Title of Your Dissertation

A Suitable Subtitle

on Multiple Lines

Published by

LOT
Trans 10
3512 JK Utrecht
The Netherlands

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

Cover illustration: The Tower of Babel, by Pieter Bruegel

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

Title of Your Dissertation

A Suitable Subtitle

on Multiple Lines

Nederlandstalige Titel

De Tweede Regel van Je Titel,

ook Vrij Lang

(met een samenvatting in het Nederlands)

Proefschrift

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

door

Konstantinos Kogkalidis

geboren 19 Juli 1991
te Thessaloniki, Greece

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

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

todo

Contents

Preface

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

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

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

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

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

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

What this thesis is about

todo
contributions
sectioning

CHAPTER I

Introduction

"In the beginning, there were types."

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

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

In a more niche (but equally beautiful) fragment of the academic world, and in parallel to the above developments, applied logicians and formally inclined linguists have been demonstrating a stunning perserverance in their self-imposed quest of modeling natural language syntax and semantics, making do only with the vocabulary provided by formal logics. The modern incarnation of this noble endeavour is due to Jim Lambek, who was the first to point out that the grammaticality of a natural language utterance can be equated to provability in a certain logic (or type inhabitation, if one is to borrow the terminology of constructive type theories), if the grammar (a collection of empirical linguistic rules) were to be treated as a substructural logic (a collection of formal mathematical rules). Funnily enough, the kind of logics Lambek would employ for his purposes would be exactly those at the interesection of intuitionistic and linear logic, the latter only made formally explicit in a breakthrough paper by Jean-Yves Girard almost three decades later. By that time, Richard Montague had already come up with the fantastically novel idea of seeing no distinction between formal and natural languages, single-handedly birthing and popularizing the field of formal semantics (which would chiefly invole semantic computations using λ-calculus notation), thus fulfilling 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

[1] In proof theory, at least.

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

the Curry-Howard correspondence, and are therefore able to drive semantic composition virtually for free (in fact one could go as far as to say that they are the only kind of machinery that can accomplish such a feat without being riddled with ad-hoc transformations). This revelation, combined with the contemporary bloom of substructural logics, was the spark that ignited a renewed interest in Lambek's work. The culmination point for this interest was typelogical 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. No novel contributions are to be found here; the intention is to establish some common grounds before we get to proceed. If confident in your knowledge of the subject matter, `goto` Chapter II.

Key References

1 The Simple Theory of Types

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

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

1.1 Intuitionistic Logic

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

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

$$\mathrm{A}, \mathrm{B}, \mathrm{C} ::= p \mid \mathrm{A} \to \mathrm{B} \mid \mathrm{A} \times \mathrm{B} \qquad (\mathsf{IL}_{\to,\times})$$

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

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

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

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

$$\Gamma, \Delta, \Theta ::= 1 \mid \mathrm{A} \mid \Gamma, \Delta$$

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

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

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

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

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

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

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

(a) Logical rules.

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

$$\frac{\Gamma \vdash \mathrm{B}}{\Gamma, \mathrm{A} \vdash \mathrm{B}}\ Weakening \qquad \frac{\Gamma, \mathrm{A}, \mathrm{A} \vdash \mathrm{B}}{\Gamma, \mathrm{A} \vdash \mathrm{B}}\ Contraction$$

(b) Structural rules.

Figure I.1: Intuitionistic Logic.

Structural rules, on the other hand, allow us to manipulate structures; they are presented in Figure I.1b. Structural rules have a two-fold role. First, they explicate an extra propery of our structure binding operator, namely commutativity. One could also make do with an implicit $Exchange$ rule by treating structures as *multisets* rather than lists – having it explicit, however, will keep us conscious of its presence and strengthen our emotional bond to it, in turn making us really notice its absence when it will no longer be there (it also keeps the presentation tidier). Second, they give an account of the status of propositions as reusable resources. The $Weakening$ rule states that if we were able to derive a proposition B from some context Γ, we will also be able to do so if the context were to contain some arbitrary extra proposition A. Conversely, the $Contraction$ rule states that if we needed a context 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). Its branches are themselves proofs, fused together by a rule – the number of premises being the local tree's arity. At its leaves, it has identity axioms – the smallest kind of proof.

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 applications that can altogether removed).

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

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

1.2 The Curry-Howard Correspondence

The Curry-Howard correspondence asserts an equivalence between the above presentation of the logic in natural deduction, and a system of computation known as the λ-calculus. It was first formulated by Haskell Curry in the 30s before being independently rediscovered by William Alvin Howard and Nicolas Goveert de Bruijn in the 60s [Curry, 1934; de Bruijn, 1983; Howard, 1980]. The entry point for such an approach is to interpret propositions as *types* of a minimal functional programming language (a perhaps more aptly named alternative to the Curry-Howard correspondence is the *propositions-as-types interpretation*). In that sense, the set of propositional constants Prop_0 becomes the

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

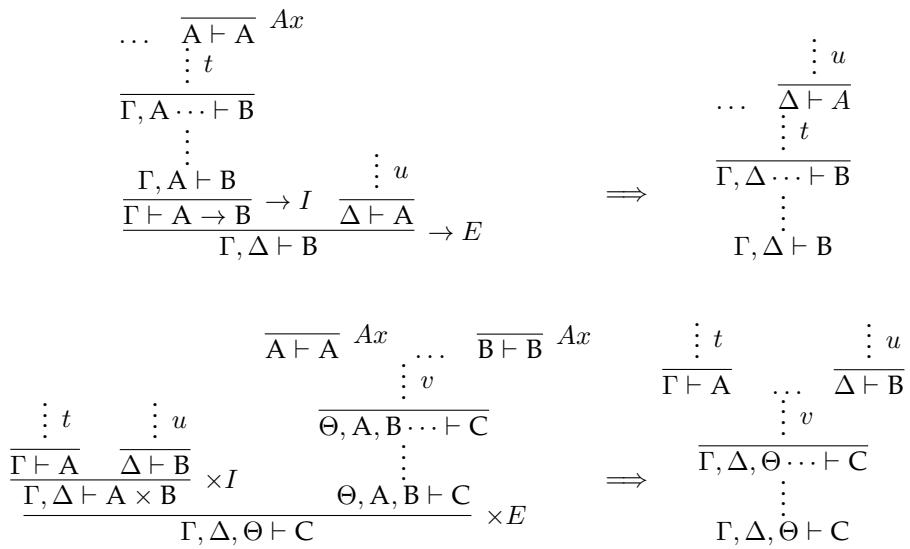


Figure I.2: Intuitionistic proof reductions.

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

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

$$\mathsf{x}_1 : \mathrm{A}_1,\ \mathsf{x}_2 : \mathrm{A}_2 \ldots\ \mathsf{x}_n : \mathrm{A}_n$$

i.e. rather than a sequence of formulas, we have a sequence of distinct variables, each of a specific type, and a judgement $\Gamma \vdash \mathsf{s} : \mathrm{B}$ will now denote the

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

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

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

$$\frac{\Gamma, \Delta \vdash \mathsf{s : A}}{\Delta, \Gamma \vdash \mathsf{s : A}}\ Exchange$$

$$\frac{\Gamma \vdash \mathsf{s : B}}{\Gamma, \mathsf{x_i : A} \vdash \mathsf{s : B}}\ Weakening \qquad \frac{\Gamma, \mathsf{x_i : A}, \mathsf{x_j : A} \vdash \mathsf{s : B}}{\Gamma, \mathsf{x_k : A} \vdash \mathsf{s}_{[\mathsf{x_i} \mapsto \mathsf{x_k}, \mathsf{x_j} \mapsto \mathsf{x_k}]} \mathsf{: B}}\ Contraction$$

Figure I.3: Simple type theory.

derivation of a term s of type B out of such an environment.

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

There's just a few catches to beware of. The first has to do with tracing variables in a proof; the concatenation of structures Γ, Δ is only valid if Γ and Δ contain no variables of the same name; if that were to be the case, we would be dealing with variable shadowing, a situation where the same name could ambiguously refer to two distinct objects (a horrible thing). The second has to do with do with the $Exchange$ 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 $\mathsf{x_i} : \mathsf{A}$ and $\mathsf{x_j} : \mathsf{B}$ can be substituted by a single (derived) term of their product type $\mathsf{s} : \mathsf{A} \times \mathsf{B}$. 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 $Contraction$ rule contains meta-notation that is not part of the term syntax itself. That is to say, $\mathsf{s}_{[\mathsf{x_i} \mapsto \mathsf{t}]}$ is *not* a valid term – even if the result of the operation it denotes is. Generally speaking, substitution of objects for others of the same type is (modulo variable shadowing) an admissible property of the type system. Mixing syntax and meta-syntax in the same system is a dirty trick we will sporadically employ; this surely invites some trouble, but conscious use of it can be worth it, since it significantly simplifies presentation.

Term Equivalences

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

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

$$\lambda x_i.x_i^{\mathsf{A}} \overset{\alpha}{\equiv} \lambda x_j.x_j^{\mathsf{A}}$$

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

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

deconstruction of a just-constructed type:

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

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. 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 fact computation. This ground-shattering discovery lies at the essence of the Curry-Howard correspondence.

η conversion In contrast to β conversion, which tells us how to simplify an introduce-then-eliminate pattern, η conversion tells us how to simplify an eliminate-then-introduce pattern. An η long form of a term is one in which the arguments to type operators are made explicit, whereas an η contracted (or pointfree) form is one where arguments are kept hidden. We refer to the simplification of an expanded form as η reduction, and to the reverse process as η expansion; either is a form of η conversion. The equivalence relation enacted by this conversion is called η equivalence.

$$\lambda x_i.s\ x_i \overset{\eta}{\equiv} s$$
$$(\text{case } s \text{ of } (x_i\ ,x_j\ ) \text{ in } x_i\ ,\ \text{case } s \text{ of } (x_k\ ,x_l\ ) \text{ in } x_l\ ) \overset{\eta}{\equiv} s$$

In place of a summary

Table I.1 summarizes the subsection.

1.3 Intermezzo: Useful Boni

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.

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

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

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

Table I.2: Polarity induction.

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

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

Function Order The implication-only fragment of the logic includes $\to$ as its sole logical connective. The resulting type theory is one that deals only with

$$\dfrac{\dfrac{\dfrac{}{\mathsf{x}_0 : \mathrm{A} \to \mathrm{C} \vdash \mathsf{x}_0 : \mathrm{A} \to \mathrm{C}}\ Ax \quad \dfrac{}{\mathsf{x}_1 : \mathrm{A} \vdash \mathsf{x}_1 : \mathrm{A}}\ Ax}{\mathsf{x}_0 : \mathrm{A} \to \mathrm{C}, \mathsf{x}_1 : \mathrm{A} \vdash \mathsf{x}_0\ \mathsf{x}_1 : \mathrm{C}}\ \to E}{\mathsf{x}_1 : \mathrm{A} \vdash \lambda \mathsf{x}_0.\mathsf{x}_0\ \mathsf{x}_1 : (\mathrm{A} \to \mathrm{C}) \to \mathrm{C}}\ \to I$$

Figure I.4: Type raising.

functions; for its types, we can define their order $\mathcal{O}$ as follows:

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

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

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

$$\dfrac{}{\mathsf{x}_\mathsf{i} : \mathrm{A} \vdash \mathsf{x}_\mathsf{i} : \mathrm{A}}\ Ax \quad \equiv \quad \dfrac{}{\mathsf{x}_\mathsf{i} : \mathrm{A}}\ Ax$$

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 most of the ones to follow) will use this alternative notation.

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

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

(a) Currying

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

(b) Uncurrying

Figure I.5: Interderivability of product and arrow.

e.g. we have:

$$\mathrm{A} \times \mathrm{B} \vdash ((\mathrm{A} \times \mathrm{B}) \to \mathrm{C}) \to \mathrm{C} \dashv\vdash (\mathrm{A} \to \mathrm{B} \to \mathrm{C}) \to \mathrm{C}$$

Keep a mental note.

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 $Contraction$ and $Weakening$ 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 $Contraction$ yields Affine Logic, a logic in which resources can be used no more than once. Removing $Weakening$ yields Relevant Logic, a logic in which resources can be used no less than once. Removing both yields Linear Logic, a logic in which resources can be used *exactly* once. The intuitionistic formulations of the above give rise to corresponding type theories [Pierce, 2004]. For the purposes of this manuscript, we will focus our presentation on linear type theory.

2.1 Linear Types

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

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

$$\mathrm{A}, \mathrm{B}, \mathrm{C} ::= p \mid \mathrm{A} \multimap \mathrm{B} \mid \mathrm{A} \otimes \mathrm{B} \mid \mathrm{A} \& \mathrm{B} \qquad (\mathsf{ILL}_{\multimap,\otimes,\&})$$

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

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

$$\frac{}{\mathsf{x_i} : \mathrm{A} \vdash \mathsf{x_i} : \mathrm{A}}\ Ax$$

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

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

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

(a) Logical rules.

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

(b) Structural rule.

Figure I.6: Linear type theory.

free once in the abstraction body s – in other words, we have no way of syntactically instantiating constant functions.

Proof & Term Reductions

The notions of proof and term equivalence discussed in the previous section hold also for linear logic. Proof normalization looks almost identical to before in the case of $\to$ and $\times$ (substituting of course for $\multimap$ and $\otimes$). The key difference lies in the absence of horizontal dots and unnamed vertical dots (since the $Contraction$ rule is no more, meaning that there can only be a single occurrence of each axiom replaced with a proof). The extra connective & introduces its own two redexes (one per eliminator); the reduction of the first projection is shown in Figure I.7. Its reading is straightforward: if one were to use Γ to independently derive A and B along two parallel proofs, proceed by constructing a choice between the two, and then also make that choice (favoring either), there was never any need for the other in the first place. The equivalent term reduction steps this time around are:

$$\lambda \mathsf{x_i}.\mathsf{s\ t} \overset{\beta}{\rightsquigarrow} \mathsf{s}_{[\mathsf{x_i} \mapsto \mathsf{t}]}$$

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

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

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

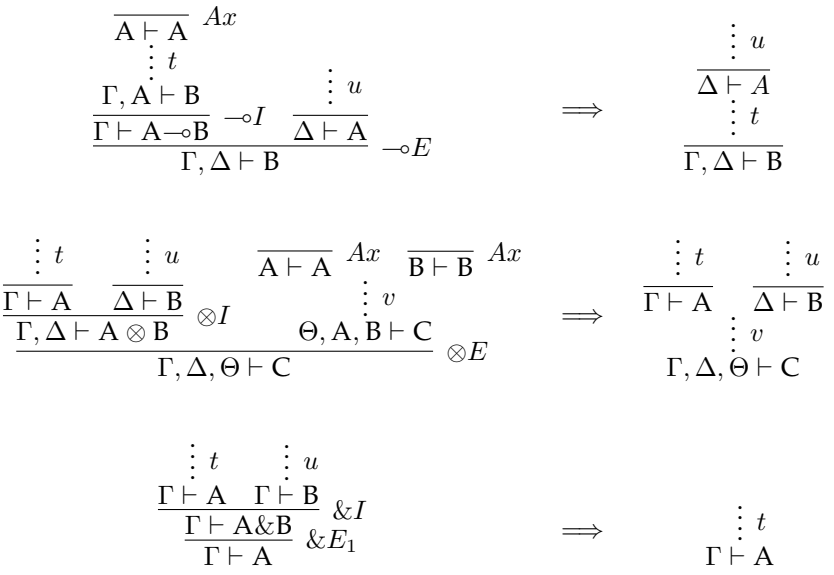


Figure I.7: Linear proof reductions.

2.2 Proof Nets

We have basked in the beauty of the natural deduction presentation adopted so far, and seen how it gives rise to a straightforward computational interpretation. We have also seen how it is at times overly bureaucratic in its explication of structural rules that are null from a computational perspective. To our luck, an additional representation makes itself available as soon as we step into linear grounds, namely *proof nets*, also due to Girard [Girard, 1987]. Proof nets are best suited for the multiplicative fragment of the logic (they are amenable to extensions with additive connectives, but things get uglier there). In our case, we have foregone the disjunctive connectives, and we have already suggested that the multiplicative conjunction $\otimes$ is (to an extent) interchangeable with the implication arrow $\multimap$ (we actually did this for intuitionistic connectives, but there is no need to repeat ourselves – the story looks identical with their linear variants). This gives us the much needed excuse to justify limiting our presenting of proof nets to the implication-only fragment of linear logic, $\mathsf{ILL}_{\multimap}$, 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

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{x_1 : B{\multimap}C}}\,Ax \quad \dfrac{\dfrac{}{\mathsf{x_0 : A{\multimap}B}}\,Ax \quad \dfrac{}{\mathsf{x_2 : A}}\,Ax}{\mathsf{x_0, x_2 \vdash x_0\ x_2 : B}}\,{\multimap}E}{\mathsf{x_1, x_0, x_2 \vdash x_2\ (x_1\ x_0) : A{\multimap}C}}\,{\multimap}E}{\mathsf{x_1, x_0 \vdash \lambda x_2.x_1\ (x_0\ x_2) : A{\multimap}C}}\,{\multimap}I}{\mathsf{x_0, x_1 \vdash \lambda x_2.x_1\ (x_0\ x_2) : A{\multimap}C}}\,Exchange$$

Figure I.8: Linear function composition.

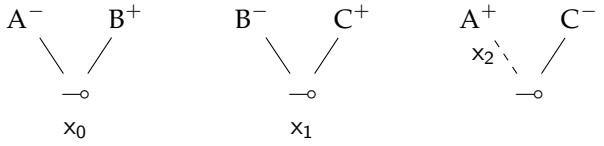
be derived simultaneously. They also have no notion of temporal precedence: everything happens in a single instant, meaning that positive subformulas are good to use without having to wait for their conditions to be met.

To see this in practice, a simple but concrete example will prove helpful. We will consider the natural deduction proof of linear function composition of Figure I.8 and translate it into its proof net equivalent of Figure I.9.

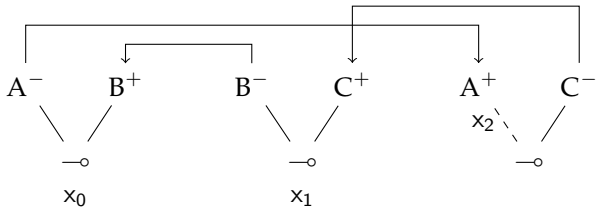
The first thing we need to do is write formulas as their *decomposition trees*; it sounds fancy, but in reality this is just recompiling formulas according to their underlying grammar, but now in tree form: propositional constants become leaves, and logical connectives become branch nodes. To make this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

A proof frame must satisfy an invariance property before the eligibility of the judgement it prescribes can even be considered. Namely, it must contain an equal number of positive and negative occurrences of each unique atomic formula that appears within it. This perfectly fits the linear logic paradigm: everything produced has to be consumed, and vice versa. In our case, we have three unique formulas, A, B and C, each one of which has a single positive and a single negative occurrence: check. The next thing to do in building a proof net is to establish a bijection between atomic formulas of negative polarity, and draw it as an extra set of edges, pointed negative atoms to their

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



(a) Proof frame for the natural deduction proof of Figure I.8.



(b) Proof structure for the natural deduction proof of Figure I.8.

Figure I.9: todo

positive matches: we call those *axiom links*. Axiom links essentially specify the elimination of function types: they identify function arguments with their concrete inputs in a geometric fashion. We do not need to put much thought for our running example: since all atoms have a single occurrence of each polarity, there is just one possible bijection to consider, presented in Figure I.9b. A proof frame with axiom links on top is called a *proof structure*, and this representation provides all of the information contained within a proof.

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

indeed be a proof net.

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

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

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

Let us revisit the natural deduction proof of Figure I.8. This time around, we will explicate polarity information, and put an identifying index to each atomic formula occurrence at its axiom leaves; the turnstile mirrors indices

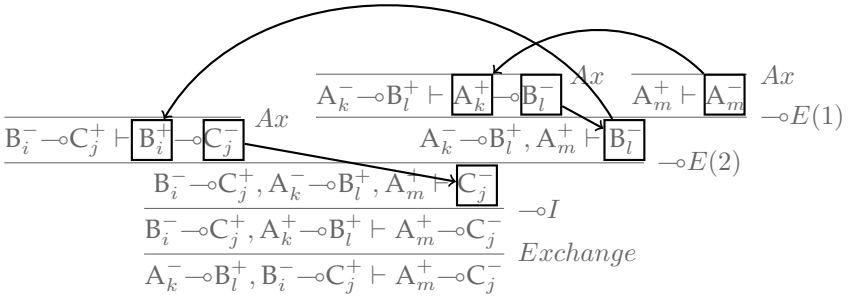


Figure I.10: Extracting axiom links from the proof of Figure I.8.

faithfully, but inverts atomic polarities. Going bottom up through Figure I.10, every encounter of an implication elimination allows us (i) to identify the set of indices coming from the functor's negative part with the set of indices provided by the counterpart positive argument and (ii) to propagate the negative indices of the functor's remainder to the succedent of the next conclusion. That is, the first elimination $\multimap E(1)$ creates the identification $\{k \leftrightarrow m\}$, and propagates the positive leftover B_l^- to the next proof step, whereas the next elimination $\multimap E(2)$ identifies $\{i \leftrightarrow l\}$ and propagates C_j^- downwards. Upon reaching the proof's conclusion, we get to merge all identifications established along the proof[1] , which can then be applied as a mapping (e.g. to the lexicographically first element) yielding a link-decorate judgement, in our case:

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

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

The question then is, when should we use which? The original intention of proof nets was to provide a compact, bureacracy-free representation of proofs that abstracts away from structural rules. In that sense, their strength is also their weakness; same as the λ-terms they prescribe, they encode the semantically essential part of a proof, but hide structural subtleties that can prove

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

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

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

hard to guess or recover. At the same time, performing search over proof nets is a horrible idea; the number of possible links we need to consider scales factorially with respect to the number of atoms in the proof frame, and checking whether a set of links is valid is in the best case linear. Due to these limitations, proof nets were envisaged as a compiled form of an existing proof, rather than a canvas to find that proof on. We will not see proof nets again for a while, but we will keep their memory warm in our hearts. Because when we do, we will challenge this perception and see how their parallel nature can actually be very convenient for proof search. Until then, we can temporarily store them in our mental backlog.

3 Dropping Commutativity

There is only one structural rule left[1]; it is time for $Exchange$ to go. Dropping structural commutativity gives us an *ordered* type system, i.e. one where variables must be used exactly in the order they were provided. If we were now to naively go about our business using ILL without commutativity, we would soon stumble upon a pitfall. Recalling the shape of the $\multimap E$ rule, we come to the realization that the functions carried over to this new type system are suddenly picky; they can only be applied to arguments to their right. This should raise some red flags: we want our type system to be directional, but not uni-directional: where shall we look for the left-biased variant of the implication? The answer is simple: the conflation between the two directions was natural, up until a moment ago; having them both explicit would not amount to much, since by $Exchange$ they would be interderivable. With $Exchange$ removed, the veil is lifted and we can now see this clearly: there were always two implications, except disguised by the same symbol!

The logic that does our newfound friend justice has come to be known as the Lambek Calculus [Lambek, 1958]. A careful reader will notice the chronological inconsistency of our presentational tour: the Lambek Calculus predates Linear Logic, despite being a refinement of its purely linear part! Formulas will now abide by the following grammar:

$$\mathrm{A}, \mathrm{B}, \mathrm{C} ::= p \mid \mathrm{A} \multimap \mathrm{B} \mid \mathrm{A} \circ\!\!- \mathrm{B} \tag{LC}$$

Alternative presentations include multiplicative and/or additive conjunction and/or disjunction, but the interesting feature lies in the two implications. The rules of this fragment are presented in Figure I.11. The directed version of the implication arrow has slightly denotational semantics: it points from argument to conclusion, meaning that what used to be $\multimap$ now becomes $\circ\!\!-$ (it looks for an argument directly to its right to produce a result). The new version of $\multimap$ is in reverse: it looks for an argument directly to its left. The use of the word *directly* is not meaningless prose. Another interesting aspect of the Lambek Calculus is that it brings forth a notion of adjacency. todo

[1] Or is there?

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

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

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

Figure I.11: todo

4 Going Modal

5 The Linguistic Perspective

Bibliography

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

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

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

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.

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.

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

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

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

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

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.

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

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.

CHAPTER II

Chapter 2: Title Pending