally in the nominal domain, we have the option of imposing structure based on word order alone. That is, we can distinguish between a noun and a noun phrase modifier depending on where the modifier is located: a modifier that occurs before the determiner must modify the entire phrase, whereas a modifier that occurs between the determiner and the head noun must modify the noun alone; see Figure III.8 for an example. This simple heuristic is as as far as we can get, but it will help homogenize our extracted proofs and types by (i) aligning function application order and word order and (ii) alleviating any unecessary typing tension between NP and N modifiers.

Ellided Constituents Other than non-finite verbal arguments, shared indexing is primarily employed by Lassy to indicate an omission of linguistic material in ellipses. The scheme Lassy employs presents the material version of a "shared" tree (be it a deep structure or a singleton node) in the first conjunct daughter of an elliptical conjunction, and a phantom copy of it in each subsequent sister. We alter this by pushing the material node to the top-level of the conjunction[1] (i.e. as a sibling to all conjuncts), and substituting the gap left behind by a new phantom node of the appropriate index. This will facilitate the easier typing of conjunctions later on.

Unary Pipes The movements and erasures performed can sometimes lead to "pipes" of unary trees. Awkward as these might be, they pose no issue to the extraction algorithm, provided they consist solely of *hd* labeled edges. We keep them as is so as to avoid the tedious and error-prone work of unecessarily reindexing nodes.

Labelless Conjunctions The *conj* label, used as an umbrella category to classify all conjunctions, carries the same risk as the *mwu* label, namely of polluting the functional type assignments of phrases outside the conjunction itself with a generic, multi-purpose argument type. We resolve this exactly like before, namely by conducting a majority voting over the categories of all conjunct siblings and propagating the elected category upwards to the conjunction node, prioritizing noun phrases over nouns over everything else in orderto account for nominalization.

[1] In the case of nested conjunctions, we stop at the first node assigned the *conj* syntactic category that is an ancestor of *all* phantom nodes of the same index and dependency.

Raising Nouns Bare nouns are assigned the *n* part of speech regardless of whether they need (or occur with) a determiner to occupy a verbal argument position. To circumvent (to the extent possible) a combinatorial explosion of meaningless N and NP argument variations, we alter the part of speech assignment of *n* nodes that are not roofed under a *np* from the former to the latter. As we will soon see, this will alter the type assignment of these nodes, in analogy to an implicit and contextual lexicalization of an explicit noun raising rule.

Assertions Prior to sending the transformed tree on its way, we make some basic checks of its structural integrity; we assert first of all it is indeed a tree (all nodes but the root have one incoming edge, nodes are all connected), that every phantom node has a material counterpart, and that no phantom nodes are labeled as multiword parts (as these would be humanly impossible to type).

11.2 Proving Lassy

With our transformations in place, the subdued corpus should be ripe for our proof extraction algorithm.

11.2.1 Proof Charming

The extraction is built around a tiny DSL written in Python, which allows one to formulate, represent and traverse valid proofs of $\mathbf{NLP}_{\Diamond,\Box}$. By invoking the DSL while traversing the dependency tree of a Lassy sample, the extraction algorithm dynamically constructs a natural deduction proof, translating tree patterns into meta-theoretical proof operations. Internalizing the syntactic validity assertions of $\mathbf{NLP}_{\Diamond,\Box}$ is a costly procedure, both in terms of processing overhead and of maintainance effort required, especially considering how unconducive the language is to formal rigor. On the other hand, it serves to eliminate the need for asynchronously interfacing with some external checker, and provides a formal guarantee that whatever the extraction algorithm produces is *correct by construction*: any syntactic missteps will be caught on the spot and raise an exception. As a bonus, the system can (and will) find use outside the scope of the extraction, as a representational intermediary for parsing and proof representation. Detailing the specifics behind the implementation shouldn't be our main concern here; it suffices to know it exists and runs as a constant safety belt in all that follows. If for whatever reason you enjoy watching people try to beat types into Python, you can take at a look at Appendix B (not for the faint of heart).

11.2.2 Parameters

We start by declaring our basic necessities. First, a translation table (or function) that maps part of speech tags and syntactic category labels to types, used to provide non-contextual type assignments to lexical nodes and phrases. The

translation table currently in use is depicted in Table III.1; it maps strictly to atomic types and takes most of the categorial labels at face value, mapping each of them to a unique image[1]. Exceptionally, the *spec* tag is cast into NP, as the tag is (inconsistently) used as a generic annotation for places, persons, events and the like. The codomain of the translation is in our case coincident with our logic's set of propositional constants, Prop_0. As hinted at earlier, the extraction is *parametric* to this translation: the domain can be any of the sets of lexical tags Lassy provides access to, and the codomain is by no means restricted to atomic types. This allows an easy adaptation to morpologically informed types, or a transition to an expanded theory (e.g. one that includes subtyping, additional axes of modal decorations, etc.). In principle, this can also allow a re-incorporation of the semantic subjects of non-finite verbal forms, but doing so would not amount to much: as promised earlier, these are already trivially recoverable by a simple morphism (readily applicable on the extracted proofs) that sends non-finite types (e.g. PPART) to the desired complex types (e.g. $\Diamond_{su}\mathrm{NP}\multimap\mathrm{PPART}$); immediate return on investment for taking source labels at face value. In any case, the translation induces a function that naively tells us for each node what type the translation table prescribes to its part of speech tag (if lexical), syntactic category (if phrasal) or the correspond translations of its material counterpart (if phantom).

Then, we need an equivalence relation on the set of dependencies so as to partition it into heads, adjuncts and complements – each non-head dependency is translated into a modal label, the union of which (together with the extraction modality) forming the set of modalities Deps. For the set of dependencies, we also need a strict partial order, serving to impose an informal ordering of the arguments of a multi-argument functor. With this, we avoid the responsibility of having to explicitly argue about an equivalence between argument order variations of the same head function: each complement will have a distinct modal decoration that decides how strongly it is attracted to the end result, yielding a canonical form for all types we'd be faced with (given the uniqueness of complements restriction). This is reminiscent of the notion of a *obliqueness hierarchy* [Dowty, 1982], which we can in fact use to produce some linguistically sensible types. From more to least oblique, we have: *svp>obcomp>vc>me>ld>hdf>pc>se>obj2>predc>obj1 >pobj1>su>sup*.

11.2.3 Tree Patterns

The algorithm takes the form of a proof assignment function, responsible for casting a local tree structure into a corresponding natural deduction proof. The function is recursively called in a top-down fashion, its original input being the complete dependency tree and its endpoints being terminal nodes. Proofs assigned to lexical nodes will correspond to lexical type assignments, whereas proofs assigned to phantom nodes will be variable instantiations. In-

[1] It basically just converts italics to smallcaps.

termediate returns of the top-level function call will be the "partial" proofs of the corresponding subtrees. To apply the appropriate operation at each slice of the tree, the algorithm distinguishes between a number of structures on the basis of the outgoing dependencies present. Context may be propagated from a local layer to the layer underneath by providing the dependency label of the edge that led to the current tree, and (optionally) a type hint. In mathy font, this would look something like:

$$\mathsf{prove} :: \mathsf{Tree} \to \mathsf{Type}^? \to \mathsf{Deps}^? \to \mathsf{Proof}$$

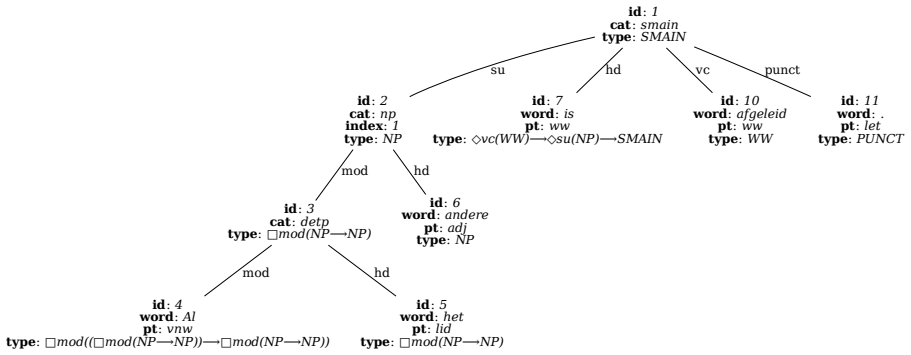
In the paragraphs to follow, we will inspect the tree structures most commonly encountered and discuss their treatment in high level terms. For ease of communication, we will trace the algorithm in reverse, going from simpler constructions to more complex ones. Despite building proofs, we will at times use notation or terminology commonly reserved for terms/programs when both convenient and applicable, e.g. a variable will denote a proof containing a single id rule, and applying a proof to a proof will mean deriving a more complex proof via modus ponens – do not be startled by this and remember your Curry-Howard.

Terminal Nodes Terminal nodes are easy-peasy. There's only three questions we have to ask, namely "do we have a type hint?", "is the node a phantom?" and "was there a dependency label that got us here and, if so, was it either an adjunct or a complement?". Ok, this is actually four questions but nonetheless. Any proof assignment we cook up must be uniquely identifiable with the node; we will use the node's identifier as the subscript of the instantiated variable or constant (we will from now on use indexing to tell constants apart, as strings do not make for trustworthy identifiers, i.e. c_i for constant i).

If we do have a type hint, we just return a new constant (if not a phantom) or variable (if phantom) of the hinted type. If we do not have a type hint, we must first map the syntactic category label of the node under scrutiny (or its material counterpart, if phantom) into a type using our translation table. If the node was lexical, we are done – we need to just instantiate the type with the lex rule. If it was a phantom, we must recall our discussion in Section 8.3.2[1] and check whether a modality needs to be added – a diamond (if we got here through a complement marking dependency), or a box (an adjunct marking one). If no label was present, or it was a heady one, we are to stick with the plain type. Either way, the type gets instantiated with the id rule.

Non-Terminal Trees In any non-terminal domain, we must first decide on the type of the current phrase. If a type hint was passed from above, we have no option other than to obey it. If no hint was passed, we will translate the phrasal category of the current root into the appropriate top type.

[1] Hypotheses come prepackaged with their modalities – you're welcome.



(a) Type-annotated sample.

$$\dfrac{\dfrac{\dfrac{}{\mathsf{c}_7 : \Diamond_{vc}\mathrm{WW} \multimap \Diamond_{su}\mathrm{NP} \multimap \mathrm{S}_{\mathrm{main}}}\ \text{lex} \quad \dfrac{\dfrac{}{\mathsf{c}_{10} : \mathrm{WW}}\ \text{lex}}{\langle \mathsf{c}_{10} \rangle^{vc} \vdash \Diamond_{vc}\mathrm{WW}}\ \Diamond_{vc}I}{\mathsf{c}_7, \langle \mathsf{c}_{10} \rangle^{vc} \vdash \Diamond_{su}\mathrm{NP} \multimap \mathrm{S}_{\mathrm{main}}}\ \multimap E \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{c}_4 : \Box_{mod}(\Box_{mod}(\mathrm{NP} \multimap \mathrm{NP}) \multimap \Box_{mod}(\mathrm{NP} \multimap \mathrm{NP}))}\ \text{lex}}{\langle \mathsf{c}_4 \rangle^{mod} \vdash \Box_{mod}(\mathrm{NP} \multimap \mathrm{NP}) \multimap \Box_{mod}(\mathrm{NP} \multimap \mathrm{NP})}\ \Box_{mod}E \quad \dfrac{}{\mathsf{c}_5 : \Box_{mod}(\mathrm{NP} \multimap \mathrm{NP})}\ \text{lex}}{\langle \mathsf{c}_4 \rangle^{mod}, \mathsf{c}_5 \vdash \Box_{mod}(\mathrm{NP} \multimap \mathrm{NP})}\ \multimap E \quad \dfrac{}{\mathsf{c}_6 : \mathrm{NP}}\ \text{lex}}{\langle \langle \mathsf{c}_4 \rangle^{mod}, \mathsf{c}_5 \rangle^{mod} \vdash \mathrm{NP} \multimap \mathrm{NP}}\ \Box_{mod}E \quad \dfrac{}{\langle \langle \mathsf{c}_4 \rangle^{mod}, \mathsf{c}_5 \rangle^{mod}, \mathsf{c}_6 \vdash \mathrm{NP}}\ \multimap E}{\langle \langle \langle \mathsf{c}_4 \rangle^{mod}, \mathsf{c}_5 \rangle^{mod}, \mathsf{c}_6 \rangle^{su} \vdash \Diamond_{su}\mathrm{NP}}\ \Diamond_{su}I}{\mathsf{c}_7, \langle \mathsf{c}_{10} \rangle^{vc}, \langle \langle \langle \mathsf{c}_4 \rangle^{mod}, \mathsf{c}_5 \rangle^{mod}, \mathsf{c}_6 \rangle^{su} \vdash \mathrm{S}_{\mathrm{main}}}\ \multimap E$$

(b) Assigned proof (unprocessed).

`WR-P-E-I-0000041235.p.1.s.123(1)`

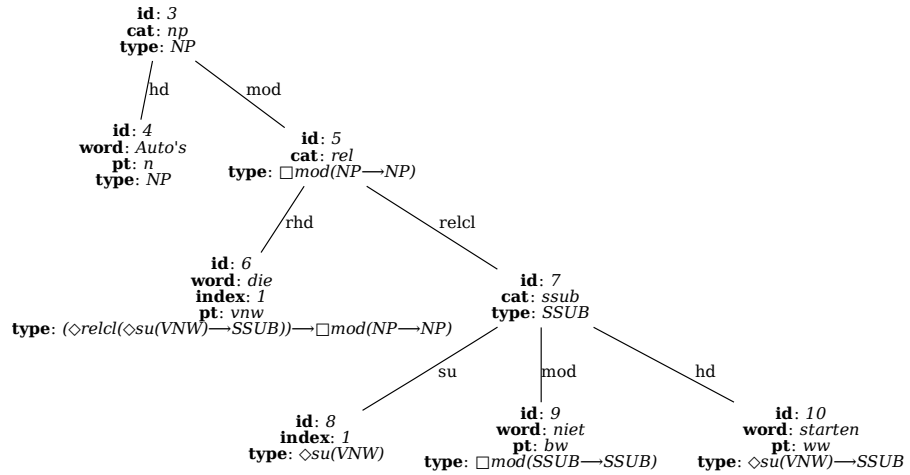
Al het andere is afgeleid.
'All the rest is derivative.'
$\mathsf{c}_7\ \triangle_{vc}\mathsf{c}_{10}\ \triangle_{su}(\blacktriangledown_{mod}(\blacktriangledown_{mod}\mathsf{c}_4\ \mathsf{c}_5)\ \mathsf{c}_6)$

Figure III.9: Proving a simple finite clause.

Matrix Clauses A simple verbal domain consists of a single head, some complements and (possibly) some adjuncts – we'll arrange them in corresponding bins, with complements sorted by their obliqueness hierarchy and adjuncts sorted by their order of appearance in the sentence. First, we'll call prove on each argument tree, passing its corresponding dependency edge and no type hint as arguments. Each proof that does not correspond to an instance of the id rule, we will apply a diamond introduction over, to enforce the appropriate complement dependency (hypotheses are excluded due to *already* being of the correct diamond type). By isolating the result types of the proofs extracted this way, we can infer the type of the phrasal head. We can thefore call prove on the head tree, passing no correspoding dependency edge, but type hinting it as the curried function from the sequentialized arguments to the top type. In a dual fashion, we can call prove on each adjunct tree, type hinting it as the endomorphism of the top type, enclosed under a box of the corresponding dependency label. With this and that, we now have proofs for each subtree underneath us – what remains to be done is fusing these proofs together. The way to go is simple: we need to first left fold the head's proof against the complements' proofs, and then apply the function composition of the "unboxed" adjuncts' proofs onto the result (unbox literally meaning using the box elimination rule to reveal the endomorphism enclosed within). Let's reiterate this for clarity. Each step of the initial fold will produce a "shorter" type, and, by the the time we run out of complements, the result's type will coincide with the top type. Each unboxed adjunct will be a function from the top type to itself – their n-ary function composition will then still be the of the same type, meaning it can be directly applied to our intermediate result. At this point, we have each tree below exactly once (yay, linearity), and we have produced a proof of the type we were asked to (or the tree prescribed), meaning we're good to go. Exceptionally to the above, nodes assigned the *punct* tag are given a plain PUNCT type that does not partake in the proof.

This simple setup already suffices to cover quite a lot of trees with simple applicative phenomena, including higher order modifiers like in the example of Figure III.9. There, nodes 1, 2 and 10 obtains types $\mathrm{S}_{\mathrm{main}}$, NP and WW by simple translation, and 7 obtains the type $\diamondsuit_{vc}\mathrm{WW} \multimap \diamondsuit_{su}\mathrm{NP} \multimap \mathrm{S}_{\mathrm{main}}$, having 10 and 2 as complements (adorned with $\diamondsuit_{vc}$ and $\diamondsuit_{su}$ diamonds respectively) and 1 as the result. Node 2 forces the type $\square_{mod}(\mathrm{NP} \multimap \mathrm{NP})$ to node 3 (as an adjunct with the *mod* label), in turn forcing the type $\square_{mod}(\square_{mod}(\mathrm{NP} \multimap \mathrm{NP}) \multimap \square_{mod}(\mathrm{NP} \multimap \mathrm{NP}))$ to node 4 for the exact same reason. Nodes 5 and 6 inherit the types of their mothers (3 and 2), having no complements. Within each domain, heads apply to their arguments and adjuncts drop their boxes to apply to the result.

Subordinate Clauses & Verbal Complements Now, if you have a sneaking suspicion that this looks oddly easy, you're right. We have not yet made any attempt to cover cases of hypothetical reasoning triggered by relative clauses, wh-questions and passive constructions. The hypotheses in such phenomena



$$\dfrac{\dfrac{\dfrac{\dfrac{}{c_6 : \Diamond_{relcl}(\Diamond_{su}\mathrm{VNW}\multimap \mathrm{S_{sub}})\multimap\Box_{mod}(\mathrm{NP}\multimap\mathrm{NP})}\ \text{lex} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{c_9 : \Box_{mod}(\mathrm{S_{sub}}\multimap\mathrm{S_{sub}})}\ \text{lex}}{\langle c_9\rangle^{mod}}\ \Box_{mod}E \quad \dfrac{}{c_{10} : \Diamond_{su}\mathrm{VNW}\multimap\mathrm{S_{sub}}}\ \text{lex} \quad \dfrac{}{x_8 : \Diamond_{su}\mathrm{VNW}}\ \text{id}}{c_{10}, x_8 \vdash \mathrm{S_{sub}}}\ \multimap E}{\langle c_9\rangle^{mod}, c_{10}, x_8 \vdash \mathrm{S_{sub}}}\ \multimap I}{\langle c_9\rangle^{mod}, c_{10} \vdash \Diamond_{su}\mathrm{VNW}\multimap\mathrm{S_{sub}}}\ \Diamond_{relcl}I}{\langle\langle c_9\rangle^{mod}, c_{10}\rangle^{relcl} \vdash \Diamond_{relcl}(\Diamond_{su}\mathrm{VNW}\multimap\mathrm{S_{sub}})}\ \multimap E}{c_6, \langle\langle c_9\rangle^{mod}, c_{10}\rangle^{relcl} \vdash \Box_{mod}(\mathrm{NP}\multimap\mathrm{NP})}\ \Box_{mod}E \qquad \dfrac{}{c_4 : \mathrm{NP}}\ \text{lex}}{\langle c_6, \langle\langle c_9\rangle^{mod}, c_{10}\rangle^{relcl}\rangle^{mod} \vdash (\mathrm{NP}\multimap\mathrm{NP})}\ \multimap E}{\langle c_6, \langle\langle c_9\rangle^{mod}, c_{10}\rangle^{relcl}\rangle^{mod}, c_4 \vdash \mathrm{NP}}\ \multimap E$$

WS-U-E-A-0000000016.p.37.s.1(3)

Auto's die niet starten.
'Cars that don't start.'
$\blacktriangledown_{mod}(c_6\ \triangle_{relcl}((\lambda x_8.\blacktriangledown_{mod} c_9\ (c_{10}\ x_8))))\ c_4$

Figure III.10: Abstracting over a relative clause gap.

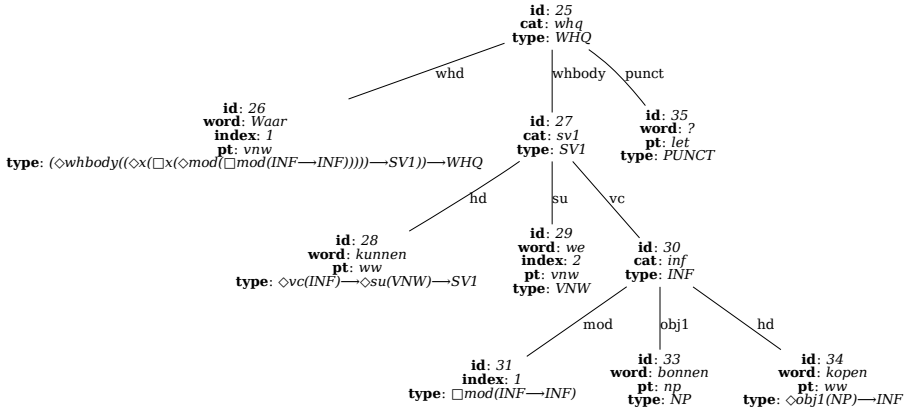
have already been established by the phantom nodes underlying them – all we need to do is know when to withdraw them. The criterion we'll follow applies to the proofs assigned to phrasal complements, and requires first that either the phrasal head (for subordinate clauses) or its subject (for passives) is indexed. If that's the case, the head is licensing hypothetical argument and must receive a higher-order type. We draw out all variables within the proof of the complement investigated, and filter the one that shares the same index as the aforementioned head or subject. Variables caught this way are abstracted over before the diamond introduction rule is applied. Figure III.10 presents a concrete example: head node 6 carries index 1 and looks for its complement in node 7. But node 7 contains x_8, and node 8 has index 1 – therefore, we must abstract the proof of node 7 over x_8 before continuing with assigning it a diamond.

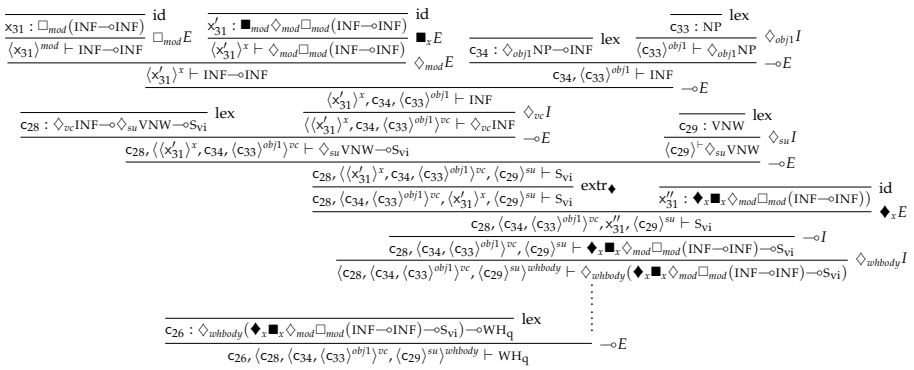
Horrors from the Deep But can we really be certain that the abstraction is indeed possible? Recalling once more our discussion from Section 8.3.2[1], there's a very real risk we might end up getting locked out of our hypotheses when these are adjuncts, deeply nested, or both. In the first case, we need to reproduce our strategy from Figure II.8. To do so, we need the tiniest of adaptations to our "unbox-and-apply" scheme from earlier on – when the adjunct that was unboxed happens to be a variable, we will immediately follow through with a diamond elimination. This way, when the time comes for us to be abstract over the reclusive adjunct (and the time will come, since our proofs must be linear), it will already have been liberated of its structural brackets. In the second case, we are in trouble. We need to employ the structural licensing pair $\blacklozenge_x \blacksquare_x$, as in Figure II.9, but there is no way for the proof assignment to have been correct preemptively: we would have needed to know that the variable is nested before ever getting to actually build its nesting context. To solve this chicken and egg problem, we need to allow ourselves an erroneous assignment, and then travel back in time to retroactively correct it. No big deal: going back in time means simply applying the substitution meta-pattern $x_i^A \mapsto \blacktriangledown_x(x_i^{\blacksquare_x A})$. In non-obscure, this translates to traversing the proof, finding the problematic hypothesis and replacing it with an x-marked (i.e. extractable) equivalent. Reconstructing the proof is not sufficient though – we also need to perform all the $\text{extr}_\blacklozenge$ rules necessary for the x-marked structure to always appear at the structural onion's outermost layer, as well as substitute it for its logical diamond counterpart. At that point, we are at long last able to abstract over the hypothesis.

The conventions described serve also to impose a canonical placement for the diamond elimination pattern. In combination with our carefully planned formulation of the $\text{extr}_\blacklozenge$ rule from Section 8.3.2[2], we have effectively relieved

[1] The variable may be free, but it could lie inaccessible behind structural brackets – don't mention it.

[2] As strictly localized and with shallow context – anytime.





WR-P-E-I-0000039352.p.3.s.7(25)

Waar kunnen we bonnen kopen?
'Where can we buy beans?'
$c_{26}\ \triangle_{whbody}(\lambda x''_{31}.\text{case}\ \triangledown_x x''_{31}\ \text{of}\ x'_{31}\ \text{in}\ (c_{28}\ \triangle_{vc}(\text{case}\ \triangledown_{mod}\blacktriangledown_x x'_{31}\ \text{of}\ x_{31}\ \text{in}\ (\blacktriangledown_{mod}x_{31}\ (c_{34}\ \triangle_{obj1}c_{33})))\ \triangle_{su}c_{29})))$

Figure III.11: Abstracting over a nested adjunct.

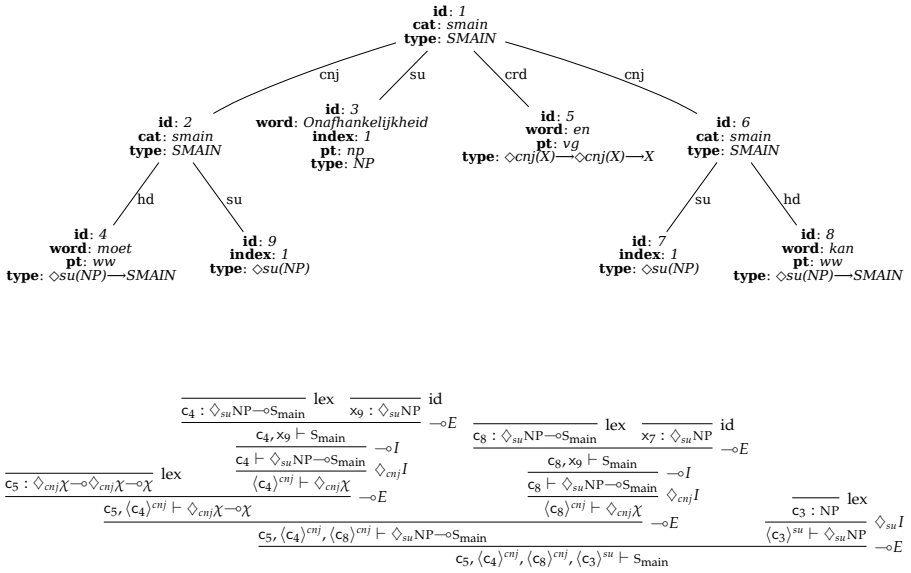
the burden of proof equivalence checking: diamond eliminations are to be performed as soon as possible and structural extractions as late as possible (but no later!) – i.e. we will avoid perpetuating temporary structures unless they have a purpose. To see this in action, let's have a look at the example of Figure III.11. There, node 31 is originally assigned $\mathsf{x}_{31} : \Box_{mod}(\text{INF}\multimap\text{INF})$. Upon unboxing it for application, we realize it's a hypothetical adjunct, therefore we follow through with a diamond elimination to $\mathsf{x}'_{31} : \Diamond_{mod}\Box_{mod}(\text{INF}\multimap\text{INF})$ to lose the *mod* brackets. Further down the line, we attempt to abstract over x'_{31}, only to find it trapped in a *vc* bracket. To facilitate the emancipation of the hypothesis, we perform the meta-syntactic substitution $\mathsf{x}'_{31} \mapsto \blacktriangledown_x \mathsf{x}'_{31}$, where the new x'_{31} is of type $\blacksquare_x\Diamond_{mod}\Box_{mod}(\text{INF}\multimap\text{INF})$. Empowered by its black box, the variable breaks free of its chains with the $\text{extr}_\blacklozenge$ rule, and gets diamond eliminated to $\mathsf{x}''_{31} : \blacklozenge_x\blacksquare_x\Diamond_{mod}\Box_{mod}(\text{INF}\multimap\text{INF})$. The latter is bracketless and exactly where we want it – we can finally perform the abstraction.

Nominal Domain The situation is not much different in the nominal domain, except for a change in the order we do things in. First, we must call prove on the head; unlike before, it receives no type hint, as we do not expect it to come out as a functor. Next, we requisit a proof for the determiner, hinted as a function from the head's type to the top type (with a complementary box on top, of course). Last, we ask for a proof for each adjunct, again as the boxed endomorphism of the result. Same as before, we take the function composition of all adjuncts; determiner first, since it's the one responsible for raising the noun to a noun phrase, followed by the garden variety modifiers in order of appearance. At this point, we must thank our past selves for the separation of noun and noun-phrase modifiers in two different tree layers, as this has saved us the trouble of having to scratch our head contemplating how to organize adjuncts.

Conjunctions We are almost there. The last bit remaining is unfortunately also the hardest one: conjunctions. Conjunctions are the infernal harbinger of torment and despair. Their ability to use linguistic material more than once challenges the linearity of our type system. To overcome their dark influence, we'll need to resort to arcane conjurations from the ancient texts (summarized for your convenience in Chapter I).

A few conjunctions are actually innocuous, i.e. when the tree inspected consists of a single coordinator and a sequence of conjuncts. Porting our intuitions on coordinator type assignments from Section 5.1.2[1], we obtain the recipe $\Diamond_{cnj}\chi\multimap\Diamond_{cnj}\chi\multimap\chi$, where χ is to range over types. In the vanilla case, we may simply call prove on each conjunct independently, hinting each as the top type (which, if not hinted, will be the translation of the syntactic category labels' majority consensus). Then, we may move on to the coordinator, which

[1]Coordinators are polymorphic types binding pairs of the same type into a conjoined pair – no problem.



wiki-6984.p.6.s.2(1)

Onafhankelijkheid moet en kan.
'Independence must and can (happen).'
$c_5\ \triangle_{cnj}(\lambda x_9.c_4\ x_9)\ \triangle_{cnj}(\lambda x_7.c_8\ x_7)\ \triangle_{su} c_3$

Figure III.12: Proving a subject ellipsis.

is hinted as a function from the ($\Diamond_{cnj}$-marked) conjuncts' types to the top type. Having used the same type hint amounts to a coercion that ensures that the parts and the whole are of the same type – in other words, we are faithful to our polymorphic recipe. A minor divergence is that our coordinator type is slightly more general than originally prescribed, being a variadic function rather than a binary one, since Lassy conjunctions are flat trees rather than hierarchical ones (punctuations carrying no annotations by default).

Unlike their well-behaved kindred, elliptical conjunctions require special treatment. Lassy rightly prefers pushing conjunctions to the topmost phrasal level. The subject ellipsis of Figure III.12, for instance, is not annotated as the conjunction of two heads applied to a single noun phrase, but rather as the conjunction of two sentences sharing the same subject (the material subject has been elevated to the same level as the conjuncts thanks to our earlier transformation). We'll gladly follow along. Same as before, we will first seek a proof for each conjunct, but now also any non-conjunct siblings. Conjuncts containing any variables whose nodes are coindexed with the non-conjunct siblings (the latter necessarily being material counterparts of conjunct-internal phantoms) will be abstracted over these variables, before being marked by the $\Diamond_{cnj} I$

rule. In the running example, conjunct 2 contains the phantom subject 9 and conjunct 6 the phantom subject 7, both coindexed with material node 3. The proofs of 2 and 6 will therefore be abstracted over their subject variables, and the instantiation of the type variable χ will in this case be $\Diamond_{su}\mathrm{NP}\multimap\mathrm{S}_{\mathrm{main}}$ (intentionally left implicit in the Figure for space economy). The proof we get back is coincidentally an η-expanded version of the heads' conjunction.

What if the elided item is not a complement, though? Not much changes, really: we must still look for hypotheses "shared" between adjuncts and materially present in the current conjunction branch. The difference is that withdrawing them this time around yields proofs of higher-order types. To see this in practice, let's consider the example of Figure III.13[1]. Conjuncts 14 and 24 contain two phantom nodes each: 21 and 33, and 30 and 25, respectively. The first of each pair (21 and 30) are the phantom objects attributed to the past participles. Their indices, 1 and 3, are properly contained within each conjunct – therefore they are of no interest to us here. The second ones (31 and 25) stand in for the missing auxilliary verb heading each conjunct, implementing higher-order functions of type: $\Diamond_{vc}(\Diamond_{obj1}\mathrm{NP}\multimap\mathrm{PPART})\multimap\Diamond_{su}\mathrm{NP}\multimap\mathrm{S}_{\mathrm{main}}$. These two share the same index 2, which materializes in node 19 – *these* are the variables we must abstract over. The abstraction will result in type variable χ being instantiated as the third order type:

$$(\Diamond_{vc}(\Diamond_{obj1}\mathrm{NP}\multimap\mathrm{PPART})\multimap\Diamond_{su}\mathrm{NP}\multimap\mathrm{S}_{\mathrm{main}})\multimap\mathrm{S}_{\mathrm{main}}$$

in turn producing a fourth order type hint for the coordinator which doesn't even fit in the line... oof.

Complicated as it might be, this type should not be alien to you, diligent reader. Let's flip things around and focus on what we have instead of what we miss. The dual view of having two sentences missing their head is that we have two pairs of floating proofs: pair one grounded in nodes 15 and 22, and pair two in nodes 26 and 31. If we were to allow products in our type calculus, we could encode each pair as an item of type $\Diamond_{su}\mathrm{NP}\otimes\Diamond_{vc}(\Diamond_{obj1}\mathrm{NP}\multimap\mathrm{PPART}))$, and use this to instantiate the polymorphic scheme. By type raising, we could alternatively use them to derive corresponding proofs of type:

$$((\Diamond_{su}\mathrm{NP}\otimes\Diamond_{vc}(\Diamond_{obj1}\mathrm{NP}\multimap\mathrm{PPART}))\multimap\mathrm{A})\multimap\mathrm{A}$$

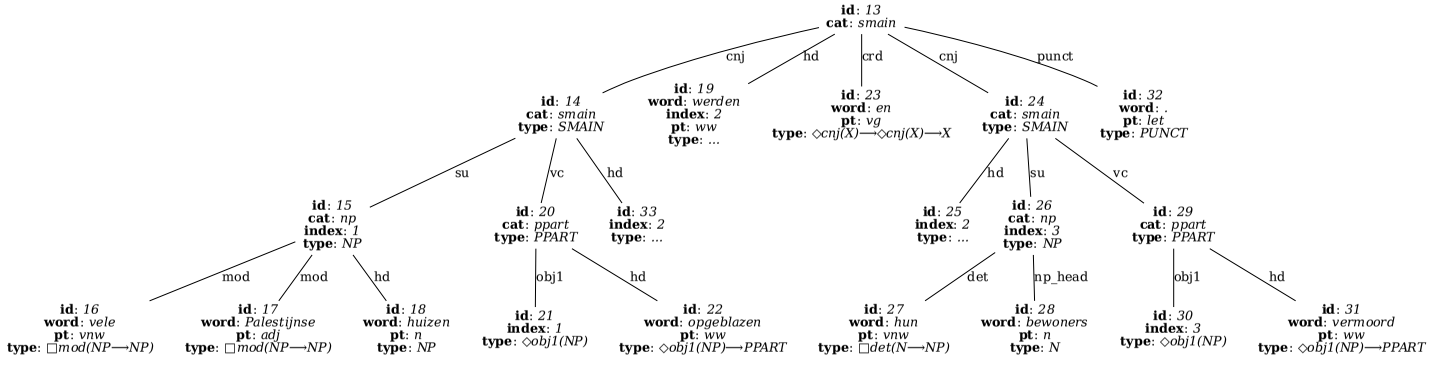
for any A of our liking. The product can then be curried into:

$$(\Diamond_{su}\mathrm{NP}\multimap\Diamond_{vc}(\Diamond_{obj1}\mathrm{NP}\multimap\mathrm{PPART}))\multimap\mathrm{A})\multimap\mathrm{A}$$

Ring a bell yet? Using $\mathrm{S}_{\mathrm{main}}$ in place of A, we end up exactly where we started.[2] Now, toying around with raised types is by no means ideal, as it reinforces the

[1] Give it up for our first ever landscape proof!

[2] If this made little sense, you should actually read the introduction instead of relying on footnote clues. Also I take "diligent" back.



$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{c}_{23} : \Diamond_{cnj}\chi \multimap \Diamond_{cnj}\chi \multimap \chi}\ \text{lex} \quad \dfrac{\dfrac{\dfrac{\mathsf{x}_{33}, \langle \mathsf{c}_{22}\rangle^{vc}, \langle\langle \mathsf{c}_{16}\rangle^{mod}, \langle \mathsf{c}_{17}\rangle^{mod}, \mathsf{c}_{18}\rangle^{su} \vdash \mathsf{S}_{\mathsf{main}}}{\langle \mathsf{c}_{22}\rangle^{vc}, \langle\langle \mathsf{c}_{16}\rangle^{mod}, \langle \mathsf{c}_{17}\rangle^{mod}, \mathsf{c}_{18}\rangle^{su} \vdash \chi}\ \multimap I}{\langle\langle \mathsf{c}_{22}\rangle^{vc}, \langle\langle \mathsf{c}_{16}\rangle^{mod}, \langle \mathsf{c}_{17}\rangle^{mod}, \mathsf{c}_{18}\rangle^{su}\rangle^{cnj} \vdash \Diamond_{cnj}\chi}\ \Diamond_{cnj} I}{\mathsf{c}_{23}, \langle\langle \mathsf{c}_{22}\rangle^{vc}, \langle\langle \mathsf{c}_{16}\rangle^{mod}, \langle \mathsf{c}_{17}\rangle^{mod}, \mathsf{c}_{18}\rangle^{su}\rangle^{cnj} \vdash \Diamond_{cnj}\chi \multimap \chi}\ \multimap E \quad \dfrac{\dfrac{\dfrac{\mathsf{x}_{25}, \langle \mathsf{c}_{31}\rangle^{vc}, \langle\langle \mathsf{c}_{27}\rangle^{det}, \mathsf{c}_{28}\rangle^{su} \vdash \mathsf{S}_{\mathsf{main}}}{\langle \mathsf{c}_{31}\rangle^{vc}, \langle\langle \mathsf{c}_{27}\rangle^{det}, \mathsf{c}_{28}\rangle^{su} \vdash \chi}\ \multimap I}{\langle\langle \mathsf{c}_{31}\rangle^{vc}, \langle\langle \mathsf{c}_{27}\rangle^{det}, \mathsf{c}_{28}\rangle^{su}\rangle^{cnj} \vdash \Diamond_{cnj}\chi}\ \Diamond_{cnj} I}{\mathsf{c}_{23}, \langle\langle \mathsf{c}_{22}\rangle^{vc}, \langle\langle \mathsf{c}_{16}\rangle^{mod}, \langle \mathsf{c}_{17}\rangle^{mod}, \mathsf{c}_{18}\rangle^{su}\rangle^{cnj}, \langle\langle \mathsf{c}_{31}\rangle^{vc}, \langle\langle \mathsf{c}_{27}\rangle^{det}, \mathsf{c}_{28}\rangle^{su}\rangle^{cnj} \vdash \chi}\ \multimap E \quad \dfrac{}{\mathsf{c}_{19} : \Diamond_{vc}(\Diamond_{obj1}\mathrm{NP} \multimap \mathrm{PPART}) \multimap \Diamond_{su}\mathrm{NP} \multimap \mathsf{S}_{\mathsf{main}}}\ \text{lex}}{\mathsf{c}_{23}, \langle\langle \mathsf{c}_{22}\rangle^{vc}, \langle\langle \mathsf{c}_{16}\rangle^{mod}, \langle \mathsf{c}_{17}\rangle^{mod}, \mathsf{c}_{18}\rangle^{su}\rangle^{cnj}, \langle\langle \mathsf{c}_{31}\rangle^{vc}, \langle\langle \mathsf{c}_{27}\rangle^{det}, \mathsf{c}_{28}\rangle^{su}\rangle^{cnj}, \mathsf{c}_{19} \vdash \mathsf{S}_{\mathsf{main}}}\ \multimap E$$

WR-P-E-I-0000015007.p.1.s.146(13)

Vele Palestijnse huizen werden opgeblazen en hun bewoners vermoord.
'Many Palestinian houses were destroyed and their residents murdered.'
$\mathsf{c}_{23}\ \triangle_{cnj}(\lambda \mathsf{x}_{33}.\mathsf{x}_{33}\ \triangle_{vc}(\lambda \mathsf{x}_{21}.\mathsf{c}_{22}\ \mathsf{x}_{21})\ \triangle_{su}(\blacktriangledown_{mod}\mathsf{c}_{16}\ (\blacktriangledown_{mod}\mathsf{c}_{17}\ \mathsf{c}_{18}))\ \triangle_{cnj}(\lambda \mathsf{x}_{25}.\mathsf{x}_{25}\ \triangle_{vc}(\lambda \mathsf{x}_{30}.\mathsf{c}_{31}\ \mathsf{x}_{30})\ \triangle_{su}(\blacktriangledown_{det}\mathsf{c}_{27}\ \mathsf{c}_{28}))\ \mathsf{c}_{19}$

Figure III.13: Proving a verb phrase ellipsis with a higher-order head.

propensity of coordinators to become very long and complex. On the other hand, it absolves us from having to incorporate an explicit product and protects us from the headaches it comes with (like proof equivalence under product elimination, type equivalence under different product branchings, more latex symbols to render in console pretty printing, etc., just to name a few). On the less cynical side, it also has the merit of permitting more flexible semantic interpretations for the user-to-be. Conjoining a series of arguments and feeding the result to the predicate means that the semantic interpretation of the coordinator would be inescapably bound to a local scope. Conjoining a series of λ abstractions and applying the result to the shared predicate grants the coordinator access to the full context of the conjunction, opening the door to a future semantic interpretation that can duplicate and distribute meaning if and as desired.

More generally, the proof of each conjunct will contain a mixture of constants and variables, the latter being any combination of heads, complements and adjuncts. All "shared" variables will be abstracted over. To maintain the homogeneousness of functors regardless of the context they appear in, the order of abstraction is reverse to the obliqueness hierarchy we have used so far. Head- and adjunct- functors, falling outside the hierarchy, will have priority over complements, appearing as the first argument of the higher-order types they help form. Any secondary coordinator appearing within the same conjunction (its dependency now labeled *cor*, due to our earlier transformation), will be the first argument to be consumed by the main coordinator (the two are thus modeled as a derived and discontinuous coordinator phrase).

11.2.4 Post-Processing

The proofs we get back don't need to be type-checked: the extraction has already taken care of that for us. From the type-correctness perspective, all we need to do is assert their linearity in terms of both variables and constants, i.e. make sure that all words of the input sample were used (exactly once each), and ditto for variables, except they must also be bound (in the off chance a phantom node was never abstracted over). For the sake of uniformity, all proofs obtained are β and η reduced. From the representation perspective, our proofs are still overly attached to their Lassy roots. Constants and variables are named according to their node origins, which will no longer make any sense after the tree has been discarded. We therefore rename constants according to their order of appearance in the sentence, and variables by enumeration, following a left-first depth-first traversal of the proof tree (preferrable to de Bruijn indexing for human legibility). To allow the incorporation of extra-theoretical information, proofs are repackaged into a Sample record containing fields proof, lexical_phrases, name and subset. The first field is self-explanatory. The second field contains a variadic tuple of lexical phrases, each element being itself a variadic container of lexical items, given a single type assignment as a whole. Each lexical item contains node attributes from the original Lassy

annotation, like word, pos, pt and lemma. This organization is in line with the more liberal type lexicon demanded by multiword units, and allows us to preserve lexical information that would be lost if we were to simply just squeeze them into a single "word". Multiwords aside, it faithfully presents a sentence together with its punctuation marks, despite them not (usually) appearing in the compositional analysis. Finally, it disassociates proofs from the concrete lexical constants justifying them, allowing us to easily compare, filter and aggregate proofs detached from the sentences they were assigned to. The third field simply associates the sample to its Lassy identifier, whereas the fourth field suggests a standard train/dev/test split for machine learning applications. The collection of extracted proofs we will call Æthel, for Automatically Extracted Theorems from Lassy, thus resolving the mystery of this section's cryptic name for those that made it this far.

11.3 Analysis

11.3.1 Quantitative Snoozefest

We are done with proving; time to get to counting. But first, a disclaimer. The numbers reported below have been relatively stable, but may well get to differ in between the time I write these words and the time you get to read them. They are correct in my subjective frame of reference (version), namely `1.0.0a4`.

From Lassy to Æthel From the 65 200 trees of Lassy Small, we discard 2 607 for being a single word or punctuation mark. The remaining 62 593 are pruned into 69 583 discourse-free cuttings (1.11 cuttings per source tree, on average). From these, the extraction algorithm produces 68 763 theorems, bringing its (processed) corpus coverage to gratifying 98.82%. All failures are due to problematic tree structures; mostly conjunctions without a coordinator, or, rarely, linearity breaches (free variables that cannot be justified by an existing proof pattern). A random 80/10/10 split is applied to the source Lassy samples, which translates into 56 875 (82.7%) train, 6 118 (8.9%) dev and 5 770 (8.4%) test Æthel samples (81.7/8.8/8.3). Performing the split on Lassy rather than directly on Æthel ensures consistency between different revisions, and asserts that any sample overlap from subtree duplications during pruning will be contained within the same subset, keeping cross-contamination to a minimum.

To quantititatively measure the impact of the preprocessing transformations (i.e. see how close the derived dataset is to the original), we first examine how many of Lassy's samples are proven unaltered. In line with our observations on discourse level annotations from earlier, we find that 48 057 (73.78%) of the filtered Lassy samples can be uniquely mapped to an Æthel proof. Counting the number of samples enumerating a certain number of words, we arrive at the graph of Figure III.14. Surprising noone, the graph reveals that both the original and the derived corpus exhibit a right-skewed

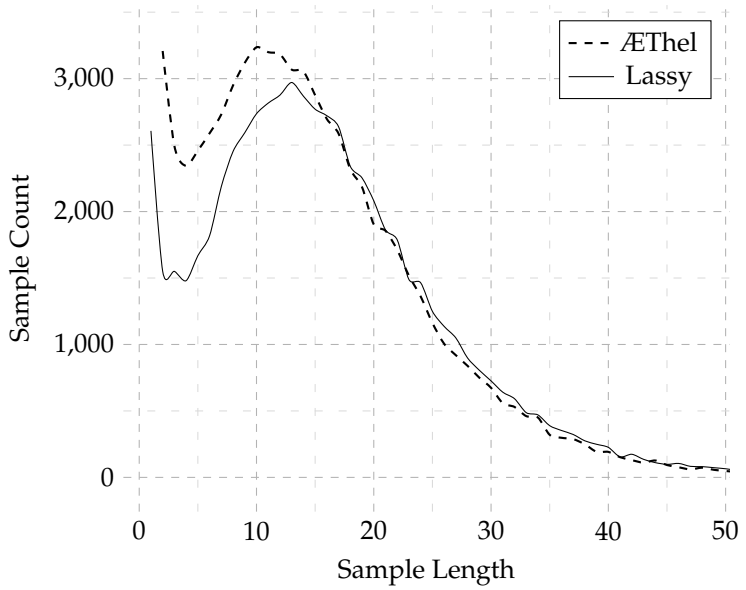


Figure III.14: Sample counts by length.

distribution of sample lengths, the latter being a left-shifted version of the former. Concretely, Lassy has a mode of 13 and a median of 17.4 (ignoring the one-word sentences, for fairness), whereas Æthel has a mode of 10 and a median of 15. The difference between them is not that striking and really only affects their left tails and centers; in fact, the two distributions go almost hand in hand towards their right tails. Practically, we lose a small proportion of originally long and medium-sized samples, and the broken parts accumulate into a bulk of short and really short samples (hence the outlier peak at 2). Not bad, considering.

Theorems To quantify the proof-theoretic diversity of the 68 763 extracted samples, we abstract away from the sentences and end up with a total of 55 026 unique theorems. Figure III.15 groups and displays them according to how common they are, i.e. how many times we see the same theorem associated to a different sentence. Evidently, the vast majority of unique theorems (52 637 or 95.7%) occur but a single time within the dataset (i.e. they are assigned to a single sample), with logarithmically$_{10}$ fewer theorems being assigned to exponentially$_2$ more sentences, making Æthel a rich and intricate resource.

Proof Ambiguity We've already made peace with the fact that our type logic is syntactically underspecified and prone to overgeneration. But a side-effect of this concession is our inability to tell well- and ill- founded ambiguities

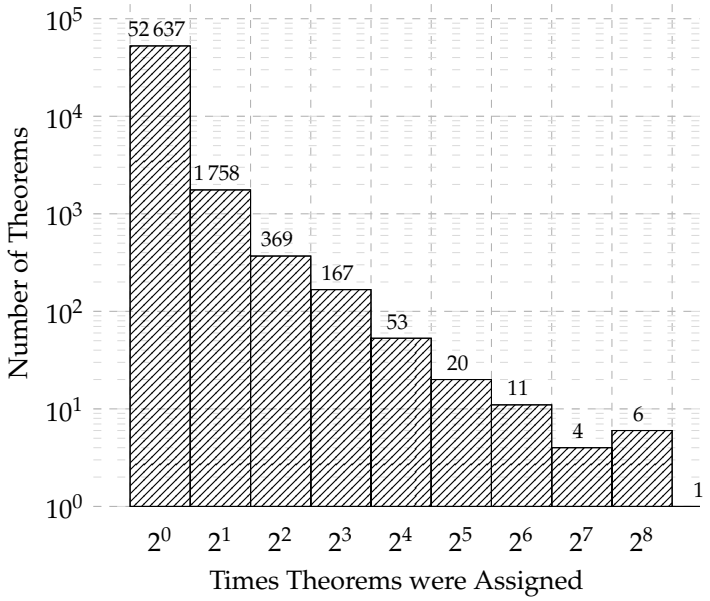


Figure III.15: Proof sparsity in Æthel.

apart, i.e. we have no means of knowing whether a second proof derivable from the exact same proof frame (sequence of type assignments) is a clear-cut error or indeed a linguistically plausible alternative reading. Lassy itself only provides the derivation of one single reading (and it is not our place to ask which reading that is). We can use this to our advantage to try and estimate the *real* derivational ambiguity of Æthel, by first isolating the proof frame of each sample (taking the type sequence independent of the sentence it was assigned to), and then counting the amount of unique proofs derived from each frame.[1] In total, there's 56 099 unique proof frames, but 53 794 (95.9%) of those occur only once, thus being of little use to our purposes (since each will unavoidably be mapped to a single proof). Of the remaining 2 305, 2 259 (97.4%) are mapped to one unique proof, whereas 61 (2.6%) may derive one of two unique proofs. Using this as a guideline, we find 472 potentially ambiguous samples in the entirety of Æthel (a measly 0.7%), their frames being among the 61 suspects. Manual inspection reveals most of them to really be ambiguous (e.g. the proof assigned prescribes one of two possible modifier attachments), although in some cases the alternative reading is linguistically ruled out on the basis of selectional restrictions. Repeating the above after first casting the proof and type assignments to **NLP** (dropping any dependency information

[1]Note that we are gathering *just* the sequence of types assigned to a sample and *not* the proof's antecedent, since the modal bracketing structure already disambiguates the proof assignment.

via a stripping morphism), we end up with 55 118 unique frames for 54 172 unique proofs, with 2 492 frames occurring more than once. Of these, 2 405 are behind 1 unique proof, 111 behind 2, 6 behind 3, and 1 is behind 5. Potentially ambiguous samples are now doubled (904, or 1.3% of the total), and ambiguities are more commonly artificial; would-be alternative readings for the newly ambiguous samples are usually ruled out by morphosyntactic constraints invisible to the theory.

What are these boring numbers trying to tell us? Well, two things. First, that using an undirectional logical core doesn't really estrange the grammar from the lexicon. Proof frames are sparse, and the bulk of them are pointing to a single proof; we are still as lexicalized as it gets. That's not to say that choosing the correct type assignments will by any means suffice for parsing, but rather that it will certainly narrow down the options of a (logically and linguistically) correct proof to almost exactly one. Second, it tells us that dependency decorations seem to serve an auxiliary role, additional to the one intended: they increase the count of unique frames and decrease the number of proofs grown per frame, trading derivational ambiguity for lexical type ambiguity. This is not really surprising: hypotheses launched by higher-order functors are restricted to a predetermined grammatical role, which, by the exclusion principle (no two complements of the same role), forces overt complements to assume the leftover role, implicitly resolving ambiguity.

Lexical Type Ambiguity This brings us to the lexicon: the binary relation connecting words (or strings, really) to types, and obtained by the aggregation of all our proofs' type assignments. Æthel as a whole contains 1 033 858 words chunked into 992 385 phrases, each chunk carrying a single assignment. These amount to 74 812 (67 883 after case normalization) unique words and 76 746 (71 077) unique phrases, the latter being essentially the lexicon's domain. On the codomain's side, we have a grand total of 6 008 unique $\mathbf{NLP}_{\Diamond,\Box}$ types. Stripped of their modalities, these fall back to 3 641 pure linear types, evidencing the import of the added dependency axis. The first question to ask then is: how functional is this lexicon? Answering that question is Figure III.16, which in short says "not much, really". Despite most lexical entries (56 176) having a single unique image, ambiguities are quite common, the number of assignments exponentially$_2$ increasing for a logarthmically$_{10}$ declining number of entries. At the end of the tail we find several chameleon words appearing in multiple guises; chief among them is the usual suspect *en* 'and', counting as many as 1 046 (!) unique types in just 41 614 occurrences – the price of braving conjunctions with concretely instanced, pre-raised types. But this is painting an overly grim picture; seen as random variables, lexical entries actually have most of their probability masses concentrated on their modes. The average probability of the most common type per entry lies at a (very high) 88.97%, while naively assigning the mode to each lexical assignment corpus-wide (disregarding context) establishes a comfortable 66.97% baseline, asserting that

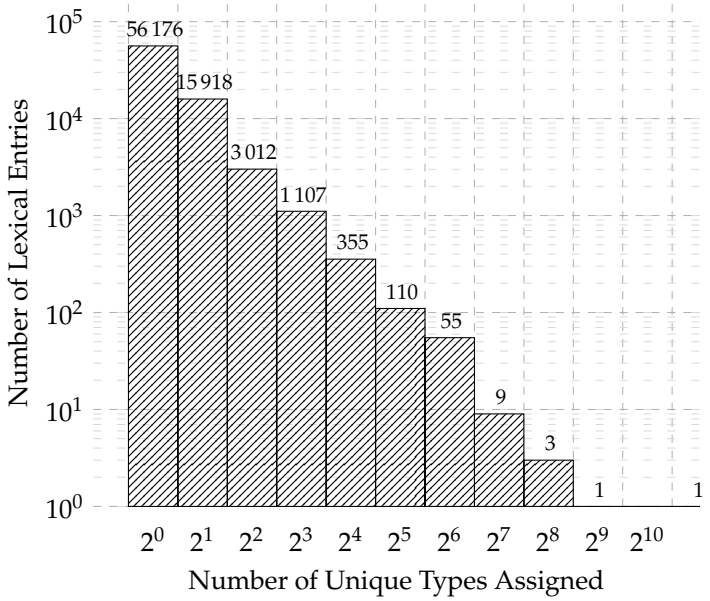


Figure III.16: Lexical Type Ambiguity

our assignments are in practice quite consistent within their entries.

Lexical Type Sparsity So ambiguity is not a major statistical threat, despite the high average number of options per entry; some opposing force must be counterweighting its effect. In reality, the only reason we are able to naively guess assignments with some reasonable accuracy is exactly because the less-than-most-frequent options are in fact very infrequent – the demon's name is sparsity. To diagose the problem and quantify its extent, we perform a number of tests. First, we measure the proportion of the 6 008 types that occur more than n times, and plot the result in Figure III.17 (if you prefer jargon, this is the inverse empirical cumulative distribution of type occurrences). The dash-dotted line leaves little room for interpretation: 2 868 (47.7%) of the total types have only a single occurrence, 4 807 (80%) have less than 10 occurrences, and only 316 (5%) are common enough to boast more than 100 occurrences – our types are quite sparse alright. This goes to show that uncommon type assignments are uncommon globally, and not just in the context of the lexical entry that contains them; in other words, a type rarely associated with *some* lexical entry is likely rare to find among *any* entry.

To analyze the real impact of sparse types, we have to holistically inspect their corpus-wide distribution. To that end, we repeat the above measurement, this time focusing on type assignments and samples rather than just types. Intuitively, we are interested in the proportion of the 992 385 type as-

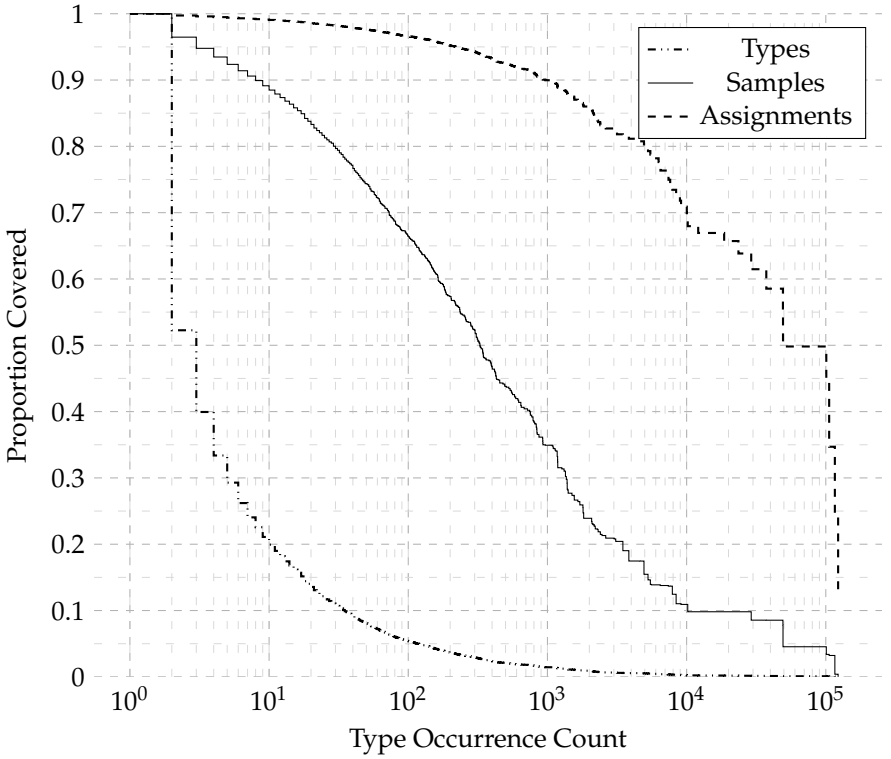


Figure III.17: Proportion of types, type assignments and samples covered as a function of a minimum type occurrence threshold.

signments we could resolve when considering only types occurring more than n times, and the proportion of the 68 763 samples containing resolved assignments only. Things look somewhat less scary here: on the type assignment front, the 5% most common types cover 97% of the total assignments, and the top 1% (occurring more than 1 000 times) is still good enough for 90% of the assignments. Sentential coverage declines more rapidly, indicating that rare types are, to our discontent, evenly distributed within our samples – discarding types with less than 10 occurrences, for instance, already brings sentential coverage down to 88%. This will haunt us later, but for now let's keep lingering in the bliss of ignorance.

Lexical Entry Sparsity On the other side of the lexicon lie lexical entries – the words and phrases we may use to index the lexicon. Lexical entry sparsity is completely external to the extraction algorithm (words and their occurrence statistics being directly inherited from Lassy and its choice of corpora). It's also no longer a real practical consideration, since distributed word vec-

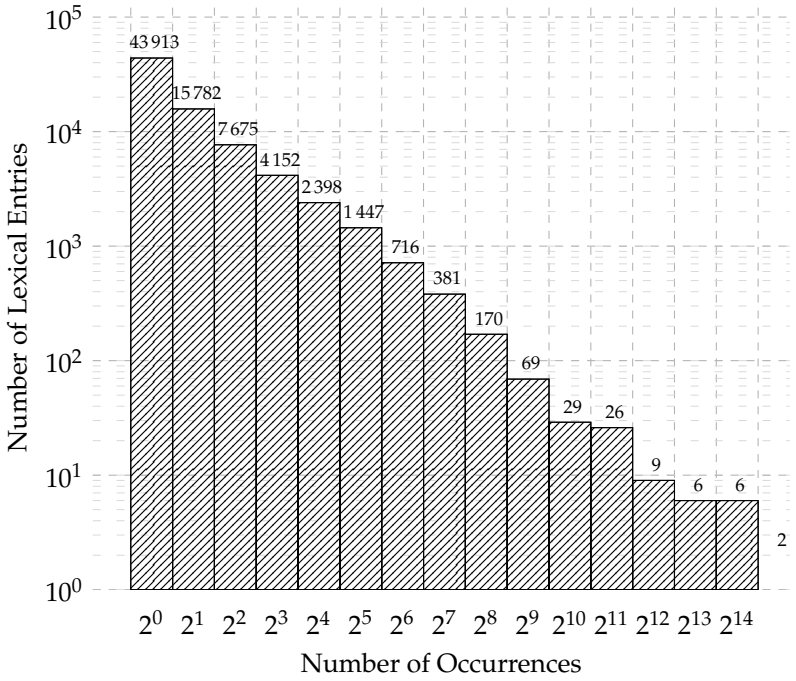


Figure III.18: Lexical Entry Sparsity

tors and pretrained language models have long superseded the lexically fixed components of the modern NLP pipeline. Still, for the sake of completeness (and since the lexicon is a tiny resource of its own), we may as well have a quick glance of the statistics of the lexicon's domain. Figure III.18 shows the distribution of lexical entry occurrences; most entries occur just once, whereas a handful occur up to a few thousand times (interpolate between most and a handful in the $\log_{10}/\log_2$ plane to find any point of interest). The most common word of Æthel is the gendered definite determiner *de*, claiming the throne with an incontestible 56 925 occurrences (making up for 5.7% of all words just by itself).

11.3.2 Quality Control

We made it through the boring part! With all the numbers laid down, all that's left to do is expose how Æthel treats common and idiosyncratic constructions of Dutch.

todo

12 Key References & Further Reading

The chapter has presented Æthel and detailed the process toward its construction. The baby version of Æthel made an early public appearance in my thesis [Kogkalidis, 2019] in what feels like ages ago (but actually was three-something years). Back then, it didn't yet have a name, and was just only a partially worked out type lexicon, but its statistical kinks were already becoming obvious. The full resource gradually took form as a proof bank proper, and was first described in Kogkalidis et al. [2020]. The chapter is a significantly more mature version of the paper, sharing the same general design philosophy but boasting lighter preprocessing (leading to longer and higher quality proofs) and wider coverage (improving linguistic variation). More importantly, the extraction algorithm has been revised and reimlemented to ensure type safety, guaranteeing the formal correctness of the extracted proofs and fixing several issues and inconsistencies pertaining to the modal decorations of higher order types.

My little bubble aside, corpus extraction is a fan favorite for the computationally inclined linguists (or the linguistically inclined computists). The endeavour was popularized by the CCGbank [Hockenmaier and Steedman, 2007], a semi-automatically extracted corpus of combinatory categorial grammar derivations. Following a similar endeavour in German [Hockenmaier, 2006], the corpus has become a flagship for the applied categorial grammarian, spawning many offsprings across different languages [Bos et al., 2009; Tse and Curran, 2010; Ambati et al., 2018, *inter alia*]. More akin to our proof-theoretic regime and forebearers to this work are the French TLGbank [Moot, 2010b], and the type-logical conversion of the spoken Dutch corpus [Moot, 2010a], both following the multimodal Lambek tradition.

Chapter Bibliography

B. R. Ambati, T. Deoskar, and M. Steedman. Hindi ccgbank: A ccg treebank from the Hindi dependency treebank. *Language Resources and Evaluation*, 52 (1):67–100, 2018.

L. Augustinus. *Complement raising and cluster formation in Dutch*. PhD thesis, KU Leuven, 2015.

S. Barbiers. *Word order variation in three-verb clusters and the division of labour between generative linguistics and sociolinguistics*, volume 265, pages 233–264. John Benjamins Publishing, Nederland, 2005. ISBN 902724779X.

J. Bos, C. Bosco, and A. Mazzei. Converting a dependency treebank to a categorial grammar treebank for Italian. In *Eight international workshop on treebanks and linguistic theories (TLT8)*, pages 27–38. Educatt, 2009.

G. E. P. Box. Science and statistics. *Journal of the American Statistical Association*, 71(356):791–799, 1976.

D. Dowty. Grammatical relations and montague grammar. In *The nature of syntactic representation*, pages 79–130. Springer, 1982.

D. Heylen. Underspecification in type-logical grammars. In *Selected Papers from the Second International Conference on Logical Aspects of Computational Linguistics*, LACL '97, page 180–199, Berlin, Heidelberg, 1997. Springer-Verlag. ISBN 3540657517.

J. Hockenmaier. Creating a ccgbank and a wide-coverage ccg lexicon for German. In *Proceedings of the 21st international conference on computational linguistics and 44th annual meeting of the association for computational linguistics*, pages 505–512, 2006.

J. Hockenmaier and M. Steedman. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396, 09 2007. ISSN 0891-2017. doi: 10.1162/

coli.2007.33.3.355. URL https://doi.org/10.1162/coli.2007.33.3.355.

R. Huybregts. The weak inadequacy of context-free phrase structure grammars. *Van periferie naar kern*, pages 81–99, 1984.

K. Kogkalidis. Extracting and learning a dependency-enhanced type lexicon for dutch, 2019. URL https://arxiv.org/abs/1909.02955.

K. Kogkalidis, M. Moortgat, and R. Moot. ÆTHEL: Automatically extracted typelogical derivations for Dutch. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 5257–5266, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.647.

L. Macken, O. De Clercq, and H. Paulussen. Dutch parallel corpus: A balanced copyright-cleared parallel corpus. *Meta: Journal des traducteurs/Meta: Translators' Journal*, 56(2):374–390, 2011.

R. Moot. Extraction of type-logical supertags from the spoken Dutch corpus, 2010a.

R. Moot. Semi-automated Extraction of a Wide-Coverage Type-Logical Grammar for French. In *TALN 2010*, Proceedings TALN 2010, Montréal, Canada, July 2010b. URL https://hal.inria.fr/inria-00494062.

C. Pollard. Type-logical HPSG. In *Proceedings of Formal Grammar*, pages 107–124, 2004.

C. Pollard and I. A. Sag. *Head-driven phrase structure grammar*. University of Chicago Press, 1994.

S. M. Shieber. Evidence against the context-freeness of natural language. In *Philosophy, language, and artificial intelligence*, pages 79–89. Springer, 1985.

D. Tse and J. R. Curran. Chinese ccgbank: extracting ccg derivations from the penn Chinese treebank. In *Proceedings of the 23rd international conference on computational linguistics (Coling 2010)*, pages 1083–1091, 2010.

G. van Noord. At last parsing is now operational. In *Actes de la 13ème conférence sur le Traitement Automatique des Langues Naturelles. Conférences invitées*, pages 20–42, 2006.

G. van Noord, G. Bouma, F. van Eynde, D. De Kok, J. van der Linde, I. Schuurman, E. Tjong Kim Sang, and V. Vandeghinste. Large scale syntactic annotation of written Dutch: Lassy. 2013.

P. Wadler. Propositions as types. *Communications of the ACM*, 58(12):75–84, 2015.

CHAPTER IV

Proof Learning

todo

Reflections are normally reserved for the end of a chapter, but I'm in a pensive mood so I'll do them here instead. This thesis to-be is part of a broader whole, a project set off with the noble goal of developing a composition calculus for vector-based semantic modelling. In its days of inception, the goal was not just noble but also technically reasonable. Distributional semantics and word vectors were in their heyday, machine learning was rapidly taking off, parsers all of a sudden started becoming reliable; everything seemed to point towards the imminent bloom of a new era in natural language processing, an era where the wisdoms of old would meet the machines of today, hinting at a bright and prosperous future for structure-aware semantic composition models. And things did seem to go that way, at least for a few years. But, unbeknownst to all, the new era in the evolutionary history of NLP would actually turn out to also be its dumbest one. The advent of data efficient neural architectures (combined with their immense potential for commercial applications, and its allure to big corporations) brought large language models into the game: unsophisticated, blatantly over-parameterized general purpose systems, fed unprocessed texts for weeks on end until they learn to convincingly imitate its use. Large language models usurped the heir apparent and condemned structure-aware semantic computation to obscurity; the future of computational semantics is to be opaque, boring, exclusive to big tech and provided as-a-service instead. My thesis got caught in the blast of this change of power, necessitating a clear positioning in the current state of affairs, and a careful motivation for the chapter to follow; anything and everything done as

– or in the name of – science requires justification after all. So here goes.

Parsing is good. Converting raw signals intro structured representations thereof allows us to standardize their machine processing, and elevates automated reasoning away from form and into substance. The more well-behaved the representational format chosen, the more powerful, transparent and verifiable the reasoning can be. The less localized and problem-specific the representational format chosen, the more adaptive and better understood the reasoning can be. On the basis of these observations, λ calculi make for an ideal representation format. Choice of format aside, a formal system operating on formal representations is not prone to implicit biases, latent variables, ambiguity, or inconsistency: erroneous outputs are the result of bad input or bad programming. Specifically in the natural language domain, advancing the conversion of text into formal representations is promoting accountable automation of textual processing, and eliminates the anthromorphic delusion of the ghost in the machine. Parsing is therefore only superficially in competition with large language models, and its seeming obsoletion is just a by-product of the ephemeral and rapidly shifting pop science trends of the "AI" race.

That said, machine learning is not bad. Shifting the focus away from the algorithm and toward the data can often be a reasonable concession in the automation of complex or labor-intensive tasks, provided that the task is not risk critical and that no intelligence is attributed to the end system. This is especially the case if "almost correct" is almost as good as correct, or the problem being modeled is intractable, making an approximation the best we can hope for. But employing machine learning has to be thought of as either a shortcut, or an admission of defeat. In opting for a machine learning solution, one assigns more faith to a generic data cruncher in solving a problem, than to oneself in designing a solution to that problem.

Interweaving symbolic and subsymbolic reasoning is then the responsible engineer's out. Disassembling a system into two tiers of components promotes the selective expenditure of formal effort where it really is needed, while still benefiting from the high horsepower of brute force statistical machinery. Complicated but decipherable components, rich in hierarchical or recursive structure, and requiring or greatly benefiting from formal transparency are to be tackled explicitly, whereas components that are laborious but uninteresting, data intensive, or intactable are to be isolated and outsourced to a machine worker. In this here context, I'm claiming that large language models should be treated not as a substitute but as complementary to logic-based systems. This is exactly the route we'll follow in this chapter, where we'll go through the hoops of designing and implementing a formally disciplined yet accurate and robust wide-coverage parser, a novel neurosymbolic architecture aimed at substructural logics of the linear lineage, instantiated here for $\mathbf{NLP}_{\Diamond,\Box}$ and trained on Æthel.

13 Building a Categorial Parser

We'll start with a high-level conceptualization of the categorial grammar parser. In the infancy of categorial grammars, the parser would be thought of as nothing other than a lexicon and a theorem prover: the lexicon enumerating any and all the possible type assignemnts for each word, the theorem prover exhaustively iterating the combinatorial space of assignments to produce all possible proofs for each possible assignment (Figure IV.1).

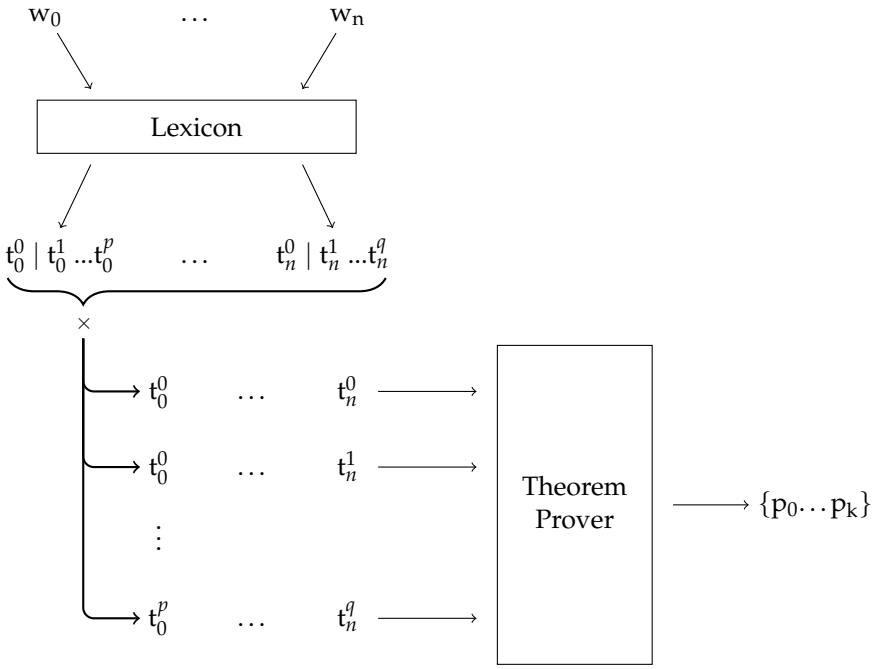


Figure IV.1: The archetypical categorial grammar parsing pipeline.

Obviously, this setup hits a brick wall in the sheer complexity of real human language. As we have discussed in earlier chapters, a type system enacting a strict grammar logic is not just hard to design, but also entails a prohibitively ambiguous type lexicon. Even if the theorem prover is perfectly optimized, the architecture will become bottlenecked at its input. The total number of assignments to consider in a sentence increases exponentially with its length, so even a minor increase in the average number of types per lexical entry will have a tremendous impact in processing time. At the same time, a fixed lexicon is a severely limiting factor, as it effectively forbids processing sentences containing unseen lexical entries, i.e. if a (word, type) pair is missing from the lexicon. Relaxing the structural properties of the type system to ease lexical pressure is not a panacea either. Other than the prover becoming

ambiguous, more proofs will become accessible, and inacessible proofs will take longer to reject. In sum, from the implementer's perspective lexicon and grammar are not synergistic but in conflict with one another, and a middle ground must be found for them to work well in tandem.

These very real problems require equally real solutions for the categorial program to come to fruition: the practitioner must often resort to tricks aimed at compressing or efficiently navigating the enormous search space. The modern pipeline commonly pipes a lexical disambiguation component, usually referred to as the *supertagger*. The supertagger is tasked with ranking the possible assignments to a single entry, given its context of appearance. Assignments are ranked according to their likelihood, in turn approximated on the basis of some training data. Depending on the quality and speed of the statistical estimator employed, the entries returned are truncated depending on some threshold likelihood (or just by their count). This (partially) sidesteps the explosive combinatorics of considering all potential assignments, setting an upper boundary to the cardinality of the parser's input. The parser may also be sped up by allowing yet another statistical model to guide its actions anytime more than a single option is available. As with all real solutions, perfect is unattainable; this time/space efficiency usually coming at the cost of approximation errors that translate in foregone rigidness, correctness and/or coverage.

The strategy we'll follow does not challenge this general model, but contributes some new insights to the operationalization of its components.

14 Supertagging

A supertagger is a statistical model, a parametric function f_θ tasked with producing the most likely type assignment sequence $t_0 \dots t_n$ for a given sentence $w_0 \dots w_n$.

$$f_\theta(w_0 \dots w_n) \approx \underset{t_0 \dots t_n}{\mathsf{argmax}}\ p(t_0 \dots t_n \mid w_0 \dots w_n, \theta) \tag{IV.1}$$

To do so, it should in theory approximate the probability of a type assignment sequence conditional on the input; in other words, feeding f_θ with any element of the product space $\mathcal{L}^k$ should implicitly produce a total order over the product space $\mathcal{U}^k$, where $\mathcal{L}$ the set of words in the language, $\mathcal{U}$ the type universe, and k ranging over $\mathbb{N}$. If that looks stupidly intractable, it's because it is. Both domain and codomain are practically infinite: regardless of what the cardinality of $\mathcal{L}$ and $\mathcal{U}$ are, the number of combinations between different sequences thereof quickly exceeds our current estimates for stars in the universe as the sequence length increases. Put simply, no amount of sample data would ever be able to overcome the problem's inherent sparsity and allow for a direct attempt at an approximation. Therefore, in practice, some truncations and independence assumptions are necessary in how we choose to formulate

the sequence-wide conditional assignment probability:

$$p(t_0 \dots t_n \mid w_0 \dots w_n) \tag{IV.2}$$

before we get to even contemplate implementing the model. The choice of assumptions and truncations made can (and do) have a deep impact on the model's performance, but also its suitability towards a specific grammar. This last fact seems to have largely been dismissed by the broader practitioner community, who treat the problem with consistent indifference, changing the viewing lens only according to the quirks and fashions of contemporary machine learning standards. We will shamelessly fall into the same last trap, but in our downfall we will be conscious of the intellectual and ideological roots the earlier chapters have established; those of revealing structure previously hidden, and paying that structure its due respect.

14.1 A Brief History of Supertagging

But to actually perceive the structure, we must first see its absence – therefore (and for maintaining supsense), we will first outline the short but dense history of supertagging, and sketch out the paradigm shifts it has undergone throughout.

14.1.1 Origins

Supertagging (both the term and the idea) are due to early insights of Joshi and Bangalore [1994]. The two correctly pointed out that, for a strongly lexicalized grammar (in their case, a Tree Adjoining Grammar), assining the correct grammatic descriptor, or *supertag* (in our case, a type), to each lexical entry within a sentence amounts to *almost* parsing, and that even just weeding out some of the erroneous assignments significantly facilitates parsing. To counteract sparsity, and in alignment with the *modus operandi* at the time, early supertaggers opted for an independence assumption between words and supertags outside a local context window.

For a so-called unigram model (full independence between subsequent words), (IV.2) boils down to:

$$\prod_{i}^{n} p(t_i \mid w_i) \tag{IV.3}$$

where each local conditional can be estimated on the basis of corpus frequencies. On its own, this is not good enough either, as rare and unknown entries would hardly give sufficient data to extract an empirical distribution from. As a solution, plain part of speech tags would find use as an intermediary, i.e. w_i would in practice be substituted by pos_i, which would in turn be supplied by an external tagger. todo: too simple

Invoking Bayes' rule and factoring out the denominator has (IV.2) rewrite to the proportionate quantity:

$$\propto p(w_0 \dots w_n \mid t_0 \dots t_n)\ p(t_0 \dots t_n) \tag{IV.4}$$

Extending the context to a window of size κ, allows supertag assignments at position i to excert infuence on the supertags of future positions via the approximation of the *contextual probability* $p(t_0 \dots t_n)$ as:

$$p(t_0 \dots t_n) \approx \prod_i^n (t_i \mid t_{i-\kappa} \dots t_{i-1}) \tag{IV.5}$$

Going one step further and making the assumption that the *emission probability* $p(w_0 \dots w_n \mid t_0 \dots t_n)$ is position-seperable and independent allows its rewrite to:

$$p(w_0 \dots w_n \mid t_0 \dots t_n) \approx \prod_i^n p(w_i \mid t_i) \tag{IV.6}$$

Putting (IV.5) and (IV.6) together, we get an approximation to

Chapter Bibliography

S. Bangalore and A. Joshi. Supertagging: An approach to almost parsing. *Computational linguistics*, 25(2):237–265, 1999.

A. K. Joshi and S. Bangalore. Disambiguation of super parts of speech (or supertags) almost parsing. In *Proceedings of the 15th conference on Computational linguistics-Volume 1*, pages 154–160, 1994.

Appendix

A Abbreviations

Abbreviation	Meaning	Abbreviation	Meaning
1	first person	NMLZ	nominalization
3	3rd person	NN	non-neuter
ACC	accusative	NOM	nominative
ADV	adverbial	PL	plural
DEF	definitive	PTCP	participle
DIM	neuter	PTV	partitive
F	feminine	PRS	present
IMP	imperative	PST	past
INF	infinitive	SG	singular
M	masculine	SUP	superlative
N	neuter		

Table V.1: Gloss abbreviations.

B NLP$_{\diamondsuit,\square}$ with Python