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. Bert van der Zwaan,
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

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 reviewing 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, four 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 irrelevant bureaucracies to avoid, but objects of interest in themselves, hidden in

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

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. Of you, I ask to remain conscious of what you are reading and aware of my own biases and limitations; feel free to skip ahead when something reads trivial, and do not judge too harshly when you encounter an explanation you find insufficient.

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... 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 progamss, bridging in essence the seemingly disparate fields of mathematic 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 mathematical world, and in parallel to the above developments, applied logicians and formally inclined linguists have demonstrated a stunning perserverance in their self-imposed quest of modeling natural language syntax and semantics, but 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 almost three decades later by Jean-Yves Girard. 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), finally 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 the Curry-Howard correspon-

[1] In proof theory, at least.

[2] In some corners of the world, this part of the prophecy has not yet transpired.

dence, thus being 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, feel free of course to skip ahead.

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 Bertrand Russel and Alonzo Church [Rus08, Chu40]. 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 (or rather the multiplicative fragment of it). 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 [Hey30], 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 claim to be proving.

Focusing only on the *multiplicative* fragment of the logic[3], we have a tiny recursive language that allows us to define the various shapes of logical *propositions* (or *formulas*). Given some finite set of propositional constants Prop_0, and A, B, C arbitrary well-formed propositions, the language of propositions is inductively defined as

$$\mathrm{A}, \mathrm{B}, \mathrm{C} ::= p \mid \mathrm{A} \to \mathrm{B} \mid \mathrm{A} \times \mathrm{B}$$

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

[3]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 multiplicative fragment.

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.

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.

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 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 $\mathrm{A} \rightarrow \mathrm{B}$ 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 $\mathrm{A} \rightarrow \mathrm{B}$. 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 $\mathrm{A} \times \mathrm{B}$ from Γ, Δ.

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

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

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

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

(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

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.

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

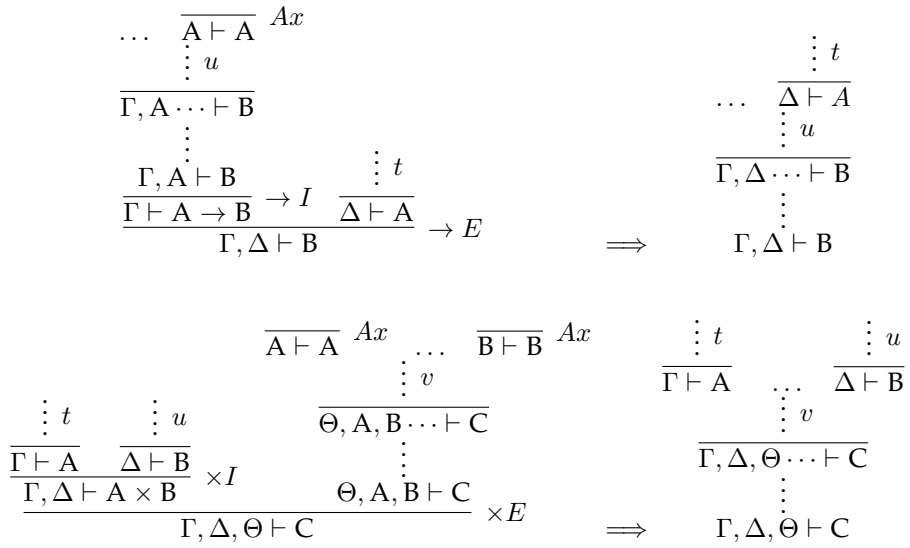

Figure I.2: Proof Reduction Patterns

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 [Gro99]. Figure I.2 presents rewrite instructions for the two detour patterns we may encounter (one per logical connective). Read bottom-up[4], the first one suggests that if one were to hypothesize a proposition A, use it within an (arbitrarily deep) proof u together with extra context Γ to derive a proposition B, before finally redacting the hypothesis and composing with a proof t that derives A from context Δ, it would have been smarter (and more concise!) to just plug in t directly when previously hypothesizing A, since then no redaction or composition would have been necessary. In a similar vein, the second suggests that if one were to derive and merge proofs 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.

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

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. The entry point for such an approach is to interpret propositions as *types* of a minimal functional programming language (a perhaps more aptly named alternative to the Curry-Howard correspondence is the propositions-as-types interpretation). In that sense, the set of propositional constants Prop_0 becomes the programming language's basic set of *primitive* types (think of them as built-ins). Implicational formulas $\mathrm{A} \to \mathrm{B}$ are read as *function* types, and conjunction formulas are read as *tuples*. 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 derivation of a term s of type B out of such an environment.

Inspecting Figure I.3, things for the most part look good. The implication elimination rule $\to E$ provides us with a composite term $\mathsf{s}\ \mathsf{t}$ that denotes the function application of s on t. Its dual, $\to I$, allows us to create (so-called anonymous) functions by deriving a result s dependent on some hypothesized argument x_i which is then *abstracted* over as $\lambda \mathsf{x}_\mathsf{i}.\mathsf{s}$. Any occurrence of x_i within s is then *bound* by the abstraction; variables that do not have a binding abstraction are called *free*. The conjunction introduction $\times I$ allows us to create tuple objects (s, t) through their parts s and t. Its dual $\to E$ gives us the option to identify the two coordinates of a tuple s with variables x_i and x_j, when the latter are hypothesized assumptions for deriving some program t. If our assumptions are not in order, blocking the applicability of some rule, we can put them back where they belong with $Exchange$. With $Contraction$ we

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

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

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

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

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

Figure I.3: Simple Type Theory

can pretend to be using two different instances x_i and x_j of the same type before identifying the two as a single object x_i in term s with term t"); note here the meta-notation for *variable substitution*, $s_{[x_i \mapsto t]}$, which reads as "replace any occurrence of variable x_i. And finally, we can introduce throwaway variables into our typing environment with $Weakening$ (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 $x_i : A$ 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 $Contraction$ rule contains meta-notation that is not part of the term syntax itself. That is to say, $s_{[x_i \mapsto t]}$ is *not* a valid term – even if the result

of the operation it denotes is. Generally speaking, substitution of objects for others of the same type is (modulo variable shadowing) an admissible property of the type system. Mixing syntax and meta-syntax in the same system is a dirty trick we will sporadically employ; this surely invites some trouble, but conscious use of it can be worth it, since it significantly simplifies presentation.

Term Equivalences

There exist three kinds of equivalence relations between terms, each given an identifying greek letter.[5]

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

$$\begin{aligned} \lambda \mathsf{x.s\ t} &\implies \mathsf{s}_{[\mathsf{x} \mapsto \mathsf{t}]} \\ \mathsf{case\ (s,\ t)\ of\ (x_i,\ x_j)\ in\ u} &\implies \mathsf{u}_{[\mathsf{s} \mapsto \mathsf{x_i}, \mathsf{t} \mapsto \mathsf{x_j}]} \end{aligned}$$

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

[5] 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.

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.

$$\lambda \mathsf{x}.\mathsf{s}\ \mathsf{x} \iff \mathsf{s}$$
$$(\text{case } \mathsf{s} \text{ of } (\mathsf{x_i},\mathsf{x_j}) \text{ in } \mathsf{x_i}, \text{ case } \mathsf{s} \text{ of } (\mathsf{x_k},\mathsf{x_l}) \text{ in } \mathsf{x_l}) \iff \mathsf{s}$$

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 the age 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 [Pie04]. For the purposes of this manuscript, we will focus our presentation on linear type theory.

2.1 Linear Logic

Linear logic is due to Jean-Yves Girard [Gir87]. The full logic includes additive connectives as well as a modality that allows one to incorporate non-linear propositions into the presentation, but we will happily forget about those.

For the multiplicative only fragment there is not much we have to do. We will note first that the nature of the implication arrow changes from material implication to a 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 a process that transforms a single A into a single B, consuming the former in the process. The new, weird-looking arrow of the linear implication is read as

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

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

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

(a) Logical Rules

Figure I.4: Linear Logic

lolli(pop) due to its suggestive appearance.[6] Conjunction $\times$ is now separated into two distinct operators, $\otimes$ and $\&$. The first denotes a linear tuple, and $\mathrm{A} \otimes \mathrm{B}$ is read as *both* A *and* B. The second denotes a choice, and $\mathrm{A} \& \mathrm{B}$ is read as A *with* B, or *choose one of* A *or* B. The subset of the logic concerning these connectives is presented in Figure I.4 (contexts, judgements, rules and proofs look just like before).

3 Lambek Calculi

4 Going Modal

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

Bibliography

[Chu40] Alonzo Church. A formulation of the simple theory of types. *The journal of symbolic logic*, 5(2):56–68, 1940.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical computer science*, 50(1):1–101, 1987.

[Gro99] Philippe 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.

[Hey30] Arend Heyting. Die formalen regeln der intuitionistischen logik. *Sitzungsbericht PreuBische Akademie der Wissenschaften Berlin, physikalisch-mathematische Klasse II*, pages 42–56, 1930.

[Pie04] Benjamin C Pierce. *Advanced topics in types and programming languages*. MIT press, 2004.

[Rus08] Bertrand Russell. Mathematical logic as based on the theory of types. *American journal of mathematics*, 30(3):222–262, 1908.