Dependency as Modality, Parsing as Permutation

A Neurosymbolic Perspective on Categorial Grammars

Published by

LOT
Binnengasthuisstraat 9
3512 JK Utrecht
The Netherlands

phone: +31 20 525 2461
lot@uva.nl
http://www.lotschool.nl

Cover Illustration: Konstantinos Kogkalidis, *Let's learn GIMP in case the whole academia thing goes awry* (2022)

ISBN: todo
DOI: 10.33540/1721
NUR: todo

Dependency as Modality, Parsing as Permutation

A Neurosymbolic Perspective
on Categorial Grammars

Dependenties als Modale Operatoren, Ontleding als Permutatie

Een Neurosymbolische Blik
op Categoriale Grammatica's

(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
vrijdag 9 juni 2023 des ochtends te 10.15 uur

door

Konstantinos Kogkalidis

geboren 19 Juli 1991
te Thessaloniki, Griekenland

Promotor:
Prof. dr. M. J. Moortgat

Copromotor:
Dr. R. Moot

Beoordelingscommissie:
Prof. dr. A. P. J. van den Bosch (voorzitter)
Prof. dr. M. Sadrzadeh
Prof. dr. J. Bos
Prof. dr. S. Clark
Prof. dr. S. Chatzikyriakidis

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).

How fleeting are all human passions compared with the massive continuity of ducks.

Dorothy L. Sayers, *Gaudy Night*

Contents

Acknowledgements

I thank everyone that was directly or indirectly involved in the completion of this thesis, and everyone who has supported my academic & emotional well-being over the last five years. Y'all know who you are; I appreciate you and you are dear to me. Now skip the drama and go read the actual thing.

Abstract

Since their inception, categorial grammars have been front runners in the quest for a formally elegant, computationally attractive and adequately flexible theory of linguistic form and meaning. As a result of developments in theoretical computer science, Lambek-style categorial grammars have gradually been recognized for what they truly are: type-systems proper. Words enact typed constants, and interact with one another via means of grammatical rules enacted by type inferences, composing larger phrases in the process. The end result is at the same time a parse, a proof and a program, bridging the seemingly disparate fields of linguistics, formal logics and computer science; a testament to the holy triptych of language, logic and computation. The transition from form to meaning is traditionally handled in a Montague-style fashion via a series of homomorphic translations that gradually remove or simplify nuances of the syntactic type calculus to move towards a uniform and expressive semantic calculus. Alluring as this might be, it poses pragmatic problems for the whole programme to come to fruition. For the setup to work on the semantic level, one has no choice but to start from the hardest part, namely the type-theoretic treatment of natural language syntax. Phenomena like movement, word-order variation, discontinuities and the like require careful treatment that needs to be both general enough to encompass the full range of grammatical utterances, yet strict enough to ward off ungrammatical derivations.

Breaking away from tradition, this thesis takes an operational shortcut in targeting a "deeper" calculus of grammatical composition, engaging only minimally with surface syntax. Where previously functional functional syntactic types would be position-conscious, requiring their arguments in predetermined positions upon a binary tree, they are now agnostic to both tree structure and sequential order, alleviating the need for fine-grained syntactic refinements. This simplification comes at the cost of a misalignment between provability and grammaticality: the laxer semantic calculus permits more proofs than linguistically allowed. To partially circumvent this underspecification, the thesis takes an additional step away from the established

norm, proposing the incorporation of unary type operators extending the analytical axis from plain function-argument structures to function-argument structures with fixed grammatical roles. The new type calculus produces mixed unary/n-ary trees, each unary tree denoting a dependency domain, and each n-ary tree underneath it denoting the phrases which together form that domain. Although still underspecified, these peculiar structures directly subsume non-projective labeled dependency trees. More than that, they have their roots set firmly in type theory, paving the way to their meaningful semantic interpretation.

On more practical grounds and in order to investigate the formalism's expressive adequacy, an extraction algorithm is designed and employed to convert syntactic analyses of Dutch sentences represented as dependency graphs (stemming from the Lassy small corpus) into proofs of the target logic. The vast majority of input analyses is successfully handled, giving rise to a large and versatile proofbank, a collection of sentences paired with tectogrammatic theorems and their corresponding programs, and an elaborate type lexicon, providing type assignments to almost one million lexical tokens within a given linguistic context.

The proofbank and the underlying lexicon both find use as training data in the design and implementation of a neurosymbolic proof search system able to efficiently navigate the logic's expansive theorem space. The system consists of three major components that alternate role within the processing pipeline. Component number one is a supertagger responsible for assigning a type to each input word — the tagger is formulated on the basis of a hyper-efficient heterogeneous graph convolution kernel that boasts state-of-the-art accuracy among categorial grammar datasets. Rather than produce type asignments in the form of conditional probabilities over a predefined type vocabulary, the supertagger instead constructs types dynamicaly, following their algebraic decomposition. As such, it is unconstrained by sparsity and data underrepresentation, generalizing well to rare assignments and even producing correct assignments for types never seen during the course of training. Component number two is a neural permutation module that exploits the linearity constraint of the target logic in order to simplify proof search as optimal transport learning, associating resources (conditional validities) to the processes that require them (conditions). This reformulation allows for a massively parallel and easily optimizable implementation, unobstructed by the structure manipulation breaks common in conventional parsers. Component number three is the type system itself, responsible for navigating the produced structures and thus asserting their well-formedness. Results suggest efficiency superior to, and performance on par with, established baselines across categorial formalisms, despite the ambiguity inherent to the logic.

CHAPTER 0

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 this communication channel is asynchronous and unidirectional so I will be doing double duty for the both of us.

So let's start with you. A range of scenarios are plausible as to how you came to be reading these letters. The safest bet (and easiest way to score some easy points on precision) is to assume this is some obligation of sorts, in which case you have my sympathy and gratitude. Also likely, you might be acquaintance of mine – academic, social, or both – driven by the curiosity to figure out what it is I spent 5 (and counting) years in Utrecht for; well, pretty much this. Less likely, you could just lazilly be scrolling through the opening pages, contemplating whether I'd be a good fit for some organization you are representing; if so, you should definitely go for me.[1] Least likely, could it even be that you are *actually* interested in the subject matter of this thesis? Exciting, but also slightly alarming; I feel somewhat conscious knowing that you'll be putting my words under a critical lens – I'll do my best not to fail your expectations. In the wildcard scenario where 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. More realistically, if noone 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 call myself Kokos. I had the enormous luck of crossing paths with my supervisor, Michael, during my first weeks of graduate studies in

[1] Exceptionally, if this happens to be some big corp reptile den, scram – and shame on you, future me.

Utrecht. The repercussions of this encounter were (and still are) unforeseeable. Coming from an applied background with an innate repulsion to anything formal, his course offered 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 the most 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 the tiniest of 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 my mental model of you is a purely artificial construct of 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 is this about The quote below was received almost verbatim as a review. Mean spirited as it may be, it provides an adequate high-level summary of this thesis' contents:

> [The paper] 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 paper] and their relevance.
>
> Unknown reviewer, 2019.

Thanks to this fellow scientist's earnest reviewing work, all I need to do here is first align the above summary with the manuscript through a chapter breakdown, and then explicate the thesis' contributions in a childproof way.

Chapter Breakdown

Chapter 0 greets the reader and provides a chapter summary, while also setting the tonal precedents for what is to follow. You are currently in it.

Chapter I is an attempt at a painless introduction to simple type theory and its substructural variants, with an emphasis on their significance for linguistics. We set things off in Section 1 with a crash course on simple type theory, a formal model of computation and logical deduction. Removing the ability to erase or duplicate logical propositions, we transition to linear type theory in Section 2 – a place where the motto is resource consciousness and truth is not for free. Following along the same path, in Section 3 we take the extra step of making propositions immovable and bracket-bound to their surroundings, revealing the landscape of Lambek calculi **(N)L(P)**. To regain some of the expressivity lost in the passage, we call modalities to the aid in Section 4 – these allow us to reinstate the implicit equivalences of before as explicit rules with limited and controllable applicability. In Section 5, the theoretical wisdoms amassed through our substructural expedition find use in defining categorial grammars: type-driven frameworks formalizing the syntax and semantics of natural languages. We discuss two relevant and related paradigms: multi-modal type-logical grammars and abstract categorial grammars; both use constraints imported by types to control grammatical composition, converting parsing to a process of logical deduction. At long last, we have all the foundational knowledge needed to move on.

Chapter II offers a non-standard usecase for the structural control modalities of the multimodal Lambek family $\mathbf{(N)L(P)}_{\diamond,\square}$. We begin in Section 7 with a face-off between the two strands of grammar flavors that have dominated computational linguistics in the past decades – namely constituency and dependency grammars – and see how they compare to categorial grammars. Unsatisfied by the comparison, we move on to Section 8, where we appropriate the modalities we resorted to earlier, repurposing them now as dependency domain demarcators.

Chapter III instantiates a type system based on this new envisaging of modalities, and employs it as a derivational semantics logic[1], in alignment with real-world corpus data. Section 10 sets the stage with a backstory motivating the design choices made and describing the source corpus. We proceed to the real thing in Section 11, where we illustrate how the corpus' analyses can be recast as proof-theoretic inhabitants in our framework. We detail the extraction process, the resulting view of the corpus, and its practical evaluation as a stand-alone resource.

Chapter IV makes for a drastic change of scenery, offering a collection of insights on the neural parsing of substructural grammar logics. We first

[1] Or abstract syntax logic, depending on which side of the dividing line you stand at.

paint a picture of the archetypical categorial grammar parser in Section 13, pinpointing the tension points between abstract theory and applied practice and exposing the need for a disciplined merger between the formal and the informal. In Section 14 we trace along the history of supertagging: the statistical disambiguation process (and the machinery empowering it) through which a system can automatically infer the most likely type for a lexical item in context (i.e. a word in a sentence). We start from its (not-so-ancient) origins and go all the way to today, motivating the abolition of a set-in-stone type vocabulary as a natural step in its evolutionary progress and providing two convincing implementations to that end. For the grand finale, in Section 15 we propose a neural operationalization for the proof nets of linear logic. Invoking their bureaucracy-free format, we uncover a brand new paradigm for the statistical parsing of substructural grammar logics in the **(N)L(P)** lineage.

Chapter V wraps things up and waves the reader goodbye.

Contributions Contributions produced and presented in this thesis, organized in bullet points for your convenience and reading pleasure, include:

- a **type-driven** model of compositional syntax that simultaneously captures **dependency-** and **function-argument- structures**; not a world first, but a close second by 30 years
- a big, open-source, **well-typed dataset** of proof-derivations for written Dutch
- the **first supertagger** to correctly **construct novel type assignments**, operating without a fixed lexicon
- the current **state of the art** supertagger that outperforms accuracy benchmarks **across grammar frameworks**, without foregoing the ability to predict rare and unseen assignments – essentially an assurance that sparse and elaborated categorial grammars are of practical use, despite prior disdains
- a **neural operationalization** of linear logic's **proof nets** into a massively parallel, differentiable and hyper-performant proof search engine – essentially a reconciliation between the modern neural toolbox and the Lambekian tradition, and a call back to typing discipline for categorial practitioners

How to README The thesis is best read in the order presented; each chapter is (weakly) dependent on its predecessors. That said, you can skip Chapter I if already familiar with type theory and grammar logics – you can always refer back to it later in case of emergency. Chapter II is self-standing and provides background that is necessary for Chapter III to make sense – skipping it is ill-advised. But while Chapter III should prove helpful in appreciating the empirical results of Chapter IV, it is by no means a prerequisite. If you are indifferent to the dataset and its construction, and only interested in the neural stuff from a high-level perspective, you could skip through straight to Chapter IV (use your power of imagination to fill in any gaps). If you're not interested in the neural stuff either, you probably downloaded the wrong document and may as well stop reading now. Assuming you're still with me, I wish you a pleasant reading.

Publications Chapter II is a novel, extended collage of work taking secondary role in Kogkalidis et al. [2020a] and Moortgat et al. [2023]. Chapter III is an extended version of Kogkalidis et al. [2020a]. Chapter IV is based on Kogkalidis et al. [2019, 2020b] and Kogkalidis and Moortgat [2022, preprint] – a practical summary is compiled in Kogkalidis et al. [2023]. The whole manuscript is a bigger, better, faster, stronger (or so I'd like to think) version of early work delivered as part of my master's thesis [Kogkalidis, 2019].

Papers this dissertation is based on

K. Kogkalidis. Extracting and learning a dependency-enhanced type lexicon for Dutch. Master's thesis, Utrecht University, 2019. URL https://studenttheses.uu.nl/handle/20.500.12932/32880.

K. Kogkalidis and M. Moortgat. Geometry-aware supertagging with heterogeneous dynamic convolutions, 2022. URL https://arxiv.org/abs/2203.12235.

K. Kogkalidis, M. Moortgat, and T. Deoskar. Constructive type-logical supertagging with self-attention networks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 113–123, Florence, Italy, Aug. 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4314. URL https://aclanthology.org/W19-4314.

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 2020a. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.647.

K. Kogkalidis, M. Moortgat, and R. Moot. Neural proof nets. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 26–40, Online, Nov. 2020b. Association for Computational Linguistics. doi: 10.18653/v1/2020.conll-1.3. URL https://aclanthology.org/2020.conll-1.3.

K. Kogkalidis, M. Moortgat, and R. Moot. SPINDLE: Spinning raw text into lambda terms with graph attention. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, Dubrovnik, Croatia, 2023. Association for Computational Linguistics. To Appear.

M. Moortgat, K. Kogkalidis, and G. Wijnholds. Diamonds are forever: Theoretical and empirical support for a dependency-enhanced type logic. In

R. Loukanova, P. LeFanu Lumsdaine, and R. Muskens, editors, *Logic and Algorithms in Computational Linguistics 2021 (LACompLing2021)*, volume 1081 of *Studies in Computational Intelligence SCI*. Springer, March 2023.

CHAPTER I

Introduction

In the beginning 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 famous 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. The system would enact 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 formalist hubris, 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; a fact both important and interesting, but which 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 intersection 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 machinery for Montague's program, seeing as they admit the Curry-Howard correspondence, and are therefore able to drive semantic composition virtu-

[1] In proof theory, at least.

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

ally for free (in fact one could go as far as to say they are the *only* kind of machinery that can accomplish such a feat without being riddled with ad hoc designs). 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 compositional manipulations they dictate upon computational 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 Brouwer's intuitionism. It is a restricted version of classical logic, where the laws of the excluded middle (*tertium non datur*) and the elimination of the double negation no longer hold universally. The first states that one must choose between a proposition A and its negation $\neg\mathrm{A}$ ($\mathrm{A} \vee \neg\mathrm{A}$), whereas the second states that negation is its own inverse ($\neg\neg\mathrm{A} \equiv \mathrm{A}$). The absence of these two laws implies that several theorems of classical logic are no longer derivable in intuitionistic logic, meaning that the logic is weaker in terms of expressivity. On the bright side, it has the pleasant effect that proofs of intuitionistic logic are constructive, i.e. they explicitly demonstrate the formation of a concrete instance of whatever proposition they claim to be proving.

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

$$\textsc{a}, \textsc{b}, \textsc{c} := p \mid \textsc{a}{\rightarrow}\textsc{b} \mid \textsc{a} \times \textsc{b} \tag{I.1}$$

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

$$\Gamma, \Delta, \Theta := \varnothing \mid \textsc{a} \mid \Gamma, \Delta \tag{I.2}$$

In other words, structures are an inductive set closed under the operator $\_,\_$ which satisfies associativity and is equipped with an identity element $\varnothing$ (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 $\Gamma \vdash \textsc{a}$. We read such a statement as a suggestion that from *assumptions* Γ (i.e. a structure of *hypotheses*) one can derive the proposition $\textsc{a}$. Formulas occurring within Γ are said to occur in *antecedent* position, whereas $\textsc{a}$ is in *succedent* position.

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

Rules may be split in two conceptual categories. *Logical* rules, on the one hand, provide instructions for eliminating and introducing logical connec-

[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.

tives. 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 $A \rightarrow B$ 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 $A \rightarrow B$. In a similar manner, the elimination of the conjunction $\times E$ states that, given a proof of a proposition $A \times B$ 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 $A \times B$, 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, the exchange rule ex explicates an extra property 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 formalizes *deletion*; it 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 formalizes *copying*; it 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 subtrees 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 in two kinds: syntactic equivalences (i.e. sequences of rule applications that can safely be rearranged) and redundant detours (i.e. sequences of rule appli-

$$\frac{}{\textsc{a} \vdash \textsc{a}}\ \mathsf{id}$$

$$\frac{\Gamma \vdash \textsc{a} \to \textsc{b} \quad \Delta \vdash \textsc{b}}{\Gamma, \Delta \vdash \textsc{b}}\ \to E \qquad \frac{\Gamma, \textsc{a} \vdash \textsc{b}}{\Gamma \vdash \textsc{a} \to \textsc{b}}\ \to I$$

$$\frac{\Gamma \vdash \textsc{a} \times \textsc{b} \quad \Delta, \textsc{a}, \textsc{b} \vdash C}{\Gamma, \Delta \vdash C}\ \times E \qquad \frac{\Gamma \vdash \textsc{a} \quad \Delta \vdash \textsc{b}}{\Gamma, \Delta \vdash \textsc{a} \times \textsc{b}}\ \times I$$

(a) Logical rules.

$$\frac{\Gamma, \textsc{b}, \textsc{a}, \Delta \vdash \textsc{c}}{\Gamma, \textsc{a}, \textsc{b}, \Delta \vdash \textsc{c}}\ \mathsf{ex}$$

$$\frac{\Gamma \vdash \textsc{b}}{\Gamma, \textsc{a} \vdash \textsc{b}}\ \mathsf{weak} \qquad \frac{\Gamma, \textsc{a}, \textsc{a} \vdash \textsc{b}}{\Gamma, \textsc{a} \vdash \textsc{b}}\ \mathsf{contr}$$

(b) Structural rules.

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

cations that can altogether be removed).

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

The second kind is more interesting and slightly more involved. A proof pattern in which a logical connective is introduced, only to be immediately eliminated, is called a *detour* (or β redex). Detours can be locally resolved via proof rewrites – the fix-point of performing all applicable resolutions is called *proof normalization* and yields a canonical proof form. The strong normalisation property guarantees that a canonical form exists for any proof in the logic, and in fact the choice of available rewrites to apply at each step is irrelevant, as all paths have the same end point [de Groote, 1999]. Figure I.2 presents rewrite instructions for the two detour patterns we may encounter (one per logical connective). Read bottom-up[1], the first one suggests that if one were to hypothesize a proposition A, use it within an (arbitrarily deep) proof s together with extra assumptions Γ to derive a proposition B, before finally redacting the hypothesis and composing with a proof t that derives A from assumptions Δ, it would have been smarter (and more concise!) to just plug in t directly when previously hypothesizing A, since then no redaction or composition would

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

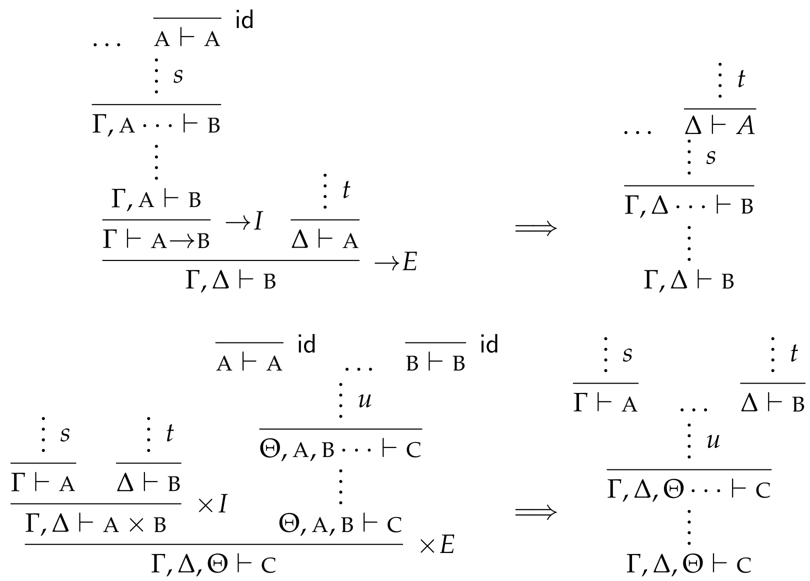


Figure I.2: Intuitionistic β redexes.

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 atomic formulas Prop_0 becomes the programming language's basic set of *primitive* or *base* types (think of them as built-ins). Implicational formulas A→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 corre-

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

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

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

$$\frac{\Gamma, B, A, \Delta \vdash s : C}{\Gamma, A, B, \Delta \vdash s : C}\ \mathsf{ex}$$

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

Figure I.3: Simple type theory.

spondence 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 $\{x_i, x_j, x_k, x_l, \ldots\}$, and denoting arbitrary but well-formed terms with s, t, u, we will use $s : A$ (or s^A) to indicate that term s is of type A. Assumptions Γ, Δ will now denote a *typing environment*:

$$x_i : A,\ x_j : A',\ x_k : A'' \ldots \tag{I.3}$$

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

Inspecting Figure I.3, things for the most part look good. The implication elimination rule ${\to}E$ provides us with a composite term $s\,t$ that denotes the function application of *functor* s on *argument* t. Function application is left-associative: s t u is the bracket-economic presentation of (s t) u – we have no choice but to use brackets if want to instead denote s (t u). The dual rule, $\rightarrow I$, 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 λx_i.s. Any occurrence of x_i within s is then *bound* by the abstraction; variables that do not have a binding abstraction are called *free*. The conjunction introduction $\times I$ allows us to create tuple objects (s, t) through their parts s and t. Its dual, $\times E$, 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_k in term s; note here the meta-notation for *variable substitution*, $s_{[x_i \mapsto t]}$, which reads as "replace any occurrence of variable x_i with term t. And finally, we can introduce throw-away variables into our typing environment with weak (arguably useful for creating things like constant functions).

There's just a few catches to beware of. The first has to do with tracing variables in a proof; the concatenation of structures Γ, Δ is only valid if Γ and Δ contain no variables of the same name; if that were to be the case, we would be dealing with variable shadowing, a situation where the same name could ambiguously refer to two distinct objects (a horrible thing). The second has to do with 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 $[x_i \mapsto x_j]$ described above. Despite seeming innocuous at a first glance, α conversion is an 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.

$$\lambda x_i . x_i^A \overset{\alpha}{\equiv} \lambda x_j . x_j^A \tag{I.4}$$

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:

$$(\lambda x_i . s)\ t \overset{\beta}{\rightsquigarrow} s_{[x_i \mapsto t]} \tag{I.5}$$

$$\text{case } (s, t) \text{ of } (x_i, x_j) \text{ in } u \overset{\beta}{\rightsquigarrow} u_{[x_i \mapsto s, x_j \mapsto t]} \tag{I.6}$$

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

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

[1]The denotational significance of these letters I have yet to understand – legend has it that the spirit of Curry will gently whisper it in your ear after having successfully written your 100th 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

Figure I.4: The Curry-Howard correspondence in tabular form.

"just" computation. This 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

Figure I.4 summarizes the subsection.

1.3 Intermezzo

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

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

Table I.1: Polarity induction.

$$\dfrac{\dfrac{\dfrac{}{\mathsf{x}_i : A\rightarrow B \vdash \mathsf{x}_i : A\rightarrow B}\,\textsf{id} \qquad \dfrac{}{\mathsf{x}_j : A \vdash \mathsf{x}_j : A}\,\textsf{id}}{\mathsf{x}_i : A\rightarrow B, \mathsf{x}_j : A \vdash \mathsf{x}_i\ \mathsf{x}_j : B}\,\rightarrow E}{\mathsf{x}_j : A \vdash \lambda \mathsf{x}_i.\mathsf{x}_i\ \mathsf{x}_j : (A\rightarrow B)\rightarrow B}\,\rightarrow I$$

Figure I.5: Type raising.

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 succedent 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 the atomic level 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.1. 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 $A^+ \vdash (A^- \rightarrow B^+) \rightarrow B^-$ is a derivable theorem of intuitionistic logic presented in Figure I.5. It states that for A, B arbitrary propositions, from A one can derive its raised form $(A\rightarrow B)\rightarrow B$. The converse, i.e. type lowering, does not generally hold: $(A^+ \rightarrow B^-) \rightarrow B^+ \nvdash A^-$.

Function Order The implication-only fragment of the logic includes $\rightarrow$ 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:

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

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

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

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: $(\mathrm{A} \times \mathrm{B}){\rightarrow}\mathrm{C} \dashv\vdash \mathrm{A}{\rightarrow}\mathrm{B}{\rightarrow}\mathrm{C}$. The forward direction is called currying, and the backward uncurrying; you can find a proof for each in Figure I.6. 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:

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

Keep a mental note.

Proof Search Attempting to derive a judgement of the form $\Gamma \vdash \mathrm{A}$ amounts to searching for a suitable proof of that statement, a process called *proof search*. We distinguish two directions of proof search: the *backward chaining* (or top-down) approach starts from the goal judgement and iteratively expands it into judgements with smaller assumptions – one judgement per premise generated by the rule of inference applied – with the intention being the eventual deconstruction of all branches into axioms of identity. The other direction is called

$$\dfrac{\dfrac{\dfrac{}{x_i : (A \times B)\to C}\,\text{id} \qquad \dfrac{\dfrac{}{x_j : A}\,\text{id} \quad \dfrac{}{x_k : B}\,\text{id}}{x_j, x_k \vdash (x_j, x_k) : A \times B}\,\times I}{x_i, x_j, x_k \vdash x_i\ (x_j, x_k) : C}\,\to E}{x_i \vdash \lambda x_j x_k . x_i\ (x_j, x_k) : A\to B\to C}\,\to I(x2)$$

(a) Currying

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x_i : A \times B}\,\text{id} \qquad \dfrac{\dfrac{\dfrac{}{x_j : A\to B\to C}\,\text{id} \quad \dfrac{}{x_k : A}\,\text{id}}{x_j, x_k \vdash x_j\ x_k : B\to C}\,\to E \quad \dfrac{}{x_l : B}\,\text{id}}{x_j, x_k, x_l \vdash x_j\ x_k\ x_3 : C}\,\to E}{x_i, x_j \vdash \text{case } x_i \text{ of } (x_j, x_k) \text{ in } x_j\ x_k\ x_l : C}\,\times E}{x_j, x_i \vdash \text{case } x_i \text{ of } (x_j, x_k) \text{ in } x_j\ x_k\ x_l : C}\,\text{ex}}{x_j \vdash \lambda x_i.\text{case } x_i \text{ of } (x_j, x_k) \text{ in } x_j\ x_k\ x_l : (A \times B)\to C}\,\to I$$

(b) Uncurrying

Figure I.6: Interderivability of product and arrow.

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 are confronted with 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 the contr rule yields Affine Logic, a logic in which resources can be used no more than once. Removing the weak rule 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 must be used *exactly* once. The intuitionistic formulations of the above give rise to corresponding type theories [Pierce, 2004]. For the purposes of this thesis, 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 separate 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 $\mathrm{A} \rightarrow \mathrm{B}$ to denote that B logically follows from A, we will now have $\mathrm{A} \multimap \mathrm{B}$ to denote an irreversible process that transforms a single A into a single B, consuming the former in the process (we can think of this as a perfect chemical reaction). The new, weird-looking arrow of linear implication is read as lolli(pop) due to its suggestive appearance.[1] Conjunction $\times$ is now separated into two distinct operators, the multiplicative $\otimes$ and the additive &. The first denotes a linear tuple, and $\mathrm{A} \otimes \mathrm{B}$ is read as *both* A *and* B. A linear tuple offers no possibility of projection: we will need to use both coordinates going forward. The second denotes a choice, and A&B is read as A *with* B, or *choose one of* A *or* B. This choice is *external*, as the freedom of applying it lies with the operator rather than the proof, and is manifested by the presence of two eliminators for our new connective: a left projection $\&E_1$ and a right projection $\&E_2$; choosing one means we lose the possibility of obtaining the other. Unique to the $\&I$ rule is the fact that two proof branches used to

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

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

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

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

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

(a) Logical rules.

$$\frac{\Gamma, \mathsf{x}_j : \mathrm{B}, \mathsf{x}_i : \mathrm{A}, \Delta \vdash \mathsf{s} : \mathrm{C}}{\Gamma, \mathsf{x}_i : \mathrm{A}, \mathsf{x}_j : \mathrm{B}, \Delta \vdash \mathsf{s} : \mathrm{C}}\ \mathsf{ex}$$

(b) Structural rule.

Figure I.7: Linear Logic $\mathbf{ILL}_{\multimap,\otimes,\&}$ and its type theory.

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.7, together with its term rewrites (assumptions, judgements, rules and proofs look just like before). For the sake of homogeneity and explicitness, ex still makes an appearance as the sole structural rule.

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

2.1.1 Proof & Term Reductions

The notions of proof and term equivalence discussed in the previous section hold also for linear logic. Proof normalization looks almost identical to before in the case of $\rightarrow$ 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

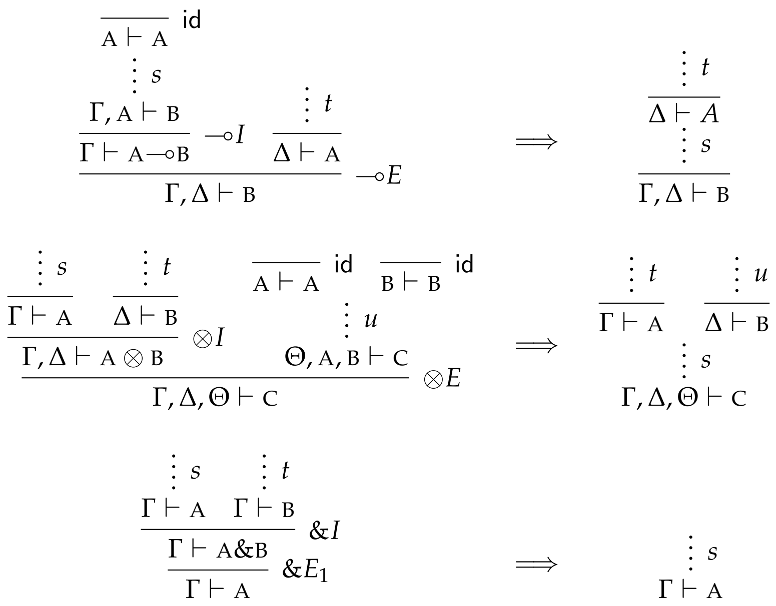


Figure I.8: Linear β redexes.

shown in Figure I.8. 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:

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

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

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

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

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

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

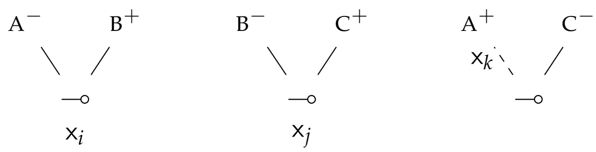
Figure I.9: Linear function composition.

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.

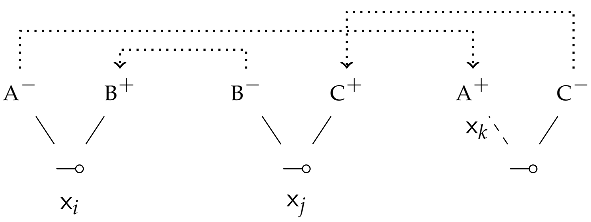
In natural deduction, our proofs are built sequentially. We start with some hypothesized variables and combine them via rules to derive more complex terms, which then serve as premises for a next iteration of rule applications. As long as we are careful not to get stuck in some detour loop hell, we rinse 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.9 and translate it into its proof net equivalent of Figure I.10.

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: atomic formulas become leaves, and logical connectives become branch nodes. To make this a tiny bit more interesting, we will decorate atomic subformulas with a superscript denoting their polarity, according to the schema of Subsection 1.3. For formula trees with positive roots (i.e. trees occuring left of the turnstile), we will put a label beneath them to identify them with the variable that provides them. We will also distinguish tree edges originating from a negative implication and pointing to a positive tree by marking them with dashed lines – these denote positively rooted trees that are either nested within a higher-order positive im-



(a) Proof frame.



(b) Proof structure.

Figure I.10: Proof net construction for the proof of Figure I.9.

plication, 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.10a.

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, pointing negative atoms to their positive matches: we call those *axiom links*. Axiom links essentially specify the elimination of function types: they identify function arguments with their concrete inputs in a geometric fashion. We do not need to put much thought for our running example: since all atoms have a single occurrence of each polarity, there is just one possible bijection to consider, presented in Figure I.10b. 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 proof structures to proof nets is not one-to-one:

[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 (⊗) and par (⅋) links.

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 [De Groote and Retoré, 1996; Lamarche, 2008]. 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 $\lambda\mathsf{x}_k.(\dots)$. 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, getting $\lambda\mathsf{x}_k.\mathsf{x}_j(\dots)$. We proceed in negatative mode to B^-, cross over to B^+ and repeat the above, getting $\lambda\mathsf{x}_k.\mathsf{x}_j\ (\mathsf{x}_i\ \dots)$ 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 $\lambda\mathsf{x}_k.\mathsf{x}_j\ (\mathsf{x}_i\ \mathsf{x}_k)$ 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.9: a joyful outcome! If we consider

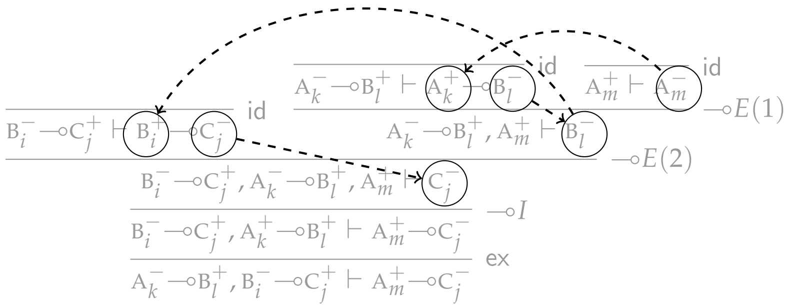


Figure I.11: Extracting axiom links from the proof of Figure I.9.

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.10a to the proof structure of Figure I.10b. A rare and lucky coindidence, or perhaps the result of carefully planned concealment? Whichever the case, 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.9. 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.11, 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.10b – 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 $\mathbf{ILL}_{\multimap}$.

Proof Nets and Search 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 in Chapter IV, we will challenge this perception, and see how their parallel nature can actually be very convenient for heuristic 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** 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

[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?

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 be 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! Nonetheless, 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* – in the literal sense – to Linear Logic; 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:

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

The rules of this fragment are presented in Figure I.12. Alternative presentations can include additive conjunction and/or either of the disjunctions, but the key feature of interest lies in the two implications, / and \. The intuitive way of reading those is as directed fractionals, the formula hidden under the cover of the slash being the denominator, and the formula lying on it the numerator. 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 $\Gamma \vdash \text{A} \otimes \text{B}$ holds. The notions of structure and substructure depend of course on the logic used – in the current setting, Δ is a sequence, to which Γ is a subsequence. The reformulation of the rule is necessary to arbitrate elimination of nested products, since their extraction to the right or left edge of an

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

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

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

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

Figure I.12: Lambek calculus **L**.

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 $\Delta[\![A, B]\!]$). Generally speaking, the empty structure $\varnothing$ is now disallowed.

The Lambek calculus hails from an intuitionistic tradition, and is thus amenable to a propositions as types interpretation [Wansing, 1990]. Adorning 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 $\lambda\!\!\!\backslash$ to denote the two kinds of anonymous functions.

3.1.1 Proof & Term Reductions

The proof reductions of Figure I.13 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

$$\frac{\dfrac{\begin{array}{c}\dfrac{}{\mathrm{A}\vdash \mathrm{A}}\,\text{id}\\ \vdots\, s\\ \Gamma, \mathrm{A}\vdash \mathrm{B}\end{array}}{\Gamma\vdash \mathrm{B}/\mathrm{A}}\,/I \quad \dfrac{\vdots\, t}{\Delta\vdash \mathrm{A}}}{\Gamma,\Delta\vdash \mathrm{B}}\,/E \quad\Longrightarrow\quad \begin{array}{c}\dfrac{\vdots\, t}{\Delta\vdash \mathrm{A}}\\ \vdots\, s\\ \hline \Gamma,\Delta\vdash \mathrm{B}\end{array}$$

$$\frac{\dfrac{\vdots\, t}{\Gamma\vdash \mathrm{A}}\quad \dfrac{\begin{array}{c}\dfrac{}{\mathrm{A}\vdash \mathrm{A}}\,\text{id}\\ \vdots\, s\\ \mathrm{A},\Delta\vdash \mathrm{B}\end{array}}{\Delta\vdash \mathrm{A}\backslash \mathrm{B}}\,\backslash I}{\Gamma,\Delta\vdash \mathrm{B}}\,\backslash E \quad\Longrightarrow\quad \begin{array}{c}\dfrac{\vdots\, t}{\Gamma\vdash \mathrm{A}}\\ \vdots\, s\\ \hline \Gamma,\Delta\vdash \mathrm{B}\end{array}$$

$$\frac{\dfrac{\dfrac{\vdots\, s}{\Gamma\vdash \mathrm{A}}\quad\dfrac{\vdots\, t}{\Delta\vdash \mathrm{B}}}{\Gamma,\Delta\vdash \mathrm{A}\otimes \mathrm{B}}\,\otimes I \quad \begin{array}{c}\dfrac{}{\mathrm{A}\vdash \mathrm{A}}\,\text{id}\quad \dfrac{}{\mathrm{B}\vdash \mathrm{B}}\,\text{id}\\ \vdots\, u\\ \Theta[\![\mathrm{A},\mathrm{B}]\!]\vdash \mathrm{C}\end{array}}{\Theta[\![\Gamma,\Delta]\!]\vdash \mathrm{C}}\,\otimes E \quad\Longrightarrow\quad \begin{array}{c}\dfrac{\vdots\, s}{\Gamma\vdash \mathrm{A}}\quad\dfrac{\vdots\, t}{\Delta\vdash \mathrm{B}}\\ \vdots\, u\\ \Theta[\![\Gamma,\Delta]\!]\vdash \mathrm{C}\end{array}$$

Figure I.13: Lambek β redexes.

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

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

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

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

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

$$\frac{}{\mathsf{x}_i : \textsc{a} \vdash \mathsf{x}_i : \textsc{a}}\ \mathsf{id}$$

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

$$\frac{\Gamma \vdash \mathsf{s} : \textsc{a} \quad \Delta \vdash \mathsf{t} : \textsc{a}\backslash\textsc{b}}{(\Gamma \cdot \Delta) \vdash \mathsf{s} \blacktriangleright \mathsf{t} : \textsc{b}}\ \backslash E \qquad \frac{(\mathsf{x}_i : \textsc{a} \cdot \Gamma) \vdash \mathsf{s} : \textsc{b}}{\Gamma \vdash \lambda\!\!\backslash \mathsf{x}_i.\mathsf{s} : \textsc{a}\backslash\textsc{b}}\ \backslash I$$

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

Figure I.14: Non-associative Lambek calculus **NL**.

to:

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

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.14 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, $\textsc{a}$ and $\textsc{b}$ 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 intuitively translated to a constraint on the applicability of abstraction. Namely, the variable to abstract over needs to occur at the top-most level of a function application in the term's inductive

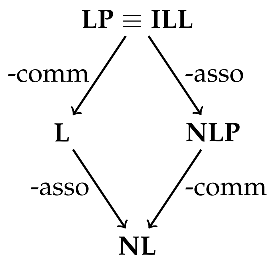


Figure I.15: **(N)L(P)**: **ILL** and substructural friends.

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.

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 accu-

[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).

mulated 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 $(\_\cdot\_)$ cast 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**. In the case for checking the satisfiability of a judgement (i.e. searching for *any* valid proofs), 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.[1] What if there was a way to keep our logic computationally tractable but with temporary, on-demand access to normally excluded reasoning tools?

---

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

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

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

Figure I.16: Logical rules of modal inference.

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 $\Box$.[1]

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.16. The presentation is intentionally detached from a specific substructural strand – modalities are plug-and-play to any member of the **(N)L(P)** family. Their incorporation adds a new kind of structure to the ones provided by the underlying logic, altering judgements accordingly:

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

Angular brackets denote unary tree branches that behave slightly differ-

[1] Note that these are not the necessity/possibility duals of modal logic, but rather *inverse* duals, $\Box$ being *past* necessity and $\Diamond$ being *future* possibility. Read (I.24) as "what is, has always been bound to be" and (I.25) as "what will have always been, is".

ent to the rest; they act as an impenetrable barrier that permits or hinders the introduction or elimination of modal connectives in a judgement. The box elimination rule $\Box E$ grants us the option of removing a logical box from the succedent of the premise (as long as it is its main connective), but encloses the premises in angular brackets in the process. Its introduction counterpart $\Box I$ does the exact opposite: it frees a judgement's assumptions from their brackets, but puts the succedent proposition under the scope of a box. The diamond behaves just the other way around. Its introduction rule $\Diamond I$ is straightforward: it offers the possibility of putting the succedent under the scope of a diamond, in exchange wrapping the antecedents with brackets. The elimination rule $\Diamond E$ is more of a problem child, behaving akin to a unary product. Without locality restrictions, it inspects a proof of $\textsc{b}$, the assumptions of which contain a substructure $\langle \textsc{a} \rangle$ 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 $\Diamond\textsc{a}$.

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 $\langle \textsc{a} \rangle$ 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. As for what the computational interpretation of these esoteric term rewrites is, our insights are limited to the fact they form a residuated pair that respects the normalizations prescribed by the typing rules.

4.1.1 Proof & Term Reductions

The proof patterns of Figure I.17 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 $\textsc{a}$ hypothesized in the other branch s. On the term level, these correspond to normalizations:

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

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

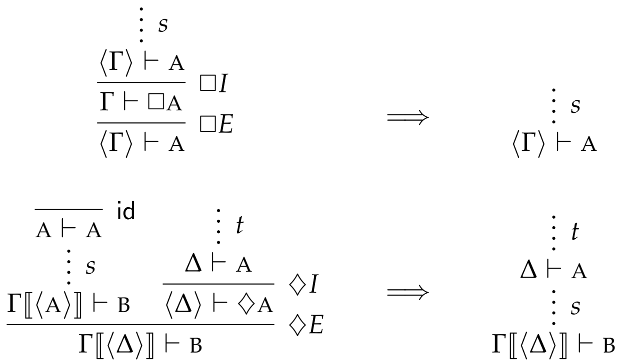


Figure I.17: Modal β redexes.

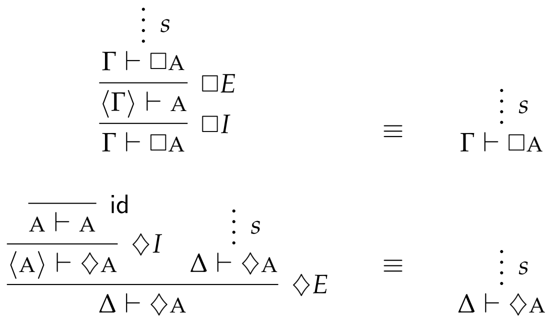


Figure I.18: Modal η redexes.

The dual direction of η equivalences also holds – since these are discovered in the literature less frequently than the more pedestrian implication and product equivalences, we explictly present them in Figure I.18. 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:

$$\frac{\Gamma[\![\langle \mathsf{x}_i : \textsc{a} \rangle]\!] \vdash \mathsf{s} : \textsc{b} \quad \Delta \vdash \mathsf{t} : \Diamond\textsc{a}}{\Gamma[\![\Delta]\!] \vdash \mathsf{s}_{[\mathsf{x}_i \mapsto \triangledown \mathsf{t}]} : \textsc{b}} \; \Diamond E$$

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 the 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. 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.19 presents the proof transformations equivalent to the properties foretold: (a) and (b) for monotonicity, (c) and (d) for composition, and (e) and (f) for the two directions of the residuation law.

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

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

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

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.20 presents proofs of the above in both directions.

$$\dfrac{\dfrac{\begin{matrix}\vdots\; s\\ x_i : A \vdash s : B\end{matrix}}{\langle x_i : A\rangle \vdash \vartriangle s : \Diamond B}\,\Diamond I \qquad \dfrac{}{x_j : \Diamond A \vdash x_j : \Diamond A}\,\mathsf{id}}{x_j : \Diamond A \vdash \text{case } \triangledown x_j \text{ of } x_i \text{ in } s : \Diamond B}\,\Diamond E$$

(a) Monotonicity of the diamond.

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots\; s\\ x_i : A \vdash s : B\end{matrix}}{\vdash \lambda x_i.s : A \multimap B}\,{\multimap} I \qquad \dfrac{\dfrac{}{x_j : \Box A \vdash x_j : \Box A}\,\mathsf{id}}{\langle x_j : \Box A\rangle \vdash \blacktriangledown x_j : A}\,\Box E}{\langle x_j : \Box A\rangle \vdash (\lambda x_i.s)\ \blacktriangledown x_j : B}\,{\multimap} E}{x_j : \Box A \vdash \blacktriangle(\lambda x_i.s)\ \blacktriangledown x_j : \Box B}\,\Box I$$

(b) Monotonicity of the box.

$$\dfrac{\dfrac{\begin{matrix}\vdots\; s\\ \Gamma \vdash s : A\end{matrix}}{\langle \Gamma\rangle \vdash \vartriangle s : \Diamond A}\,\Diamond I}{\Gamma \vdash \blacktriangle\vartriangle s : \Box\Diamond A}\,\Box I$$

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

$$\dfrac{\dfrac{\dfrac{}{x_i : \Box A \vdash x_i : \Box A}\,\mathsf{id}}{\langle x_i : \Box A\rangle \vdash \blacktriangledown x_i : A}\,\Box E \qquad \begin{matrix}\vdots\; s\\ \Gamma \vdash s : \Diamond\Box A\end{matrix}}{\Gamma \vdash \text{case } \triangledown s \text{ of } x_i \text{ in } \blacktriangledown x_i : A}\,\Diamond E$$

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

$$\dfrac{\dfrac{\begin{matrix}\vdots\; s\\ x_i : A \vdash s : \Box B\end{matrix}}{\langle x_i : A\rangle \vdash \blacktriangledown s : B}\,\Box E \qquad \dfrac{}{x_j : \Diamond A \vdash x_j : \Diamond A}\,\mathsf{id}}{x_j : \Diamond A \vdash \text{case } \triangledown x_j \text{ of } x_i \text{ in } s : B}\,\Diamond E$$

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

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots\; s\\ x_i : \Diamond A \vdash s : B\end{matrix}}{\vdash \lambda x_i.s : \Diamond A \multimap B}\,{\multimap} I \qquad \dfrac{\dfrac{}{x_j : A \vdash x_j : A}\,\mathsf{id}}{\langle x_j : A\rangle \vdash \vartriangle x_j : \Diamond A}\,\Diamond I}{\langle x_j : A\rangle \vdash \text{case } \triangledown\vartriangle x_j \text{ of } x_i \text{ in } \lambda x_i.s : B}\,\Diamond E}{x_j : \Diamond A \vdash \blacktriangle(\text{case } \triangledown\vartriangle x_j \text{ of } x_i \text{ in } \lambda x_i.s) : B}\,\Box I$$

(f) Ditto, the other way around.

Figure I.19: Derivations for the various aspects of residuation.

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

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

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

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

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

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

$$
\dfrac{
\dfrac{\dfrac{\text{(I.19c)}}{x_i : A \vdash \blacktriangle\triangle x_i : \Box\Diamond A}}{\langle x_i : A \rangle \vdash \triangle\blacktriangle\triangle x_i : \Diamond\Box\Diamond A}\,\Diamond I
\qquad
\dfrac{}{x_j : \Diamond A \vdash x_j : \Diamond A}\,\text{id}
}{x_j : \Diamond A \vdash \text{case } \triangledown x_j \text{ of } x_i \text{ in } \triangle\blacktriangle\triangle x_i : \Diamond\Box\Diamond A}\,\Diamond E
$$

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

Figure I.20: The triple laws for the two modalities in both directions.

4.2 Structural Reasoning

This detour may have proven lengthy, but has hopefully helped us acquire a first taste for modalities. We now know how to introduce and eliminate them and what the effect of doing so is on the antecedent structure, and got a first glimpse of their properties, the term rewrites they prescribe and the type inequalities (in the form of unidirectional derivations) they give rise to. The question then becomes how to actually use them for the task at hand, namely disciplined traversal between substructural logics. Structural reasoning is accomplished via structural postulates, rules of inference that enact commutativity and associativity (or combinations thereof), except in a controlled fashion. These are permissible only under strict conditions on the *shape* of the antecedent structure and its constituents – 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.

5 The Linguistic Perspective

Despite their presentation having intentionally been left vague and abstract, the ideas explored so far have been a keystone element of computer science, from its inception until recent modernity. Beyond that, they form the common theoretical underpinnings for the formal treatment of natural languages and their various aspects, where they manifest as so-called *Categorial Grammars*. Categorial grammars is a heavily overloaded term that refers to a wide and diverse family of related formalisms, each with its own ambitions, goals, strengths and weaknesses. The most encompassing way of defining a categorial grammar is thus best accomplished through a high-level intersection of their common points. A categorial grammar is 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

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

Figure I.21: The Curry-Howard correspondence applied in linguistics.

be developed and presented later in this thesis.

5.1 Type-Logical Grammars

The earliest take at a categorial grammar are the AB grammars attributed to Kazimierz Adjukiewicz [Ajdukiewicz, 1935] and Yehoshua Bar-Hillel [Bar-Hillel, 1953], but it was Jim Lambek that raised the existing notation and operations into the glory of a fully-fledged type logic. 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 Figure I.21; 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:

$$\mathsf{Prop}_0 := \{\textsc{n}, \textsc{np}, \textsc{s}_{\text{main}}, \textsc{pp}\}$$

for a grammar able to reason about nouns $\textsc{n}$, noun phrases and bare nouns $\textsc{np}$, sentential clauses $\textsc{s}_{\text{main}}$, prepositional phrases $\textsc{pp}$ and functions thereof in English. One might wonder: what happened to the remaining kinds of

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	::	$\text{ADV}_\backslash$	:= (NP\S)\(NP\S)
never	::	$\text{ADV}_/$	:= (NP\S)/(NP\S)
litten	::	(NP\NP)/PP	
may	::	AUX	:= (NP\S)/(NP\S)

Table I.2: Toy lovecraftian lexicon of pure Lambek types.

phrasal categories, like verbs, adjectives and adverbs? These would indicate grammatical functions, and in fact should be represented as such. An intransitive phrase, for instance, is a grammatical function that would consume a left-adjacent noun phrase to produce a sentence, therefore it would materialize as $\text{NP}\backslash\text{S}_{\text{main}}$. It follows that a transitive phrase or copula would then be of type $(\text{NP}\backslash\text{S}_{\text{main}})/\text{NP}$, a function that requires a right-adjacent noun phrase to produce an intransitive, whereas a bitransitive, requiring two, would be $((\text{NP}\backslash\text{S}_{\text{main}})/\text{NP})/\text{NP}$, etc. In the same vein, determiner phrases NP/N consume right-adjacent nouns and lift them to noun phrases, whereas prenominal adjectives NP/NP are noun phrase (or noun) endomorphisms modifying them but keeping their type intact (and the other way around for postnominal use). Adverbs would also be endomorphisms, except this time higher-order – $(\text{NP}/\text{NP})/(\text{NP}/\text{NP})$ for adjectival and $(\text{NP}\backslash\text{S})\backslash(\text{NP}\backslash\text{S})$ 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,

$$\dfrac{\dfrac{}{\text{strange} \vdash \textsc{np}/\textsc{np}}\,\mathsf{lex} \qquad \dfrac{}{\text{dolphins} \vdash \textsc{np}}\,\mathsf{lex}}{\text{strange} \cdot \text{dolphins} \vdash \textsc{np}}\,/E$$

(a) Derivation for *strange dolphins*.

$$\dfrac{\dfrac{}{\text{the} \vdash \textsc{np}/\textsc{n}}\,\mathsf{lex} \qquad \dfrac{}{\text{eye} \vdash \textsc{n}}\,\mathsf{lex}}{\text{the} \cdot \text{eye} \vdash \textsc{np}}\,/E$$

(b) Derivation for *the eye*.

$$\dfrac{\dfrac{}{\text{litten} : (\textsc{np}\backslash\textsc{np})/\textsc{pp}}\,\mathsf{lex} \qquad \dfrac{\dfrac{}{\text{by} : \textsc{pp}/\textsc{np}}\,\mathsf{lex} \qquad \dfrac{}{\text{suns} : \textsc{np}}\,\mathsf{lex}}{\text{by} \cdot \text{suns} \vdash \textsc{pp}}\,/E}{\text{litten} \cdot (\text{by} \cdot \text{suns}) \vdash \textsc{np}\backslash\textsc{np}}\,/E$$

(c) Derivation for *litten by suns*.

$$\dfrac{\dfrac{}{\text{sea-nymphs} : \textsc{np}}\,\mathsf{lex} \qquad \dfrac{\dfrac{}{\text{of} : (\textsc{np}\backslash\textsc{np})/\textsc{np}}\,\mathsf{lex} \qquad \dfrac{\dfrac{}{\text{unremeberable} : \textsc{np}/\textsc{np}}\,\mathsf{lex} \qquad \dfrac{}{\text{deeps} : \textsc{np}}\,\mathsf{lex}}{\text{unrememberable} \cdot \text{deeps} \vdash \textsc{np}}\,/E}{\text{of} \cdot (\text{unrememberable} \cdot \text{deeps}) \vdash \textsc{np}\backslash\textsc{np}}\,/E}{\text{sea-nymphs} \cdot (\text{of} \cdot (\text{unrememberable} \cdot \text{deeps})) \vdash \textsc{np}}\,\backslash E$$

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

$$\dfrac{\dfrac{\dfrac{}{\text{opiate} : \textsc{np}/\textsc{np}}\,\mathsf{lex} \qquad \dfrac{}{\text{oceans} : \textsc{np}}\,\mathsf{lex}}{\text{opiate} \cdot \text{oceans} \vdash \textsc{np}}\,/E \qquad \dfrac{\dfrac{}{\text{poured} : \textsc{np}\backslash\textsc{s}}\,\mathsf{lex} \qquad \dfrac{}{\text{there} : \textsc{adv}\backslash}\,\mathsf{lex}}{\text{poured} \cdot \text{there} \vdash \textsc{np}\backslash\textsc{s}}\,\backslash E}{(\text{opiate} \cdot \text{oceans}) \cdot (\text{poured} \cdot \text{there}) \vdash \textsc{s}}\,\backslash E$$

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

Figure I.22: Deriving simple multiplicative phrases in **NL**.

> *and having in their whirlpools strange dolphins and sea-nymphs of unrememberable deeps.*
>
> H.P. Lovecraft, *Azathoth* (1938). In *Leaves (2).*

Let's pave the way towards this ambitious goal with the miniature mock-up lexicon of Table I.2, and see just how far it can get us.

Figure I.22 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.22e and I.22c 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.

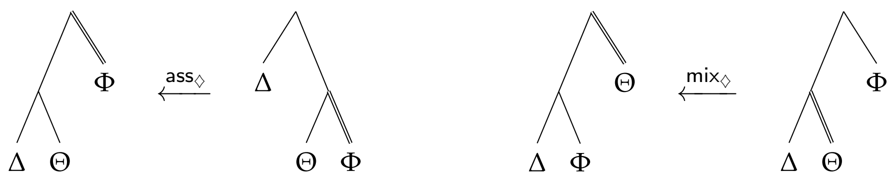
One could of course attept to cop out of the problem by just introducing ad hoc raised forms for movable parts, one per distinct position they can be found at. The repercussions of such a move would soon, however, prove catastrophic. On the one hand, the once reliably concise lexicon would become overpopulated 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 (at least not for the formal syntactician). 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, modalities have played an indispensable role in the history and development of type-logical grammars [Hendriks, 1995; Moortgat, 1996; Kurtonina and

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

(a) In rule format.



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

Figure I.23: Controlled associativity/mixed commutativity.

Moortgat, 1997; Moortgat, 1997; Vermaat, 1999]. They find use as either licensors or inhibitors of structural rewrites, now in the form of movement and rebracketing of words and phrases. Prime examples and standard items for consideration include the controlled associativity and mixed associativity-commutativity rules of Figure I.23a (and the corresponding tree transformations of Figure I.23b, if you have a disdain for brackets). The first rule $\mathsf{ass}_{\Diamond}$ 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 $\mathsf{mix}_{\Diamond}$ allows a unary $\langle\Theta\rangle$ to swap position with its right-adjacent neighbor Φ, disassociating from its left neighbour Δ in the process.

Figure I.25 progresses our agenda by accounting for the presence of a (hypothetical) movable postnominal modifier via the rules of Figure I.23. 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 $\mathsf{mix}_{\Diamond}$ 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 sought-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.24 for a *proof* of concept.[1]

This methodology is in fact adopted from Moortgat [1999], where it finds

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\text{opiate}\cdot\text{oceans}\vdash \textsc{np}} \quad \dfrac{\dfrac{}{\mathsf{x}_i : \Box(\textsc{np}\backslash\textsc{np})}\,\mathsf{id}}{\langle \mathsf{x}_i\rangle \vdash \textsc{np}\backslash\textsc{np}}\,\Box E}{(\text{opiate}\cdot\text{oceans})\cdot\langle \mathsf{x}_i\rangle\vdash \textsc{np}}\,\backslash E \quad \dfrac{\vdots}{\text{poured}\cdot\text{there}\vdash \textsc{np}\backslash\textsc{s}}}{((\text{opiate}\cdot\text{oceans})\cdot\langle \mathsf{x}_i\rangle)\cdot(\text{poured}\cdot\text{there})\vdash \textsc{s}}\,\backslash E}{((\text{opiate}\cdot\text{oceans})\cdot(\text{poured}\cdot\text{there}))\cdot\langle \mathsf{x}_i\rangle\vdash \textsc{s}}\,\mathsf{mix}_\Diamond$$

(a) Extracting a hypothetical postnominal modifier...

$$\dfrac{\dfrac{\text{(I.24a)}}{((\ldots)\cdot(\ldots))\cdot\langle \mathsf{x}_i\rangle\vdash \textsc{s}} \quad \dfrac{\dfrac{}{\text{litten} : \Diamond\Box(\textsc{np}\backslash\textsc{np})/\textsc{pp}}\,\mathsf{lex} \quad \dfrac{\dfrac{}{\text{by} : \textsc{pp}/\textsc{np}}\,\mathsf{lex} \quad \dfrac{}{\text{suns} : \textsc{np}}\,\mathsf{lex}}{\text{by}\cdot\text{suns}\vdash \textsc{pp}}\,/E}{\text{litten}\cdot(\text{by}\cdot\text{suns})\vdash \Diamond\Box(\textsc{np}\backslash\textsc{np})}\,/E}{((\text{opiate}\cdot\text{oceans})\cdot(\text{poured}\cdot\text{there}))\cdot(\text{litten}\cdot(\text{by}\cdot\text{suns}))\vdash \textsc{s}}\,\Diamond E$$

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

Figure I.24: Deriving long-distance postnominal modification with the aid of type assignment (I.33).

similar use in dealing with the grammatical ambivalence of relativizers like *that* or *which*. Bound relative clauses headed by complementizers like the above contain a subordinate sentence with a *gap*, which can vary in its position. Let's make things unnecessarily convoluted for the sake of clichéd self-referentialism by considering the relative clause *which can vary 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:

$$\text{that} :: \textsc{rel}_\text{s} := (\textsc{np}\backslash\textsc{np})/(\textsc{np}\backslash\textsc{s}) \qquad \text{(I.34)}$$

$$\text{that} :: \textsc{rel}_\text{o} := (\textsc{np}\backslash\textsc{np})/(\textsc{s}/\Diamond\Box\textsc{np}) \qquad \text{(I.35)}$$

The second version launches a mobile NP hypothesis via the same diamond-box pattern showcased earlier. The proof of Figure I.25 employs this typing in combination with the $\mathsf{ass}_\Diamond$ 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.24 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

$$
\begin{prooftree}
\AxiomC{$\mathsf{that} : \textsc{rel}_o$}
\RightLabel{lex}
\UnaryInfC{}
\AxiomC{(I.22b)}
\UnaryInfC{$\mathsf{the} \cdot \mathsf{eye} \vdash \textsc{np}$}
\AxiomC{}
\RightLabel{lex}
\UnaryInfC{$\mathsf{may} : \textsc{aux}$}
\AxiomC{}
\RightLabel{lex}
\UnaryInfC{$\mathsf{never} : \textsc{adv}_{/}$}
\AxiomC{}
\RightLabel{lex}
\UnaryInfC{$\mathsf{behold} : \textsc{tv}$}
\AxiomC{}
\RightLabel{id}
\UnaryInfC{$\mathsf{x}_i : \Box\textsc{np}$}
\RightLabel{$\Box E$}
\UnaryInfC{$\langle \mathsf{x}_i \rangle \vdash \textsc{np}$}
\RightLabel{$/E$}
\BinaryInfC{$\mathsf{behold} \cdot \langle \mathsf{x}_i \rangle \vdash \textsc{np}\backslash\textsc{s}$}
\RightLabel{$/E$}
\BinaryInfC{$\mathsf{never} \cdot (\mathsf{behold} \cdot \langle \mathsf{x}_i \rangle) \vdash \textsc{np}\backslash\textsc{s}$}
\RightLabel{$/E$}
\BinaryInfC{$\mathsf{may} \cdot (\mathsf{never} \cdot (\mathsf{behold} \cdot \langle \mathsf{x}_i \rangle)) \vdash \textsc{np}\backslash\textsc{s}$}
\RightLabel{$\backslash E$}
\BinaryInfC{$(\mathsf{the} \cdot \mathsf{eye}) \cdot (\mathsf{may} \cdot (\mathsf{never} \cdot (\mathsf{behold} \cdot \langle \mathsf{x}_i \rangle))) \vdash \textsc{s}$}
\RightLabel{$\mathsf{ass}_\Diamond$}
\UnaryInfC{$(\mathsf{the} \cdot \mathsf{eye}) \cdot (\mathsf{may} \cdot ((\mathsf{never} \cdot \mathsf{behold}) \cdot \langle \mathsf{x}_i \rangle)) \vdash \textsc{s}$}
\RightLabel{$\mathsf{ass}_\Diamond$}
\UnaryInfC{$(\mathsf{the} \cdot \mathsf{eye}) \cdot ((\mathsf{may} \cdot (\mathsf{never} \cdot \mathsf{behold})) \cdot \langle \mathsf{x}_i \rangle) \vdash \textsc{s}$}
\RightLabel{$\mathsf{ass}_\Diamond$}
\UnaryInfC{$((\mathsf{the} \cdot \mathsf{eye}) \cdot (\mathsf{may} \cdot (\mathsf{never} \cdot \mathsf{behold}))) \cdot \langle \mathsf{x}_i \rangle \vdash \textsc{s}$}
\AxiomC{}
\RightLabel{id}
\UnaryInfC{$\mathsf{x}_j : \Diamond\Box\textsc{np}$}
\RightLabel{$\Diamond E$}
\BinaryInfC{$((\mathsf{the} \cdot \mathsf{eye}) \cdot (\mathsf{may} \cdot (\mathsf{never} \cdot \mathsf{behold}))) \cdot \mathsf{x}_j \vdash \textsc{s}$}
\RightLabel{$/I$}
\UnaryInfC{$(\mathsf{the} \cdot \mathsf{eye}) \cdot (\mathsf{may} \cdot (\mathsf{never} \cdot \mathsf{behold})) \vdash \textsc{s}/\Diamond\Box\textsc{np}$}
\RightLabel{$/E$}
\BinaryInfC{$\gamma := \mathsf{that} \cdot ((\mathsf{the} \cdot \mathsf{eye}) \cdot (\mathsf{may} \cdot (\mathsf{never} \cdot \mathsf{behold}))) \vdash \textsc{np}\backslash\textsc{np}$}
\end{prooftree}
$$

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

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{lex}
\UnaryInfC{$\mathsf{litten} : \Diamond\Box(\textsc{np}\backslash\textsc{np})/\textsc{pp}$}
\AxiomC{}
\RightLabel{lex}
\UnaryInfC{$\mathsf{by} : \textsc{pp}/\textsc{np}$}
\AxiomC{}
\RightLabel{lex}
\UnaryInfC{$\mathsf{suns} : \textsc{np}$}
\AxiomC{(I.25a)}
\UnaryInfC{$\gamma \vdash \textsc{np}\backslash\textsc{np}$}
\RightLabel{$\backslash E$}
\BinaryInfC{$\mathsf{suns} \cdot \gamma \vdash \textsc{np}$}
\RightLabel{$/E$}
\BinaryInfC{$\mathsf{by} \cdot (\mathsf{suns} \cdot \gamma) \vdash \textsc{pp}$}
\RightLabel{$/E$}
\BinaryInfC{$\mathsf{litten} \cdot (\mathsf{by} \cdot (\mathsf{suns} \cdot (\mathsf{that} \cdot ((\mathsf{the} \cdot \mathsf{eye}) \cdot (\mathsf{may} \cdot (\mathsf{never} \cdot \mathsf{behold})))))) \vdash \Diamond\Box(\textsc{np}\backslash\textsc{np})$}
\end{prooftree}
$$

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

Figure I.25: An object-relative clause in action, prompted by type assignment (I.35).

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 necessarily expressed with a distinct type. In our case, we need the type:

$$\text{having} :: (\Diamond\Box(\textsc{np}\backslash\textsc{np})/\textsc{np})/\textsc{pp} \tag{I.36}$$

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 postnominal 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:

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

and the subscript enumerating each of the possible instantiations in the context of a single sentence. In such a 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

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

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.[1]

Armed with this last trick, we are now in possession of all the knowledge necessary to finally tackle the full derivation. First, we must instantiate the polymorphic coordinator once by substituting χ for NP to derive the noun phrase conjunction *strange dolphins and sea-nymphs of unremememberable deeps*, as portrayed in Figure I.26a. 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.26b. At this point, we must employ another instance of the polymorphic coordinator, this time substituting χ for $\Diamond\Box(\textsc{np}\backslash\textsc{np})$ – 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.24. 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.

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.26a:

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

---

[1]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].

$$\frac{\Gamma, \alpha : \textsc{Type} \vdash \mathsf{M} : \sigma}{\Gamma \vdash \lambda\alpha.\mathsf{M} : \Pi\alpha.\sigma}\ \Pi I \qquad \frac{\Gamma \vdash \mathsf{M} : \Pi\alpha.\sigma \quad \Delta \vdash \mathsf{B} : \textsc{Type}}{\Gamma, \Delta \vdash \mathsf{M}\ \textsc{b} : \sigma_{[\alpha \mapsto \textsc{b}]}}\ \Pi E$$

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 $(\textsc{a}\backslash\textsc{a})/\textsc{a}$. 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.

$$\dfrac{\dfrac{\text{(I.22a)}}{\text{strange}\cdot\text{dolphins}\vdash \textsc{np}} \quad \dfrac{\dfrac{}{\text{and} : (\textsc{np}\backslash\textsc{np})/\textsc{np}}\,\text{lex} \quad \dfrac{\text{(I.22d)}}{\text{sea-nymphs}\cdot(\text{of}\cdot(\text{unrememberable}\cdot\text{deeps}))\vdash \textsc{np}}}{\text{and}\cdot(\text{sea-nymphs}\cdot(\text{of}\cdot(\text{unrememberable}\cdot\text{deeps})))\vdash \textsc{np}\backslash\textsc{np}}\,/E}{\delta := (\text{strange}\cdot\text{dolphins})\cdot(\text{and}\cdot(\text{sea-nymphs}\cdot(\text{of}\cdot(\text{unrememberable}\cdot\text{deeps}))))\vdash \textsc{np}}\,\backslash E$$

(a) Deriving noun-phrase coordination...

$$\dfrac{\dfrac{\dfrac{}{\text{having} : (\Diamond\Box(\textsc{np}\backslash\textsc{np})/\textsc{np})/\textsc{pp}}\,\text{lex} \quad \dfrac{\dfrac{}{\text{in} : \textsc{pp}/\textsc{np}}\,\text{lex} \quad \dfrac{\dfrac{}{\text{their} : \textsc{np}/\textsc{np}}\,\text{lex} \quad \dfrac{}{\text{whirlpools} : \textsc{np}}\,\text{lex}}{\text{their}\cdot\text{whirlpools}\vdash \textsc{np}}\,/E}{\text{in}\cdot(\text{their}\cdot\text{whirlpools})\vdash \textsc{pp}}\,/E}{\text{having}\cdot(\text{in}\cdot(\text{their}\cdot\text{whirlpools}))\vdash \Diamond\Box(\textsc{np}\backslash\textsc{np})/\textsc{np}}\,/E \quad \dfrac{\text{(I.26a)}}{\delta\vdash \textsc{np}}}{(\text{having}\cdot(\text{in}\cdot(\text{their}\cdot\text{whirlpools}))\cdot\delta)\vdash \Diamond\Box(\textsc{np}\backslash\textsc{np})}\,/E$$

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

Figure I.26: Filling in the missing bits using the polymorphic type (I.38).

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 bedtime:

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

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

Montague's Insights A full exposition to Montague grammar is beyond the scope of this thesis, but a brief introduction to some of its foundations will go a long way in helping us perceive its relevance to the type-logical approach. Richard Montague was disillusioned with the tackling of natural language semantics at the time, which he found formally inadequate and lacking the elegance of contemporary approaches to mathematical syntax. He sought 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 attributable 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

$$\begin{array}{ccccc} \text{Surface Syntax} & & \text{Derivational Semantics} & & \text{Target Semantics} \\ \Sigma := \mathbf{NL}_{\diamondsuit,\square} & \xrightarrow{\lceil.\rceil} & \mathrm{T} := \mathbf{ILL}_{\multimap} & \cdots\cdots\rightarrow & ? \end{array}$$

Figure I.27: The syntax-semantics interface in the type-logical setting.

the details of the translations. The only constraint imposed is the only one that matters: the high-level principle of compositionality needs to hold, i.e. the function-argument structure specified by syntax needs to be carried through to 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 $\mathbf{NL}_{\diamondsuit,\square}$ 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 base types, 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 $\eta : \mathcal{U}^\Sigma \rightarrow \mathcal{U}^\mathrm{T}$ and $\theta : \mathsf{Terms}^\Sigma \rightarrow \mathsf{Terms}^\mathrm{T}$, 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 base types 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.3. 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 $\diamondsuit E$ rule, as shown in Table I.4. As an example, applying $\lceil.\rceil$ to the proof of Figure I.25 should yield the derivational term:

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

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

Table I.3: Translating $\mathbf{NL}_{\Diamond,\Box}$ types to $\mathbf{ILL}_{\multimap}$.

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

Table I.4: 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 thought 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 a quantum (!) interpretation of the control modalities.

The Role of the Lexicon The sentiments of the previous paragraph could be met with some skepticism. A reader with 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 semantic black boxes. 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 obey 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 from 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.9, so as to allow the iteration of the intersective modifiers:

$$\lambda \mathsf{x}_i \mathsf{x}_j \mathsf{x}_k . \mathsf{x}_i \ (\mathsf{x}_j \ \mathsf{x}_k) :: (* \multimap *) \multimap (* \multimap *) \multimap * \multimap * \tag{I.40}$$

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 $\wedge : * \rightarrow * \rightarrow *$ for set-theoretic intersection ($*$ now an arbitrary set), to deliver the recipe:

$$\lambda \mathsf{x}_i \mathsf{x}_j \mathsf{x}_k . (\mathsf{x}_i \ \mathsf{x}_k) \wedge (\mathsf{x}_j \ \mathsf{x}_k) :: (* \rightarrow *) \rightarrow (* \rightarrow *) \rightarrow * \rightarrow * \tag{I.41}$$

[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.

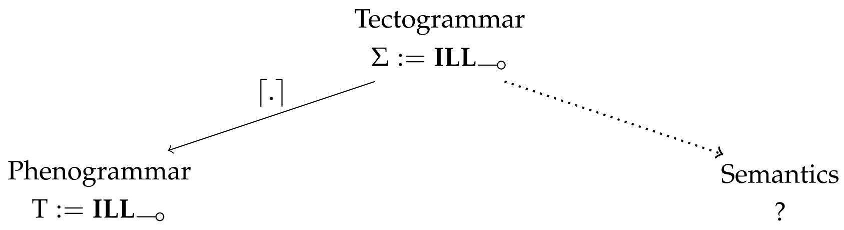


Figure I.28: The syntax-semantics interface in the abstract categorial setting.

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 formalisms make extensive use of $\mathbf{ILL}_{\multimap}$ and the Curry-Howard isomoprhism to obtain a phenogrammatic realizations via the homomorphic translation of a tectogrammatic parse. Our presentation will stick to the former, as they are closer in spirit to what is to come later.

5.2.1 Basic Definitions

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 instantiation 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 homomorphism $\lceil . \rceil$, 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 $p_d \mapsto \texttt{str}$, forcing terms of the object language to evaluate to strings. Note that, despite appearances, $\texttt{str}$ 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), with the identity function enacting the empty string.

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 **(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.29 by all abstract constants denoting base "types"[2] and "type" constructors. The target logic T will be our phenogrammatic printer tasked with translating abstract terms ("types") to object terms (strings). We will need a single object base type $\mathsf{Prop}_0^{\mathrm{T}} := \{*\}$, such that $\eta_0(\textsc{Type}) = \texttt{str}$, the type alias of $* \multimap *$. Some auxiliary object constants are necessary before we

[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.29: Abstract lexicon for the language of $\mathbf{(N)L}_{\Diamond,\Box}$ 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.5: Object translation for the lexicon of Figure I.29.

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.5: 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.30 presents the construction of the type previously assigned to *litten*, $\Diamond\Box(\textsc{np}\backslash\textsc{np})/\textsc{pp}$ (contexts are intentionally left empty and axioms re-

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

Figure I.30: Constructing the type assignment of (I.33).

placed by abstract constants for brevity). Applying the homomorphic translation to its abstract term yields a printout in the form of the object term below (source function-argument brackets substituted with indendation levels for legibility):

$$
\begin{aligned}
&\lceil \mathsf{rdiv}\ (\mathsf{dia\ box}\ (\mathsf{ldiv\ np\ np})))\ \mathsf{pp} \rceil \\
= {} & \lambda \mathsf{x}_i \mathsf{x}_j . \underline{(} + \mathsf{x}_j + \underline{/} + \mathsf{x}_i + \underline{)} \\
&\quad \underline{\text{PP}} \\
&\quad \lambda \mathsf{x}_k . \underline{\diamondsuit} + \mathsf{x}_k \\
&\qquad \lambda \mathsf{x}_l . \underline{\square} + \mathsf{x}_l \\
&\qquad\quad \lambda \mathsf{x}_m \mathsf{x}_n . \underline{(} + \mathsf{x}_m + \underline{\backslash} + \mathsf{x}_n + \underline{)} \\
&\qquad\qquad \underline{\text{NP}} \\
&\qquad\qquad \underline{\text{NP}} \\
\overset{\beta *}{\rightsquigarrow} {} & \underline{(\diamondsuit\square(\text{NP}\backslash\text{NP})/\text{PP})}
\end{aligned} \tag{I.42}
$$

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), the subset that encapsulates context-free grammars.[1]

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 otherwise simply be 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

[1] Or (2,2) if we set sail from $\mathcal{O}(p) = 1$ for the base case $p \in \mathsf{Prop}_0$ in (I.9), as commonly done in abstract categorial grammar literature.

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 aliases in the source domain, outsourcing their expansion to a parallel grammar for concrete semantics – though this is less of a solution and more of a deferral. Beyond practicality, there are also foundational issues 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.

The appealing simplicity and elegance of the tectogrammatic logic is therefore 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. 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 Chapter IV.

5.3 Alternatives

Type-logical and abstract categorial grammars have so far monopolized our interest, but not due to an absence of alternatives. The extended type-logical family includes relatives like the displacement calculus [Morrill et al., 2011] and hybrid type-logical grammars [Kubota and Levine, 2012]; each offers a unique perspective in treatment in the tackling of discontinuities, but neither concerns us much here, for our agenda is different.

Further away from the radiant warmth of type theory, we find a deviant from the categorial tradition in 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 with uninteresting semantics. To circumvent the problem, 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, allowing types to be raised once before being administered to the derivation, thus forcing a flip in the local function-argument structure and semantic scope. 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 theory, but ad hoc schemata, fixed a priori to fit their designated purpose. To counteract overgeneration, these rules are to be made available only to some empirically defined subset of the full lexicon. 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. Unlike before, 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 type-logical grammars and combinatory categorial ones are aligned, at least at a high level: they both stipulate the existence of syntactic universals that guide structure formation, utilize them as a pathway to semantics à la Montague, and acknowledge the need for language-specific fine-tuning; one exercising proof-theoretic control via 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 relatives – to the point where the term categorial grammars has become an almost synonym of combinatory categorial grammars. I hope that, by its end, this thesis will have slightly adjusted the scales towards a healthier epistemological pluralism.

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 [1992] 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 I 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.

P. De Groote and C. Retoré. On the semantic readings of proof-nets. In *Formal grammar 1996*, pages 57–70. FoLLI, 1996.

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 Preußische 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.

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

F. Lamarche. Proof nets for intuitionistic linear logic: Essential nets, 2008.

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. 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*. Kluwer Academic Publishers, 1994.

G. Morrill, O. Valentín, and M. Fadda. The displacement calculus. *Journal of Logic, Language and Information*, 20(1):1–48, 2011.

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. *CSLI Lecture Notes 29*, 3, 1992.

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 (spoiler: 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.

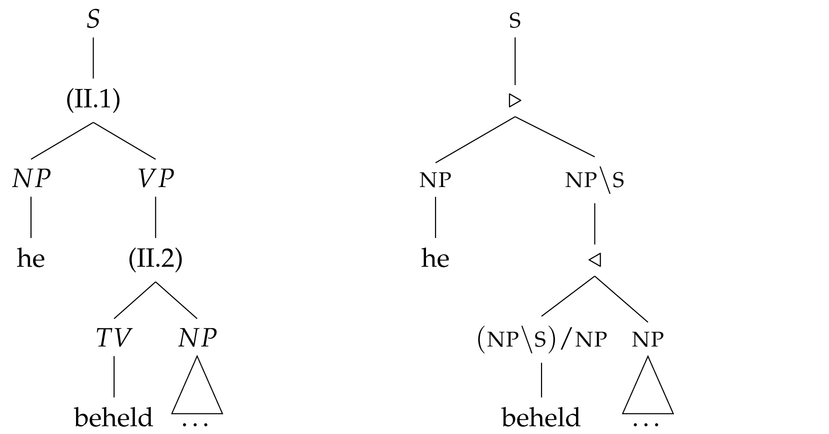
7.1 Phrase Structure Grammars

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, at least for the purposes of phrasal composition. Phrases are inventorized according to their syntactic *categories*. If one so wishes, one 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. he · (beheld · (the · city))
ii. he · (beheld · (the · ((glittering · minarets) · (of · (the · city)))))
iii. he · (beheld · ((such · beauty) · (of · ((red · (and · white)) · flowers))))
iv. he · (beheld · ((some · (feature · (or · arrangement))·(which · (he · ((had · known) · 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 discontinuity, 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 discontinuity, 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.22 to I.26 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 dominates 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 (UD) [de Marneffe et al., 2021], claiming a broad collection of multi-lingual treebanks [Nivre et al., 2020] and tools. In UD, words are usually assigned a label pulled from a rudimentary set of part of speech tags and lexical identifiers; 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] UD explicitly 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.

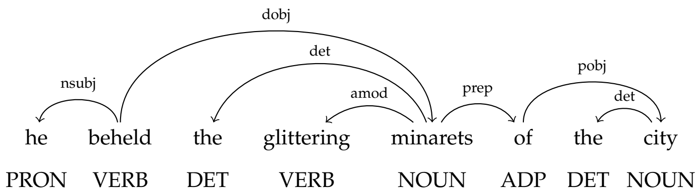


Figure II.2: A sample dependency parse in the universal dependencies format.

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 we could ever hypothesize between functional predicateness and syntactic headedness. 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 bidimensional 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.

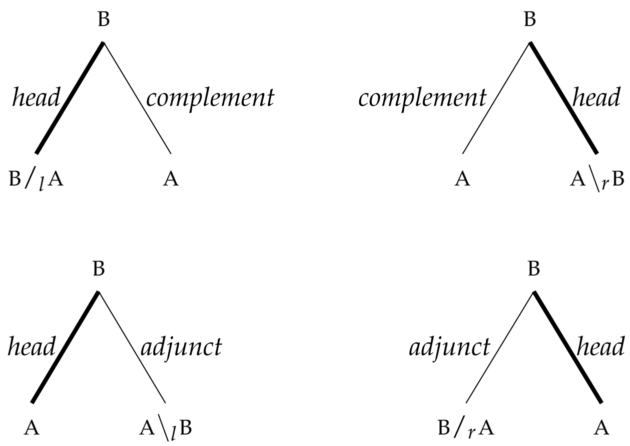


Figure II.3: The four implications of **DNL**.

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:

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

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

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.16 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 instead of the latter. The solution manifests itself 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.16, 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; type-checking 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 straightforward. 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 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

[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, 1976].

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 transformations, 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:

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

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 these 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.2, for instance, would still be typed as plain NPs – there's no telling in advance whether they will occur as subjects, direct objects 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.[1]

8.3.2 Dependencies & Structural Reasoning

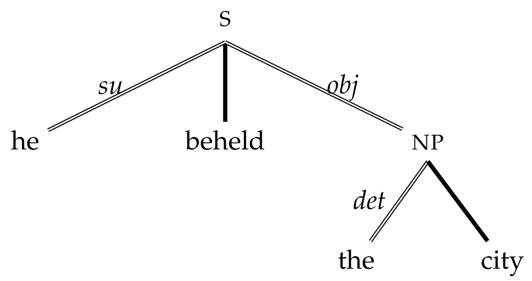
Lambek vs. Lambek (vs. van Benthem) The next thing to consider is how the inclusion of dependency modalities alters the structural core of each base logic. In any case, modalities induce a multi-labeled unary bracketing structure, inpenetrable by the the structural rules the core logic assigns to its structural binder. In that sense, they act akin to the *blocking* modalities of Morrill [1994]. 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 shallower and wider 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 **LP** – 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

[1] This is more of a serving suggestion – we won't really be using it anywhere.

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

(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 **LP** 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 crossroads 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:

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

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 unrelated to the choice of 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.25a (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 base types 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 $\text{ADJ}_{/}$ 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 when stripped of their modalities. This is perfectly in line with our agenda of revealing previ-

[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. $\diamondsuit_{su}\text{NP}\backslash\square_{cl}\text{S}$ 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	$:= \text{ITV}/\Diamond_{vc}\text{ITV}$

Table II.1: Dependency-enhanced lovecraftian lexicon.

ously 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.

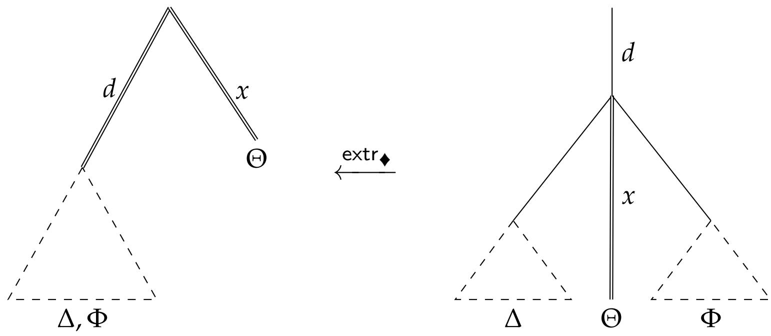
Naively, we would assume that the absence of binary bracketing structure in **L** would allow us direct access to the gap within the relative clause body, counteracting the need for the control modalities of (I.35). But the gap is still enclosed, except this time under layers of impenetrable dependency 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:

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

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

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

(a) Controlled extraction rule.



(b) Corresponding tree transformation.

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

is now *movable*.

Mobility, however, also means something different now. Taking inspiration from the established structural vocabulary (see Figure I.23 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 knowledge, 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 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

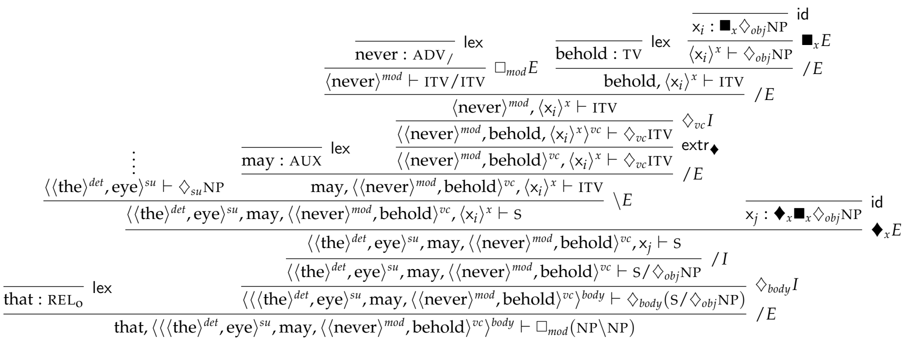


Figure II.6: Dependency-enhanced adaptation of the proof of Figure I.25a, exemplifying the interaction between dependency and control modalities.

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 **(N)L** and/or their binary brackets in **NL** still calls for different structural rules with possibly different control modalities that will need to coexist with the $\mathsf{extr}_{\blacklozenge}$ 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 instantiating 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 logical equivalent (wrapped under the necessary control brackets) via the $\Diamond E$ rule – consult Figure II.7 and contrast with Figure I.18 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

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

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

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 it, except this time it will be the interior combination of Figure I.19d that we are invoking, which is 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.22e, 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:

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

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 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 $\mathsf{extr}_{\blacklozenge}$

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

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

(a) Structurally freeing a hypothesized adjunct...

$$
\dfrac{\dfrac{}{\text{where} : \textsc{rel}_{\text{loc}}}\,\textsf{lex} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\text{sterile} : \textsc{adj}_{/}}\,\textsf{lex}}{\langle\text{sterile}\rangle^{mod} \vdash \textsc{np}/\textsc{np}}\,\Box_{mod}E \qquad \dfrac{}{\text{twilight} : \textsc{np}}\,\textsf{lex}}{\langle\text{sterile}\rangle^{mod}, \text{twilight} \vdash \textsc{np}}\,/E}{\langle\langle\text{sterile}\rangle^{mod}, \text{twilight}\rangle^{su} \vdash \Diamond_{su}\textsc{np}}\,\Diamond_{su}I \qquad \dfrac{\text{(II.8a)}}{\text{reigned}, \mathsf{x}_j \vdash \textsc{itv}}}{\langle\langle\text{sterile}\rangle^{mod}, \text{twilight}\rangle^{su}, \text{reigned}, \mathsf{x}_j \vdash \textsc{s}}\,\backslash E}{\langle\langle\text{sterile}\rangle^{mod}, \text{twilight}\rangle^{su}, \text{reigned} \vdash \textsc{s}/(\Diamond_{mod}\Box_{mod}(\textsc{itv}\backslash\textsc{itv}))}\,/I}{\langle\langle\langle\text{sterile}\rangle^{mod}, \text{twilight}\rangle^{su}, \text{reigned}\rangle^{body} \vdash \Diamond_{body}(\textsc{s}/(\Diamond_{mod}\Box_{mod}(\textsc{itv}\backslash\textsc{itv})))}\,\Diamond_{body}I}{\text{where}, \langle\langle\langle\text{sterile}\rangle^{mod}, \text{twilight}\rangle^{su}, \text{reigned}\rangle^{body} \vdash \Box_{mod}(\textsc{np}\backslash\textsc{np})}\,/E
$$

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

Figure II.8: Deriving a locative relative clause.

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.

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.

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{x_i : \Box_{mod}(\textsc{itv}\backslash\textsc{itv})}\,\mathsf{id}}{\langle x_i\rangle^{mod} \vdash \textsc{itv}\backslash\textsc{itv}}\,\Box_{mod}E \qquad \dfrac{\dfrac{}{x_j : \blacksquare_x \Diamond_{mod}\Box_{mod}(\textsc{itv}\backslash\textsc{itv})}\,\mathsf{id}}{\langle x_j\rangle^{x} \vdash \Diamond_{mod}\Box_{mod}(\textsc{itv}\backslash\textsc{itv})}\,\blacksquare_x E}{\langle x_j\rangle^{x} \vdash \textsc{itv}\backslash\textsc{itv}}\,\Diamond_{mod}E \\ \vdots\, s \\ \langle\Gamma\rangle^{d}, \langle x_j\rangle^{x} \vdash A \qquad \dfrac{}{x_k : \blacklozenge_x\blacksquare_x\Diamond_{mod}\Box_{mod}(\textsc{itv}\backslash\textsc{itv})}\,\mathsf{id}}{\dfrac{\langle\Gamma\rangle^{d}, x_k \vdash A}{\vdots}}\,\blacklozenge_x E
$$

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

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:

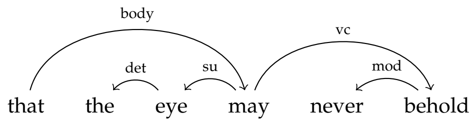
$$
\Gamma := \underbrace{\langle\Gamma_0\rangle^{d_0}\ldots\langle\Gamma_{n-1}\rangle^{d_{n-1}}}_{\text{adjuncts}}\ \underbrace{\langle\Gamma_n\rangle^{d_n}\ldots\langle\Gamma_{n+m-1}\rangle^{d_{n+m-1}}}_{\text{complements}}\ \underbrace{\kappa}_{\text{head}} \tag{II.13}
$$

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

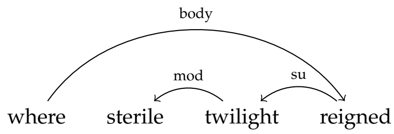
$$
\begin{gathered}
\mathsf{deptree} :: \mathsf{Struct} \to \mathsf{Head} \to \mathsf{Deps} \to \mathsf{Set}[\mathsf{Arc}] \\
\mathsf{deptree}\ \left(\langle\Gamma_0\rangle^{d_0}\ldots\langle\Gamma_{n+m-1}\rangle^{d_{n+m-1}}\ \kappa\right)\ \mathsf{root}\ \mathsf{label} = \\
\{\mathsf{root} \xrightarrow{\mathsf{label}}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

Figure II.10 shows the function's yield applied to two of the section's exam-

[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.



(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).

ple 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 superficial 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 and allowing for a stronger syntax-semantics interface – as for what these translations might be, quite a few reasonable paths are presented. *Monads* make for a natural translation choice, both because of their affinity with modalities [Kobayashi, 1997; Corfield, 2020] as well as their progressively maturing adoption by the linguistics community [Asudeh and Giorgolo, 2020]. 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, while still distinguishing between optional and necessary ones [Asudeh et al., 2012]. 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. [2023] 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 and craving 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 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.

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

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

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 in themselves the grammar's base types [Dekhtyar et al., 2015].

Chapter II Bibliography

A. Asudeh and G. Giorgolo. *Enriched meanings: Natural language semantics with category theory*, volume 13. Oxford University Press, 2020.

A. Asudeh, G. Giorgolo, M. Butt, and T. H. King. Flexible composition for optional and derived arguments. In *Proceedings of the LFG12 Conference*, pages 64–84, 2012.

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.

D. Corfield. *Modal homotopy type theory: The prospect of a new logic for philosophy*. Oxford University Press, 2020.

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. 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, 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. In R. Loukanova, P. LeFanu Lumsdaine, and R. Muskens, editors, *Logic and Algorithms in Computational Linguistics 2021 (LACompLing2021)*, volume 1081 of *Studies in Computational Intelligence SCI*. Springer, March 2023.

G. Morrill. *Type Logical Grammar: Categorial Logic of Signs*. Kluwer Academic Publishers, 1994.

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. To ascertain the model's utility without resorting to promises of theoretical universality, cognitive plausibility, linguistic intrinsicness and the like, we have two tried and tested ways to proceed. The first is the way of the scholar, requiring a rare combination of high intelligence and charisma with a pinch of luck. It involves equal doses or profound thinking and aggressive campaigning, stirred vigorously in hope of the torque becoming self-sustaining – 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 stamina, 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, but also a public resource for the world to do with as they please.

10 Preliminaries

10.1 Dutch

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 an overview of the Dutch grammar here; I'm certain that interested parties will be able to easily get their hands on some introductory guide way more adequate than any I could have ever written.[1] In any case, knowledge of Dutch or its grammar is by no means a prerequisite to moving on[2] – the framework we will describe might be instantiated for Dutch (thus inheriting typological labels and conventions typical to the language), but is actually language-agnostic; it's not a framework *made for* the language, but a framework *applied to* the language. What could be of use, however, is the briefest of expositions to Dutch word order.

Even though main clauses emanate a false sense of safety, coming off as SVO at first glance, Dutch surface word order is SOV canonically, where exceptionally for matrix clauses the verb appears in the *second* position. The effect becomes apparent when employing a preverbal adverb – contrast (III.1a)

[1] An online version of the standard Dutch reference grammar can be found at https://e-ans.ivdnt.org/ (in Dutch).

[2] Speaking from experience.

with (III.1b), where both the subject and its predicate complement follow the verb in a VSO pattern.

(III.1) a. *Frans verkoop-t kaas.*
Frans sell-3SG cheese
'Frans sells cheese.'

b. *Morgen verkoop-t Frans kaas.*
tomorrow sell-3SG Frans cheese

Subordinate sentences are unaffected by the V2 rule. As a corollary, the infinitives of (non-nested) verbal complements (III.2a) are pushed to the end of their clause, as are the verbal heads of indirect questions (III.2b).

(III.2) a. *Frans wil koopman worden.*
Frans wants merchant become[INF]
'Frans wants to be a merchant.'

b. *Weet je wie kaas verkoop-t?*
know you who cheese sell-3SG
'Do you know who sells cheese?'

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 morphological 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 (III.3) below, where the first one is amenable to two different readings.

(III.3) a. *het geheim dat een bos verbergt*
the secret[N] that a forest[N] hides
i. 'the secret that 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 characterized 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 simplified common observations.

[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].

For starters, bare infinitives usually follow their governor – but: this is not necessarily the case for clusters of two 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.4) 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*

A similar phenomenon is observed in the passive voice and the perfect tense, both of which are formed periphrastically using a past participle and the appropriate auxiliary verb. There, the past participle may occur either to the left or the right of the auxiliary, leading to either a German- or an English- like construction.

(III.5) a. *... omdat ik de eend ge-zien heb*
... because I the duck PTCP-see have-1SG

b. *... omdat ik de eend heb ge-zien*
... because I the duck have-1SG PTCP-see

'... because I have seen the duck'

This gets further 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 – whether this substitution is mandatory, optional or altogether impossible is lexically decided [Augustinus, 2015].

(III.6) a. *Ik heb de eend ge-zien.*
I have the duck 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.'

There's of course a lot more to the story, but we will steer clear of the rabbit hole. The matter is of more than typological interest, though – 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 putting an end to any delusion that

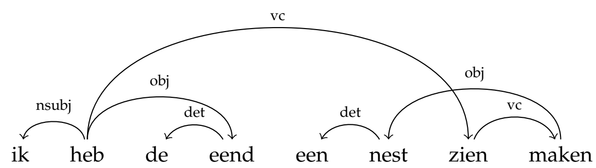


Figure III.1: Crossing dependencies in the 2-verb cluster of Gloss III.6c.

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. Long story short, Dutch word order is a complex puzzle that is not to be trifled with.

10.2 Parsing: Recognition vs. Discovery

Our journey has to start with some painful observations. A natural language grammar is a rather complicated construct that extends beyond the reach of compositional structure. Nominals come with a ton of morphosyntactic rules; 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 [Heylen, 1997; Pollard, 2004, inter alia]; 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 and historical 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, which puts 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

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

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 the morphological concessions we have already been silently making. But since the focus here is on compositionality, we're given the chance for a much more radical leap: we can *cut* the knot by skipping 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.27: 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.28, 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. $\mathbf{LP}_{\diamond,\square}$ 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, in Chapter IV.

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].

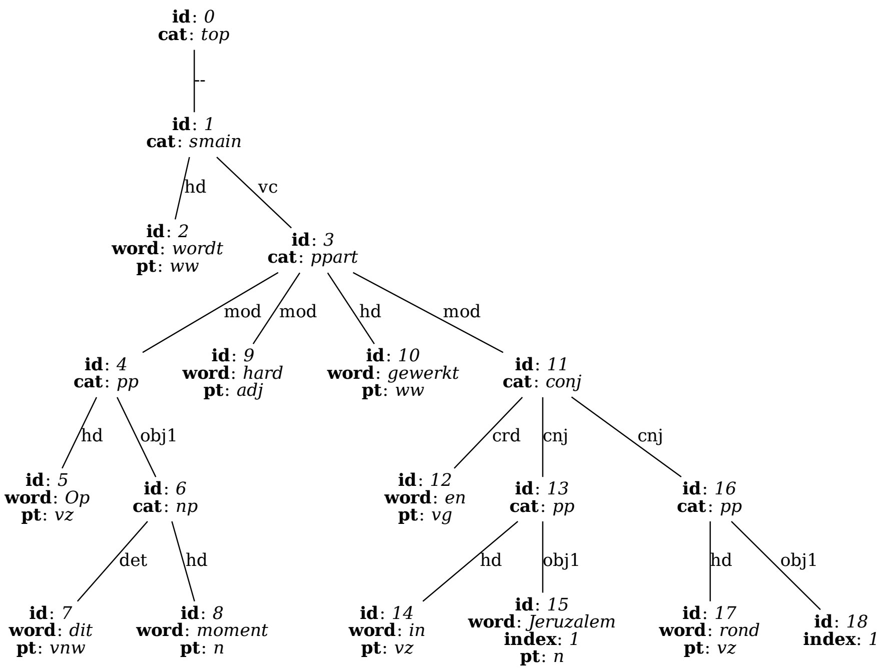
[1] An atonement to the ghost of Montague, returned to claim his dues.

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.

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. It is a gold standard corpus, meaning 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

[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.



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.

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 (i.e. silver standard). 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]

[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.

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	$\mathrm{S}_{\mathrm{main}}$
ssub	SOV Clause	0.8	$\mathrm{S}_{\mathrm{sub}}$
sv1	VSO Clause	<0.1	S_{vi}
svan	*van* Clause	<0.1	$\mathrm{S}_{\mathrm{van}}$
ti	*te* Infinitive	1.8	TI
whq	Main WH-Q	0.1	WH_{q}
whrel	Free Relative	0.2	$\mathrm{WH}_{\mathrm{rel}}$
whsub	Subordinate WH-Q	0.2	$\mathrm{WH}_{\mathrm{sub}}$
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	$\diamondsuit_{whbody}$
body	Relative Clause Body	0.1	$\diamondsuit_{relcl}$
body	Complementizer body	2	$\diamondsuit_{cmpbody}$
cnj	Conjunct	4.3	$\diamondsuit_{cnj}$
crd	Coordinator	1.8	–
crd	Second Element of Correlative	<0.1	$\diamondsuit_{cor}$
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	$\diamondsuit_{hdf}$
ld	Locative Complement	0.5	$\diamondsuit_{ld}$
me	Measure Complement	0.1	$\diamondsuit_{me}$
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	$\diamondsuit_{obcomp}$
obj1	Direct Object	10.8	$\diamondsuit_{obj1}$
obj2	Secondary Object	0.2	$\diamondsuit_{obj2}$
pc	Prepositional Complement	10.6	$\diamondsuit_{pc}$
pobj1	Preliminary Direct Object	<0.1	$\diamondsuit_{pobj1}$
predc	Predicative Complement	1.3	$\diamondsuit_{predc}$
predm	Predicative Modifier	0.1	$\Box_{predm}$
sat	Satellite	0.2	n/a
se	Obligatory Reflexive Object	0.7	$\diamondsuit_{se}$
su	Subject	6.9	$\diamondsuit_{su}$
sup	Preliminary Subject	<0.1	$\diamondsuit_{sup}$
svp	Separable Verbal Participle	0.7	$\diamondsuit_{svp}$
vc	Verbal Complement	2.8	$\diamondsuit_{vc}$
tag	Appendix	0.1	$\diamondsuit_{tag}$
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 extracted 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
	Newsletters
	Teletext Pages
	Web Sites
	Wikipedia
		`WR-P-E-C`
		`WR-P-E-E`
		`WR-P-E-H`
		`WR-P-E-I`
		`WR-P-E-J`
			14 420	232 631

WR-P-P	Books
	Brochures
	Guides & Manuals
	Legal Texts
	Newspapers
	Periodicals & Magazines
	Policy Documents
	Proceedings
	Reports
	Surveys
		`WR-P-P-B`
		`WR-P-P-C`
		`WR-P-P-E`
		`WR-P-P-F`
		`WR-P-P-G`
		`WR-P-P-H`
		`WR-P-P-I`
		`WR-P-P-J`
		`WR-P-P-K`
		`WR-P-P-L`
			17 691	281 424

WS-U	Auto Cues
	News Scripts
	Text for the Visually Impaired
		`WS-U-E-A`
		`WS-U-T-A`
		`WS-U-T-B`
			14 032	184 611

Table III.3: Breakdown of Lassy Small contents.

11 Æthel

The stage is set. We need to devise an algorithm that accepts trees like the one of Figure III.2 and emits proofs of $\mathbf{LP}_{\diamond,\Box}$. 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 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 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 unintended 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.

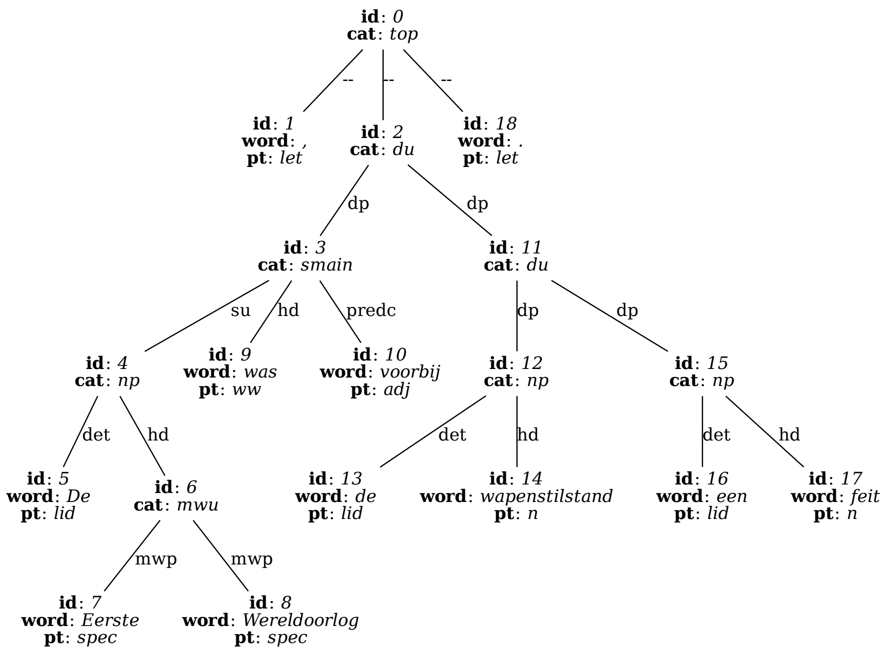
***det* or *mod*?** Lassy is occassionally inconsistent with the use of the determiner *det* and modifier *mod* labels in the nominal domain, marking either as the other in various contexts. Examples include marking indefinite, demonstrative or possessive pronouns as modifiers, and, the other way around, 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 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 concessions 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 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



`wiki-4941.p.4.s.2`

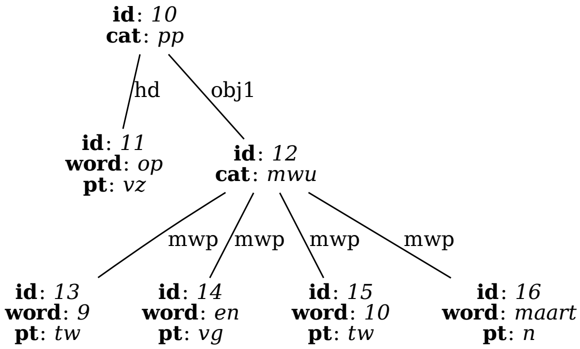
De Eerste Wereldoorlog was voorbij, de wapenstilstand een feit.
'The first world war was over, the armistice a fact.'

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

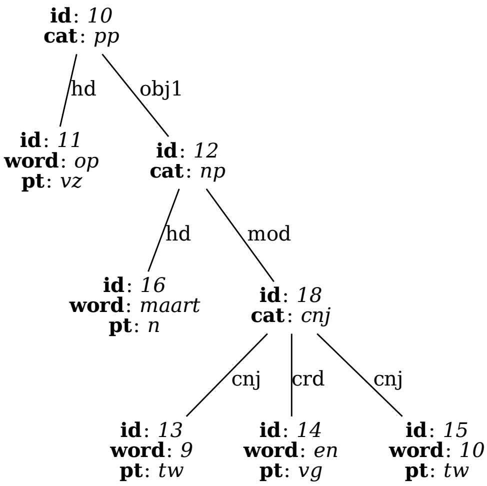
syntactic justification for the elided verb. Unfortunatey, there is no algorithmic way to mend these pretend annotations, in part due to their wildly general use, but mostly due to the fact they give us nothing to work with. This phenomenon is unpleasantly common; discourse level annotations sum up to about 2% of the total dependency edges in the corpus, and are present in some 11 700 samples, affecting 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.

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 subcase 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 stunning 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 keyed by 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. 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.

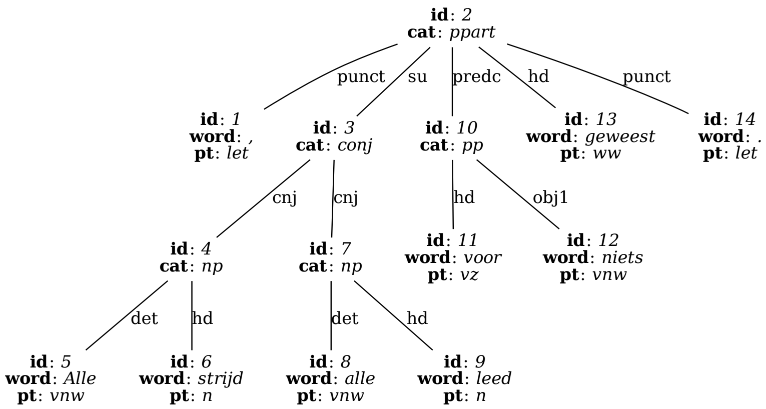
These minor changes suffice to cut down the frequency of multiword expressions to a more manageable 4.5% (an overall reduction of 25%). Unresolved expressions have their parts merged into a single node; the resulting node gets a new syntactic label, that being the most common part of speech tag of the 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 associating it to its material counterpart.

11.1.3 There Can Be Only One (Head)

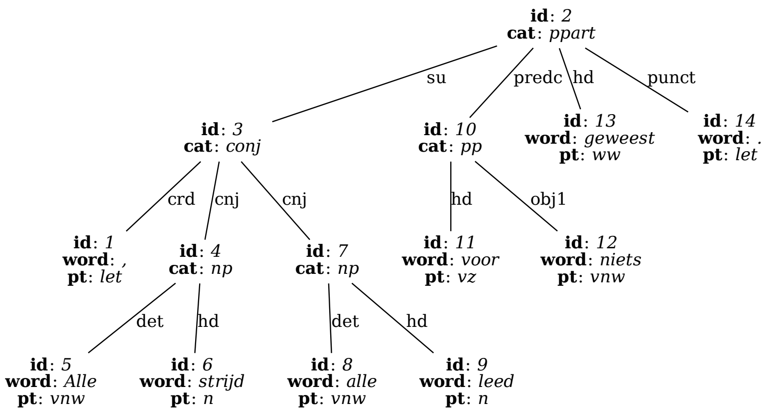
Having dealt with structureless structures, the next thing to tackle are subtrees that fail to elect a single head, falling to civil war (when multiple nodes compete for the role) or rising to anarchy (when all relinquish it). Here, we'll need to assume the interventive role of a self-appointed stabilizing 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 flown 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.5 for an example.

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 such adjustments involve removing or reorganizing subtrees and edges, we run the risk of accidentally removing the material tree that grounds a set of phantom



(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.'

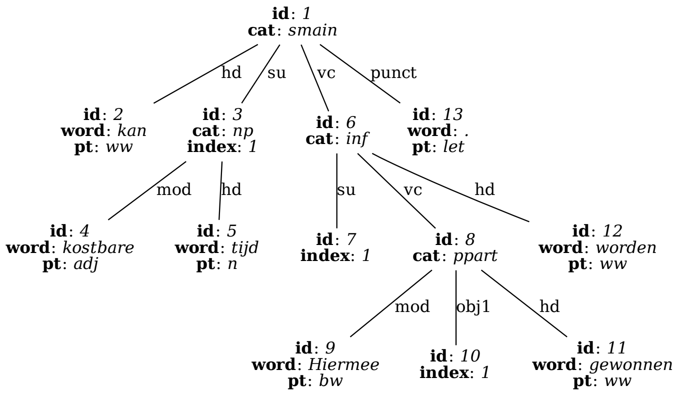
Figure III.5: Reannotating a headless conjunction.

nodes sharing the same index. As a precautionary measure, we enforce the convention of having the material tree appear as close to the root as possible.

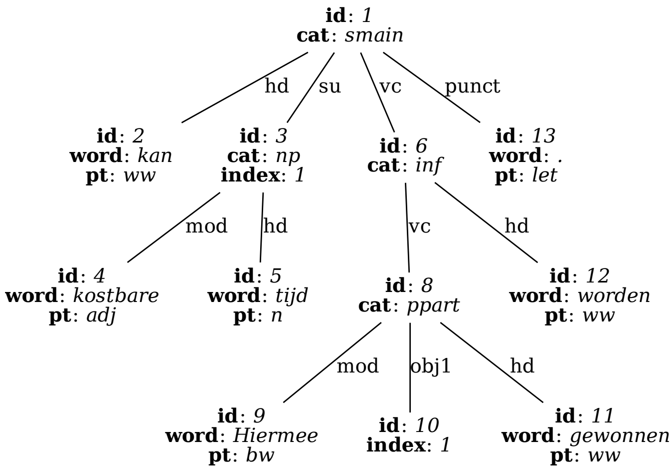
(Mis)understood Arguments Lassy treats the non-finite verbal forms (participles and infinitives) as verbal elements proper, selecting for all the arguments their finite counterparts would. Obviously, participles used for the passive or the perfect and infinitives in verbal complements cannot possibly 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 – we'll let it slide. 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.6 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 to the phrasal domain they are part of, rather than recursively 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

[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.

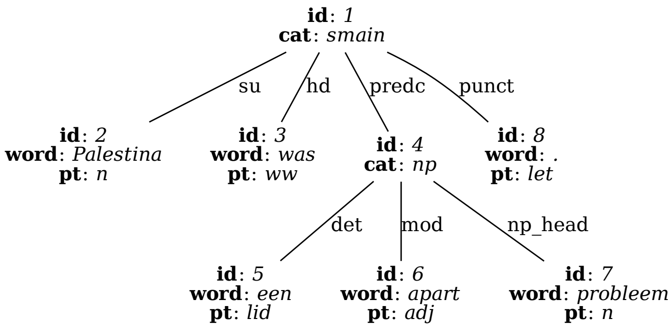


(b) After.

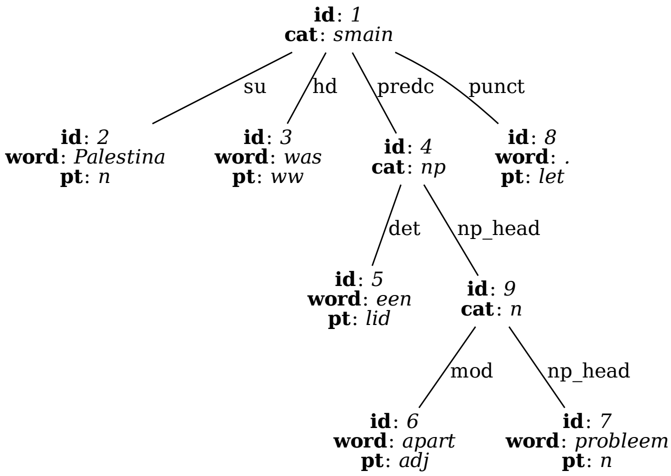
`dpc-gaz-001006-nl-sen.p.52.s.4(1)`

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

Figure III.6: 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.7: Inserting an intermediate layer for nominal modification.

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.7 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 dominance hierarchy 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.

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 turning N to NP.

[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.

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 domain-specific language, written in Python and allowing one to formulate, represent, transform and traverse valid proofs of $\mathbf{LP}_{\diamondsuit,\square}$. By invoking the language 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 $\mathbf{LP}_{\diamondsuit,\square}$ is a costly procedure, both in terms of processing overhead and of maintainance effort required, especially considering how unconducive Python 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 1 (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 atomic formulas, Prop_0. As hinted at earlier, the extraction is *parametric* to this translation: the domain can be any of the sets of lexi-

[1]It basically just converts italics to smallcaps.

cal 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. $\diamondsuit_{su}\mathrm{NP}\multimap\mathrm{PPART}$); 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 corresponding 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 a canonical 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. Intermediate 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.

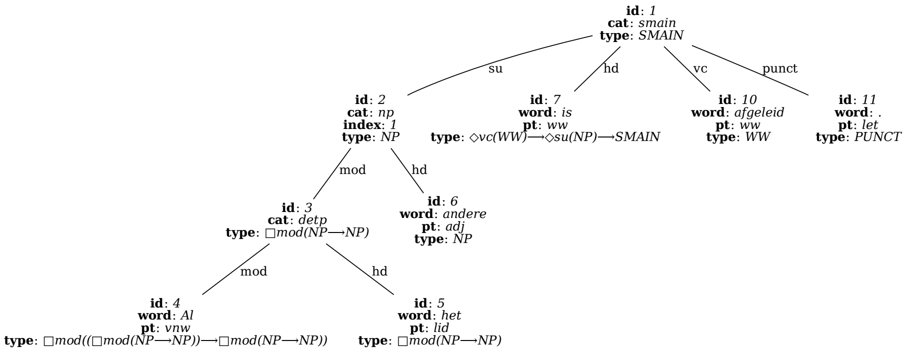
Terminal Nodes Terminal nodes are easy-peasy. There's only three questions we have to ask, namely "do we have a type hint?", "is the node a phantom?" and "was there a dependency label that got us here and, if so, was it either an adjunct or a complement?". Ok, this is actually four questions but nonetheless. 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.

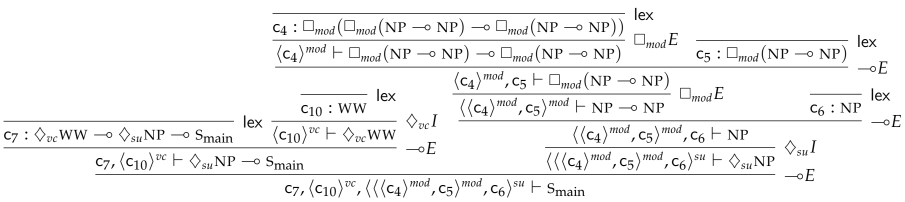
Non-Terminal Trees In any non-terminal domain, we must first decide on the type of the current phrase. If a type hint was passed from above, we have no option other than to obey it. If no hint was passed, we will translate the phrasal category of the current root into the appropriate top type.

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

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



(a) Type-annotated sample.



(b) Assigned proof (unprocessed).

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

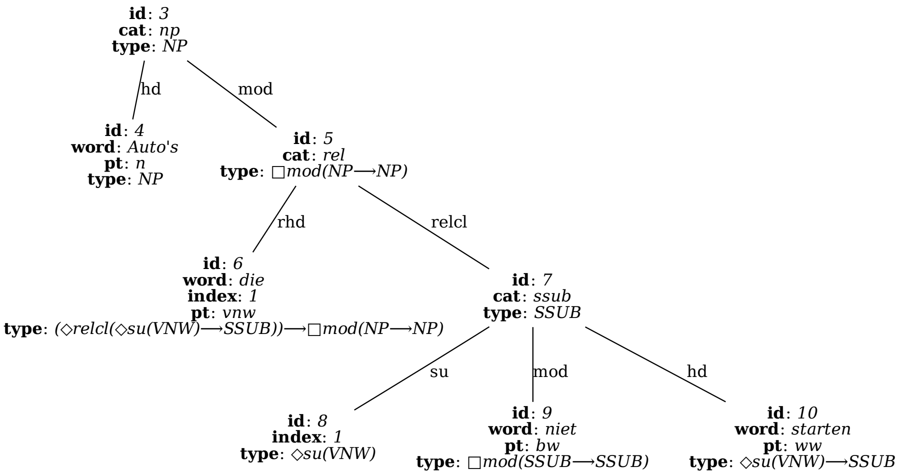
Al het andere is afgeleid.
'All the rest is derivative.'
$c_7 \vartriangle_{vc} c_{10} \vartriangle_{su} (\blacktriangledown_{mod}(\blacktriangledown_{mod} c_4\ c_5)\ c_6)$

Figure III.8: Proving a simple finite clause.

this way, we can infer the type of the phrasal head. We can thefore call prove on the head tree, passing no corresponding 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 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 used each tree below exactly once (adherent to 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.8. There, nodes 1, 2 and 10 obtain types $\mathrm{S_{main}}$, NP and WW by simple translation, and 7 obtains the type $\diamond_{vc}\mathrm{WW}\multimap\diamond_{su}\mathrm{NP}\multimap\mathrm{S_{main}}$, having 10 and 2 as complements (adorned with $\diamond_{vc}$ and $\diamond_{su}$ 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 have already been established by the phantom nodes underlying them – all we need to do is find out when to withdraw them. The criterion we'll follow applies to the proofs assigned to phrasal complements, and inspects the variables contained therein. Variables and nodes are in a one-to-one relation, owing to their common naming scheme; from a variable, we can refer to its node, and from a node we can extract its index, inducing a many-to-one variable to index relation. Any variable that maps to an index coinciding with that of the phrasal head (for subordinate clauses), the phrasal subject (for passives), or the phrasal subject of any ancestor phrase (for passives nested under auxiliaries), the variable is abstracted over. The complement then becomes a



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

$$\dfrac{\langle c_6, \langle \langle c_9 \rangle^{mod}, c_{10} \rangle^{relcl} \rangle^{mod} \vdash (\mathrm{NP} \multimap \mathrm{NP}) \quad c_4 : \mathrm{NP}}{\langle c_6, \langle \langle c_9 \rangle^{mod}, c_{10} \rangle^{relcl} \rangle^{mod}, c_4 \vdash \mathrm{NP}}\,\multimap E$$

`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.9: Abstracting over a relative clause gap.

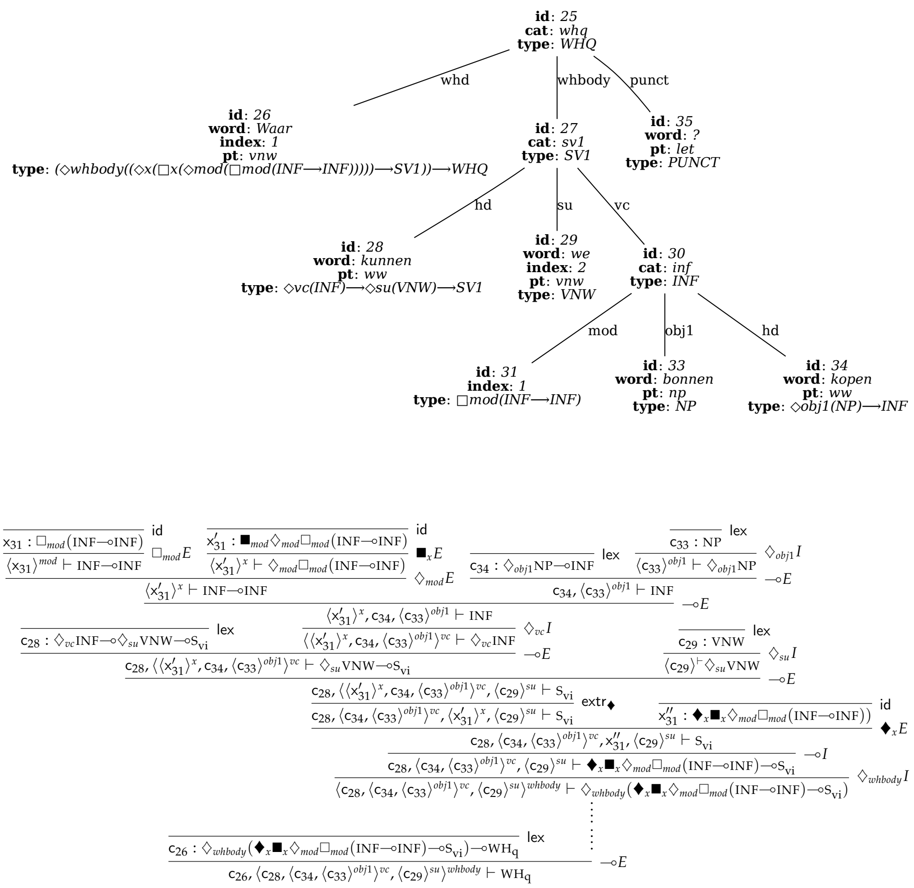
functional type, before the diamond introduction rule is applied. This forces the head to attain a higher-order type that *licenses* the hypothetical argument. Figure III.9 presents a concrete example: head node 6 carries index 1 and looks for its complement in node 7. But node 7 contains $\mathtt{x}_8$, and node 8 has index 1 – therefore, we must abstract the proof of node 7 over $\mathtt{x}_8$ 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 $\mathtt{x}_i^{\mathrm{A}} \mapsto \blacktriangledown_x(\mathtt{x}_i^{\blacksquare_x \mathrm{A}})$. 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 $\mathsf{extr}_\blacklozenge$ 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 $\mathsf{extr}_\blacklozenge$ rule from Section 8.3.2[2], we have effectively relieved 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.10. There, node 31 is originally assigned $\mathtt{x}_{31} : \Box_{mod}(\mathrm{INF} \multimap \mathrm{INF})$. Upon unboxing it for application, we realize it's a hypothetical adjunct, therefore we follow through with a diamond elimination to $\mathtt{x}'_{31} : \Diamond_{mod}\Box_{mod}(\mathrm{INF} \multimap \mathrm{INF})$ to lose the *mod* brackets. Further down the line, we attempt to abstract over

[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 vouchers?'

$\mathsf{c}_{26}\ \triangle_{whbody}(\lambda \mathsf{x}''_{31}.\text{case } \triangledown_{x}\mathsf{x}''_{31} \text{ of } \mathsf{x}'_{31} \text{ in } (\mathsf{c}_{28}\ \triangle_{vc}(\text{case } \triangledown_{mod}\blacktriangledown_{x}\mathsf{x}'_{31} \text{ of } \mathsf{x}_{31} \text{ in } (\blacktriangledown_{mod}\mathsf{x}_{31}\ (\mathsf{c}_{34}\ \triangle_{obj1}\mathsf{c}_{33})))\ \triangle_{su}\mathsf{c}_{29})))$

Figure III.10: Abstracting over a nested adjunct.

x'_{31}, only to find it trapped in a *vc* bracket. To facilitate the emancipation of the hypothesis, we perform the meta-syntactic substitution $x'_{31} \mapsto \blacktriangledown_x x'_{31}$, where the new x'_{31} is of type $\blacksquare_x \Diamond_{mod} \Box_{mod}(\text{INF} \multimap \text{INF})$. Empowered by its rectangular black flag, the variable breaks free of its chains with the $\text{extr}_{\blacklozenge}$ rule, and gets diamond eliminated to $x''_{31} : \blacklozenge_x \blacksquare_x \Diamond_{mod} \Box_{mod}(\text{INF} \multimap \text{INF})$. The latter is bracketless and exactly where we want it – we can finally perform the abstraction.

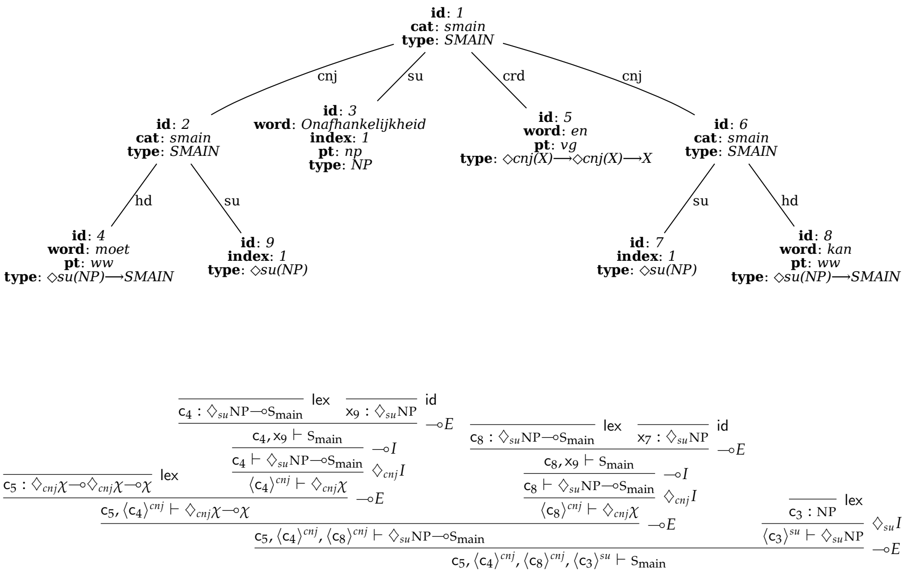
Nominal Domain The situation is not much different in the nominal domain, except for a change in the order we do things in. First, we must call prove on the head; unlike before, it receives no type hint, as we do not expect it to come out as a functor. Next, we requisition a proof for the determiner, 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 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

[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).'
$c_5 \vartriangle_{cnj} (\lambda x_9 . c_4\ x_9) \vartriangle_{cnj} (\lambda x_7 . c_8\ x_7) \vartriangle_{su} c_3$

Figure III.11: Proving a subject ellipsis.

level. The subject ellipsis of Figure III.11, 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.12.[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 passive 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:

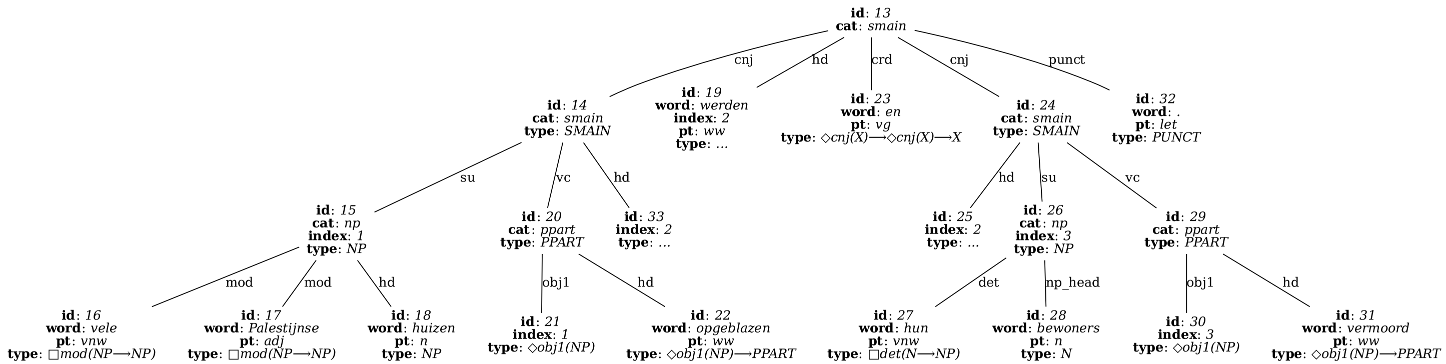
$$(\Diamond_{vc}(\Diamond_{obj1}\mathrm{NP}\multimap\mathrm{PPART})\multimap\Diamond_{su}\mathrm{NP}\multimap\mathrm{S}_{\mathrm{main}})\multimap\mathrm{S}_{\mathrm{main}}$$

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:

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

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



$$\dfrac{\dfrac{}{\mathsf{c}_{23} : \Diamond_{cnj}\chi \multimap \Diamond_{cnj}\chi \multimap \chi}\ \textsf{lex} \qquad \dfrac{\dfrac{\dfrac{\vdots}{\mathsf{x}_{33}, \langle \mathsf{c}_{22}\rangle^{vc}, \langle\langle \mathsf{c}_{16}\rangle^{mod}, \langle \mathsf{c}_{17}\rangle^{mod}, \mathsf{c}_{18}\rangle^{su} \vdash \mathsf{S}_{\text{main}}}}{\langle \mathsf{c}_{22}\rangle^{vc}, \langle\langle \mathsf{c}_{16}\rangle^{mod}, \langle \mathsf{c}_{17}\rangle^{mod}, \mathsf{c}_{18}\rangle^{su} \vdash \chi}\ \multimap I}{\langle\langle \mathsf{c}_{22}\rangle^{vc}, \langle\langle \mathsf{c}_{16}\rangle^{mod}, \langle \mathsf{c}_{17}\rangle^{mod}, \mathsf{c}_{18}\rangle^{su}\rangle^{cnj} \vdash \Diamond_{cnj}\chi}\ \Diamond_{cnj} I}{\mathsf{c}_{23}, \langle\langle \mathsf{c}_{22}\rangle^{vc}, \langle\langle \mathsf{c}_{16}\rangle^{mod}, \langle \mathsf{c}_{17}\rangle^{mod}, \mathsf{c}_{18}\rangle^{su}\rangle^{cnj} \vdash \Diamond_{cnj}\chi \multimap \chi}\ \multimap E$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\mathsf{x}_{25}, \langle \mathsf{c}_{31}\rangle^{vc}, \langle\langle \mathsf{c}_{27}\rangle^{det}, \mathsf{c}_{28}\rangle^{su} \vdash \mathsf{S}_{\text{main}}}}{\langle \mathsf{c}_{31}\rangle^{vc}, \langle\langle \mathsf{c}_{27}\rangle^{det}, \mathsf{c}_{28}\rangle^{su} \vdash \chi}\ \multimap I}{\langle\langle \mathsf{c}_{31}\rangle^{vc}, \langle\langle \mathsf{c}_{27}\rangle^{det}, \mathsf{c}_{28}\rangle^{su}\rangle^{cnj} \vdash \Diamond_{cnj}\chi}\ \Diamond_{cnj} I$$

$$\dfrac{\mathsf{c}_{23}, \langle\langle \mathsf{c}_{22}\rangle^{vc}, \langle\langle \mathsf{c}_{16}\rangle^{mod}, \langle \mathsf{c}_{17}\rangle^{mod}, \mathsf{c}_{18}\rangle^{su}\rangle^{cnj} \vdash \Diamond_{cnj}\chi \multimap \chi \qquad \langle\langle \mathsf{c}_{31}\rangle^{vc}, \langle\langle \mathsf{c}_{27}\rangle^{det}, \mathsf{c}_{28}\rangle^{su}\rangle^{cnj} \vdash \Diamond_{cnj}\chi}{\mathsf{c}_{23}, \langle\langle \mathsf{c}_{22}\rangle^{vc}, \langle\langle \mathsf{c}_{16}\rangle^{mod}, \langle \mathsf{c}_{17}\rangle^{mod}, \mathsf{c}_{18}\rangle^{su}\rangle^{cnj}, \langle\langle \mathsf{c}_{31}\rangle^{vc}, \langle\langle \mathsf{c}_{27}\rangle^{det}, \mathsf{c}_{28}\rangle^{su}\rangle^{cnj} \vdash \chi}\ \multimap E$$

$$\dfrac{\begin{matrix}\vdots \\ \cdots\end{matrix} \qquad \dfrac{}{\mathsf{c}_{19} : \Diamond_{vc}(\Diamond_{obj1}\mathrm{NP} \multimap \mathrm{PPART}) \multimap \Diamond_{su}\mathrm{NP} \multimap \mathsf{S}_{\text{main}}}\ \textsf{lex}}{\mathsf{c}_{23}, \langle\langle \mathsf{c}_{22}\rangle^{vc}, \langle\langle \mathsf{c}_{16}\rangle^{mod}, \langle \mathsf{c}_{17}\rangle^{mod}, \mathsf{c}_{18}\rangle^{su}\rangle^{cnj}, \langle\langle \mathsf{c}_{31}\rangle^{vc}, \langle\langle \mathsf{c}_{27}\rangle^{det}, \mathsf{c}_{28}\rangle^{su}\rangle^{cnj}, \mathsf{c}_{19} \vdash \mathsf{S}_{\text{main}}}\ \multimap E$$

`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.'
$\mathsf{c}_{23} \vartriangle_{cnj}(\lambda \mathsf{x}_{33}.\mathsf{x}_{33} \vartriangle_{vc}(\lambda \mathsf{x}_{21}.\mathsf{c}_{22}\ \mathsf{x}_{21}) \vartriangle_{su}(\blacktriangledown_{mod}\mathsf{c}_{16}\ (\blacktriangledown_{mod}\mathsf{c}_{17}\ \mathsf{c}_{18})) \vartriangle_{cnj}(\lambda \mathsf{x}_{25}.\mathsf{x}_{25} \vartriangle_{vc}(\lambda \mathsf{x}_{30}.\mathsf{c}_{31}\ \mathsf{x}_{30}) \vartriangle_{su}(\blacktriangledown_{det}\mathsf{c}_{27}\ \mathsf{c}_{28}))\ \mathsf{c}_{19}$

Figure III.12: Proving a verb phrase ellipsis with a higher-order head.

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

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

Ring a bell yet? Using S_{main} in place of A, we end up exactly where we started.[1] Now, toying around with raised types is by no means ideal, as it reinforces the 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. Such a lifted interpretation is aligned in spirit with the long line of work on categorial conjunction semantics [Oehrle, 1987; Hendriks, 1995, inter alia], except simplified in *not* having to account for the surface discontinuity caused by the elision of linguistic material.

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 a theoretical 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 a representational perspective, our

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

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 depth-first left-first traversal of the proof tree (preferrable to de Bruijn indexing for human legibility). 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. For a practical user's guide to Æthel, refer to Appendix 2. For moving on, it suffices to know that an analysis is essentially a nicely packaged tuple, containing a proof on one coordinate and a tokenized and type-annotated sentence on the other.

11.3 Analysis

11.3.1 Quantitative Obligations

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 `1.0.0a5`.

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 a 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.13. Surprising noone, the graph reveals that both the original and the derived corpus exhibit a right-skewed distribution of sample lengths, the latter being a left-shifted version of the for-

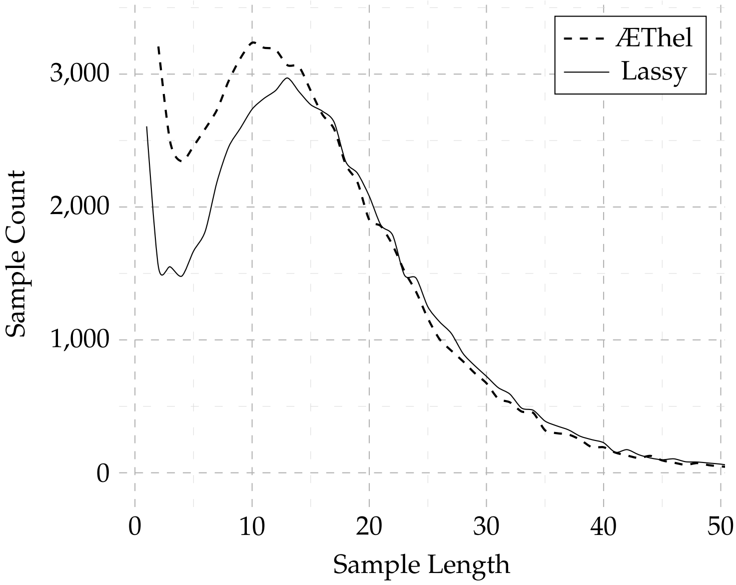


Figure III.13: Sample counts by length.

mer. 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 lexical content and end up with a total of 55 108 unique theorems, where constants are identified indexically (and not by their string). Figure III.14 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 (52 739 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 apart, i.e. we have no means of knowing whether a second proof derivable

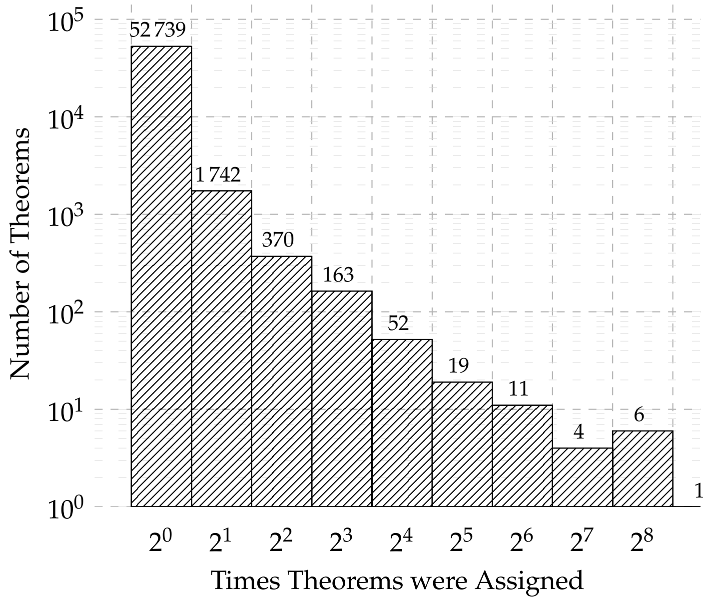


Figure III.14: Proof sparsity in Æthel.

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 56 098 unique proof frames, but 53 867 (96%) 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 2 292, 2 231 (97.3%) are mapped to one unique proof, whereas 61 (2.7%) 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 might prescribe 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 **LP** (dropping any dependency information via a stripping morphism), we end up with 56 185 unique frames for 54 170 unique proofs, with 2 471 frames occurring more than once. Of these,

[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.

2 374 are behind 1 unique proof, 112 behind 2, 6 behind 3, and 1 is behind 5. Potentially ambiguous samples are now doubled (909, 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 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 keys (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 5 762 unique $\mathbf{LP}_{\diamondsuit,\Box}$ types. Stripped of their modalities, these fall back to 3 542 pure linear types, evidencing the import of the added dependency axis. The first question to ask then is: how functional is the word to type relation encoded by the lexicon? Answering that question is Figure III.15, which in short says "not much, really". Despite most lexical keys (56 224) having a single unique image, ambiguities are quite common, the number of assignments exponentially$_2$ increasing for a logarthmically$_{10}$ declining number of keys. 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 1 045 (!) unique types in just 41 614 occurrences. No less than 1 363 types contain a $\diamondsuit_{cnj}$ marking, and are thus associated with coordinators – the price of braving conjunctions with concretely instanced, pre-raised types; Table III.4 presents the most commonly occurring coordinator types. But this is painting an overly grim picture; seen as random variables, lexical keys actually have most of their probability masses concentrated on their modes. The average probability of the most common type per key lies at a (very high) 89.04%, while naively assigning the mode to each lexical assignment corpus-wide (disregarding context) establishes a comfortable 67.08% baseline, asserting that our lexical entries are

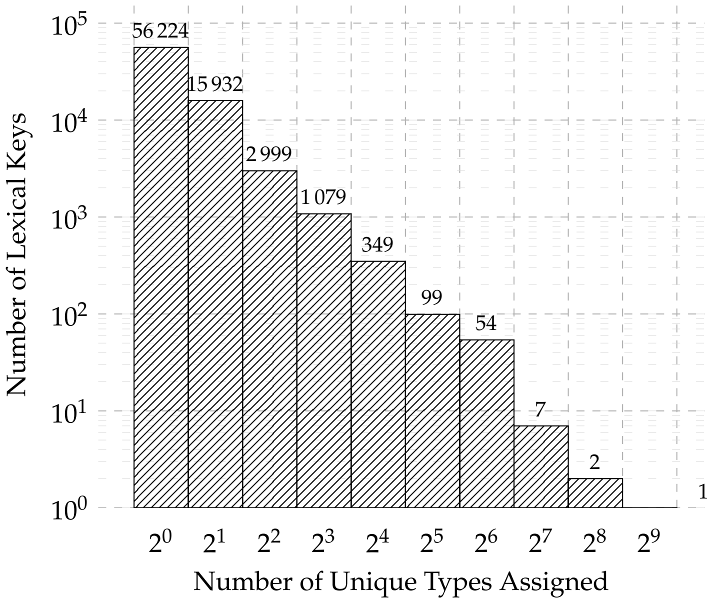


Figure III.15: Lexical Type Ambiguity

in practice quite consistent when grouped by their keys.

Lexical Type Sparsity So ambiguity is not a major statistical threat, despite the high average number of options per key; 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 5 762 types that occur more than n times, and plot the result in Figure III.16 (if you prefer jargon, this is the inverse empirical cumulative distribution of type occurrences). The dash-dotted line leaves little room for interpretation: 2 747 (47.7%) of the total types have only a single occurrence, 4 603 (79.9%) have less than 10 occurrences, and only 310 (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 key that maps to them; in other words, a type rarely associated with *some* lexical key is likely rare to find among *any* key.

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-

Concrete Type	# Occurrences	Concrete Type	# Occurrences
NP	7,620	$\square_{mod}(\mathrm{N}\multimap\mathrm{N})$	582
$\mathrm{S_{main}}$	2 623	$\square_{app}(\mathrm{NP}\multimap\mathrm{NP})$	528
$\square_{mod}(\mathrm{NP}\multimap\mathrm{NP})$	1 655	NP ($\times 4$)	400
NP ($\times 3$)	1 366	PP	361
$\Diamond_{mod}\square_{mod}(\mathrm{NP}\multimap\mathrm{NP})\multimap\mathrm{NP}$	1 147	$\Diamond_{su}\mathrm{VNW}\multimap\mathrm{S_{main}}$	344
$\Diamond_{su}\mathrm{NP}\multimap\mathrm{S_{main}}$	1 113	TI	302
$\Diamond_{det}\square_{det}(\mathrm{N}\multimap\mathrm{NP})\multimap\mathrm{NP}$	592	$\square_{mod}(\mathrm{S_{main}}\multimap\mathrm{S_{main}})$	282

Table III.4: Most common concrete types instantiating the polymorphic coordinator, and relevant occurrence counts. Coordinator arity is 2 unless specified.

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 98% 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 Key Sparsity On the other side of the lexicon lie lexical keys – the words and phrases we may use to index the lexicon. Lexical key 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 vectors 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.17 shows the distribution of lexical key occurrences; most keys occur just once, whereas a handful occur up to a few thousand times.[1] The most common word of Æthel is the gendered definite determiner *de*, claiming the throne with an incontestible 56 925 occurrences and making up for 5.7% of all words just by itself.

[1] Interpolate between most and a handful in the $\log_{10}/\log_2$ plane to find any point of interest.

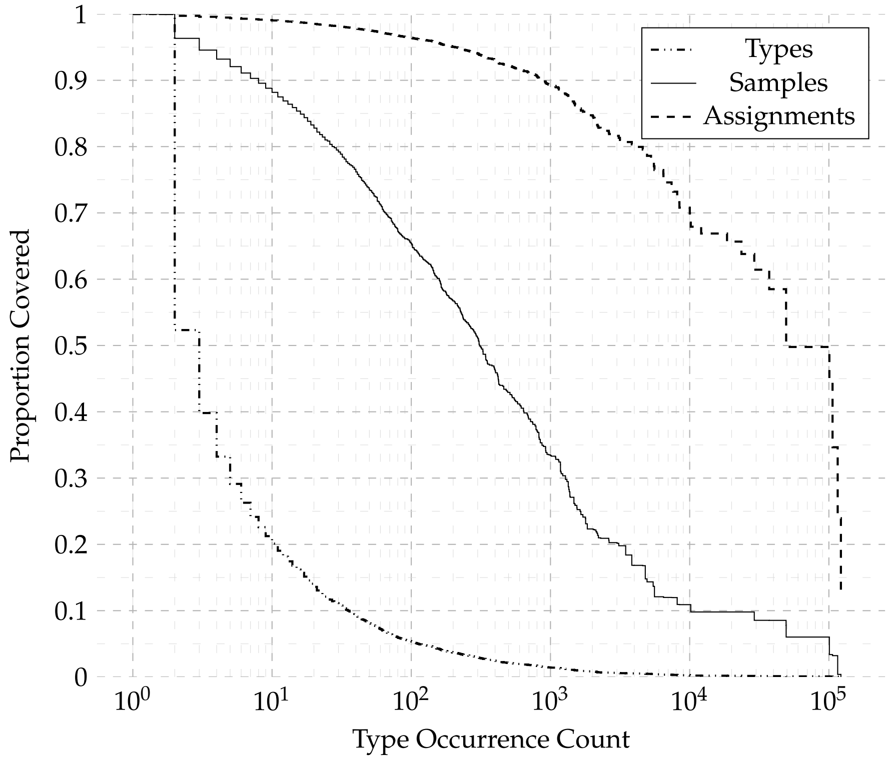


Figure III.16: Proportion of types, type assignments and samples covered as a function of a minimum type occurrence threshold.

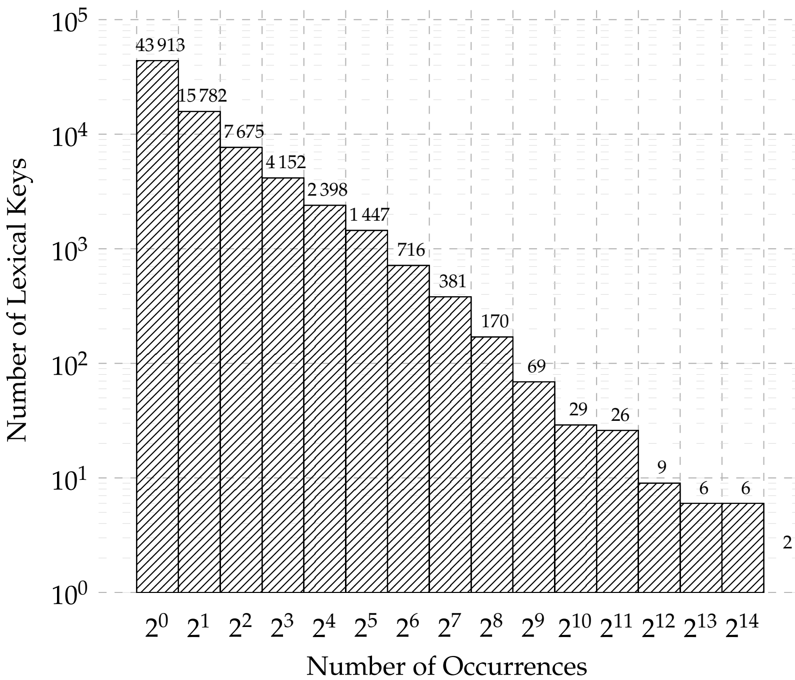


Figure III.17: Lexical Key Sparsity

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 showcase how Æthel treats common and idiosyncratic constructions of Dutch. For space economy, we'll drop the natural deduction presentation in favor of the compactified λ terms – we've seen enough back-and-forths by now to take the training wheels off. The glossed examples to follow contain a third line for type assignments, and the prescribed λ term before the free translation.

Higher-Order Reasoning The extraction recipe applied to coindexed phantom nodes seems to generalize well, producing elegant higher-order proof patterns that capture the corresponding syntactic reentrancies prescribed from the Lassy graph quite well. In (III.7), *waarom* both heads the wh-question and provides a hypothetical argument for the verb-initial direct question nested within. In (III.8), the auxiliary *worden* properly accounts for the passive by providing a hypothetical object for the transitive past participle *behandeld*.

(III.7) `dpc-vla-001161-nl-sen.p.107.s.1(1)`

Waarom	*werken*	*we*	*?*
Why	work	we	?
$c_0 :: \Diamond_{whbody}(\Diamond_{mod}\Box_{mod}(\mathrm{S_{vi}} \multimap \mathrm{S_{vi}}) \multimap \mathrm{S_{vi}}) \multimap \mathrm{WH_q}$	$c_1 :: \Diamond_{su}\mathrm{VNW} \multimap \mathrm{S_{vi}}$	$c_1 :: \mathrm{VNW}$	PUNCT

$c_0 \vartriangle_{whbody}(\lambda x_0.\text{case } \triangledown_{mod} x_0 \text{ of } x_1 \text{ in } (\blacktriangledown_{mod} x_1\ (c_1 \vartriangle_{su} c_2)))$

'Why do we work?'

(III.8) `WR-P-P-J-0000000001.p.38.s.2(1)`

Deze	*worden*	*behandeld*
these	become	treated
$c_0 :: \mathrm{VNW}$	$c_1 :: \Diamond_{vc}(\Diamond_{obj1}\mathrm{VNW} \multimap \mathrm{PPART}) \multimap \Diamond_{su}\mathrm{VNW} \multimap \mathrm{S_{main}}$	$c_2 :: \Diamond_{obj1}\mathrm{VNW} \multimap \mathrm{PPART}$

in	*hoofdstuk*	*4*	*.*
in	chapter	4	.
$c_3 :: \Diamond_{obj1}\mathrm{NP} \multimap \Box_{mod}(\mathrm{PPART} \multimap \mathrm{PPART})$	$c_4 :: \mathrm{NP}$	$c_5 :: \Box_{app}(\mathrm{NP} \multimap \mathrm{NP})$	PUNCT

$c_1 \vartriangle_{vc}(\lambda x_0.\blacktriangledown_{mod}(c_3 \vartriangle_{obj}(\blacktriangledown_{app} c_5\ c_4)\ (c_2\ x_0))) \vartriangle_{su} c_0$

'These will be treated in chapter 4.'

Relative Clauses The derivational ambiguity attributed to relative clauses is now implicitly cast into lexical type ambiguity – the relativizer obtains a different type assignment depending on the type of relative clause it introduces. Examine the type assignments to the relativizer *die* in the examples below, and notice that the hypothesis of (III.9) is forced to occupy the object slot of *oproepen*, as opposed to the hypothesis of (III.10) which is forced to occupy the subject slot of *volgen*. This is not just due to the decorations assigned, but also because of the atomic type distinction between pronouns VNW and noun phrases NP[1], implicitly filtering out implausible argument associations.

(III.9) `WS-U-E-A-0000000215.p.25.s.1(1)`

Beelden	*die*	*slechte*	*herinneringen*	*oproepen*	*.*
Pictures	that	bad	memories	evoke	.
$c_0 :: \mathrm{NP}$	$c_1 :: \Diamond_{relcl}(\Diamond_{su}\mathrm{VNW} \multimap \mathrm{S_{sub}}) \multimap \Box_{mod}(\mathrm{NP} \multimap \mathrm{NP})$	$c_2 :: \Box_{mod}(\mathrm{NP} \multimap \mathrm{NP})$	$c_3 :: \mathrm{NP}$	$c_4 :: \Diamond_{obj1}\mathrm{NP} \multimap \Diamond_{su}\mathrm{VNW} \multimap \mathrm{S_{sub}}$	PUNCT

$\blacktriangledown_{mod}(c_1 \vartriangle_{relcl}(c_4 \vartriangle_{obj1}(\blacktriangledown_{mod} c_2\ c_3)))\ c_0$

'Pictures that evoke bad memories.'

(III.10) `pc-ind-001645-nl-sen.p.12.s.1(13)`

de	*strategie*	*die*	*ze*	*volgen*	*is*	*eeuwenoud*	*.*
the	strategy	that	they	follow	is	centuries.old	.
$c_0 :: \Box_{det}(\mathrm{N} \multimap \mathrm{NP})$	$c_1 :: \mathrm{N}$	$c_2 :: \Diamond_{relcl}(\Diamond_{obj1}\mathrm{VNW} \multimap \mathrm{S_{sub}}) \multimap \Box_{mod}(\mathrm{NP} \multimap \mathrm{NP})$	$c_3 :: \mathrm{VNW}$	$c_4 :: \Diamond_{obj1}\mathrm{VNW} \multimap \Diamond_{su}\mathrm{VNW} \multimap \mathrm{S_{sub}}$	$c_5 :: \Diamond_{predc}\mathrm{ADJ} \multimap \Diamond_{su}\mathrm{NP} \multimap \mathrm{S_{main}}$	$c_6 :: \mathrm{ADJ}$	PUNCT

$c_5 \vartriangle_{predc} c_6 \vartriangle_{su}(\blacktriangledown_{mod}(c_2 \vartriangle_{relcl}(\lambda x_0.c_4\ x_0 \vartriangle_{su} c_3))\ (\blacktriangledown_{det} c_0\ c_1))$

'the strategy they follow is centuries old'

Word Order & Sentential Types Similarly, word order may not be formally treated, but the subcategorization of sentential types provides a heuristic clue toward what the correct association between verbal arguments and their sentential positioning should be. Compare the distinct positioning of the verbal head *heeft* in the examples below and how it reflects on its type assignment. In (III.11), the result type is fixed to being a main clause, whereas in (III.12) the result type accounts for direct questions being verb initial. This goes to show that lexical type ambiguity is not the adverse effect of a misbehaved extraction or an ill-thought logic, but actually a reflection of the real morphosyntactic plurality of Lassy (and by extension, Dutch).

(III.11) `WR-P-P-I-0000000130.p.1.s.1(1)`

Vet	*heeft*	*geen*	*smaak*	*.*
Fat	has	no	taste	.
$c_0 :: \mathrm{NP}$	$c_1 :: \Diamond_{obj1}\mathrm{NP} \multimap \Diamond_{su}\mathrm{NP} \multimap \mathrm{S_{main}}$	$c_2 :: \Box_{mod}(\mathrm{NP} \multimap \mathrm{NP})$	$c_3 :: \mathrm{NP}$	PUNCT

$c_1 \vartriangle_{obj1}(\blacktriangledown_{mod} c_2\ c_3) \vartriangle_{su} c_0$

'Fat has no taste.'

[1]Where relative clause gaps are always the former, due to being coindexed with the relativizer.

(III.12) `WR-P-P-C-0000000008.p.16.s.1(2)`

Heeft	*uw*	*woning*	*mechanische*
Has	your	residence	mechanical
$c_0 :: \Diamond_{obj1}\mathrm{NP} \multimap \Diamond_{su}\mathrm{NP} \multimap \mathrm{S_{vi}}$	$c_1 :: \Box_{det}(\mathrm{N} \multimap \mathrm{NP})$	$c_2 :: \mathrm{N}$	$c_3 :: \Box_{mod}(\mathrm{NP} \multimap \mathrm{NP})$
ventilatie	*?*		
ventilation	?		
$c_4 :: \mathrm{NP}$	PUNCT		

$c_0 \vartriangle_{obj1} (\blacktriangledown_{mod} c_3\ c_4) \vartriangle_{su} (\blacktriangledown_{det} c_1\ c_2)$

'Does your residence have mechanical ventilation?'

Discontinuities Horizontal (i.e. word-order) discontinuities are trivialized by the non-directional type system. Neither the discontinuous *om..te* 'to' nor the displaced participle of the separable verb *wegnemen* 'take away' have any difficulty finding their place in the derivation of (III.13).

(III.13) `WR-P-P-C-0000000048.txt-162(6)`

Om⟩	*eventuele*	*twijfel*	*weg*	*⟨te*
To⟩	potential	doubts	away	
$c_0 :: \Diamond_{cmpbody}\mathrm{TI} \multimap \mathrm{OTI}$	$c_1 :: \Box_{mod}(\mathrm{NP} \multimap \mathrm{NP})$	$c_2 :: \mathrm{NP}$	$c_3 :: \mathrm{BW}$	$c_4 :: \Diamond_{cmpbody}\mathrm{INF} \multimap \mathrm{TI}$
nemen				
take				
$c_5 :: \Diamond_{svp}\mathrm{BW} \multimap \Diamond_{obj1}\mathrm{NP} \multimap \mathrm{INF}$				

$c_0 \vartriangle_{cmpbody} (c_4 \vartriangle_{cmpbody} (c_5 \vartriangle_{svp} c_3 \vartriangle_{obj1} (\blacktriangledown_{mod} c_1\ c_2)))$

'To cast away any doubts'

Likewise, types don't have to vary for the two different placements of past participles in the perfect or the passive; inspect the positioning of the auxiliary *heeft* in relation to the past participles *geïnformeerd* and *bezield* in Examples (III.14) and (III.15) below.

(III.14) `WS-U-E-A-0000000046.p.8.s.6(1)`

Hij	*erkent*	*dat*	*hij*
He	admits	that	he
$c_0 :: \mathrm{VNW}$	$c_1 :: \Diamond_{vc}\mathrm{CP} \multimap \Diamond_{su}\mathrm{VNW} \multimap \mathrm{S_{main}}$	$c_2 :: \Diamond_{cmpbody}\mathrm{S_{sub}} \multimap \mathrm{CP}$	$c_3 :: \mathrm{VNW}$
de	*kamer*	*onjuist*	*geïnformeerd*
the	chamber	wrongly	informed
$c_4 :: \Box_{det}(\mathrm{N} \multimap \mathrm{NP})$	$c_5 :: \mathrm{N}$	$c_6 :: \Box_{mod}(\mathrm{PPART} \multimap \mathrm{PPART})$	$c_7 :: \Diamond_{obj1}\mathrm{NP} \multimap \mathrm{PPART}$
heeft	*.*		
has	.		
$c_8 :: \Diamond_{vc}\mathrm{PPART} \multimap \Diamond_{su}\mathrm{VNW} \multimap \mathrm{S_{sub}}$	PUNCT		

$c_1 \vartriangle_{vc} (c_2 \vartriangle_{vc} (\blacktriangledown_{mod} c_6\ (c_7 \vartriangle_{obj1} (\blacktriangledown_{det} c_4\ c_5)))) \vartriangle_{su} (c_0)$

'He admits that he has wrongly informed the chamber.'

(III.15) `WR-P-P-I-0000000173.p.1.s.2(1)`

Hij	*moet*	*begrijpen*
He	must	understand
$c_0 :: \text{VNW}$	$c_1 :: \Diamond_{vc}\text{INF} \multimap \Diamond_{su}\text{VNW} \multimap \text{S}_{\text{main}}$	$c_2 :: \Diamond_{vc}\text{WH}_{\text{sub}} \multimap \text{INF}$

wat	*de*	*verdachte*
what	the	suspect
$c_3 :: \Diamond_{whbody}(\Diamond_{su}\text{VNW} \multimap \text{S}_{\text{sub}}) \multimap \text{WH}_{\text{sub}}$	$c_4 :: \Box_{det}(\text{WW} \multimap \text{NP})$	$c_5 :: \text{WW}$

heeft	*bezield*	.
has	possessed	.
$c_6 :: \Diamond_{vc}\text{PPART} \multimap \Diamond_{su}\text{VNW} \multimap \text{S}_{\text{sub}}$	$c_7 :: \Diamond_{obj1}\text{NP} \multimap \text{PPART}$	PUNCT

$$c_1 \vartriangle_{vc}(c_2 \vartriangle_{vc}(\vartriangle_{whbody}(c_6 \vartriangle_{vc}(\vartriangle_{obj1}(\blacktriangledown_{det} c_4\ c_5))))) \vartriangle_{su} c_0$$

'He must understand what possessed the subject.'

Example (III.15) is of special interest, showcasing also the absence of "conversion" rules – the nominalized present participle *verdachte* is not cast into a N, but used as-is, forcing a non-standard type assignment to the determiner.

Vertical (i.e. dependency domain) discontinuities are a tad trickier, requiring the $\blacklozenge_x\blacksquare_x$ type pattern to unlock. Combined with our position-explicit formulation of the $\Diamond E$ rules, this has the side-effect of producing rather complicated proofs and terms, even for relatively short and simple sentences. In the example below, the hypothetical modifier seeks to apply to a *complete* past participle, forcing the transitive participle forming the passive to *really* apply to a hypothetical object noun phrase, only to then immediately abstract over it, before the whole $\text{extr}_{\blacklozenge}$ contraption is set to work. The result is a term with as many variables as constants – not a particularly pleasant sight to behold, even if keeping terms and proofs representationally proximal.[1]

(III.16) `dpc-bmm-001078-nl-sen.p.1.s.1(1)`

Hoe
How
$c_0 :: \Diamond_{whbody}(\blacklozenge_x\blacksquare_x\Diamond_{mod}\Box_{mod}(\text{PPART} \multimap \text{PPART}) \multimap \text{S}_{\text{vi}}) \multimap \text{WH}_{\text{q}}$

worden	*modellen*	*ontwikkeld*
become	models	developed
$c_1 :: \Diamond_{vc}(\Diamond_{obj1}\text{NP} \multimap \text{PPART}) \multimap \Diamond_{su}\text{NP} \multimap \text{S}_{\text{vi}}$	$c_2 :: \text{NP}$	$c_3 :: \Diamond_{obj1}\text{NP} \multimap \text{PPART}$

$$c_0 \vartriangle_{whbody}(\lambda x_0.\text{case } \triangledown_x x_0 \text{ of } x_1 \text{ in}$$
$$c_1 \vartriangle_{vc} (\lambda x_2.\text{case } \triangledown_{mod}\blacktriangledown_x x_1 \text{ of } x_3 \text{ in } (\blacktriangledown_{mod} x_3\ (c_3\ x_2))) \vartriangle_{su} c_2)$$

'How are models developed'

Conjunctions A similar effect is observed with conjunctions. Even though the polymorphic scheme works wonders with simple coordination, ellipses

[1]Selectively using the shortcut notation of Section 4.1.2 at the user's behest is an option, producing the somewhat more legible: $c_0 \vartriangle_{whbody}(\lambda x_0.c_1 \vartriangle_{vc}(\lambda x_1.\blacktriangledown_{mod}\triangledown_{mod}\blacktriangledown_x\triangledown_x x_1\ (c_3\ x_1)) \vartriangle_{su} c_2)$.

can quickly turn types humongous and analyses illegible. In (III.17) our defense of Lassy's tendency to distribute modifiers among conjuncts backfires, as we end up with a coordinator joining not two noun phrases, but two noun phrases missing a mobile modifier. The proof abides by our design decision to allow for flexible semantic interpretations, but in doing so gives us a headache to visually parse. It's worth pointing out that the actual modifier does not actually need to be mobile, being just an argument to the coordinator. This goes to show that the polymorphic scheme is a gentle suggestion and not a strict mandate, allowing for sentence-specific adjustments and variations.

(III.17) `WS-U-E-A-0000000243.p.1.s.1(14)`

Dertig	*vrouwen*	*en*	*kinderen*
Thirty	women	and	children
$\mathsf{c}_0 :: \Box_{mod}(\mathrm{NP}\multimap\mathrm{NP})$	$\mathsf{c}_1 :: \mathrm{NP}$	$\mathsf{c}_2 :: \Diamond_{cnj}\chi\multimap\Diamond_{cnj}(\chi)\multimap\chi^\star$	$\mathsf{c}_3 :: \mathrm{NP}$

$$\chi := \Diamond_{mod}\Box_{mod}(\mathrm{NP}\multimap\mathrm{NP})\multimap\mathrm{NP}$$

$$\chi^\star := \Box_{mod}(\mathrm{NP}\multimap\mathrm{NP})\multimap\mathrm{NP}$$

$$\mathsf{c}_2\ \triangle_{cnj}(\lambda \mathsf{x}_0.\mathsf{case}\ \triangledown_{mod}\mathsf{x}_0\ \mathsf{of}\ \mathsf{x}_1\ \mathsf{in}\ (\blacktriangledown_{mod}\mathsf{x}_1\ \mathsf{c}_3))$$
$$\triangle_{cnj}(\lambda \mathsf{x}_2.\mathsf{case}\ \triangledown_{mod}\mathsf{x}_2\ \mathsf{of}\ \mathsf{x}_3\ \mathsf{in}\ (\blacktriangledown_{mod}\mathsf{x}_3\ \mathsf{c}_1))$$
$$\mathsf{c}_0$$

'Thirty women and children'

Multiwords & Discourse Parts Multiword phrases not caught by our ad hoc transformations find their way to Æthel's derivations, polluting the lexicon with opaque, non-compositional phrasal types; in (III.18), for instance, the phrase *Op het laatst* is assigned a single type. This is suboptimal, but except for manually reannotating all multiword annotations, no immediate solution presents itself. The example is also interesting in showing how Æthel treats isolated non constituents torn apart from their phrasal context. The phrase is obviously a pruned adjunct of sorts, but the usual type assigment of a boxed endomorphism is impossible, since the phrasal head is absent. Not knowing what the wider phrasal type is, the extraction algorithm falls back to calling the non-constituent what it is: an adverbial. The same approach is followed universally, meaning that all of Æthel's samples derive atomic types.

(III.18) `WR-P-E-I-0000050381.p.1.s.531(3)`

Op het laatst	*van*	*de*	*oorlog*
At the end	of	the	war
$\mathsf{c}_0 :: \mathrm{ADV}$	$\mathsf{c}_1 :: \Diamond_{obj1}\multimap\Box_{mod}(\mathrm{ADV}\multimap\mathrm{ADV})$	$\mathsf{c}_2\Box_{det}(\mathrm{N}\multimap\mathrm{NP})$	$\mathsf{c}_3 :: \mathrm{N}$

$$\blacktriangledown_{mod}(\mathsf{c}_1\ \triangle_{obj1}(\blacktriangledown_{det}\mathsf{c}_2\ \mathsf{c}_3)\ \mathsf{c}_0$$

'At the end of the war'

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]. Back then, it didn't yet have a name, being only a partially worked out type lexicon – nevertheless, its statistical kinks were already apparent. The full resource gradually took form as a proof bank proper, and was initially described in Kogkalidis et al. [2020]. The chapter is a significantly more mature version of the paper, describing an overhauled version that shares the same general design philosophy but with lighter preprocessing (leading to longer and higher quality proofs) and wider coverage (improving linguistic variation). The most important feature of the overhaul is the revised extraction algorithm, reimlemented so as to ensure type safety, guaranteeing the formal correctness of the extracted proofs and fixing some suboptimal annotations 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 proof of concept in German [Hockenmaier, 2006]. CCGbank has since become a flagship for the applied categorial grammarian, spawning many offsprings, monolingual [Bos et al., 2009; Tse and Curran, 2010; Ambati et al., 2018, inter alia], and aligned [Bos et al., 2017; Abzianidze et al., 2017]. 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 III Bibliography

L. Abzianidze, J. Bjerva, K. Evang, H. Haagsma, R. van Noord, P. Ludmann, D.-D. Nguyen, and J. Bos. The Parallel Meaning Bank: Towards a multilingual corpus of translations annotated with compositional meaning representations. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 242–247, Valencia, Spain, Apr. 2017. Association for Computational Linguistics. URL https://aclanthology.org/E17-2039.

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. In L. Cornips and K. Corrigan, editors, *Syntax and Variation. Reconciling the Biological and the Social*, volume 265, pages 233–264, Nederland, 2005. John Benjamins Publishing. 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.

J. Bos, V. Basile, K. Evang, N. Venhuizen, and J. Bjerva. The groningen meaning bank. In N. Ide and J. Pustejovsky, editors, *Handbook of Linguistic Annotation*, volume 2, pages 463–496. Springer, 2017.

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

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

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.

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. Master's thesis, Utrecht University, 2019. URL https://studenttheses.uu.nl/handle/20.500.12932/32880.

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.

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.

R. T. Oehrle. Boolean properties in the analysis of gapping. In *Discontinuous constituency*, pages 201–240. Brill, 1987.

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. T. K. Sang, and V. Vandeghinste. Large scale syntactic annotation of written dutch: Lassy. *Essential Speech and Language Technology for Dutch*, page 147, 2013.

CHAPTER IV

Learning to Prove

Like sudoku, but with types.

This thesis was originally envisaged as a stepping stone toward an integrated approach at structural reasoning and meaning representation; a type-driven model of syntax reflected in a vector-based model of meaning. When the plans were first laid down, such a deliverable was not just theoretically possible, but also technically relevant – distributional semantics and word vectors were in their heyday, machine learning was rapidly taking off, parsers all of a sudden were becoming reliable, structured attention was a buzzword. Everything seemed to point towards the imminent bloom of a new era in natural language processing, where the wisdoms of old would meet the machines of today, hinting at a bright and prosperous future for neurosymbolic and structure-aware models of semantic composition. And things did seem to go that way, at least for a few years; but the advent of data efficient ultra deep architectures brought large language models into the game. These are unsophisticated, wildly over-parameterized, general purpose systems, fed unprocessed texts for weeks on end until they learn to convincingly imitate its use. With their sheer force, large language models usurped the heir apparent and condemned structure-aware semantic computation to obscurity. 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 statement of purpose before we get to the chapter's content – 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. But choice of format aside, a formal system operating on formal representations is not prone to implicit biases, latent variables, ambiguity, inconsistency, or any of the the modern pestilences of the sort. Erroneous outputs are the result of *bad input* or *bad programming*; there's always someone to blame. 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; for to imitate linguistic form is not to understand linguistic meaning. Parsing is therefore only superficially in competition with large language models, and its seeming obsoletion is just a by-product of ephemeral and rapidly shifting pop science trends.

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 could really ever hope for. But employing machine learning has to be thought of as either a shortcut or an admission of defeat, not as an end goal in itself. In opting for a machine learning solution, one assigns more faith to a generic data cruncher in pretending to solve a problem than to oneself in designing a solution to that same problem.

Interweaving symbolic and subsymbolic reasoning is then the responsible engineer's out. Complicated but decipherable components, rich in hierarchical or recursive structure, requiring or greatly benefiting from formal transparency are to be tackled explicitly. Components that are laborious but uninteresting, data intensive or intractable are to be isolated and outsourced to a machine worker. This partition promotes the expenditure of formal effort where it really is needed, while having it benefit from (rather than compete with) the high horsepower of brute force statistical machinery. 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, deviating from the original plan in going not from structures of form to vectors of meaning, but from vectors of form to structures of meaning. In practical terms, we'll go through the hoops of designing and implementing a formally disciplined but accurate and robust wide-coverage parser, a neurosymbolic architecture aimed at substructural logics of the linear lineage, instantiated here for $\mathbf{LP}_{\diamond,\Box}$ and trained on Æthel.

13 The Categorial Parser

A high-level conceptualization of the categorial grammar parser should make for a good starting point. 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 assignments 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).

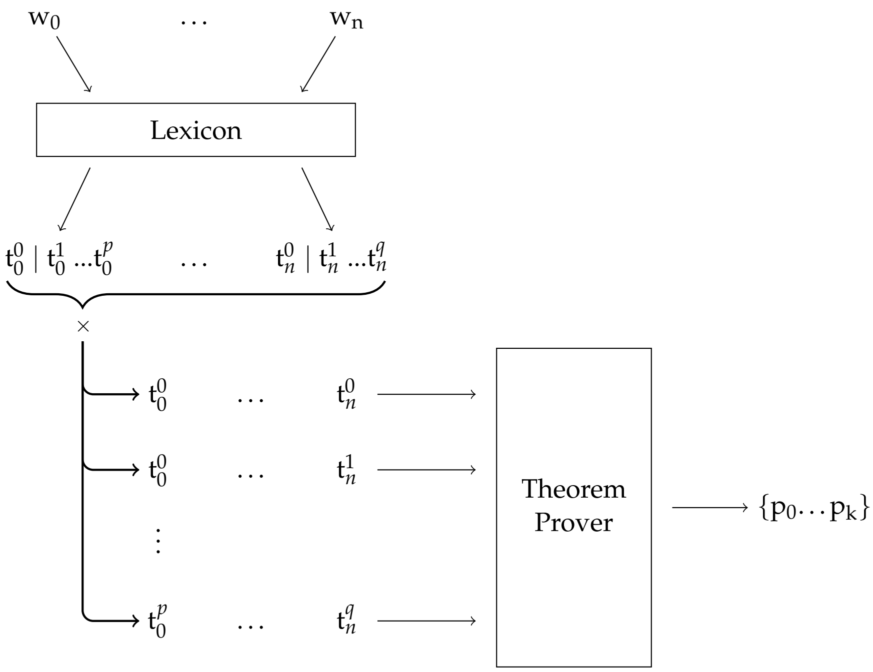


Figure IV.1: The archetypical categorial grammar parsing pipeline.

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 key will have a high 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. in cases of a ⟨word, type⟩ pair missing from the lexicon). Relaxing the structural properties of the type system to ease lexical pressure is not a panacea either. With the parser becoming increasingly ambiguous, more possibilities become accessible during search, and putative proofs become harder to reject. 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 play together peacefully.

For the categorial program to come to fruition, these very real problems require equally real solutions. The practitioner must often resort to tricks aimed at compressing or efficiently navigating the enormous search space. The modern pipeline commonly outsources lexical disambiguation to a statistical component, referred to as the *supertagger*. The supertagger is tasked with ranking the possible assignments to a single key, 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 candidates returned are truncated depending on some threshold likelihood or 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 it hits a decision point. As with all real solutions, perfect is unattainable; this time/space efficiency usually comes at the cost of approximation errors that translate to 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.

A foreword before we get to it: I imagine a crash course in machine learning to be redundant in the current day and age. In any case, my intention is to help the purists make sense of what's going on, yet without obfuscating the implementation from fellow hackers. To that end, I'll try keep the gory technical details contained and separate from the high level, abstract descriptions. I hope the result is sensible and inclusive.

14 Supertagging

A supertagger is a statistical model, a parametric function f_θ tasked with producing the most likely type assignment sequence $\mathbf{t}_{0:n} := t_0 \ldots t_n$ for a given sentence $\mathbf{w}_{0:n} := w_0 \ldots w_n$.

$$f_\theta(\mathbf{w}_{0:n}) \approx \underset{\mathbf{t}_{0:n}}{\operatorname{argmax}}\; p(\mathbf{t}_{0:n} \,|\mathbf{w}_{0:n}, \theta) \tag{IV.1}$$

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:

$$p(\mathbf{t}_{0:n} \mid \mathbf{w}_{0:n}) \tag{IV.2}$$

The decomposition of the seemingly innocuous (IV.2) will basically monopolize this section, because a choice of assumptions and truncations is a prerequisite for us to even contemplate the model's implementation. Each choice can (and will) have a deep impact on the model's performance, most notably in phenomena inhabiting the more remote regions of the probability density function's landscape. As a corollary, each choice will alter how suitable a model is to one single grammar depending exactly on how that landscape looks. 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 at least 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

To actually perceive the structure, we must first notice its absence – therefore (and for maintaining suspense), 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) is due to the 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), assigning the correct grammatic descriptor, or *supertag* (in our case, a type), to each word in a sentence amounts to *almost* parsing, and that even just weeding out some of the erroneous assignments significantly facilitates parsing. Early literature was characterized by an almost single-minded attachment to localized computation, the justification being that supertagging must remain localized for it not to become "too much like parsing" [Bangalore and Joshi, 1999]. With the benefit of hindsight, we can see this for what it was: an attempt to justify a pragmatic consideration, and an artifact of the times, with the scene then largely dominated by window-based models.

A so-called unigram model assumes full independence between subsequent words, and (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. Despite competely breaking apart sequential sparsity, this formulation is not much good on its own either; rarely occurring lexical keys hardly provide sufficient data for an empirical distribution to be extracted. 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. The resulting model is, alas, too simple to find real use: the assumptions made are exceedingly naive, and lexicalization is heavily bottlenecked by the coarse and undescriptive part of speech tags; we need to do better.

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

$$\propto p(\mathbf{w}_{0:n} \mid \mathbf{t}_{0:n})\ p(\mathbf{t}_{0:n}) \tag{IV.4}$$

Extending the context to a window of size κ, allows local decisions to excert direct influence to the next κ predictions (and thus indirectly affect all future ones). This requires approximating the *contextual probability* $p(\mathbf{t}_{0:n})$ as:

$$p(\mathbf{t}_{0:n}) \approx \prod_{i}^{n} (t_i \mid \mathbf{t}_{i-\kappa:i-1}) \tag{IV.5}$$

Going one step further and making the assumption that the *emission probability* $p(\mathbf{w}_{0:n} \mid \mathbf{t}_{0:n})$ is position-separable and independent allows its rewrite to:

$$p(\mathbf{w}_{0:n} \mid \mathbf{t}_{0:n}) \approx \prod_{i}^{n} p(w_i \mid t_i) \tag{IV.6}$$

Putting (IV.5) and (IV.6) together, we get an approximation to (IV.2) that is blatantly wrong. Nevertheless, it is also workable, in having efficiently circumvented sparsity, adequate, in having accounted for the very important axis of output-to-output interactions, and practical, in allowing a tangible implementation as a hidden Markov model. Simple structural constraints would then be applied to filter out candidate predictions on the basis of admissibility criteria related to the shape and content of the supertag as well as the surrounding lexical context.

14.1.2 CCGbank and the Original Sin

The problem garnered attention and gained significant traction with the release of the CCGbank, the large size and gold standard nature of which offered an excellent test bed for molding the first generation of supertaggers. The first incarnation of a combinatory categorial grammar supertagger was the original work of Clark [2002]. The model diverged from the implementation of Bangalore and Joshi [1999] in opting for a larger window size and foregoing the contextual effect of output-to-output dependencies. In that setting, and for a window of size $2\kappa + 1$, (IV.2) takes the form:

$$\prod_{i}^{n} p(t_i \mid \mathbf{w}_{i-\kappa:i+\kappa}) \tag{IV.7}$$

The model would materialize as a log-linear feature weighter trained as a maximum entropy estimator. The input would include several sparse heuristics, including morphological features and boolean context predicates, allowing a partial soft bypass of the fixed-key lexicon.

Novelty and ingenuity aside, the work set a number of precedents; some of those, reasonable as they may have been at the time, have since permeated through the problem statement, becoming de facto practices rather than conscious design decisions. Structural constraints were dropped, in part because they are less straightforward to deduce in frameworks other than tree adjoining grammars; they never found their way back into the mainstream, athough admittedly they never were particularly sophisticated to begin with. This step away from structural discipline is exacerbated by having also dropped the supertag-to-supertag dependencies, since the model now has no chance of learning how to statistically filter out mutually incompatible assignments either. To counteract the problem, the paper opted for a yet more radical solution: abandoning the sequential formulation (i.e. no more argmaxing over the product) in favor of a *multitagging* approach (i.e. returning all categories whose local probability exceeds some fixed ratio of the highest ranked candidate). This was shown to greatly improve coverage (by outsourcing heavier duty to the parser), but has to be understood as a practical overcorrection, an emergency measure to sidestep the model's inherent disregard for output-level sequential interactions. The limitation is acknowledged by Clark and Curran [2004], and an attempt at resolution is offered by Curran et al. [2006]. There, the forward-backward algorithm is employed to efficiently calibrate the probability of an assignment (in the multitagging setup) as the sum of all sequential assignments containing it:

$$p(t_i \mid \mathbf{w}_{0:n}) = \sum_{\mathbf{t}_{0:i-1}, \mathbf{t}_{i+1:n}} p(\mathbf{t}_{0:i-1}, t_i, \mathbf{t}_{i+1:n} \mid \mathbf{w}_{0:n}) \tag{IV.8}$$

This does reinstate a notion of output-to-output dependencies in the form of estimated posteriors, but computational considerations have diffused the potential for widespread adoption in later frameworks. Finally, rare supertags, which were particularly problematic or near impossible to learn, were found to have very limited impact on overall coverage; this set the grounds for their statistically near-inconsequential erasure, a choice that gradually became ingrained as a mandatory step of data sanitation and preprocessing.

14.1.3 Distributed Word Vectors & Neural Networks

The advent of word embeddings and the gradual substitution of sparse features with continuous vectors paved the way for the incorporation of artificial neural networks. Lewis and Steedman [2014] employed a collection of pretrained embeddings combined with a window-based two-layer network in a "semi-supervised" manner (in today's jargon, a pretty much fully supervised separable convolution), to a dual effect. On the one hand, the pretrained embeddings offered a natural generalization from the fixed size lexicon to the (still fixed, but much larger) set of pretrained' embeddings, single handedly obsoleting the long standing problem of tackling rare and unseen words [Thomforde and Steedman, 2011; Deoskar et al., 2011, 2014]. On the other hand, the parameter-sharing convolution improved the accuracy/ambiguity ratio and overall efficiency of the (then standard) log-linear supertagger of Clark and Curran [2007]. Unlike before, words were allowed to associate to any supertag, regardless of whether or not a ⟨word, supertag⟩ pair was observed during training; the lexicon thus turning from a hard imperative to a soft guideline. Additional experiments involving a conditional random field were mildly successful, but abandoned due to the prohibitively slow decoding – yet, it was obvious to the people involved that something critical was amiss. Xu et al. [2015] took the approach a step further by utilizing a simple recurrent network (RNN), and, in doing so, claimed to sidestep the locality of the previous neural model. To escape the unidirectional constraint of the classical recurrent network (or perhaps out of force of habit?), they continued incorporating window-based features that provided a minimal amount of right context κ, thereby rewriting (IV.2) as:

$$\prod_{i}^{n} p(t_i \mid \mathbf{w}_{0:i+\kappa}) \tag{IV.9}$$

And while their approach does indeed offer a wider receptive field, it is focused solely on the input side; output interactions are still nowhere to be seen.

14.1.4 Autoregressive Modeling

By now (and despite earlier aphorisms), it is becoming increasingly evident that nothing deep or spiritual restricts supertagging to remaining local, as advancements in machine learning are progressively offering more opportunities for fast and efficient incorporation of ever wider context. But the absence of output-to-output dependencies remains unresolved, despite them being a recurrent theme in the literature. This changes with the work of Vaswani et al. [2016] who score two major points with their resourceful use of long short-term memory networks (LSTM). First, they replace the simple recurrence of Xu et al. [2015] with a bidirectional one (thus allowing unbounded left- and right- input interactions) – an idea explored in parallel by multiple contemporary works [Ling et al., 2015; Xu et al., 2016; Lewis et al., 2016, inter alia]. More importantly and in addition to that, they introduce an intermediate recurrence that is to serve as a supertag-level language model, fusing the prediction history with the input context to produce each local prediction. The two together alter (IV.2) into a version far more elaborate than previous proposals:

$$\prod_{i}^{n} p(t_i \mid \mathbf{t}_{0:i-1}, \mathbf{w}_{0:n}) \tag{IV.10}$$

Under a modern lens, this is akin to a somewhat idiosyncratic implementation of an autoregressive sequence-to-sequence model, with the decoder benefiting from perfect alignment between input and output tokens. Unlike prior work, no occurrence threshold was imposed, and explicit evaluations over sparse lexical relations were provided. The added expressivity and significantly wider receptive field granted LSTM models the state of the art badge, which was to remain uncontested for a surprisingly long two human years[1].

14.1.5 Superwhat?

More than just a testament to the LSTM's strengths, this momentary pause makes for a discontinuity in the velocity of progress; not because people suddenly lost interest in supertagging, but rather because machine learning architectures and their applications had slowly become exhausted. This coincides with a stall across NLP in general, and a concurrent paradigm shift; specialized models started becoming outfashioned, and improvements would no longer be enabled by domain expertise and task-specific engineering, but rather by higher quality unsupervised and semi-supervised training routines over larger and larger models. A case in point is the next major landmark, in fact reached by a structurally simplified model [Clark et al., 2018], a plain bidirectional sequence encoder using the factorization:

$$\prod_{i}^{n} p(t_i \mid \mathbf{w}_{0:n}) \tag{IV.11}$$

Despite taking a step backwards in terms of structural sophistication, the model managed a performance leap comparable to that of switching from a separable neural function to a recurrent one (see Figure IV.2), all by "simply" incorporating multiple tasks and losses in its training loop. The same paradigm is today more dominant than ever, and has pushed conventional NLP outside of the spotlight, putting an end to an exciting but short golden era.

[1] Approx. three centuries in machine learning years.

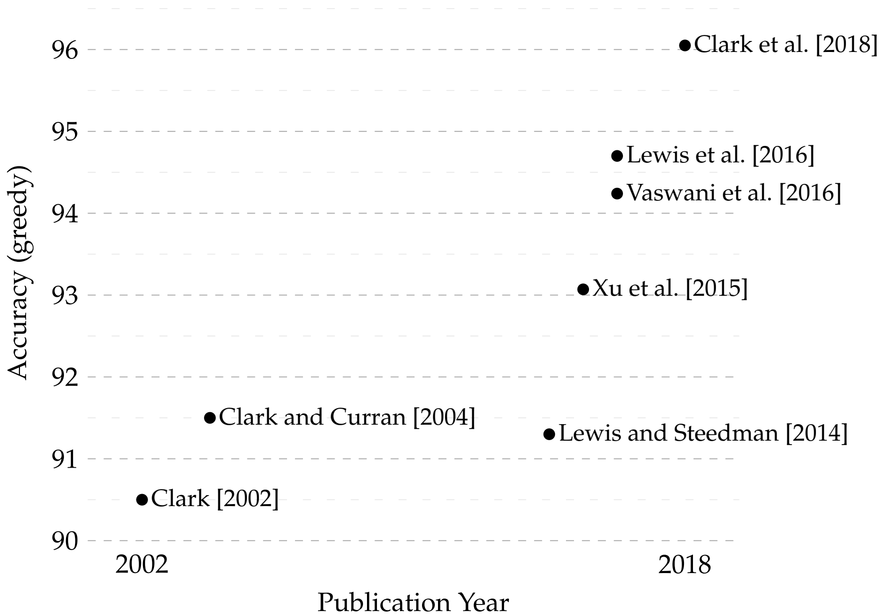


Figure IV.2: Supertagging performance in the CCGbank historically.

14.2 Constructing Types

The supertagging architectures reviewed are, from first to last, variations on a theme. Regardless of whether the underlying statistical machinery is a hidden Markov model, a maximum entropy model, a neural sequence tagger or a sequence-to-sequence transducer, a single commonality characterizes them all: they start from the assumption of a finite codomain. More than that, they don't just assume but *require* that the zipfian tail of lexical type sparsity is practically irrelevant for the corpus, and, by extension, for language at large. In other words, they require that most of the probability mass of type occurrences is concentrated around a central region of a few common types, and that exceptionally rare types are nothing but statistical artifacts which can safely be ignored. This bias is not to be mistaken for a vice, nor for a deeply motivated ruling; it is a practical compromise that became an unwritten rule, similar to the (once proclaimed as necessary) locality of supertagging – a notion since abandoned and forgotten with minimal remorse and deliberation as soon as technology allowed. The issue is really quite shallow: statistical models have always had a very hard time dealing with under-represented samples (in our case, supertags), and correctly recognizing items outside the training data is an open problem with no general solution.

That is not to say the compromise is an unjust one; its heedless proliferation does come with two major side effects, though. One, it forces parsers to give up on potentially rare syntactic phenomena, at least when those manifest through unique and uncommon supertags. Even though a parser should in principle be able to handle any valid supertag (regardless of its statistical properties), the a priori exclusion of rare ones corresponds to an externally imposed restriction to its generalization. In other words, exactly those difficult phenomena that would benefit from the linguistic expertise of a robust parser are to be discarded in the first place. There's a bit of a self-fulfilling defeatism here: we'll always only parse what we *can* parse, sure, but we'll never be able to parse what we won't ever *try* to parse. Two, in becoming part of the first page of the (as of yet unwritten) supertagging bible, the concept implicitly reinforces the belief that grammars not following a distribution of occurrences similar to (or denser than) that of the CCGbank are practically unusable. A densely featured type system and its overpopulated lexicon have become demonized as pitfalls we have to steer away from. This is actually quite the contradiction – we came up with supertagging to treat lexicalized grammars, but we won't push lexicalized grammars further because of a lingering fear that our supertaggers are not good enough; lexicalization is good, but only as long as it's not too lexicalized...

Epistemological ramblings aside and back to reality, the type grammar we have developed is not among the lucky few. Æthel is way sparser than the CCGbank, containing five times as many types, while test samples with at least one rare type (i.e. a type with less than 10 occurrences in the joined train and dev subsets) appear four times as often as in the CCGbank (14.5% vs 3.5%). Disregarding rare types is making ourselves content with an idealized (unachievable) peak sentential parsing accuracy of 86.5%, which is far from an aspiring start. Worse yet, these statistics are suggestive of a vast type universe, of which we likely have a observed only but a glimpse through the lens of Æthel. In practical terms, we messed up, and no existing technology will save us now. But, as the proverb goes, "necessity is the mother of invention" and we're definitely in need here, so we may as well invent something.

In reality, what we need is less of an invention and more of an observation. The important thing to observe is that supertags (be them combinatorial categories, type-logical types or anything resembling them) are not ad hoc, opaque and dissimilar *units*, but highly regular, transparent and decomposable *structures*, made of a small set of primitives and the operations that piece them together. In our setup, complex types are the result of type forming operators applied to "smaller" types, the smallest types available being atomic propositions; recall the (strategically placed) exercise of Figure I.30. This insight is not a particularly deep one; it won't come as a surprise to anyone that has even superficially dabbled in the joys of algebraic data types, context-free grammars, inductive tree structures, or any sort of the hierarchical recursions common in computer science. Theoretically unsurprising as it may be, it offers an interesting applied perspective: why teach a statistical model how to *disambiguate* (i.e. choose) between some candidate assignments (however many), when we could instead teach it to *construct* (i.e. inductively describe) the most suitable assignments instead? A system able to consult the present linguistic context in order to construct well-formed and well-motivated types would amount to the first ever specimen of a new species: a supertagger with an unrestricted codomain. We'll call this species of supertaggers *constructive*. This perspective is in a sense orthogonal to the transition from fixed, corpus-extracted assignment frequencies to word vectors. Whereas one generalizes over rare and unseen items in the first coordinate of lexical entries (⟨word, type⟩ pairs), the other does so over the second. The two, combined, lift the closed world assumption, paving the way for the last supertagger we'll ever need – one able to reliably predict the correct type (be it rare or unseen) for any word (be it rare or unseen).

14.3 Supertagging as NMT

The first attempt at a constructive supertagger is described in detail in Kogkalidis et al. [2019]. Like all first attempts, it is characterized by a degree of naivety combined with an overeager execution. Types are first viewed as the corresponding formula trees, and then traversed in a depth-first left-first fashion (i.e. read off in Polish notation). Each type thus yields a type-word, viewed as the produce of a tiny recursive grammar (a context free one), the alphabet of which would be the union of atomic formulas and logical connectives. A sequence of types is represented as the concatenation of the sequentialized types, each type-word separated from the next by an (in hindsight unecessary) special alphabet token. This expansion of a type-word into multiple symbols inadvertently breaks the input-to-output alignment; words are no longer associated to a single output symbol, but rather a sequence thereof. As expected, this means that a sequence tagger is no longer a fitting backend for our experimental ventures. Thankfully, the two biggest buzzwords of machine learning in 2018 are both surprisingly relevant here.

14.3.1 Buzzwords

Neural Machine Translation Neural machine translation (NMT) is the modern paradigm to machine translation, the task of automatically translating text from some source language to a target one. The term made its explosive first appearance halfway through the last decade, taking the field by storm [Kalchbrenner and Blunsom, 2013; Cho et al., 2014b; Bahdanau et al., 2015]. The dominant approach rests on a sequence-to-sequence neural model [Cho et al., 2014b; Sutskever et al., 2014], which consists of two parts: a sequence *encoder*, which builds a contextual representation of the input sequence, and a sequence *decoder* which uses the input representation to iteratively produce the output sequence on a token by token basis. For an input sequence $\mathbf{x}_{0:M}$ mapped to an output sequence $\mathbf{y}_{0:N}$, this corresponds to a conditional language model trained to maximize

$$p(y_i \mid \mathbf{y}_{0:i-1}, \mathbf{x}_{0:M}) \tag{IV.12}$$

The above conditional is identical to (IV.10); in fact the supertag language model of Vaswani et al. [2016] *is* a degenerate case of neural machine translation, where **y** is **t** and **x** is **w**, and M and N coincide. This is not a one-off, but rather an instace of a broader trend, referred to as *generalized* machine translation. The generalized part stems from the fact that neither the source nor the target language are in any way constrained to being natural (or human) languages; either of the two (or both!) may well be artificial languages. The actually interesting bit is that they don't actually even need to be languages *per se*; any complex data structure that can be canonically traversed into an unambiguous sequentialization makes for a valid input/output. Vinyals et al. [2015] explore the idea in training a sequence-to-sequene parser by directly translating the input sentence into a linearized constituency tree; the model is surprisingly accurate in learning both *how* to create valid trees (only occasionally producing malformed output), and *which* valid tree to create for a given sentence (with an accuracy comparable or matching previous established models).

The paradigm per se is rather bland, making no assumptions about the output structure and requiring little to no task-specific tuning. For the exact same reasons, it is also highly appealing, and a good starting point for experimentation – we can just apply it virtually unchanged to the task at hand. In our domain, the goal sequence **y** would be the sum of symbols together forming our sequence of type-words, and **x** will be none other than the sentence itself. Using the doubly indexed $s_{i,j}$ to denote the j-th symbol of the i-th type (symbol enumeration following the depth-first left-first traversal of the formula tree), the conditional becomes:

$$\prod_{i}^{n} \prod_{j}^{||t_i||} p(s_{i,j} \mid s_{k,:} : k < i, s_{i,k} : k < j, \mathbf{w}_{0:n}) \qquad \text{(IV.13)}$$

where $||t_i||$ the number of symbols of type-word i. We will refer to this operationalization as a *symbol sequential* supertagger. Note that the above is essentially an expanded version of (IV.10), in the sense of containing intermediate evaluations in between full supertags. This view allows drawing a parallel between type-words made of primitive symbols and words made of subword units [Sennrich et al., 2016]. The two share the same high-level purpose of improving "translation" to rare (type-)words, even though the structural decomposition of types is much more regular and consistent than the morphological decomposition of words.

Neural Attention Encoding the input sequence to a fixed length vector is essentially lossy neural compression. The longer the input and output sequences are, the more this compression may prove catastrophic in capturing long range dependencies [Cho et al., 2014a]. As an alternative, attention-based models circumvent the need for compression by simply building a contextually informed representation of the full input, distributed evenly among its tokens (one representation per sequence element). These representations can be dynamically weighted and summed, yielding a distinct view of the same structure based on an external aggregation context (a query). Attention has its roots in neural image processing [Larochelle and Hinton, 2010; Mnih et al., 2014, inter alia], but its application to language was essentially the catalyst that set the field ablaze [Bahdanau et al., 2015].

Even though attention was originally used as an enhancement on top of RNNs, the code of conduct today is basically attention only. The instigator of that paradigm shift was the transformer architecture [Vaswani et al., 2017], which by now enjoys an unprecedented pop status (saving me the hassle of having to regurgitate yet another "transformers explained" pamphlet). In high level terms (and consciously oversimplifying), the transformer is a heteroassociative memory mechanism. It builds three distinct representations for each sequence token: *queries* dictate what each token looks for, *keys* dictate what each token associates with, and *values* correspond to memory storage. A distance metric (commonly a scaled dot-product) is used to induce a weighting over the keys matrix for each query vector; we may say that queries *attend* to keys. The resulting weights are normalized to sum to one, and act as multiplicative factors in the weighted averaging of the values matrix, yielding a vector acting as a distinct evaluation of the full sequence for each query. This basic operation is trivial to parallelize, both across tokens within the same sequence, as well as across independent sequences, thus allowing an efficient many-to-many message passing contextualization that can be stacked multiple times in depth for extra expressivity. Using this as a decoder is just as easy, since queries may come from a different sequence than keys and values, provided their dimensionalities (not the counts!) match. The only requirement is a masking strategy that disallows autoregressed tokens from attending to their future while training (since that would be cheating). This is significant for training in the NMT setup, as it circumvents the linear temporal delay of the RNN by trading it for the quadratic memory cost of the attention matrix (quadratic because all tokens must attend to all tokens); the trade-off does not carry through to inference, where one has to suffer both the temporal delay (since there's no oracle supplying the future anymore) as well as the memory penalty.

14.3.2 Implementation

Our problem is ripe with long distance dependencies. Moreover, these are not confined to being only between encoder-decoder token pairs, but may also occur within decoder token pairs alone. Consider that the misalignment between input and output means that we must consult the full input sequence at each decoding step, while the structurally liberal type logic means that cues to the current step may be found locally (within the same type), or multiple types (and thus even more steps) away. For this reason alone, the transformer seemed like a good candidate architecture. Adhering to evidence that pretrained language models seem to benefit either side of the encoder-decoder pipeline, the encoder would consist of a Dutch version of ELMo, the de facto language model at the time [Peters et al., 2018; Che et al., 2018]. To account for domain adaptation without having to compute the costly gradient updates for the over-parameterized language model, a single transformer encoder was used to contextualize ELMo's precomputed representations. The encoder was connected to a tiny transformer decoder of two layers, allowing unhindered access to the full input and all previous outputs.

14.3.3 Experiments & Results

Training The model was trained with teacher forcing, i.e. predicting the current step assuming perfect rather than predicted (noisy) context. For regularization, and in order to discourage the model from memoizing common type patterns, the Kullback-Leibler divergence was employed as the loss function, computed between the model's predictions and the ground truth, with 20% of the probability mass evenly distributed across the non-true entries (basically a naive implementation of the label smoothed cross entropy loss [Szegedy et al., 2016]). The training data would consist of samples counting less than 20 words, pulled from the version of Æthel then current. This historical version of the dataset diverges considerably from its present incarnation, the core difference being the use of **LP** as the type logic, with an informal decoration of the implication standing for today's modalities. Despite formal and representational divergences, the distribution of types is practically identical in between the two versions; as a fun trivia, only about 85% of the total unique types were present in the training split used. Modulo exact numbers, insights gained from this past venture do carry over to the present.

Evaluation Unlike work in CCGbank, evaluation cannot be done on a comparative basis, due to the absence of established baselines[1]. Cross-framework comparisons are also irrelevant due to the vastly different problem formulations (i.e. different linguistic framework, corpus, language); to drive the point across, consider that accuracy was measured over a set of 5 700 types, which is 1 order of magnitude above CCGbank's 425 non-thresholded categories. What's worth exploring instead is (i) the architecture's potential at supertagging, and (ii) its ability to learn reasonable generalizations beyond its training data. To that end, we may view constructive and discriminative supertagging not as two orthogonal approaches, but as the extreme points of a continuum. At the intermediate points between these extremes, there exist alphabets containing composite symbols that correspond to notational shorthands for the most common type and sub-type patterns. As more of notational shorthands are introduced, the target output's length is significantly decreased, but the

[1] There's basically noone to beat.

model is exposed to progressively less constructions of full types. This becomes useful in approximately mapping the landscape between a fully constructive supertagger and a fully discriminative one.

On a purely numerical basis, the results are not astounding. Constructive accuracy lies at a disheartening 88%, which is far from sufficient for downstream parsing. What is intriguing, though, is that accuracy gradually declines with the introduction of notational shorthands, falling all the way down to 87.2% with the eventual collapse to a discriminative autoregressive tagger. Let's repeat this once more: obfuscating type structure hinders performance. The story looks even more promising when it comes to the far end of the zipfian tail: 19.2% of type assignments involving unseen types are correctly predicted, as are 45.7% of those involving rarely seen types; these plunge to an unavoidable 0 and 23.9%, respectively, with the transition to a discriminative setup. Furthermore, not a single type is malformed, indicating that the grammar of type formation is indeed learnable, even when incorporated within a challenging sequence labeling task. Raw numbers aside, the results suffice to deem the experiment an objective success: we generated concrete evidence that a full dismantling of the lexicon is not just *possible* but in fact also *beneficial* for supertagging a sparse type grammar.

14.3.4 Insights & Observations

Advantages The prime advantage is the acquisition of rare and unseen supertags, which is a major accomplishment in its own right. Secondary advantage #1 is the unintended provision of trained representations for zeroary and n-ary primitives[1], either contextual (i.e. as provided by the decoder) or standalone (i.e. as provided by the embedding layer). In the first case, they enact contextual representations that live in the disputed zone between the input sentence and the output derivation, suggesting new routes to parsing – we'll see about that in Section 15. In the second case, they may find use as high-granularity supertag representations, allowing the dynamic representation of *any* valid supertag, akin to character-level embeddings for a character level model – supertag representations could then find use in downstream applications as an extralingual input [Kasai et al., 2017]. Secondary advantage #2 is the possibility for a hyper-articulated heuristic search during decoding, as we are now able to branch off to different sequences of assignments by sampling not only *across* types, but also *within* them. A different symbol might drastically alter and affect the future of the decoding, locally within the current type or globally across the full sequence. Other than potentially improving the sample efficiency of beam search, this can further be used to strictly enforce structural constraints, as we will also see in Section 15.

[1] Replace with appropriate framework-specific terminology, e.g. atomic propositions and logical connectives, atomic categories and categorial combinators, etc.

Limitations With the benefit of hindsight, it is also clear the approach suffers from a series of limitations. First and foremost, there's the superficial fact that overall accuracy is far from groundbreaking, pointing to the need for architectual search and hyper-optimization adjustments. A deeper issue is the computational penalty of the naive application of the transformer; unfolding supertags to primitive symbols has added a second product in the formulation of (IV.10). The sequential decoding inherited from NMT means that this extra product excerts a multiplicative influence to decoding time, made quadratic in terms of memory footprint. The model is computationally expensive, slow and bulky to optimize. At the same time, we have not fully kept our initial promise; structure may have been revealed, but it was not paid the respect due. Supertags were brutally leveled into one-dimensional decals, their original treeness reflected neither in the representations nor in the structural inductive bias of the learning machine. We still have to do better.

14.4 Geometric Constraints

Prange et al. [2021] notice the problem and seek to resolve it by explicating the categorial tree structure. Their methodology abides by the encoder-decoder paradigm, but with one crucial, task-specific adaptation: the decoders experimented with are *tree recursive*, making them a far better fit for addressing the problem at hand. The general setup has the encoder build a contextualized representation for each word in the input, which is to serve as the initial seed for the decoding of the respective supertag; the decoder is then independently applied among all trees. Two decoders are considered; a tree-shaped variant of the gated recurrent unit [Cho et al., 2014a] and a positionally informed feed-forward network. The first recurses along the tree structure, generating each local symbol dependent on its direct ancestor. The second sums the initial seed with the projection of a feature vector describing the local position and its ancestry (both fixed choices among some predetermined possibilities).

The approach makes for a well motivated step in the right direction. The new formulation completely eliminates the burden of *how* trees are constructed, allowing the model to focus on *which* trees to construct. At the same time, the decoders considered are now token-separable, i.e. they can be applied in parallel across both sequences and trees. Where previously we would have to perform $\sum_i^n ||t_i||$ decoding steps, this now shrinks to $\max_i^n \text{depth}(t_i)$ – practically a constant, and a reduction of at least one order of magnitude. Furthermore, words and supertags are now structurally aligned, relieving the model from having to learn the implicit soft alignments necessary at each decoding step. On the practical side, numbers are significantly improved across the board (except for the far end of the zipfian tail), making the model a real alternative to the discriminative status quo. This becomes even more relevant considering how easily the setup lends itself to the multitagging paradigm (an insight that escaped the authors), as multiple trees may be obtained by following along the path of the factorization (modulo accounting for depth-width smoothing):

$$p(t_i \mid \mathbf{w}_{0:n}) = \prod_{j}^{||t_i||} p(s_{i,j} \mid s_{i,k} : k \in \text{ancestors(j)}, \mathbf{w}_{0:n}) \qquad \text{(IV.14)}$$

All these merits come, however, at a heavy price: in parallelizing decoding across trees, the architecture loses the ability to model auto-regressive interactions between output nodes belonging to *different* trees; interactions that can be crucial at the granularity scale we are now at. The task is morally reduced to a sequence classification once more, albeit now with a dynamically adaptive classifier; we are back at (IV.11), except for each local decision being elaborated according to (IV.14).[1]

The sequential and tree-biased approaches seem to be at odds, but the tension between them is highly artificial. Both merely suffer from the naivety of conflating problem-specific structural biases and general purpose decoding order: one forgets about tree structure in opting for a sequential decoding, whereas the other does the exact opposite, forgetting about sequential structure in opting for a tree-like decoding. What we need to do is disentangle the two concepts, observing first that the output type is neither $\mathsf{Seq}[\mathsf{s}]$ nor $\mathsf{Tree}[\mathsf{s}]$ but $\mathsf{Seq}[\mathsf{Tree}[\mathsf{s}]]$. And that's it. Having done that, the work that remains is of purely technical nature; we just need to come up with the spatiotemporal dependencies that abide by *both* structural axes, and then a neural architecture that can accommodate them. The choice of a temporal (decoding) order is easy: Prange et al. [2021] make a very compelling case for depth-parallel decoding, given that it's insanely fast (we are not temporally bottlenecked by left-to-right sequential dependencies) but also structurally elegant (trees are only built when/if licensed by non-terminal nodes, ensuring structural correctness virtually for free). Sticking with depth-parallel decoding means necessarily foregoing some autoregressive interactions: we certainly cannot look to the future (i.e. tree nodes located deeper than the current level, since these should depend on the decision we are about to make), but neither to the present (i.e. tree nodes residing in the current level, since these will be all decided simultaneously). This leaves some leeway as to what could constitute the decision context, and here's where we can improve upon prior work: in adding the missing structural dependencies. The maximalist position is nothing less than the entire past, i.e. *all* the nodes we have so far decoded. Crucially, this abolishes conservative ancestry biases, establishing "diagonal" structural interactions between autoregressed nodes without requiring them to be directly linked to one another, or even share the same ancestral heritage (belong

[1]Interestingly, Liu et al. [2021], who concurrently explore a similar operationalization with tiny word-level parsers, consider this a strength, arguing that it helps counteract error accumulation.

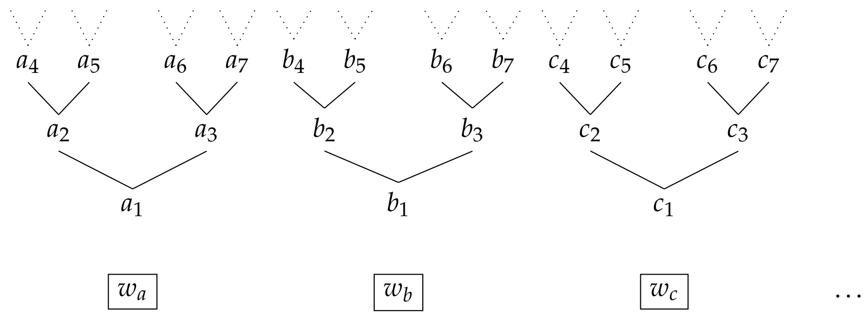


Figure IV.3: Abstract canvas of a constructive supertagger's I/O structure.

to the same tree). The liberal position casts (IV.2) to:

$$\prod_{i}^{n}\prod_{j}^{||t_i||} p(s_{i,j} \mid p(s_{:,k} : \text{level}(k) < \text{level}(j), \mathbf{w}_{0:n}) \qquad \text{(IV.15)}$$

The point might seem stretched but it is really just subtle. If you're having trouble following along, take a look at Figure IV.3, displaying an abstract (partial) canvas of the constructive supertagger's input/output space, where w_a, w_b, w_c are the first three words of the input sequence, with corresponding goal trees a, b and c, the nodes of which are enumerated according to a depth-first left-first traversal. Focusing on autoregressive interactions alone, the sequential approach we started from would have each node depend on all nodes to its left and below; without loss of generality, b_6 would for instance depend on all of a, but also b_1, b_2, b_3, b_4 and b_5, as well as any descendants of the last two. The tree-biased approach would have each node depend on its ancestors; for b_6, these would be just b_3 and b_1. The tree-sequential approach envisaged here has each node depend on all nodes below it; the prediction of b_6 is now informed by the contents of nodes $[a/b/c/\ldots]_{1,2,3}$. The convention is that shallow nodes (presumably the easiest ones) are decoded first, unraveling the next layer of the canvas (we won't need to waste any compute on predicting, say, b_6 if either of b_3 and b_1 was a terminal symbol), while providing disambiguation context for deeper nodes (presumably harder) along the entire sequence.

14.4.1 Geometry-Aware Supertagging

A suggestive operationalization of this novel approach was made public in Kogkalidis and Moortgat [2022, preprint]; we'll expand upon it here. First off, the spatiotemporal dependencies we seek to implement do not follow the inductive biases of any run-of-the-mill architecture we may find precompiled in some machine learning library. The closest paradigm available are graph neural networks (GNNs), which are essentially the most general class of neural architectures, suitable for learning on arbitrary graphs and manifolds (points, sequences, canonical grids, trees – these are all just very specific instances of graphs: every neural network is a subclass of a graph neural network). GNNs are usually formulated on the basis of some graph structure, where primitive graph entries (edges, nodes or both) are iteratively updated in a series of so-called *message passing* rounds. The concrete implementation of the messaging scheme (including what the flow of communication is and how messages are constructed) are up for deliberation.

In our case, it would be straighforward to add direct messaging components that implement exactly the spatiotemporal dependencies described earlier. But this lacks subtlety, making no attempt at exploiting the regularity of the output space; sure – it may be neither sequence nor tree, but it's not an ad-hoc graph either! Computationally, this would not bode very well either; the number of interactions to compute would be upper bound by the series:

$$\sum_{k=0}^{m:=\max_i^n \text{depth}(t_i)} \Big(\underbrace{2^k n}_{\text{\# prediction targets}} \times \big(\underbrace{\textstyle\sum_{k'=0}^{k-1} 2^{k'} n}_{\text{\# context nodes}} + \underbrace{n}_{\text{input length}} \big) \Big) \tag{IV.16}$$

whose memory footprint grows as $O(2^{2m}n^2)$, scaling quadratically with sequence length and exponentially with twice the maximal tree depth – yikes. To keep this beast under check, we would do well to utilize the output's geometric constants, namely the words. A reasonable way to do that would be as state tracking vectors (fixed both in count and in length). Akin to RNN hidden states, these shall be iteratively updated by the decoding process, while simultaneously reining it in. Practically, each decoding step shall be conditioned on the current states, with each state (word) informing only the nodes it is associated with (the supertag it will decode into) in a one-to-many fashion, i.e. n parallel messaging rounds, each from a *single* state to the (maximally) 2^k nodes above. Conversely, after the step has concluded, states will receive feedback from the nodes last predicted, again respecting word boundaries, now in a many-to-one fashion, i.e. again n parallel messaging rounds, now from the 2^k freshly decoded states back to the single state they are assocciated with (originate from). Unlike the naive approach, the setup maintains the word/supertag alignment while also structurally fusing the input- and output-level interactions sources. Nodes are indirectly informed by all *local* nodes below, with a much more endearing complexity of just:

$$\sum_{k=0}^{m} \underbrace{2^k n}_{\text{prediction messages}} + \underbrace{2^k n}_{\text{feedback messages}} \tag{IV.17}$$

which now grows as $O(2^m n)$.

Of course, something is amiss: the depth-wise intra-tree interactions may well be captured, but the inter-tree ones are unaccounted for. In the same vein as before, we may bypass this by having the state vectors communicate with one another after each local feedback around, allowing non-local autoregressive context flows. Having this done globally (all words communicating with all words) is certainly feasible and still preferrable to (IV.16), but suboptimal: it inserts a mn^2 memory complexity component (m messaging rounds in the cartesian product of words). A better alternative can be found in the dusty scriptures of old: sliding windows. Regulating and thresholding state interactions according to their relative distance reinstates computational well-behavedness[1], substituting n^2 with $n\kappa$ for window size κ (basically linear in sequence length) and setting the final memory footprint of the decoder at $O(2^m n + mn)$.

Computational considerations aside, this formulation is also conducive to learning. Having interactions modulated and bottlenecked by state tracking vectors reduces the number of statistical confounds accessible to the model, acting as an implicit regularizer and enforcing a degree of locality to the (otherwise distributed) neural representations. It also justifies a heterogeneous formulation, which would have different graph elements inhabit different vector spaces. State vectors are recurrent across depth and inter-communicating across width, thus meriting from high-dimensional representations; with that in mind, they can initially be supplied by an external high-horsepower encoder, solving the initial interfacing with the input sentence. Tree nodes, on the other hand, encode a decision over a very small vocabulary and are use-and-forget, justifying a low-dimensional representation. Finally, implementing the forward, backward and horizontal message passing rounds as separable, parameter-sharing convolutions repeated both across depth as well as width reduces the model's parameter count and provides the inductive biases needed for strong generalization.

Tree Parallel Decoding Summarizing, the decoding algorithm consists of the following steps:

i. State vectors are initialized by some external encoder.
ii. An empty fringe consisting of n blank nodes is instantiated, one such per word, rooting the corresponding supertag trees.
iii. Until a fix-point is reached (there is no longer any fringe):
   (a) States project class weights to their respective fringe nodes in a one-to-many fashion. Depending on the arity of the decoded symbols, a next fringe of unfilled nodes is constructed at the appropriate positions.
   (b) Each state vector receives feedback in a many-to-one fashion from the just decoded nodes above (what used to be the fringe), yielding tree-contextual states.

[1] Maybe there was something to the locality of supertagging after all.

(c) The updated state vectors emit and receive messages within their local neighborhoods in a many-to-many fashion, yielding tree- and sequence- contextual states.

A single iteration of step (iii) over the abstract canvas of Figure IV.3 is presented in Figure IV.4.

14.4.2 Implementation

The paragraphs to follow detail how the abstract pipeline is executed in practice. Consider yourself warned: you are urged to skip to the next section if sensitive to machine learning jargon, or the calendar year in your frame of reference is greater or equal to 2026 (I expect every single word to be obsolete by then).

Node Embeddings State vectors are temporally dynamic and of size d_w; they are initialized to $\mathbf{h}^0_{0:n} \in \mathbb{R}^{n \times d_w}$ by some external encoder, and are then updated through the fix-point iteration of three message passing rounds, as described in the next paragraphs. Tree nodes, on the other hand, are not subject to temporal updates, but instead become dynamically "revealed" by the decoding process. Their representations of size d_n are computed on the basis of (i) their primitive symbol and (ii) their position within a tree.

Primitive symbol embeddings are obtained from a standard embedding table $W_e : \mathcal{S} \rightarrow \mathbb{R}^{d_n}$ that contains a distinct vector for each symbol in the set of primitives $\mathcal{S}$. When it comes to embedding positions, we are presented with a number of options. It would be straightforward to fix a vocabulary of positions, and learn a distinct vector for each. But this is neither inclusive nor elegant: it imposes an ad hoc bound to the shape and size of tree nodes that can be encoded (contradicting the constructive paradigm), and fails to account for the compositional nature of trees. The structure-conscious route requires noting that *paths* over binary branching trees form a semi-group, i.e. they consist of two primitives (namely a left and a right path), and an associative non-commutative binary operator that binds two paths together into a single new one. The archetypical example of a semigroup is matrix multiplication; we therefore instantiate a tensor $P \in \mathbb{R}^{2 \times n_d \times n_d}$ encoding each of the two path primitives as a linear map over symbol embeddings. From the above we can derive a function p that converts positions to linear maps, by performing consecutive matrix multiplications of the primitive weights, as indexed by the binary word of a node's position; e.g. the linear map corresponding to position $12_{10} = 1100_2$ would be $p(12) = P_0 P_0 P_1 P_1 \in \mathbb{R}^{d_n \times d_n}$. We flatten the final map by evaluating it against an initial seed vector $\rho_0 \in \mathbb{R}^{d_n}$, corresponding to the tree root (or the initial hidden state in the RNN paradigm). To stabilize training and avoid vanishing or exploding weights and gradients, we model paths as *unitary* transformations by parameterizing the two matrices of P to orthogonality using the exponentiation trick on skew-symmetric bases [Bader et al., 2019; Lezcano Casado, 2019]. Now, let tree node $s_{i,k}$ contain symbol

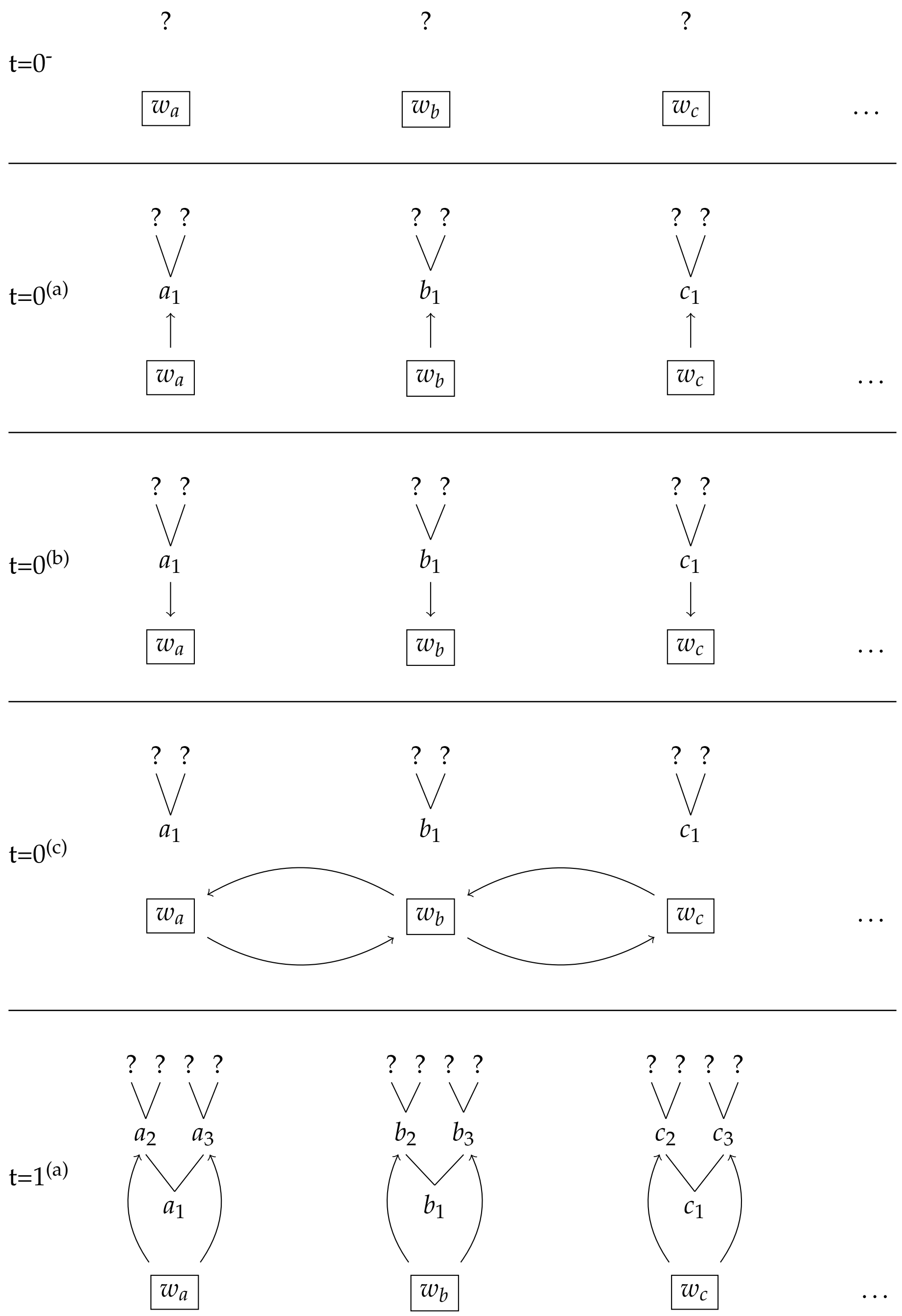


Figure IV.4: A frame by frame view of the first decoding step.

$\sigma \in \mathcal{S}$; its embedding $n_{i,k}$ will be agnostic to its tree index i and given as the element-wise product of its tree-positional and content embeddings:

$$n_{i,k} = p(k)(\rho_0) \odot (W_e(\sigma)) \in \mathbb{R}^{d_n} \tag{IV.18}$$

The embedder is then essentially an instantiation of a binary branching unitary RNN [Arjovsky et al., 2016], the choice of which hidden-to-hidden map to follow at each step depending on the node's position relative to its ancestor.[1] Since paths are shared across trees, their representations are in practice efficiently computed once per batch for each unique tree position during training, and stored as fixed embeddings during inference.

Node Prediction Assuming at step τ a sequence of globally contextualized states $\mathbf{h}^{\tau}_{0:n}$, we need to use each element h^{τ}_{i} to obtain class weights for all of the node neighborhood $\mathcal{N}_{i,\tau}$ consisting of all nodes (if any) of tree t_i that lie at depth τ.[2] We start by down-projecting the state vector into the node's dimensionality using a linear map W_n. The resulting feature vectors are indistinguishable between all nodes of the same tree – to tell them apart (and obtain a unique prediction for each), we gate the feature vectors against each node's positional embedding. From the latter, we obtain class weights by matrix multiplying them against the transpose of the symbol embedding table, as standard practice compels [Press and Wolf, 2017]:

$$\text{weights}_{i,k} = (p(k)(\rho_0) \odot W_n h^{\tau}_{i})\, W_e^{\top} \tag{IV.19}$$

The above weights are converted into a probability distribution over the alphabet symbols $\mathcal{S}$ by application of the softmax function.

Autoregressive Feedback To update the states for the next iteration, we must first provide autoregressive feedback from the last decoded nodes. We do so using a heterogeneous message-passing scheme based on graph attention networks [Veličković et al., 2018; Brody et al., 2021]. First, we use a a linear map W_b to down-project the state vector into the nodes' dimensionality. For each position i and corresponding state h^{τ}_{i}, we compute a self-loop score:

$$\tilde{\alpha}_{i,\circlearrowleft,\tau} = w_a \cdot (W_b(h^{\tau}_{i}) \,||\, \mathbf{0}) \tag{IV.20}$$

where $w_a \in \mathbb{R}^{2d_n}$ a dot-product weight and $\mathbf{0}$ a d_n-dimensional zero vector. Then we use the (now decoded) neighborhood $\mathcal{N}_{i,\tau}$ to generate a heteroge-

[1]Concurrently, Bernardy and Lappin [2022] follow a similar approach in teaching a unitary RNN to recognize Dyck words, and find the unitary representations learned to respect the compositional properties of the task. Here we go the other way around, using the unitary recurrence exactly because we expect them to respect the compositional properties of the task.

[2]That's a lot of indexing operations. Look at figure IV.3 and assume an enumeration that starts from 0 for timesteps and sequence positions and 1 for node positions. Then $\mathcal{N}_{1,2}$ would be $\{b_4, b_5, b_6, b_7\}$.

neous attention score for each node $s_{i,k} \in \mathcal{N}_{i,\tau}$:

$$\tilde{\alpha}_{i,k,\tau} = w_a \cdot (h_i^\tau \mid\mid n_{i,k}) \tag{IV.21}$$

Scores are passed through a leaky rectifier non-linearity before being normalized to attention coefficients α. These are used as weighting factors that scale the self-loop and input messages, the latter upscaled by a linear map W_m:

$$\tilde{h}_i^\tau = \sum_{s_{i,k} \in \mathcal{N}_{i,\tau}} \alpha_{i,k,\tau} W_m n_{i,k} + \alpha_{i,\circlearrowleft,\tau} h_i^\tau \tag{IV.22}$$

This can also be seen as a dynamic residual connection – $\alpha_{i,\circlearrowleft,\tau}$ acts as a gate that decides how open the state's representation should be to node feedback (or conversely, how strongly it should retain its current values). States receiving no node feedback (i.e. states that have completed decoding one or more time steps ago) are thus protected from updates, preserving their content. In practice, attention coefficients and message vectors are computed for multiple attention heads independently as done by Vaswani et al. [2017], but these are omitted from the above equations to avoid cluttering the notation.

Sequential Feedback At the end of the node feedback stage, we are left with a sequence of locally contextualized states $\tilde{h}_i^\tau$. The sequential structure can be seen as a fully connected directed graph, nodes being states (words) and edges tabulated as the square matrix $\mathcal{E}$, with entry $\mathcal{E}_{i,j}$ containing the relative distance between words i and j. We embed these distances into the encoder's vector space using an embedding table $W_r \in \mathbb{R}^{2\kappa \times d_w}$, where κ the maximum allowed distance, a hyper-parameter. Edges escaping the maximum distance threshold are truncated rather than clipped, in order to preserve memory and facilitate training, leading to a natural segmentation of the sentence into (overlapping) chunks. Following standard practices, we project states into query, key and value vectors [Vaswani et al., 2017], and compute the attention scores between words i and j using relative-position weighted attention [Shaw et al., 2018]:

$$\tilde{a}_{i,j} = d_w^{-1/2} \left(W_q \tilde{h}_i^\tau \odot W_r \mathcal{E}_{i,j}\right) \cdot W_k \tilde{h}_j^\tau \tag{IV.23}$$

From the normalized attention scores we obtain a new set of aggregated messages:

$$m'_{i,t} = \sum_{j \in \{0..s\}} \frac{\exp(\tilde{a}_{i,j}) W_v \tilde{h}_j^\tau}{\sum_{k \in \{0..s\}} \exp(\tilde{a}_{i,k})} \tag{IV.24}$$

Same as before, queries, keys, values, edge embeddings and attention coefficients are distributed over many heads. Aggregated messages are passed through a swish-gated feed-forward layer [Dauphin et al., 2017; Shazeer, 2020] to yield the next sequence of state vectors:

$$h_i^{\tau+1} = W_3 \left(\text{swish}_1(W_1 m'_{i,\tau}) \odot W_2 m'_{i,\tau}\right) \tag{IV.25}$$

	CCGbank		*TLGbank*	*Æthel*
	original	*rebank*		(1.0.0a5)
Primitives	37	40	27	81
Zeroary	35	38	19	31
Binary	2	2	8	50
Categories	1 323	1 619	851	5 762
in train	1 286	1 575	803	5 146
depth avg.	1.94	1.96	1.99	1.82
depth max.	6	6	7	35*
Test Sentences	2 407	2 407	1 571**	5 770
length avg.	23.00	24.27	27.58	16.52
Test Tokens	55 371	56 395	44 302	95 331
Frequent (100+)	54 825	55 690	43 289	91 503
Uncommon (10-99)	442	563	833	2 639
Rare (1-9)	75	107	149	826
Unseen (OOV)	22	27	31	363

*A beautiful conjunction of 35 noun phrases.
**Random but consistent train/dev/test split of 80/10/10.

Table IV.1: Bird's eye view of datasets employed and relevant statistics. Test tokens are binned according to their corresponding categories' occurrence count in the respective dataset's training set. Token counts are measured before preprocessing. Unique primitives and tree depths for the type-logical datasets are counted *after* binarization.

where $W_{1,2}$ are linear maps from the encoder's dimensionality to an intermediate dimensionality, and vice versa for W_3.

Putting Things Together We compose the previously detailed components into a single layer, which acts as a sequence-wide, recurrent-in-depth decoder. We insert skip connections between the input and output of the message-passing and feed-forward layers [He et al., 2016], and subsequently normalize each using root mean square normalization [Zhang and Sennrich, 2019].

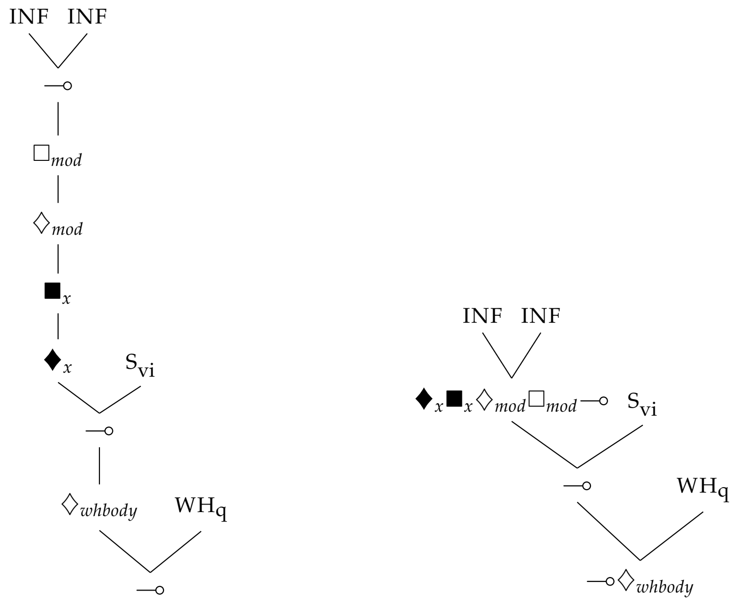
14.4.3 Experiments & Results

Datasets Testing the architecture on multiple datasets is good practice for multiple reasons, the least cynical being that it helps us better affirm its potential. Hence, we shall employ it not just on Æthel but also on the two versions of the CCGbank, as well as the French TLGbank; in total, 4 different datasets spanning three languages and as many grammar formalisms.

A high-level overview of the datasets is presented in Table IV.1. The English CCGbank and its refined version [Honnibal et al., 2010, rebank] stand out in having combinatory categories as their supertags, built with the aid of two binary slash operators. Combinatory rules take care of shifting, raising and function composition, allowing the lexicon to remain small and simple. The key difference between the two versions lies in their tokenization and the plurality of categories assigned, the latter containing more assignments and a more fine-grained set of syntactic primitives, which in turn make it a slightly more challenging evaluation benchmark. On more familiar grounds we have the French TLGbank, Æthel's distant but cherished uncle. It uses modalities for control purposes, licensing or restricting the applicability of rules related to non-local syntactic phenomena. Its supertags are therefore multimodal Lambek types, the tree representations of which are not strictly binary; to attune unary operators with the architecture, we cast them into pseudo-binaries by inserting an artificial terminal tree within a fixed position. A similar strategy is applied to Æthel (which I assume is by now familiar). Unary branches are shortened by first merging diamond-box pairs into a single composite operator, and then iteratively merging adjunct (resp. complement) markers (either plain or composite) with the subsequent (resp. preceding) binary operator. The new symbols correspond to notational and temporal shorthands for multiple decisions compressed in a single time step, making for an unambiguous and invertible representational translation at the cost of an enlarged primitive alphabet. These shorthands are not to be confused with the ones we experimented with earlier in Section 14.3. Back then, we were creating shorthands for frequent (sub-)types (self standing trees), whereas here we are establishing representational shorthands for composite type operators (tree constructors) – see Figure IV.5 for an example.[1]

Training A single hyper-parameter setup is shared among all experiments, obtained after a minimal logarithmic search over sensible initial values. Specifically, we set the node dimensionality d_n to 128 with 4 heterogeneous attention heads and the state dimensionality d_w to 768 with 8 homogeneous attention heads. The window size for the state-to-state messaging passing rounds is set to 14 (1 self connection, 6 right neighbours and as many left, 1 direct connection to the sequence summary token). We train using AdamW [Loshchilov and Hutter, 2019] and a variable learning rate scaled by a linear warmup and cosine decay schedule over 25 epochs, scaled by 10% for the encoder. During training we provide strict teacher forcing and apply feature and edge dropout

[1] If you're really really observant here, you might see a potential problem with this. If you can't, I'll spoil it for you. Some of the composite alphabet symbols may appear in the test set (or in the wild) without having ever appeared in the training set. These would be impossible to predict, even in the constructive setting – not because they're absent from the vocabulary, but because there's no usage examples to learn from! In fact, one such symbol exists, corresponding to the type/tree pattern $\blacklozenge_x \blacksquare_x \lozenge_{cmpbody} \llcorner \multimap \llcorner$ for hypothesizing a deeply nested body of a complementizer – it has 3 occurrences in the dev set and 1 in the test set. This is basically the zipfian tai of the zipfian tail – we'll let it slide without rearranging the train/dev/test split, as it poses an innocuous and fun little challenge and an easter egg of shorts. Obviously, dropping the practical restriction for binarity would solve the problem, at the cost of elongated trees and the loss of architectural uniformity.



(a) Original decomposition. (b) With notational shorthands.

Figure IV.5: Compactifying the horrifying type of Figure III.10.

at 20% chance. The loss signal is derived as the label-smoothed negative log-likelihood between the network's prediction and the ground truth label. Base-sized BERT variants are procured from the transformers library [Wolf et al., 2020]: RoBERTa for English [Liu et al., 2019], BERTje for Dutch [de Vries et al., 2019] and CamemBERT for French [Martin et al., 2020], all fine-tuned during training. The model is trained to be responsible for its own chunking, merging subsequent words into a multiword phrase by assigning a merge-left meta-symbol to multiword parts.

Evaluation We perform model selection on the basis of validation accuracy, and gather the corresponding test scores according to the frequency bins of Table IV.1. Table IV.2 presents our results compared to relevant published literature. Evidently, our model surpasses established benchmarks in terms of overall accuracy, matching or surpassing the performance of both traditional supertaggers on common categories and constructive ones on the tail end of the frequency distribution.

To investigate the relative impact of each network component, we conduct an ablation study where message passing components are removed from their network in their entirety. Removing the state feedback component collapses the network into a token-wise separable recurrence, akin to a graph-featured RNN without a hidden-to-hidden affine map. Removing the node feedback

model	accuracy (%) overall	frequent	uncommon	rare	unseen
CCGbank (original)					
Symbol Sequential LSTM /w n-grams [Liu et al., 2021]	95.99	96.40	65.83	8.65[!]	
Tree Recursive (Feature Embedding) [Prange et al., 2021]	96.09	96.44	68.10	**37.40**	**3.03**
Cross-View Training [Clark et al., 2018]	96.10	–	–	–	n/a
Attentive Convolutions [Tian et al., 2020]	**96.25**	**96.64**	71.04	n/a	n/a
Geometry-Aware Convolutions (this work)	**96.29** $_{\pm 0.04}$	**96.61** $_{\pm 0.04}$	**72.06** $_{\pm 0.72}$	34.45 $_{\pm 1.58}$	**4.55** $_{\pm 2.87}$
CCGbank (rebank)					
Symbol Sequential Transformer† [Kogkalidis et al., 2019]	90.68	91.10	63.65	34.58	**7.41**
Tree Recursive (GRU) [Prange et al., 2021]	94.62	95.10	64.24	25.55	2.47
Tree Recursive (Feature Embedding) [Prange et al., 2021]	94.70	95.11	68.86	**36.76**	4.94
Geometry-Aware Convolutions (this work)	**95.07** $_{\pm 0.04}$	**95.45** $_{\pm 0.04}$	**71.40** $_{\pm 1.15}$	**37.19** $_{\pm 1.81}$	3.70 $_{\pm 0.00}$
French TLGbank					
ELMo & LSTM Classification [Moot, 2019]	93.20	95.10	75.19	25.85	n/a
Geometry-Aware Convolutions (this work)	**95.92** $_{\pm 0.01}$	96.40 $_{\pm 0.01}$	**81.48** $_{\pm 0.97}$	**55.37** $_{\pm 1.00}$	**7.26** $_{\pm 2.67}$
Æthel (0.4)					
Symbol Sequential Transformer⋆ [Kogkalidis et al., 2020]	83.67	84.55	64.70	50.58	**24.55**
Æthel (1.0.0)					
Geometry-Aware Convolutions (this work)	**94.08** $_{\pm 0.02}$	95.16 $_{\pm 0.01}$	**75.55** $_{\pm 0.02}$	**58.15** $_{\pm 0.00}$	**18.37** $_{\pm 2.73}$

[!] Accuracy over both bins, with a frequency-truncated training set.
†Numbers from Prange et al. [2021].
⋆Parser-integrated model trained with tree sequences spanning less than 140 nodes in total.

Table IV.2: Model performance across datasets and compared to recent studies. Numbers are taken from the papers cited unless otherwise noted. For our model, we report averages and standard deviations over 6 runs. Bold face fonts indicate (within standard deviation of) highest performance.

	node feedback only	state feedback only	no feedback
CCGbank *(original)*	-0.05	-0.01	-0.08
CCGbank *(rebank)*	-0.12	-0.04	-0.07
French TLGbank	-0.13	-0.14	-0.23
Æthel (1.0.0)	-0.24	-0.12	-0.39

Table IV.3: Absolute difference in overall accuracy when removing the state and node feedback components (averages of 3 repetitions).

component turns the network into a Universal Transformer [Dehghani et al., 2018] composed with a dynamically adaptive classification head. Removing both is equatable to a 1-to-many contextualized token classification that is structurally unfolded in depth. Our results, presented in Table IV.3, verify first a positive contribution from both components, indicating the importance of both information sharing axes. In three out of the four datasets, the relative gains of incorporating state feedback outweigh those of node feedback, and are most pronounced in the case of Æthel, likely due to its positionally agnostic types. With the exception of CCGrebank, relinquishing both kinds of feedback largely underperforms having either one, experimentally affirming their compatibility.

14.4.4 Insights & Observations

Advantages Obtaining state of the art performance while still being able to predict rare and unseen types with relative reliability is a definite advantage. Like before, the approach provisions contextual and self-standing representations for atomic components virtually for free. Unlike before, the architecture boasts an extremely fast inference speed that goes toe-to-toe with conventional discriminative architectures, owing to its temporal upper bound scaling with maximal tree depth (practically a constant, except for really perverse cases). Structure manipulation is kept to the bare minimum, even in the absence of oracle guidance, maximizing GPU utilization and data parallelism with high efficiency sparse routines [Fey and Lenssen, 2019]. Operations are batched and temporally iterated across all nodes and sequences without any CPU interruptions. Each next fringe is dynamically generated on a purely numerical basis.[1] The architecture's memory footprint and parameter count are also the product of careful design and thus well under check, facilitating training and parser integration (as we will soon see). The model takes approximately 5 hours to train on each dataset, and boasts a constant processing speed of 6 000 tokens/sec on a consumer-grade laptop GPU. By comparison, the sequential model

[1] By selecting decoded symbols that correspond to tree forming operators, isolating their indices, multiplying them by two (to create left children), offsetting by one (to create right children), and finally interleaving the two to yield both descendants. In the same vein, the indices of the aligned states are simply the repetition of their respective ancestors' indices.

takes about 100 hours to converge, and processes about 700 tokens/sec, scaling inversely with sentence length.

Supertagging and Sparsity Practice aside, the results obtained pose concrete evidence that lexical sparsity, historically deemed the categorial grammar's curse, might well just require a change of perspective to tame and deploy as the answer to the very problem it poses. Crucially, the architecture's relative gains scale with respect to the task's complexity. In the original version of the CCGbank, the model is only slightly superior to the next best performing model – an ad hoc graph neural network with built in lexical biases (quite literally the ideological antipode of our endeavour). The difference becomes an order of magnitude wider for the slightly more challenging rebank version. The effect is maximally pronounced for the harder type-logical datasets. For the French TLGbank, performance jumps up to CCGbank scales (despite it being significantly smaller and sparser). For Æthel, the absolute performance leap is about 10% compared to the vanilla constructive tagger. Even though there's some data distance between the two experiments (version gap, different data filtering, etc.) making strict numeric comparisons moot, the sizeable improvement is beyond doubt. This is clearly to be attributed to increased returns from the rare and uncommon bins. There is a synergistic effect between the larger population of these bins pronouncing even minor improvements, while at the same time the acquisition of rare categories apparently benefits from their plurality. Put simply, learning sparse assignments is *easier* in grammars that contain many and diverse rare assignments, and improvements there *matter* more – especially so if these don't come at the cost of stability at the higher frequency spectrum. The impact of this finding alone is bigger than the menial architecture itself – it is basically an open invitation to more elaborate, more strict and more regular lexicalized theories, and a promise that no matter how statistically unruly they might seem, there will always be an architectural solution to accommodate them. In today's machine learning frenzy, it is a statement of purpose lost: tools for the task, and not tasks for the tools.

Limitations Despite its objective success, the methodology is not without limitations. Most importantly, the parallel nature of the decoder trades inference speed for an incompatibility with greedy algorithms like beam search and an inability to produce local assignment rankings. Put plainly, obtaining more than the "best" category assignment per word is not straightforward, a fact which can prove harmful for coverage of a downstream parser. A possible solution would involve branching across multiple tree-slices (i.e. sequences of partial assignments) rather than single predictions, but efficiently computing scores and comparing between complex structures is uncharted territory and not trivial to implement. The issue is of course not unique to this system, but common to all decoders that perform multiple assignments concurrently – as such, there is some hope that insights might percolate from one field to another and eventually make their way to us.

15 Neural Proof Search

We have made significant progress with supertagging, first removing the scary-looking roadblock of lexical type sparsity, and then producing numbers bigger than any numbers seen before[1], earning a spot at at the top 10 pop hits leaderboard for the next couple of months. But supertagging alone is not going to take us to the end of the road. Far from it, in fact, since our chosen logic is extremely permissive, burning away any hope of making do using just the proof-theoretic tools we have available. The problem is simple: our type assignments, even when fully correct, still allow more proofs than desired – we need yet another statistical learner on which to outsource the duty of finding which one of these many proofs is the linguistically sensible one. And once more, the discriminator approach is not going to work: enumerating and ranking all well-formed proofs is a no-go – we have to build the correct proof from scratch instead.

To move forward we need to look back. Almost an entire book ago, we had a brief encounter with proof nets, and we saw them at work in the context of $\mathbf{ILL}_{\multimap}$ (by now rebranded as **LP**). Back then, we were quick to dismiss them after a moment of shallow appreciation, seeing as they were too complicated of a proof format for conducting search on, and too underspecific of a representational format for showing around. Lots have changed since, and not for the better; we added modalities, which impose further structure on their own, invisible to proof nets (at least in their unaltered form). Why then should we turn our attention back to them now?

The blast from the past is justified by the inadequacy of alternatives. The natural deduction presentation we have predominantly employed is hierarchical and tree-like; assuming perfect processing and regardless of search direction (bottom-up or top-down), computation is temporally bound by the proof depth – bottlenecked by decisions due, or anticipating decisions to be. The same is true for most proof-theoretic alternatives (tableaus, the sequent calculus, λ terms, etc.). Decisions cannot be detached from the structure they bind to and help form – to make a choice requires knowing your options, which in turn requires that previous choices were not just abstractly *made* but concretely *evaluated*. At the same time, even if the proof is to be dynamically constructed, thus abolishing the need for enumerating all options globally, each local junction point still needs to be exhaustively expanded for the construction to proceed. Explicit symbolic manipulation and neural representations are inseparably bound, and even though the former can be somewhat

[1] Actually, Tian et al. [2020] were reporting even bigger numbers for a while, but it later turned out that these numbers were in error and their real numbers were a lot smaller than the numbers they originally reported. Long story short, don't trust the numbers.

localized, it can never be foregone fully. Practically speaking, type safety is not for free: it costs back-and-forth between the logical and numerical back ends, inhibiting integration. Proof nets, on the other hand, are the embodiment of data parallelism, the most sought-after property of neural computation and the holy grail of efficient training and optimization. Decision making and validity testing are detached: we must first make all the decisions that are to be made, and only then can we check whether they are structurally legitimate. This laxness was the very reason we abandoned proof nets in the first place, but it is exactly what makes them so very appealing now.

15.1 Permuting Types to Alignment

So what exactly is the structure of decisions encoded in a proof net? Recalling our earlier discussion, a proof net is essentially two things and a promise. First, a sequence of decomposition trees, called a proof frame. Second, a bijection between atomic formulas of opposite polarities, called axiom links. The two together make a proof structure, i.e. a candidate proof – if it so happens that we can traverse the structure, the promise is kept: it is also a proof net. The proof frame is deterministically obtained from a sequence of type assignments, and thus irrelevant to this discussion – in principle, we can assume it to be given from the supertagging component. The promise of traversability corresponds to the deferred (and also deterministic) validity checking – it can only be done *given* a proof structure. The only thing we really have any say on is the axiom links. Our goal is therefore to design a neural model capable of manipulating sets and building bijections between them – and not just *any* bijections, but rather only those that are theoretically permitted (traversable, i.e. valid proofs) and linguistically sensible (i.e. not just valid proofs, but also valid parses).

A feeling of apprehension wouldn't be unreasonable at this point; neural models are notoriously ill-fit for discrete and combinatorial optimization. Pulling us out of this momentary lapse is the folkloric knowledge that a bijection between two ordered sets, or *chains*, can be uniquely represented as a *permutation matrix* – a boolean valued, doubly stochastic and square matrix that, left-applied to set_1 (in column form), *transports* each of its elements to its pair in set_2. Note that the order does not need to be due to some intrinsic property of the set and its elements, but may well be externally imposed – this is just simply a representational trick. Considering also that a link is only possible between different instances of the *same* atomic type, the problem naturally reduces to finding not a single, big matrix, but many smaller, independent ones.

This might all be easier to digest with a visual example. Let's turn our attention to Figure IV.6, which displays an annotated proof net for the wh-question *Wat is die rare tekening?* 'What is this weird picture?', with derivation:

$$\text{wat } \triangle_{whbody}(\lambda \mathrm{x}_0.\text{is } \mathrm{x}_0 \ \triangle_{su}(\blacktriangledown_{det}\text{die } (\blacktriangledown_{mod}\text{rare tekening}))) \tag{IV.26}$$

Formula decompositions should be easy to decipher. Following the polarization induction, we end up with polarity information for all formula nodes, all the way up to leaves – the only addition to the modus operandi of Section 2.2 are the unary modal branches, which are polarity preserving. To be able to tell distinct occurrences of the same atom apart, we associate a unique index to each leaf; the convention followed is irrelevant, but we may as well assume a depth-first left-first sequence-wide enumeration. We can then use these to refer back to the occurrences of atomic formulas they identify within the proof frame. Gathering positive and negative leaves, we end up with two ordered sets P and N; in this case $P := \{0, 2, 5, 7, 9, 10\}$ and $N := \{1, 3, 4, 6, 8, 11\}$. Each set can then be partitioned according to its elements' types, yielding as many positive as negative subsets, aligned in equinumerous pairs; in our case, positive subsets $P_{\mathrm{VNW}} = \{0\}$, $P_{\mathrm{S_{vi}}} = \{5\}$, $P_{\mathrm{WH_q}} = \{2\}$, $P_{\mathrm{NP}} = \{7\}$ and $P_{\mathrm{N}} = \{9, 10\}$, and their negative counterparts $N_{\mathrm{VNW}} = \{3\}$, $N_{\mathrm{S_{vi}}} = \{1\}$, $N_{\mathrm{WH_q}} = \{11\}$, $N_{\mathrm{NP}} = \{4\}$ and $N_{\mathrm{N}} = \{6, 8\}$. Now, rather than look for the correct bijection between P and N, we should be looking for the correct bijections between sets of the same "type" and opposite polarity. All sets except those indexed by N are singletons, their pairs having but one bijection. Each of P_{N} and N_{N} contains two elements, giving rise to two possible bijections – a one-in-two choice that captures the entire proof! The bijection we are in search for is $\pi_{\mathrm{N}} : P_{\mathrm{N}} \rightarrowtail\!\!\!\!\twoheadrightarrow N_{\mathrm{N}}$, $\pi_{\mathrm{N}}(9) = 6$, $\pi_{\mathrm{N}}(10) = 8$, which can be represented as the permutation matrix:

$$\pi_{\mathrm{N}} := \begin{array}{c|c|c|} & 6 & 8 \\ \hline 9 & \checkmark & \\ \hline 10 & & \checkmark \\ \hline \end{array} \tag{IV.27}$$

Without loss of generality, creating such matrices is what the proof search we need to conduct boils down to – even if usually we'll have more choices spread among more types. Needless to say, this is a simplification; structural constraints are only partially imported, in the sense that traversability and well-formedness are not respected by default. It is not an unreasonable one though; the format is compact, and imports at least some of the structural constraints, faithfully mirroring the definition of a proof structure at least. Besides, matrices are the bread and butter of machine learning – we're on the right track.

15.2 Neural Proof Nets

Permutation matrices may well be matrices, but they're still discrete – not something we could ever hope to differentiably produce. In our neural reimaging of axiom links, we need to go for the next best thing: their continuous relaxations. A soft version of a permutation matrix can be approximated with arbitrary precision by virtue of the Sinkhorn operator [Sinkhorn, 1964]. The operator and its underlying theorem state that the iterative normalization (alternating between rows and columns) of a square matrix with positive entries

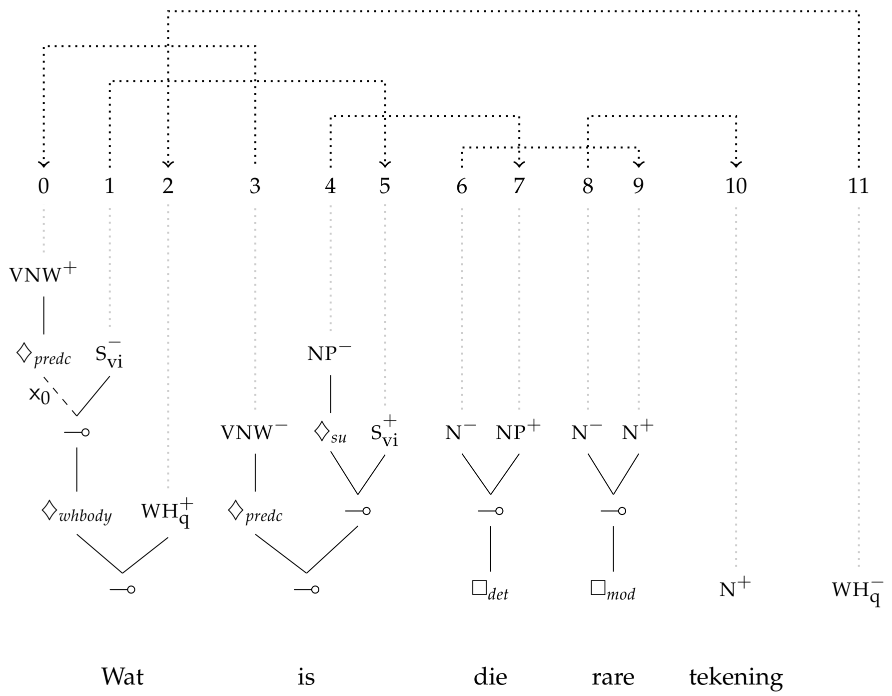


Figure IV.6: The two layers of an $\mathbf{LP}_{\Diamond,\Box}$ proof net: a proof frame (below) with its axiom links (above).

yields, in the limit, a doubly-stochastic matrix, the entries of which are *almost* binary. Almost binary is binary enough – we will do just fine without going to the limit. The positive entry constraint is a minor hickup though. To bypass it, we can move computation to the logarithmic domain, employing the log-sum-exp trick in place of standard normalization, which also helps ensure numerical stability. In that setting, the Sinkhorn normalization of a real-valued square matrix $\mathbf{X}$ is defined as:

$$\text{Sinkhorn}(\mathbf{X}) = \lim_{\tau \to \infty} \exp\left(\text{Sinkhorn}^{(\tau)}(\mathbf{X})\right) \tag{IV.28}$$

where the induction is given by:

$$\text{Sinkhorn}^{(0)}(\mathbf{X}) = \mathbf{X} \tag{IV.29}$$

$$\text{Sinkhorn}^{(\tau)}(\mathbf{X}) = \mathcal{T}_r\left(\mathcal{T}_r\left(\text{Sinkhorn}^{(\tau-1)}(\mathbf{X})\right)^\top\right) \tag{IV.30}$$

and $\mathcal{T}_r$ the row normalization in the log-space:

$$\mathcal{T}_r(\mathbf{X})_{i,j} = \mathbf{X}_{i,j} - \log \sum_{r=0}^{N-1} \exp(\mathbf{X}_{r,j} - \max(\mathbf{X}_{r,:})) \tag{IV.31}$$

Used this way, the Sinkhorn operator gives rise to a non-linear activation function that applies on matrices and pushes them towards binarity and bistochasticity, analogous to a 2-dimensional softmax that preserves assignment [Mena et al., 2018]. This exotic activation function is the key to efficiently navigating the combinatorially prohibitive landscape of axiom links.[1] Where previously we would need to either (i) thoroughly construct and rank all combinations, or (ii) iteratively decode through each element of set_1 while dynamically adjusting set_2 as candidates get excluded, we are now presented with a much more appealing alternative; a temporally bound, backtrack-free operation that translates the structural constraint of bijectivity into highly optimized and fully parallelizable linear algebraic routines.

To apply in the setup envisaged here, all we need to do is assemble matrices containing unnormalized similarity scores in the cartesian product of positive P_χ and negative N_χ occurrences of formula tree leaves, one such matrix S_χ per unique atomic proposition χ present in a frame; a similar position is advocated by Moot [2008]. Normalizing these scores with Sinkhorn and contrasting the result with the target output (the discrete ground truth permutation π_χ) amounts to teaching a network an implicit ranking of the set of bijections between the two sets, on the basis of their elements' representations. If training goes according to plan, the π_χ-image (resp. π_χ^{-1}) of each element of P_χ (resp. N_χ) will outrank all competing items in N_χ (resp. P_χ), i.e. be the largest entry of its row (resp. column) in S_χ. For this to have any chance of

[1] Recall that the number of possible bijections scales factorially with the cardinality of the sets.

success, the mechanism producing S_χ, and by extension the representations of P_χ and N_χ, will need to be highly contextual. Embedding a leaf's type, polarity and/or tree position won't cut it – sequential context is crucial to disambiguate between leaves living in distinct instances of identical types, whereas lexical association context should prove beneficial in resolving derivational ambiguities (rare as they might be). Coincidentally and to our great fortune, the two constructive supertaggers we have described and implemented in the previous section do in fact provide contextual representations, at exactly this granularity scale – imagine that! Happy coincidence aside, the operationalization described is plug-and-play for any supertagger, constructive or otherwise – the picky requirements on subtype representations can always be satisfied by some third party encoder. Employing such an encoder might even be for the best, in terms of performance alone – but for the sake of parameter compression and model reuse, we are given the chance to have the decoding architectures described earlier do double duty as "proof frame encoders". To that end, we simply need to jointly train supertagging and axiom linking in a unified, end-to-end architecture, simultaneously optimizing both objectives.[1]

15.2.1 Implementation(s)

Since the original version of Kogkalidis et al. [2020], the architecture was revised with minor micro-adjustments, and retrained with each new major adaptation of Æthel, charting a multidimensional course of only partially compatible successive stops. In what follows, I describe in detail only the current and most recent implementation [Kogkalidis et al., 2023], drawing parallels with isolated historical insights only when relevant.

Supertagger Integration The architecture combines just as easily with both the symbol sequential and the geometrically informed decoder, requiring only an adaptation of how leaf nodes are indexed and gathered. Freely reusable as they might be, the contextual representations of either decoder are imperfect: each token can only be informed by tokens that temporally preceded it. In theory, this means that disambiguation between competing link candidates is back-loaded – the weight of flipping the scales is on the tokens last decoded. It also means that the geometry-aware model is at a disadvanage, since it performs multiple assignments in parallel. In practice, this is not a major concern – both integrations perform exceptionally well and quickly fit the training set, making the addition of any extra parameters redundant and a potential threat to generalization; we can stick to using the decoder's representations as is.

[1] Another way to see this is as a flipped version of the *linear assignment problem*, where given matrices of representations P, N, a similarity metric w and target output π, we wish to learn set element representations and the parameters of a similarity metric w (if any), such that the quantity:

$$\sum_{i,j} \pi_{i,j} \odot \text{Sinkhorn}^{\tau}(S)_{i,j}$$

is maximized, where $S_{i,j} = w(P_i, N_j)$.

Choosing between the two decoders seems like a no brainer at first – the geometry-aware formulation is significantly faster and more accurate, let alone easier to train (despite these factors usually being in conflict). But in transitioning to it, we forfeit the right to algorithmically *search* in the output space. In practical terms, while we may keep demanding a new proof frame from the symbol sequential supertagger ad infinitum (or until satisfied), the geometric one won't be at all receptive to our pleas – its greedily decoded proof frame is our one and only chance at a parse. This limitation becomes especially relevant considering the impenetrable barrier placed by the count invariance property, requiring an equal count of positive and negatives for every atomic type present (no square matrix otherwise!). A proof frame that fails to satisfy that property is no good for proof search. This obstacle can be repurposed to a tool, as long as search is an option – hard-wiring the constraint into beam search yields a correct-by-construction (yet painfully slow) decoding algorithm, massively increasing computational load but practically ensuring parsability.[1] Sticking with the parallel decoder means having to wave such niceties goodbye, at least until some search algorithm is formulated and implemented. Even so, the drastically improved accuracy and speed translate to a multiplicative increase in the performance-to-compute ratio – the greedy output of the novel supertagger is practically as good as – and incomparably faster than – a vanilla beam search on the original one, but with no guarantee of structural correctness. Long story short, the geometry-aware decoder is not a no brainer, but it's still the superior choice.

Since the permutations require access to the atoms of goal (succedent) formula, we train the supertagger to produce one, using the input sequence's sentential summary token (the `[CLS]` token, in jargon) as the stateful decoding seed. Æthel's proofs are actually restricted to atomic goal types, allowing us to derive them "by hand" by simply counting which of the atomic types has a positive atom too much – but that offers no vectorial representation we can use.

Neural Permutation Module The permutation component may modulate the linking process by providing a parametric and trainable similarity metric. The current version utilizes a weight vector $w \in \mathbb{R}^{d_n}$ to compute similarity scores between positive and negative vectors as their w-weighted inner product. This is an efficient way to selectively allocate signed weights on the decoder's representations, allowing certain pairwise interactions more promi-

[1] Beam searching over the symbol sequential decoder with a beam width β means obtaining β unique sequences satisfying:

$$\text{argmax}_{\beta}\left(\prod_{i}^{n}\prod_{j}^{||t_i||} p(s_{\beta,i,j} \mid s_{\beta,k,:} : k < i, s_{\beta,i,k} : k < j, \mathbf{w}_{0:n})\right)$$

We can import the count invariance condition by overwriting the probability score $p(s_{\beta,n,||t_n||}| \dots)$ assigned by the decoder with $-\infty$ when $\mathbf{t}_{0:n}$ fails the check, essentially discarding the sequence and forcing a backtrack.

nence than others (or even imposing sparsity to disentangle the two problems, if L_1 regularization is employed). Let's take a second to restate this: the transition from supertagging to full-blown parsing incurs a cost of 128 extra parameters; a relative increase of a paltry 0.0001%. What we have in our hands is quite literally the world's *leanest* parser.

Neurosymbolic Integration Symbolic processing is handled by the tiny type system that Æthel rests on, now extended with conversion routines that allow casting proofs to proof nets and back. The conversion routines allow us to conduct neural proof search in the favorable regime of proof nets, and convert the result to natural deduction format only at the very end, for the sake of sanity testing. Crucially, the type-checker, originally designed to assert the dataset's type safety, is now repurposed to a tool for verifying the correctness of analyses constructed – a proof structure that does not constitute a proof net will fail the traversal, throwing an error and alerting us to the fact. In other words, we can blindly trust anything the parser gives us as correct, at least in the sense of syntactic validity.

15.2.2 Experiments & Results

Training The unified architecture is end-to-end differentiable, and can be jointly trained on both the supertagging and the axiom linking tasks simultaneously. The supertagging objective remains exactly as before, whereas the linking loss is obtained as the negative log likelihood between the Sinkhorn-normalized activations and the discrete ground truth labels (identical to standard multiclass classification). Given that proof frames are a priori known in training and considering that teacher forcing ensures the correct autoregressive interactions, we may simply proceed with isolating the leaves out of the decoder's representations. Leaf representations are binned according to their sentential index, atomic type and polarity (e.g. a single bin would be all occurrences of a positive NP in sentence #13 of the input batch). Each bin is zipped with its opposite polarity counterpart, and their element-wise similarity scores are computed via the chosen metric.[1] Similarity scores are normalized by a fixed number of Sinkhorn iterations – three iterations suffice to produce sharp activations without eroding the gradient updates. Similarity scores and Sinkhorn normalizations are computed in parallel and batched across bins of the same size (i.e. according to the bijected sets' cardinality) – a faster alternative would be to pad them to a fixed size using some arbitrarily low constant as a padding value, but the difference is minor and not worth the memory overhead.

[1]Note that the zipping function is not *the* similarity metric but the pairwise application of one. In other words, the agreement scores are independently computed between pairs of elements from the two sets. This is necessary to account for the variably sized sets encountered without ad hoc padding, but also to ensure that the end operator is permutation invariant.

parsability (some proof obtainable)	**coverage** (some proof obtained)
$87.35_{\pm 0.18}$	$85.56_{\pm 0.22}$
frame accuracy (correct proof obtainable)	**accuracy** (correct proof obtained)
$57.76_{\pm 0.55}$	$55.63_{\pm 0.55}$

Table IV.4: Sentential-level evaluation of the parser.

The joint architecture takes longer to train than the stand-alone tagger, owing to the slower forward and backward passes, but also due to the increased number of epochs required to reach convergence. The linking task is surprisingly fast to converge, but its inclusion is detrimental to supertagging, as its loss term dominates the sum early on. Since accurate supertagging is a prerequisite to parsing, linking loss is scaled by 10% to promote smoother training curves and a healthier task balance. To nudge the model away from local optima induced by gradient conflict, one of the two losses is occassionally zeroed out with a 20% chance.

Evaluation During evaluation and inference, the decoder has to rely on its own output, as the ground truth frame is unknown. When decoding completes, the output must first be "parsed" into types proper. Assuming no structural integrity issues, leaf positions are indexed, keeping track of their polarities and atomic types – the result is a proof frame. For the frame to be an eligible starting point for proof search, it must satisfy the count invariance property, which is dynamically asserted on the spot. If it does, leaf representations are extracted and binned and their agreement scores are computed and normalized like before (except sequentially for each sentence). In the rare event that the local discretization (i.e. rounding) of a normalized matrix does not correspond to a bijection, we resort to an explicit combinatorial optimization via the Hungarian method, using the normalized scores as assignment weights. This authorizes a sensible, static number of Sinkhorn iterations while still providing a fallback to ensure the output's structural integrity. In either case, the pairings obtained correspond to axiom links, which are traversed to produce a proof (more on that in a bit).

We proceed with evaluation with no training wheels: no pre- or post- processing, no tokenization or chunking oracles, and no length, depth or frequency thresholding. All scores reported are the average of three repetitions. Numeric evaluation requires a way to compare proofs. The first and most transparent thing to consider are sentential- (or proof-) level performance metrics, where there's two axes of interest; axis one is whether we could have gotten or did get a proof, and axis two is whether that proof could be or was the correct one. The key results are presented in Table IV.4.

	local metrics			global accuracy	
modulo	p	r	F_1	proof	frame
–	$89.36_{\pm 0.05}$	$89.46_{\pm 0.06}$	$89.17_{\pm 0.06}$	$55.63_{\pm 0.55}$	$57.76_{\pm 0.55}$
modalities	$90.82_{\pm 0.02}$	$90.94_{\pm 0.03}$	$90.68_{\pm 0.03}$	$56.25_{\pm 1.02}$	$64.61_{\pm 0.99}$
functional types	$90.91_{\pm 0.03}$	$91.02_{\pm 0.04}$	$90.72_{\pm 0.04}$	$57.39_{\pm 0.71}$	$59.38_{\pm 0.65}$
types	$92.14_{\pm 0.01}$	$92.30_{\pm 0.01}$	$92.00_{\pm 0.02}$	$58.83_{\pm 0.63}$	$68.94_{\pm 0.52}$

Table IV.5: Decomposition metrics and relaxations.

On the bottom right, accuracy corresponds to the proportion of sentences assigned a proof that satisfies *strict syntactic equality* to the ground truth one, and stands at a 55.63%. The significance of this number is easy to miss, considering the unforgiving rigidness of the metric and the demanding nature of the task – proof equality means having *perfectly* captured the input's function-argument structures, functional types and dependency roles. By comparison, the state of the art in CCGbank parsing is currently 54% [Clark, 2021] – which is only to say that the two tasks, despite their obvious differences, are now very proximal in performance, despite type-logical grammars being traditionally dismissed as "too complex". Less promising is the average coverage, i.e. the proportion of sentences assigned any proof at all, lying at a modest 85.56%. To find the culprit for this lacking result, we measure parsability, i.e. the proportion of analyses whose proof frame satisfies the count invariance property. Evidently, only 87.35% of the input is amenable to proof search at all. Despite appearances, this is actually quite reasonable considering the severe architectural limitation of being confined to the one single best proof frame. It also goes to show that the permutation strategy is incredibly robust; 97.95% of the parsable sentences are *actually* parsed, with the remaining 2.05% of the errors being due to a structural link error (i.e. a proof structure that cannot be traversed). To put this in context, only 2% of the parsable sentences contain *any* structural error among *any* of their permutation bijections among *any* of the atomic types present. In other words, the traversability condition that makes a proof structure a proof net has been almost perfectly captured, despite remaining implicit throughout the training process, both in terms of representations used and of the loss signal backpropagated. More than just incredibly robust, the permutations are extremely accurate. This is made evident when considering the proportion of correct proofs to correct frames, which lies at an astonishing 96.31%. Put plainly, the number suggests that an error-free proof is *practically guaranteed* from an error-free frame. The results as a whole paint a clear picture that takes little effort to interpret: the performance bottleneck is on the supertagging module rather than the permutation module.

To understand the sizeable gap between accuracy and coverage, we employ an adaptation of the parsing community's favorite F_1-score. Concretely, we gather all samples for which a proof was produced, and decompose both prediction and ground truth into their respective sets of subproofs. We measure tp as the two sets' intersection, fp as the difference between predicted and correct subproofs and fn as the difference between correct and predicted subproofs, from which we may obtain precision as $p = {}^{tp}/_{(tp+fp)}$, recall as $r = {}^{tp}/_{(tp+fn)}$ and their harmonic mean as $F_1 = {}^{2pr}/_{(p+r)}$. On top of the vanilla versions of these metrics, we can also examine relaxations by incorporating a combination of two modulo factors. Relaxation one targets the functional core of the logic, applying a forgetful transformation that strips proofs of their modalities in order to examine typed function-argument structures in isolation. Relaxation two targets the modal enhancement of the logic, collapsing the set of atomic types into a single point (thus treating all functional types of the same *shape* as equal) in order to examine dependency structures in isolation. Relaxing on both axes at once is essentially casting proofs into the untyped λ calculus, where all we care about are the type- and dependency-agnostic function-argument structures – this is the metric most comparable to external theories.[1] Note that relaxations are performed only *after* inference – the point being that a strict proof must have been produced for its relaxations to be considered (i.e. lax accuracy is still bottlenecked by strict coverage). The results are averaged over covered samples[2] and presented in Table IV.5. Whether they are informative or not is up to debate; practically, they suggest that allowing for the occassional error in an atomic type or modality, about 92% of the subproofs returned are correct, and about as many of the gold standard subproofs are returned, which in turn suggests that erroneous parses are likely the product of isolated, local errors and not totally butchered.

15.2.3 Insights & Observations

Advantages The position that parsing is a problem of permutation offers an appealing way of encoding the parse space, where computational efficiency and formal correctness are no longer at odds. This challenges the status quo of shift-reduce parsing, scorning iterative structure manipulation in favor of a direct translation of structural constraints into vectorial ones, directly optimizable with numerical methods. Neural proof nets embody this in being fully parallel (even *intra*-sententially) and living entirely on the GPU. They refuse to engage in the over-parameterization game; combined with a constructive architecture, they're practically parameterless – you can use them on your grandma's laptop. To call their asymptotic behavior favorable would be an understatement; modern machines can perform a Sinkhorn normalization over batches of 64 matrices in constant time (in the μs scale) for any matrix order up to 2^7 – to comprehend how extremely ridiculous this is, consider that this would amount to finding the correct bijection out of $2^7! = 3.9 \times 10^{215}$ possibilities across 64 pairs of sets in parallel. More than just efficient, they are effective – employed here for an objectively difficult, sparse, underspecified

[1] Proofs are in β and η normal, so no free points from abstractions. Identity proofs are only equal if they match in both name and type, so no free points from variable instantiations either.

[2] Averaging over the full test set would artificially inflate p and deflate r scores, since no partial proofs are returned from failing samples.

and nascent type logic, they achieve accuracy comparable to that of established parsers boasting decades of combined community wisdom and collective effort. Numbers aside, neural proof nets are messengers of unity and not confrontation. In trivializing the difficulties associated with hypothetical reasoning, they bring explicit variable binding into *practical* relevance for wide-coverage categorial grammars, discrediting decades of naysaying. The same pipeline we used here applies to *any* grammar logic in the linear tradition – type theoretic purity is no longer a foe to shy away from, but a free pass at proof nets and their vectorized forms. To piggyback to our early discussions on abstract categorial grammars, the methodology presents a way to deliver type-correct tectogrammatic proofs *directly* from surface form, without having to ever engage in the difficult game of explicating the phenogrammatic type- and term- morphism, the latter implicitly internalized within a 128 parameter long black box instead. The framework is finally open to modification; explicit structural constraints additional to linearity may always be imported, either as additional objective functions or representational adjustments.

Limitations The key issue with the end-to-end architecture is inherited from the supertagger – our inability to mix and match multiple assignments is now here to haunt us with a sharp 13% reduction in absolute coverage. Beyond that, the implementation described capitalizes on a disentanglement between neural and symbolic operations for the sake of efficiency. But doing so comes at the heavy price of a unidirectional data flow that lacks cross-component feedback; even though tagger and parser share the same representations, there's no communication from the latter to the former. Symbolically traversing the produced axiom links is done singularly for the sake of testing and verifying the neural output, but gives us no chance to emit back a new request. Failures may be caught, but they are nonetheless irrecoverable – a partial output that fails any structural constraint, however rare or common, signifies an abrupt and non-negotiable end to the processing pipeline. A better operationalization would be to use the symbolic engine to continuously ask for neural output as long as the structural constraints are not met (or the user is not satisfied with the parse provided). For this to be possible, neural components would need to be extended with some notion of backtracking. In that sense, the parallel nature not just of the supertagger but also the parser becomes now a double-edged sword, practically prohibiting us from asking for the "next-best" set of axiom links – this becomes especially hard to tackle considering how the permutations of different atoms are independently produced. On a relevant note, the permutation module is currently implemented as a greedy deterministic oracle, making no attempt to account for derivational ambiguity. This is a sensible decision for the current set of experiments, given that ambiguity is mostly captured at the type level. Still, the limitation could be lifted by cross-contextualizing and/or adding noise and incorporating sampling routines in a probabilistic learning setup.

15.3 Proof Nets in $\mathbf{LP}_{\Diamond,\Box}$

Everything has been done, but not everything has been said. A cautious reader might have noticed an argumentative sleight of hand in the current section. To clear the air of any suspicion of deception, we need to step away from neural matters and take one last detour through the loopy land of proof nets.[1] The issue at hand is none other than the use of proof nets as our representational standard, which might be seen as implicating an equivalence relation between proofs and proof nets. Such an implication would be sloppy – the two sure are closely related, and going from proof to proof net is straightforward, but the opposite direction is tricky: a proof net encodes less than necessary to allow a perfect proof reconstruction. Luckily, we are not really demanding an isomorphism between $\mathbf{LP}_{\Diamond,\Box}$ and our version of proof nets; we just want a locally one-to-one relation between the two, covering only the neighborhood of proofs inhabited by our linguistic usecases. There's a whole lot of proof patterns we don't expect to ever encounter, and can thus safely ignore, practically reducing the combinatorial space to a manageable size. This might still seem quite ambituous at a first – our proof nets have no means of encoding the positionally underspecified diamond elimination rule $\Diamond E$, and are altogether imperceptive of the structural extraction rule $\mathsf{extr}_{\blacklozenge}$ This would've indeed been the case if it wasn't for our earlier providence: we have imposed a canonical placement on both these rules, which should relieve the burden of translation. In principle, we should be able to put things in literal order.

The algorithm we'll follow requires as input a set of axiom links, a sequence of (antecedent) lexical type assignments and a (succedent) conclusion. All types are polarized and their decomposition trees indexed – this means we have invertible mappings between types and constant/variable indices, leaf indices and types, and, by extension, leaf indices and constant/variable indices. Using these, we may also produce the *mirror image* of a decomposition tree on demand: a tree of the exact same type but opposite polarity, whose leaf indices are the axiom link pairs of the original (regardless of link direction).

15.3.1 Traversal

Our traversal follows the same principles as did before – there's two traversal modes, negative and positive. Traversal begins in negative mode at the root of the conclusion. If we pass through all nodes and no red letters appear on our screen (read: no type errors are thrown) along the way, there's a pinky promise[2] that the proof structure was a proof net. If what follows reads like black magic, it's probably because it is. Obscure as it might seem, it works: followed by η and β normalizations and α conversion, it produces a faithful back-and-forth translation for all the 55 108 unique theorems of Æthel.

[1] Actually a loop is the one thing a proof net should never have.

[2] That's the closest we can do in lieu of a formal guarantee.

15.3.2 Negative Mode

Upon taking a step in negative (upward) mode traversal, we will consult the mirror image of the tree just entered; if it is mapped to either a variable or a constant, we will return the corresponding proof without actually proceeding with the traversal; this allows us to sidestep dangerous η expanded modal forms and their normalizations. If that's not the case, we have no choice other than to actually perform the traversal, taking a different action depending on what the current node is.

Atom In the easiest case, it will just be a leaf – we just need to cross over it and enter through the opposite end in positive (downward) mode. In the case of the node being a type constructor, things get complicated.

Implication When encountering a negative implication (a par node), we will first navigate the right hand side (the functor's result, which is also negative) – whatever we get back will be the *body* of a λ abstraction to-be. The variable to abstract over is indexed by the left hand side (the functor's argument, which is positive), meaning we can *usually* just proceed with the abstraction and return the result (as we know both the variable's type and its index). Exceptionally, if the left hand side tree is rooted in a $\blacklozenge_x$ node, we are in trouble. Its presence alerts us to the fact that the variable we are trying to abstract over is structurally nested, meaning (i) we have its index wrong, and (ii) it is not structurally free for us to abstract. To recover the correct index, we may simply just consult the variable index assigned to the positive tree rooted in the $\blacklozenge_x$ node – when associating variables to formula trees, we *know* that positive boxes rooted in positive diamonds are two distinct variables, to be indexed separately. Having learned its name, we are now free to manipulate it and perform all extractions necessary on the proof body, moving the problem variable to the outermost structural layer of the antecedent structure.[1] At that point, we procure a new proof for the body by a $\blacklozenge_x E$ rule, substituting the structural $\langle \_ \rangle^x$ for a logical diamond and the old variable for a new one, of the correct index. We are then able to proceed with the abstraction and return the result as we'd do normally.

Diamond A negative diamond is one of two things: an actual instruction to apply a diamond introduction rule, or part of a variable's type assignment. As we discussed earlier, the difference between these two is only artificial – but following our drive to avoid modal normalizations, we must figure out which of the two possibilities we're dealing with. To do so, we skip through the diamond to the negative tree above, traverse it as usual, and then inspect the result. If it so happens that it's a proof containing a single variable as its premise (bracketed or otherwise), we return it unaltered. If not, we apply the appropriate $\lozenge I$ rule as directed by the skipped node and return the result.

[1] Beware: this is not a rule applied locally, but a retroactive transformation of the entire proof.

Box To reach a negative box node must mean that the negative adjunct we're traversing is not directly supplied by a constant or a variable, but rather by a complex proof. As before, we skip the node, traverse the nested tree, and apply the appropriate $\square I$ rule to the result. The appearance of a $\square I$ rule shouldn't alienate us – it is just a case of a verbose η redex (considering we don't employ any closure patterns $\square\Diamond$) which will disappear after normalization.

15.3.3 Positive Mode

Upon entering a tree through a leaf in positive (downward) mode, we need to delineate its shape and content by identifying its root, i.e. the lowest positive node we can reach without changing sign (i.e. passing through a negative implication). If that root is a $\blacklozenge_x$ node, we'll go for the immediately previous root instead, i.e. a $\blacksquare_x$ node. We'll then instantiate either a variable or a constant (depending on what kind of proof the tree is associated with) and begin the traversal by pattern matching the entire tree while passing the freshly instantiated proof as context. Positive mode traversal may occassionally require a *cut* to proceed, that being a tuple of a variable and a proof – its role will become apparent in a bit. No cut is provided initially.

Atom If the tree is a singleton, we simpy return the context and call it a day.

Implication If the tree is rooted in an implication and no cut was provided, things look familiar. We simply perform a negative traversal of the left branch, apply the context to it, and then step up the tree passing the updated context. If a cut was provided, we must first unpack its contents. These are to be read as instructions from the *past*, calling for $\Diamond E$ rule to be applied on the context in order to substitute the unpacked variable for the unpacked proof. After the substitution is performed, we may proceed as before.

Diamond A positive diamond is a bit of a wildcard. What's for certain is that we are within some hypothesis, as diamonds in result position was never part of the plan. To figure out what kind of hypothesis we're dealing with, we have to inspect the contained tree above. If it's a leaf, we're inside the type assignment of a complement variable, so we may simply return the context. If it's an implication, we're inside some hypothesis of a very high order type – traversal is ill-advised, but assuredly the entire tree must be associated with a variable which we can just return untouched. If it's a box, things get interesting – we're in an interior pattern $\Diamond\square$. For this to make sense, the two modalities must match in their labels. Assuming they do, the diamond node is prescribing a cut: a substitution to be done in the *future*. We must package the variable associated with the nested (box-rooted) tree together with the passed context. We then continue with the traversal, using the variable as the reset context – when the time comes, it will be substituted for the previous context.

Box The box is again the better behaved of the two modalities. A box in positive mode is simply a call for $\Box E$ rule to be applied on the context before continuing with the traversal.

16 Key References & Further Reading

On this pleasant note, this chapter has reached its overdue end. If excited about the prospects of the work presented here, you're in luck – the field is fresh as a daisy and the possibilities for further developments endless. Below you'll find some pointers to get you started.

Despite digressions against the sequence-to-sequence paradigm, our original supertagger [Kogkalidis et al., 2019] is inspired and preceded by a ton of inventive applications of such models repurposed for structure induction [Vinyals et al., 2015; Wiseman and Rush, 2016; Dong and Lapata, 2016; Buys and Blunsom, 2017, inter alia]. Our early results have somewhat sensitivized the community to the problem of open-domain lexicalized supertagging. Bhargava and Penn [2020] essentially replicate our early experiments (i.e. same venue and paradigm but a year later, except this time on the CCG-bank). Other, serious steps forward include the ones by Prange et al. [2021] and Liu et al. [2021], who concurrently sought to account for the tree-like structure of lexical categories, the former through a tree-biased architecture and the latter through transition-based parsers applied per category at the word level.

These serve as the inspiration for the geometry-informed supertagger of Kogkalidis and Moortgat [2022], which bears semblance and owes credit to various ongoing lines of architectural work. The depth recurrence is evocative of weight-tied architectures [Dehghani et al., 2018; Bai et al., 2019] and their graph-oriented variants [Li et al., 2016], which model neural computation as the fix-point iteration of a single layer against a structured input, thus allowing for a dynamically adaptive computation "depth" – albeit with a constant parameter count. Analogously to structure-aware self-attention networks [Zhu et al., 2019; Cai and Lam, 2020, inter alia] and graph attentive networks [Veličković et al., 2018; Yun et al., 2019; Ying et al., 2021; Brody et al., 2021, inter alia], it also employs standard query/key and fully-connected attention mechanisms injected with structurally biased representations, either at the edge or at the node level. Finally, akin to dynamic graph approaches [Liao et al., 2019; Pareja et al., 2020], it forms a closed loop system that autoregressively generates its own input, in the process becoming exposed to subgraph structures that drastically differ between time steps.

Our neuralification of proof nets [Kogkalidis et al., 2020] is really just the creative application of modern breakthroughs in optimal transport learning and differentiable set representations [Cuturi, 2013; Mena et al., 2018; Grover et al., 2019; Peyré et al., 2019, inter alia], combined with existing insights and intuitions on the application of graph-theoretic machinery for proof search [Moot, 2008]. A year later, Bhargava and Penn [2021] present our operationalization anew, and apply it on a Lambek-adapted subset of the CCG-bank. As for steps forward, Moot [2022] raises a shismatic criticism of proof nets from within, but also provides stimuli and incentives for alternative operationalizations. De Pourtales et al. [2023] are more devout, adapting the architecture for use with a multimodal Lambek calculus targeted for French. Finally, Clark [2021] recounts an up-to-date tale of categorial grammar parsing, and offers new insights along the way, but from the combinatory side of history – this might prove handy if you're looking for alternative perspectives.

Chapter IV Bibliography

M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. In *International conference on machine learning*, pages 1120–1128. PMLR, 2016.

P. Bader, S. Blanes, and F. Casas. Computing the matrix exponential with an optimized Taylor polynomial approximation. *Mathematics*, 7(12):1174, 2019.

D. Bahdanau, K. H. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32, 2019.

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

J.-P. Bernardy and S. Lappin. Assessing the unitary RNN as an end-to-end compositional model of syntax. *arXiv preprint arXiv:2208.05719*, 2022.

A. Bhargava and G. Penn. Supertagging with ccg primitives. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 194–204, 2020.

A. Bhargava and G. Penn. Proof net structure for neural lambek categorial parsing. In *Proceedings of the 17th International Conference on Parsing Technologies and the IWPT 2021 Shared Task on Parsing into Enhanced Universal Dependencies (IWPT 2021)*, pages 13–25, 2021.

S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2021.

J. Buys and P. Blunsom. Robust incremental neural semantic graph parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1215–1226, 2017.

D. Cai and W. Lam. Graph transformer for graph-to-sequence learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7464–7471, 2020.

W. Che, Y. Liu, Y. Wang, B. Zheng, and T. Liu. Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium, Oct. 2018. Association for Computational Linguistics. doi: 10.18653/v1/K18-2005. URL https://aclanthology.org/K18-2005.

K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014a.

K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, 2014b.

K. Clark, M.-T. Luong, C. D. Manning, and Q. Le. Semi-supervised sequence modeling with cross-view training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1914–1925, Brussels, Belgium, Oct.-Nov. 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1217. URL https://aclanthology.org/D18-1217.

S. Clark. Supertagging for combinatory categorial grammar. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+ 6)*, pages 19–24, 2002.

S. Clark. Something old, something new: Grammar-based CCG parsing with transformer models. *CoRR*, abs/2109.10044, 2021. URL https://arxiv.org/abs/2109.10044.

S. Clark and J. R. Curran. The importance of supertagging for wide-coverage CCG parsing. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 282–288, Geneva, Switzerland, aug 23–aug 27 2004. COLING. URL https://aclanthology.org/C04-1041.

S. Clark and J. R. Curran. Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistics*, 33(4):493–552, 2007.

J. R. Curran, S. Clark, and D. Vadas. Multi-tagging for lexicalized-grammar parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 697–704, 2006.

M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26, 2013.

Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 933–941, 2017.

C. de Pourtales, J. Rabault, K. Kogkalidis, and R. Moot. DeepGrail: Neural proof nets for French. Technical report, LIRMM, 2023. forthcoming.

W. de Vries, A. van Cranenburgh, A. Bisazza, T. Caselli, G. van Noord, and M. Nissim. Bertje: A dutch bert model. *arXiv preprint arXiv:1912.09582*, 2019.

M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and Ł. Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2018.

T. Deoskar, M. Mylonakis, and K. Sima'an. Learning structural dependencies of words in the Zipfian tail. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 80–91, Dublin, Ireland, Oct. 2011. Association for Computational Linguistics. URL https://aclanthology.org/W11-2911.

T. Deoskar, C. Christodoulopoulos, A. Birch, and M. Steedman. Generalizing a strongly lexicalized parser using unlabeled data. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 126–134, 2014.

L. Dong and M. Lapata. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany, Aug. 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1004. URL https://aclanthology.org/P16-1004.

M. Fey and J. E. Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

A. Grover, E. Wang, A. Zweig, and S. Ermon. Stochastic optimization of sorting networks via continuous relaxations. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1eSS3CcKX.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

M. Honnibal, J. R. Curran, and J. Bos. Rebanking ccgbank for improved np interpretation. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 207–215, 2010.

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.

N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1700–1709, 2013.

J. Kasai, B. Frank, T. McCoy, O. Rambow, and A. Nasr. TAG parsing with neural networks and vector representations of supertags. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1712–1722, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1180. URL https://aclanthology.org/D17-1180.

K. Kogkalidis and M. Moortgat. Geometry-aware supertagging with heterogeneous dynamic convolutions, 2022. URL https://arxiv.org/abs/2203.12235.

K. Kogkalidis, M. Moortgat, and T. Deoskar. Constructive type-logical supertagging with self-attention networks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 113–123, Florence, Italy, Aug. 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4314. URL https://aclanthology.org/W19-4314.

K. Kogkalidis, M. Moortgat, and R. Moot. Neural proof nets. In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 26–40, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.conll-1.3. URL https://aclanthology.org/2020.conll-1.3.

K. Kogkalidis, M. Moortgat, and R. Moot. SPINDLE: Spinning raw text into lambda terms with graph attention. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, Dubrovnik, Croatia, 2023. Association for Computational Linguistics. To Appear.

H. Larochelle and G. E. Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. *Advances in neural information processing systems*, 23, 2010.

M. Lewis and M. Steedman. Improved CCG Parsing with Semi-supervised Supertagging. *Transactions of the Association for Computational Linguistics*, 2: 327–338, 10 2014. ISSN 2307-387X. doi: 10.1162/tacl_a_00186. URL https://doi.org/10.1162/tacl_a_00186.

M. Lewis, K. Lee, and L. Zettlemoyer. LSTM CCG parsing. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 221–231, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1026. URL https://aclanthology.org/N16-1026.

M. Lezcano Casado. Trivializations for gradient-based optimization on manifolds. *Advances in Neural Information Processing Systems*, 32, 2019.

Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow. Gated graph sequence neural networks. In *International Conference on Learning Representations*, 2016.

R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D. K. Duvenaud, R. Urtasun, and R. Zemel. Efficient graph generation with graph recurrent attention networks. *Advances in Neural Information Processing Systems*, 32, 2019.

W. Ling, C. Dyer, A. W. Black, I. Trancoso, R. Fermandez, S. Amir, L. Marujo, and T. Luís. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1176. URL https://aclanthology.org/D15-1176.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Y. Liu, T. Ji, Y. Wu, and M. Lan. Generating ccg categories. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(15):13443–13451, May 2021. doi: 10.1609/aaai.v35i15.17586. URL https://ojs.aaai.org/index.php/AAAI/article/view/17586.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

L. Martin, B. Muller, P. J. O. Suárez, Y. Dupont, L. Romary, É. V. de la Clergerie, D. Seddah, and B. Sagot. Camembert: a tasty french language model. In *ACL 2020-58th Annual Meeting of the Association for Computational Linguistics*, 2020.

G. Mena, D. Belanger, S. Linderman, and J. Snoek. Learning latent permutations with gumbel-sinkhorn networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Byt3oJ-0W.

V. Mnih, N. Heess, A. Graves, and k. kavukcuoglu. Recurrent models of visual attention. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/09c6c3783b4a70054da74f2538ed47c6-Paper.pdf.

R. Moot. Graph algorithms for improving type-logical proof search. *arXiv preprint arXiv:0805.2303*, 2008.

R. Moot. Reconciling vectors with proofs for natural language processing. Compositionality in formal and distributional models of natural language semantics, 26th Workshop on Logic, Language, Information and Computation (WoLLIC 2019), July 2019. Retrieved from https://richardmoot.github.io/Slides/WoLLIC2019.pdf.

R. Moot. Perspectives on neural proof nets. *arXiv preprint arXiv:2211.04141*, 2022.

A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl, and C. Leiserson. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5363–5370, 2020.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL https://aclanthology.org/N18-1202.

G. Peyré, M. Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

J. Prange, N. Schneider, and V. Srikumar. Supertagging the long tail with tree-structured decoding of complex categories. *Transactions of the Association for Computational Linguistics*, 9:243–260, 2021. doi: 10.1162/tacl_a_00364. URL https://aclanthology.org/2021.tacl-1.15.

O. Press and L. Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, Valencia, Spain, Apr. 2017. Association for Computational Linguistics. URL https://aclanthology.org/E17-2025.

R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, Aug. 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162.

P. Shaw, J. Uszkoreit, and A. Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2074. URL https://aclanthology.org/N18-2074.

N. Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

E. Thomforde and M. Steedman. Semi-supervised CCG lexicon extension. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1246–1256, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics. URL https://aclanthology.org/D11-1115.

Y. Tian, Y. Song, and F. Xia. Supertagging combinatory categorial grammar with attentive graph convolutional networks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6037–6044, 2020.

A. Vaswani, Y. Bisk, K. Sagae, and R. Musa. Supertagging with LSTMs. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 232–237, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1027. URL https://aclanthology.org/N16-1027.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton. Grammar as a foreign language. *Advances in neural information processing systems*, 28, 2015.

S. Wiseman and A. M. Rush. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306, 2016.

T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

W. Xu, M. Auli, and S. Clark. CCG supertagging with a recurrent neural network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 250–255, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-2041. URL https://aclanthology.org/P15-2041.

W. Xu, M. Auli, and S. Clark. Expected F-measure training for shift-reduce parsing with recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 210–220, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1025. URL https://aclanthology.org/N16-1025.

C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34, 2021.

S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

B. Zhang and R. Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

J. Zhu, J. Li, M. Zhu, L. Qian, M. Zhang, and G. Zhou. Modeling graph structure in transformer for better AMR-to-text generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5459–5468, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1548. URL https://aclanthology.org/D19-1548.

CHAPTER V

Conclusion

Because all good things (and bad ones too) must come to an end.

I am slowly running out of words to write, and you're probably running out of words to read, so we best keep this short.

In Retrospect Let's first look back on what we've gone through and try to put everything in context using the gift of hindsight. After a brief survey of substructural type theories and categorial grammars, we set off into the unknown, armed only with the basic tools of logical deduction and a residuated pair of modal operators.

Our first stop was the crossroads between phrase structure grammars and dependency grammars. Comparing them to our freshly adopted categorial heritage, we found them both lacking. Phrase structure grammars are in principle subsumed by categorial grammars, while dependency grammars play a different game altogether, devoid of higher-order interactions. Nonetheless, we saw a merit absent from the categorial vocabulary in the dependency tradition. Dependency grammars' itemization of grammatical functions amounts to an extra dimension, perpendicular to the one of function-argument structures captured by the implicative operators standard in categorial grammars. Having seen past the horizon and unwilling to become entrenched in a flatland of the vaster grammatical expanse, we rejected the dilemma and charted a new, interdimensional course. To navigate this alien territory, we repurposed the modal operators. Previously used for fine-tuning structural control, they became type-conscious ways to impose barriers on structural form and computational meaning, delineating grammatical constituents with named boundaries – and λ terms with named operations – that specify their grammatical role.

Eager to put our apparatus to the test, we sought to employ it in converting a corpus of Dutch syntactic derivations encoded as dependency graphs into a of type-safe spin-off consisting of type-logical proof-derivations. Lacking the patience to engage with the mythical beast of Dutch word order, we simply navigated around it, propelled by a logical system unmoved by phrasal structure permutation. We revelled in the resulting logic's easy and (for the most part) transparent tackling of discontinuous and non-local phenomena, deferring any pain points stemming from linguistic underspecification and lexical type ambiguity to what at the time seemed like the distant future.

Soon thereafter, future became present and the past caught up. Attempting to settle into our type-logical destination with a practically usable real-world parser, we encountered a barren landscape, where none of the blueprints of the modern neural world would readily apply. Our type universe turned out to be too broad and our type lexicon too sparse for standard sequence classification techniques to apply, necessitating a radical overhaul of the supertagging problem. Unhindered, we produced the the first ever instance of a supertagging architecture capable of correctly producing novel type assignments, pointing to a general solution to lexical type sparsity. Employing geometric constraints, we then proceeded to improve upon it, establishing the current state of the art – not just for our own niche corner of the world, but also for the more mainstream combinatorial grammar resort. Finally, we saw how the proof nets of linear logic offer a unique oportunity at quite literally the leanest (and quite possibly the meanest) substructral parser conceived to date, resolving also linguistic underspecification and experimentally proving both the theory and our tooling valid contestants to the status quo.

In Practice Everything produced as part of this thesis is 100% open source and open access since day one. To play with, contribute to, track the progress of, or ask for aid with any of the software and resources developed, I redirect you to the relevant repositories below.

- **Æthel** – https://github.com/konstantinosKokos/aethel
  Æthel, batteries included. Contains the Python backend allowing the representation and manipulation of $\mathbf{LP}_{\diamond,\square}$ proofs as necessary for the implementation of the proof extraction algorithm, and provides user facing utilities for loading, inspecting and searching through Æthel samples with minimal effort. Download links to the verified dataset and versioning changelogs are also provided.
- **Spind**$^{\mathbf{(2)}}\lambda$**e**[1] – https://github.com/konstantinosKokos/spindle
  A curated repackaging of the geometry-aware supertagger, neural proof nets and Æthel's type checker into a single package, intended as a prototype neurosymbolic parser that showcases the combined potential of its

[1]**S**pindle **p**arses **in**to **d**ependency-**d**ecorated λ **e**xpressions.

components. A user can easily invoke the whole system to efficiently parse a collection of sentences with a single command from a Python console.

The above two are actively maintained and shall remain accessible for the foreseeable future – brief user instructions for the both can be found in Appendix A. Publication-specific code is archived for historical purposes and/or to assist with replicating experiments – refer to the paper of interest for the corresponding repository links.

In the Future The future's not ours to see. If however you feel like you could use some suggestions on where this caravan could be headed next, there's quite some vistas inviting further exploration.

For starters, our discussion of modalities as dependencies barely scratches the surface of what the paradigm could offer. In our presentation and for our current purposes, we have employed $\mathbf{LP}_{\diamond,\square}$ not out of linguistic rigor but rather in absence thereof, pragmatic engineering constraints being the main driving force. A more linguistically disciplined exploration of the structures of $\mathbf{(N)LP(P)}_{\diamond,\square}$ could aim to align the agendas of structural annotation and structural control, here treated as distinct and unaware of one another. Different rule vocabularies could be used to account for the named yet empty structural *traces* left behind by movement, or simply to alter the structural properties of the logic depending on the bracketing context – be it of a dependency grammar flavor or otherwise. On another level altogther, it's intriguing to contemplate whether the peculiar, quasi-mobile structures of $\mathbf{LP}_{\diamond,\square}$ used here have any proximity to the surface form of any actual language, artificial or otherwise.

In a twist of a fate, the incorporation of dependency modalities blew up our lexicon, largely counteracting any benefits we would have hoped to reap from ignoring word order, and thus forcing us to actively account for lexical type sparsity. By now, this has lead to the birthing of a series of novel architectures, all addressing the same problem; some our own, some by others. In pushing the boundaries of supertagging, these architectures also implicitly promote the eventual abolition of parsing. Practically usable sparse grammars essentially translate to an open invitation to design and incorporate elaborate morphosyntactic constraints, which in turn reduce the need for parsing by diminishing the output search space. In the extreme, no decisions at all would be delegated to the parser; the correct derivation(s) fully preordained by the type assignments. In the less extreme, if a 128 parameter long parser is able to reliably navigate a proof space as expansive as that of our logic, imagine what we could do with a stricter type system and/or a better supertagger. In all honesty and as a message to future endeavours, if this thesis were to start today, the ingrained directive of taking design decisions that push away from sparsity would have been far less pressing.

Finally, the neuralification of proof nets opens the floodgates to a storm surge of architectural possibilities. Proof search can be recast into a greedy algorithm with the concurrent filtering of structural impossibilities, derivational ambiguity can be tackled by latent variables and noisy sampling, proofs can be encoded using graph neural networks and represented independently of the input sentence, and the list goes on.

Which (if any) of these directions shall be picked up is as much as up to you, dear reader, as it to me.

APPENDIX A

Implementation Notes

The acronym Æthel has two readings, depending on the aspect of the E. Read as automatically extract*ed* theorems from Lassy, it is an elaborate dataset of type-checked derivations of Dutch in $\mathbf{LP}_{\diamond,\Box}$. Read as automatically extract*ing* theorems from Lassy, it is a Python library for extracting, processing and representing these derivations. The two are made for one another; even though they can live independently, they are best presented together. We'll start from the second aspect and move towards the first. The exposition is honest but not exhaustive with respect to the actual code – the intention is not to write a full API reference manual but rather just a quick how-to guide, with occasional commentary motivating design decisions. The full source code is available at https://github.com/konstantinosKokos/aethel.[1] For up-to-date user instructions and data downloads please refer to the repository, which shall remain active and accessible for the foreseeable future.

1 NLP with $\mathbf{LP}_{\diamond,\Box}$

Implementing a type system in an untyped language is a perversity of nature, but makes the code easier to integrate with standard machine learning libraries, for which Python is the de facto choice – we're just planning ahead. The absence of first-class inductive types in Python means we'll have to use the object-oriented abstract class pattern instead – I apologize in advance for the mandatory eye bleach to follow. In spite of the impediments, our implementation of $\mathbf{LP}_{\diamond,\Box}$ proofs is as faithful as possible to the decomposition of

[1] Latest commit is 7e9e4c472df22582708ff03a35cf90718c17c60e.

Chapter I. We disentangle proofs and terms into distinct entities, following our observations that the latter can be less informative than the former (e.g. hiding structural brackets and structural rules, being equivalent under order variations, etc.). Practically, proofs and terms are functionally related but not equivalent, i.e. each proof *induces* or *has* a term, but a term is *not* a proof – it might be either many or none. To carve a path to our objects of interest, we will go from small to big, starting with the basic definitions of structures, types, structural primitives, terms and rules.

1.1 Structures

Structures are implemented as an abstract class, parameterizable with respect to their contents `T`. Structures must be traversable, representable and pairwise comparable, and we must be able to check whether a structure contains an object, where the notion of containment changes depending on the concrete structure under scrutiny.

```
class Structure(abc.ABC, typing.Generic[T]):
    def __repr__(self) -> str: ...
    def __eq__(self, other) -> bool: ...
    def __contains__(self, item) -> bool: ...
```

LP structures are multisets; yet in our use case we do care about order (even if only in the representational sense) – we can treat them as *sequences* instead (altering the notion of equality to account for permutation invariance if/when necessary). The $\diamondsuit, \square$ modalities impose structure in the form of brackets – we can treat them as *unary* containers. A container structure will carry a name for its brackets; strictly speaking a tagged union object, but implemented as a string. To canonicalize ambiguous representations and remain faithful to the absence of tree-like recursion as described in Section 8.4.1, we impose that a unary must necessarily contain a sequence (may as well be a singleton), while a sequence can contain either unary structures or elementary objects (i.e. no sequences of sequences). With this simple mutual induction in mind, we arrive at the definitions below (the type hints serve as little more than mental notes).

```
class Unary(Structure[T]):
    content:  Sequence[T]
    brackets: str

class Sequence(Structure[T], typing.Sequence[T]):
    structures: tuple[T | Unary[T], ...]
```

1.2 Types

Next, we take the inductive type grammar and break it into multiple classes, starting with the abstract class that all concrete patterns inherit from.[1] Types must be representable and implement equality, while also providing auxiliary functionalities like back-and-forth translations between prefix and infix notation, computing order, stripping modalities, etc.

```
class Type(abc.ABC):
    def __repr__(self) -> str: ...
    def __eq__(self, other) -> bool: ...
    def __abs__(self) -> Type: ... # removes modalities
    def order(self) -> int: ...    # see I.9
    def prefix(self) -> str: ...

    @staticmethod
    def parse_prefix(prefix: str) -> Type: ...
```

Different type patterns are then defined as different concrete classes. Atoms are simple: they just contain a sign that allows their identification. Functors are defined coordinate-wise (having an argument to the left of the arrow and a result to the right). Modal quantifications obey the same abstraction (having a content and a decoration), differing only in whether they are a diamond or a box.

```
class Atom(Type):
    sign: str

class Functor(Type):
    argument: Type
    result:   Type

class Modal(Type, ABC):
    content:    Type
    decoration: str

class Box(Modal):
    ...

class Diamond(Modal):
    ...
```

By inheritance, concrete type objects are instances of both their respective construction patterns and the abstract class `Type`. On the basis of the above, we implement a tiny calculator responsible for performing operations on types and asserting their validity.

```
class TypeInference:
    class TypeCheckError(Exception): ...
```

[1] Older implementations had `Type` be an abstract factory pattern, with each type-pattern being a concrete factory, with the intention of dynamically constructing types that are indeed distinct python "types". This would then allow the native creation of terms of the appropriate "type". In hindsight, the extra complexity was far from worth it – tangling up $\mathbf{LP}_{\diamond,\Box}$ types and Python "types" offers little practical value aside the cheap thrill of calling `type()` on some proof object and reading back an actual formula. The two implementations are mutually compatible, though.

```
    @staticmethod
    def assert_equal(a: Type, b: Type) -> None: ...
    @staticmethod
    def arrow_elim(functor: Type, argument: Type) -> Type: ...
    @staticmethod
    def box_elim(inner: Type, box: str | None) -> tuple[Type, str]: ...
    @staticmethod
    def dia_elim(inner: Type, dia: str | None) -> tuple[Type, str]: ...
```

1.3 Terms

In the exact same vein, we have to define a painstaking number of different classes to capture all the ways we can cook up a term. We'll adorn terms with types to ensure a first layer of well-typedness pertaining to logical constraints. Rather than redundantly transcribe the type of all complex terms, we compute it dynamically on the basis of their primitive parts and the operations that bind them.

```
class Term(abc.ABC):
    def __repr__(self) -> str: ...
    def __eq__(self, other) -> bool: ...
    def vars(self) -> Iterable[Variable]: ...
    def constants(self) -> Iterable[Constant]: ...

    @property
    @abstractmethod
    def type(self) -> Type: ...
```

On top of an index allowing their identification, variables and constants must then also carry their types on their sleeve.

```
class Variable(Term):
    type:  Type
    index: int

class Constant(Term):
    type:  Type
    index: int
```

All other terms consist of subterms which allow the inductive computation of their type. This is also used at instantiation time to assert that the term is well-formed.

```
class ArrowElimination(Term):
    function: Term
    argument: Term

class ArrowIntroduction(Term):
    abstraction: Variable
    body:        Term

class DiamondIntroduction(Term):
```

```
    decoration: str
    body:       Term

class BoxElimination(Term):
    decoration: str
    body:       Term

class BoxIntroduction(Term):
    decoration: str
    body:       Term
```

Since we are encoding term patterns rather than proofs, it makes sense to decompose the term rewrite prescribed by the $\Diamond E$ rule into the two different patterns it involves: one for the actual removal of a diamond, done retroactively, and one for the substitution, done locally.

```
class DiamondElimination(Term):
    decoration: str
    body:       Term

class CaseOf(Term):
    becomes:  Term
    where:    Term
    original: Term
```

This is already showing how proofs and terms diverge. A valid (sub-)term is not necessarily a valid proof: its validity can only be asserted given some external context (and under structural conditions it is blind to), in turn relying on our definition of proof.

1.4 Proofs

To close the circle, we start by mimicking the definition of a judgement as an antecedent structure of variables and constants and a succedent term (which carries a type). As before, we restrict the assumptions to being a `Sequence` for the sake of canonicalization.

```
class Judgement:
    assumptions: Sequence[Variable | Constant]
    term:        Term
```

Proof constructors are the logic's rules, which for the most part overlap with term patterns. Exactly because of the exceptional cases where they don't, we need to actually implement them anew.

```
class Rule(enum.Enum):
    def __repr__(self) -> str: ...
    def __str__(self) -> str: ...
    def __call__(self, *args, **kwargs) -> Proof: ...
```

Rules are essentially implemented as enumerated types mapped to dynamically checked operations on proofs. An organizational distinction is made between logical rules and the sole structural rule.

```
class Logical(Rule):
    Variable = ...
    Constant = ...
    ArrowElimination = ...
    ArrowIntroduction = ...
    DiamondIntroduction = ...
    BoxElimination = ...
    BoxIntroduction = ...
    DiamondElimination = ...

class Structural(Rule):
    Extract = ...
```

At long last, we have all the components necessary to define a proof. A proof is a record of zero or more premises (themselves proofs), a conclusion (a verified judgement), the last rule of inference used to bind the premises together (an identifier of the previous enumeration), and maybe a variable under focus (used to tell which variable is abstracted over or substituted, for rules $\multimap I$ and $\diamondsuit E$ respectively). A proof has a structure (that of its conclusion's antecedent), a term (that of its conclusion succedent) and a type (that of its term's). Other than being comparable, representable and yada yada, and on top of some proof-theoretic utilities, proof objects provide instance-level access to the compositional operations implemented by rules, allowing (relatively) easy bottom-up synthesis.

```
class Proof:
    premises:   tuple[Proof, ...]
    conclusion: Judgement
    rule:       Rule
    focus:      Variable | None

    def __repr__(self) -> str: ...
    def __str__(self) -> str: ...
    def __eq__(self, other) -> bool: ...

    # self-applied rule shortcuts
    def apply(self, other: Proof) -> Proof: ...
    def diamond(self, diamond: str) -> Proof: ...
    def box(self, box: str) -> Proof: ...
    def unbox(self, box: str | None) -> Proof: ...
    def undiamond(self, where: Variable, becomes: Proof) -> Proof: ...
    def abstract(self, var: Variable) -> Proof: ...
    def extract(self, var: Variable) -> Proof: ...

    def standardize_vars(self) -> Proof: ...
    def eta_norm(self) -> Proof: ...
    def beta_norm(self) -> Proof: ...
    def is_linear(self) -> bool: ...
    def subproofs(self) -> Iterator[Proof]: ...
```

Since we now have access to the full picture, rule applications are sufficiently informed to assert all validity checks necessary, both logical and structural.

1.5 Examples

To see this in action, the snippets below showcase the construction of simple proofs seen through Chapter I. But first, some type shortcuts to make our lifes easier:

```
>>> A = Atom('A')             # A
>>> B = Atom('B')             # B
>>> C = Atom('C')             # C
>>> bA = Box('a', A)          # □aA
>>> dbA = Diamond('a', bA)    # ◇a□aA
>>> dA = Diamond('a', A)      # ◇aA
>>> bdA = Box('a', dA)        # □a◇aA
```

Then a proof pattern shortcut that from a type and an index creates the identity proof of the corresponding variable:

```
>>> def var(t: Type, i: int) -> Proof:
>>>     return Logical.Variable(Variable(_type=t, index=i))
```

With these, we can create our first toy proofs, like the axiom of identity for some type A or the function composition of A⊸B and B⊸C:

```
>>> (x := var(A, 0)).abstract(x.term)
⊢ (λx0.x0) : A⊸A
>>> x = var(A, 0)
>>> f = var(Functor(A, B), 1)
>>> g = var(Functor(B, C), 2)
>>> g.apply(f.apply(x)).abstract(x.term)
x2, x1 ⊢ (λx0.x2 (x1 x0)) : A⊸C
```

The story is no different for the modalities; here's deriving the interior and closure operators:

```
>>> var(A, 0).diamond('a').box('a')
x0 ⊢ ▽a(△a(x0)) : □a(◇a(A))
>>> (x := var(bA, 0)).unbox().undiamond(where=x.term,
                                        becomes=var(dbA, 1))
x1 ⊢ case ▲a(x1) of x0 in ▼a(x0) : A
```

As long as we refrain from initializing proof objects manually or mutating their values, rules will block us from making illegal moves:

```
>>> var(Functor(A, B), 0).apply(var(B, 1)
TypeCheckError: A⊸B is not a functor of B
>>> var(A, 0).box('a')
ProofError: x0 : A is not a singleton containing a unary
```

2 Manipulating Æthel

To allow the incorporation of extra-theoretical information, proofs are repackaged with the sentence into a `Sample` record, containing also a name (the Lassy identifier of the source file, plus a node identifier indicating the pruning point) and a subset specification, imposing a canonical train/dev/test split.

```
class Sample:
    lexical_phrases:    tuple[LexicalPhrase, ...]
    proof:              Proof
    name:               str
    subset:             str

    def __len(self)__ -> int: ...
    def __repr__(self) -> str: ...
    def show_term(self, show_types: bool, show_words: bool) -> str: ...

    @property
    def sentence(self) -> str: ...
```

Tokenization and token-level annotations are provided as a record field, populated by breaking the sentence apart into a variadic tuple of lexical phrases. Each lexical phrase consists of one or more lexical items, but is given a single type assignment and enacts a singularly indexed proof constant. 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 permits faithfully presenting 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.

```
class LexicalPhrase:
    items:  tuple[LexicalItem, ...]
    type:   Type

    @property
    def string(self) -> str: ...
    def __repr__(self) -> str: ...
    def __len__(self) -> int: ...

class LexicalItem:
    word:   str
    pos:    str
    pt:     str
    lemma:  str
```

Finally, the entire dataset is packaged into a `ProofBank` record; practically a list of samples with some extra niceties on top, including indexing utilities and a version field used to tell different temporal instances apart. Loading from a binarized dump is done by a convenience static function.

```
class ProofBank:
    version: str
    samples: list[Sample]

    def __getitem__(self, item: int) -> Sample: ...
    def __len__(self) -> int: ...
    def find_by_name(self, name: str) -> list[Sample]: ...
    def __repr__(self) -> str: ...

    @staticmethod
    def load_data(path: str) -> ProofBank: ...
```

2.1 User Interface

User interface is barebones but practical. The user procures (some version of) the source code from the official repository and a compatible binarized dump of the dataset (download links are also provided there). From then on, the dataset can easily be loaded as follows:

```
>>> from LassyExtraction import ProofBank
>>> aethel = ProofBank.load_data('path/to/dump')
Loading and verifying path/to/dump...
Loaded æthel version 1.x.x containing xxxxx samples.
```

Individual samples can be indexed numerically, or found by name, and their attributes can be explored according to the earlier specifications; here's inspecting the sample of Figure III.9, for instance:

```
>>> sample = aethel.find_by_name('WS-U-E-A-0000000016.p.37.s.1(3)')[0]
>>> sample.sentence
"Auto's die niet starten"
>>> sample.lexical_phrases
(LexicalPhrase(string=Auto's, type=NP, len=1),
 LexicalPhrase(string=die, type=(◇relcl(◇su(VNW)⊸SSUB))⊸□mod(NP⊸NP), len=1),
 LexicalPhrase(string=niet, type=□mod(SSUB⊸SSUB), len=1),
 LexicalPhrase(string=starten, type=◇su(VNW)⊸SSUB, len=1))
>>> sample.lexical_phrases[0].items[0]
LexicalItem(word="Auto's", pos='noun', pt='n', lemma='auto')
>>> sample.proof.structure
⟨c1, ⟨⟨c2⟩mod, c3⟩relcl⟩mod, c0
>>> sample.proof.term
▼mod(c1 △relcl((λx0.▼mod(c2) (c3 x0)))) c0 : NP
```

2.2 Visualization

Proofs and samples can be converted to pdf format for easier inspection; the conversion utility simply follows along the inductive definitions of proofs and terms, casting them into corresponding LaTeX code. All proofs rendered in this document were built this way. Assuming you have pdflatex installed, you can replicate this by running:

```
>>> from LassyExraction.utils.tex import compile_tex, sample_to_tex
>>> tex_code = sample_to_tex(sample)
>>> compile_tex(tex_code, './output.pdf') # see Figure III.9
```

2.3 Corpus Search

To facilitate corpus exploration, we provide search utilities in the form of composable queries and a lazy search function. New queries can be made bottom-up from custom boolean predicates applied to samples, or point-free composed from existing queries using standard boolean operations. Preimplemented queries include searching for samples utilizing only (a subset of) some rules, enumerating a specific number of constants, having some specific word or lemma or being of a specific type, etc.

```
>>> from scripts.search import (search, contains_word,
>>>                             length_between, of_type)
>>> x = next(search(aethel, contains_word('vuur')
>>>                        & length_between(4, 7)))
>>> x.name
'WS-U-E-A-0000000004.p.28.s.3(1).xml'
>>> x.sentence
'Het vuur greep snel om zich heen .'
```

And here's an example of what the creation of a primitive query looks like; we start from the creation of a boolean predicate on samples that tells us whether the opening letters of each word follow a reverse alphanumeric order:

```
def alphanumerically_ordered(sample: Sample) -> bool:
    first_letters = [lp.string[0].lower()
                     for lp in sample.lexical_phrases]
    return first_letters == sorted(set(first_letters), reverse=True)
```

which we can then wrap into a `Query` and compose it with a type condition to find the longest sentence with alphanumerically descending initial word letters:

```
>>> sample = sorted(search(aethel,
>>>                        Query(alphanumerically_ordered)
>>>                        & of_type(Atom('SMAIN')),
>>>                 key=len,
>>>                 reverse=True))[0]
>>> sample.sentence
'Ze vinden nog een bombrief ...'
```

Less (or more) ad-hoc searches can be similarly written, predicating over any of the properties enclosed within a `Sample`; i.e. proof depth, maximal type order, variable count, nestedness of verbal complements, presence of a part of speech tag, or anything of the sort.

3 Neural Interfacing: Spindλe

A practical user-facing front integrating the neural proof search engine described in Section 15 and the type system's implementation can be found at https://github.com/konstantinosKokos/spindle. To install, refer to the instructions provided online; they invole downloading the source code, installing prerequisite packages and obtaining a copy of the pretrained model's parameters. Aftewards, the model can be invoked using:

```
>>> from inference import InferenceWrapper
>>> inferer = InferenceWrapper(weight_path='/path/to/model/weights')
```

and analyses requisitioned with:

```
>>> analyses = inferer.analyze(
>>>     ['omdat ik Henk haar de nijlpaarden zag helpen voeren',
>>>      'Wat is de lambda term van deze voorbeeldzin?'])
```

What we get back is a list of Python objects that partially abide by the `Sample` protocol, each containing a field for lexical phrases and a proof, one such object per input sentence. Lexical phrases are chunked by the supertagger, but lemma and part of speech information are not provided. The compatibility between parser-produced analyses and Æthel samples allows us to use on the former methods originally intended for the latter. More importantly, it asserts that what we get back is not a cheap, duck-typed imitation of a proof, but a proof proper. The parser is by construction bound to either give us some proof, or, if it fails, the reason for its failure.

```
>>> analyses[0].lexical_phrases
(LexicalPhrase(string=omdat, type=◇cmpbody(SSUB)⊸CP, len=1),
 LexicalPhrase(string=ik, type=VNW, len=1),
 LexicalPhrase(string=Henk, type=NP, len=1),
 LexicalPhrase(string=haar, type=VNW, len=1),
 LexicalPhrase(string=de, type=□det(N⊸NP), len=1),
 LexicalPhrase(string=nijlpaarden, type=N, len=1),
 LexicalPhrase(string=zag, type=◇vc(INF)⊸◇obj1(NP)⊸◇su(VNW)⊸SSUB,
 ↪  len=1),
 LexicalPhrase(string=helpen, type=◇vc(INF)⊸◇obj1(VNW)⊸INF, len=1),
 LexicalPhrase(string=voeren, type=◇obj1(NP)⊸INF, len=1))
>>> analyses[0].proof.term
c0 △cmpbody(c6 △vc(c7 △vc(c8 △obj1(▼det(c4) c5)) △obj1(c3)) △obj1(c2)
 ↪  △su(c1)) : CP
>>> analyses[1].lexical_phrases
(LexicalPhrase(string=Wat, type=(◇whbody(◇predc(VNW)⊸SV1))⊸WHQ,
 ↪  len=1),
 LexicalPhrase(string=is, type=◇predc(VNW)⊸◇su(NP)⊸SV1, len=1),
 LexicalPhrase(string=de, type=□det(N⊸NP), len=1),
 LexicalPhrase(string=lambda, type=□mod(N⊸N), len=1),
 LexicalPhrase(string=term, type=N, len=1),
 LexicalPhrase(string=van, type=◇obj1(NP)⊸□mod(NP⊸NP), len=1),
 LexicalPhrase(string=deze, type=□det(N⊸NP), len=1),
 LexicalPhrase(string=voorbeeldzin, type=N, len=1),
 LexicalPhrase(string=?, type=PUNCT, len=1))
```

```
>>> analyses[1].proof.term
c0 △whbody((λx0.c1 x0 △su(▼mod(c5 △obj1(▼det(c6) c7)) (▼det(c2)
 ↪  (▼mod(c3) c4))))) : WHQ
```

Samenvatting in het Nederlands

Sinds hun ontstaan zijn categoriale grammatica's koplopers in de zoektocht naar een formeel elegante, computationeel aantrekkelijke en voldoende flexibele theorie van vorm en betekenis in natuurlijke taal. Ontwikkelingen in de theoretische informatica hebben er gaandeweg toe geleid dat categoriale grammatica's in de traditie van J. Lambek in hun ware aard worden begrepen als volwaardige typesystemen. Woorden krijgen de status van getypeerde constanten, grammaticale regels voor hun interactie nemen de vorm aan van type inferentieregels, die grotere woordgroepen opbouwen uit kleinere bouwstenen. Het eindresultaat van dat proces is tegelijkertijd een taalkundige analyse, een logisch bewijs en een programma. Het overbrugt zo de ogenschijnlijk ongelijksoortige gebieden van taalkunde, formele logica en informatica, en vormt daarmee een manifestatie van het heilige drieluik van taal, logica en rekenen. De traditionele aanpak van de overgang van vorm naar betekenis bouwt voort op Montague's idee van structuurbehoudende vertalingen die nuances van het syntactische typesysteem vereenvoudigen of verwijderen om te komen tot een uniforme en expressieve semantische calculus. Deze aanpak is aantrekkelijk, maar hij leidt tot pragmatische problemen die de ontwikkeling van het categoriale onderzoeksprogramma in de weg staan. Om het semantische niveau van betekenisopbouw te bereiken, heeft men geen andere keuze dan te beginnen bij het moeilijkste deel, namelijk de typetheoretische behandeling van de syntaxis van natuurlijke taal. Verschijnselen zoals verplaatsing, variatie in woordvolgorde, discontinuïteiten en dergelijke vereisen een zorgvuldige behandeling die algemeen genoeg moet zijn om het volledige scala aan grammaticale uitingen te omvatten, maar tegelijkertijd strikt genoeg om ongrammaticale uitingen uit te sluiten.

Dit proefschrift breekt met de traditie en richt zich rechtstreeks op een diepere calculus van het grammaticale bouwproces die de details van de oppervlaktesyntaxis voor zover mogelijk buiten beschouwing laat. Waar functionele syntactische typen voorheen gespecificeerd waren voor informatie die hun lineaire ordening en plaats in een binaire boomstructuur bepaalt, gaan we hier over op een typetheorie die agnostisch is voor zowel hierarchische boomstructuur als lineaire volgorde, met een geringere behoefte aan fijnmazige syntactische onderscheidingen als resultaat. Deze vereenvoudiging leidt er wel toe dat logische bewijsbaarheid en grammaticaliteit niet langer in de pas lopen: de lakse typecalculus staat meer bewijzen toe dan taalkundig wenselijk is. Om deze onderspecificatie gedeeltelijk te omzeilen, neemt het proefschrift een extra stap weg van de gevestigde norm, door het typesysteem te verrijken met unaire operatoren voor die het analytische bereik uitbreiden van gewone functie-argumentstructuren naar functie-argumentstructuren met vaste grammaticale rollen. De nieuwe typecalculus produceert gemengde unaire/n-aire bomen, waarbij elke unaire boom een dependentiedomein afbakent, en elke n-aire boom eronder de uitdrukkingen die samen dit domein vormen. Hoewel ze nog steeds ondergespecificeerd zijn, laten deze nieuwe unaire/n-aire structuren zich lezen als niet-projectieve gelabelde dependentiebomen. Meer dan dat, ze hebben hun wortels stevig in de typetheorie, wat de weg vrijmaakt voor hun betekenisvolle semantische interpretatie.

Op een meer praktisch niveau en om de expressieve geschiktheid van het formalisme te onderzoeken, is een algoritme ontworpen waarmee syntactische analyses van Nederlandse zinnen, weergegeven als dependentiestructuren (afkomstig van het kleine Lassy-corpus), omgezet kunnen worden in bewijzen van de doellogica. De overgrote meerderheid van de Lassy invoeranalyses wordt met succes omgezet, wat aanleiding geeft tot een grote en veelzijdige bewijsbank, een verzameling zinnen gecombineerd met semantische stellingen en hun bijbehorende programma's, en een uitgebreid typelexicon, dat typetoewijzingen biedt voor ongeveer een miljoen lexicale tokens binnen een gegeven taalkundige context.

De bewijsbank en het onderliggende typelexicon kunnen beide worden gebruikt als trainingsdata voor het ontwerpen en implementeren van een flexibele stellingbewijzer, een neurosymbolisch systeem dat in staat is om efficiënt door de uitgebreide zoekruimte van de typenlogica te navigeren.

Het systeem bestaat uit drie hoofdcomponenten die elkaar afwisselen binnen de verwerkingspijplijn.

Component nummer één is een supertagger die verantwoordelijk is voor het toewijzen van een type aan elk invoerwoord. De tagger gaat uit van een hyperefficiënte heterogene grafenconvolutiekernel met state-of-the-art nauwkeurigheid wat betreft datasets voor categoriale grammatica's. In plaats van typetoewijzingen te produceren in de vorm van voorwaardelijke kansen over een vooraf gedefinieerd typevocabulaire, bouwt de supertagger in plaats daarvan de typen dynamisch op, in overeenkomst met hun algebraïsche decompositie. Daarmee is de tagger niet beperkt door dataschaarsheid en ondervertegenwoordiging, goed te generaliseren naar zeldzame typetoewijzingen en zelfs in staat om correcte toewijzingen te produceren voor typen die nooit tijdens de training zijn gezien. Component nummer twee is een neurale permutatiemodule die het lineaire karakter van de doellogica benut om het zoeken naar bewijzen in de vorm te gieten van een proces van optimaal transportleren, waarbij voorzieningen (voorwaardelijke validiteiten) worden gekoppeld aan de processen die ze nodig hebben (voorwaarden). Deze herformulering maakt een gemakkelijk te optimaliseren implementatie mogelijk die berust op een massief parallellisme, en die de onderbrekingen voor structuurmanipulatie van conventionele parsers vermijdt. Component nummer drie is het typesysteem zelf, dat verantwoordelijk is voor het navigeren door de geproduceerde bewijsstructuren en voor het daarmee vaststellen van hun welgevormdheid. De resultaten tonen een efficiëntie superieur aan, en prestaties vergelijkbaar met, de standaard basislijnen voor categoriale formalismen, en dat ondanks de onderspecificatie inherent aan de gebruikte typenlogica.

Curriculum Vitae

Konstantinos was born in Thessaloniki, Greece in 1991. He obtained his engineering diploma from the School of Electrical & Computer Engineering of the Aristotle University of Thessaloniki with a specialization in Electronics & Computers in 2017. Afterwards, he enrolled in the Artificial Intelligence MSc programme of Utrecht University. He graduated cum laude in 2019 with a specialization in Logic & Reasoning. During his second year as a graduate student, he was offered employment as a PhD candidate in the NWO Project "A composition calculus for vector-based semantic modelling with a localization for Dutch" hosted by the Utrecht Institute of Linguistics OTS and supervised by Michael Moortgat. This manuscript summarizes four years of research on the topic.