Multi Types

MPRI course
Logique Linéaire et Paradigmes Logiques du Calcul
Year 2023-24, Part 3, Lecture 4

Beniamino Accattoli

In this lecture we see how to connect the number of steps of weak head reduction, which is a reasonable time cost model, with a notion of type related to linear logic and inducing a denotational model. At the end of the lecture we also see the connection with the linear substitution calculus.

This note at times uses colors to help parsing symbols, but colors do not have any special meaning.

1 Preliminaries

Type systems are a form of compositional abstraction in which the behavior of programs, particularly higher-order programs such as λ-terms, is described by types, that is, specifications of the kinds of objects that programs expect in input and are supposed to produce as output. As famously stated by Milner, typed programs *cannot go wrong*, as types guarantee the absence of run-time errors. Some type systems ensure also other properties such as termination or various forms of security.

Simple Types. A cornerstone of the field is the fact that type systems for λ-calculi can be seen as logical systems. This is the Curry-Howard correspondence, whose prototypical instance is the fact that the typing rules for simple types for the λ-calculus below are exactly minimal (that is, implication only) intuitionistic logic, once terms (in blue) are erased and only types (in red) are considered.

$$\frac{}{\Gamma, x : A \vdash x : A}\,(\mathrm{var}) \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}\,(\lambda)$$

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash Mu : B}\,(@)$$

We assume the reader familiar with the basics of type systems, but let's fix nonetheless a few notions. Here the grammar of types is

$$\textsc{Simple types} \qquad \sigma, \tau \quad ::= \quad \alpha \mid \sigma \multimap \tau$$

Where α is an unspecified ground type, you may think of booleans or natural numbers—it is not important which type it is, we simply need a base type for the inductive definition of types to be well founded.

Also, Γ is a type context (also noted Γ', Γ''), that is, a partial function from variables to simple types of finite domain $\mathtt{dom}(\Gamma)$, i.e. defined only on a finite number of variables. We write $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$ if $\mathtt{dom}(\Gamma) = \{x_1, \ldots, x_n\}$ and $\Gamma(x_i) = \sigma_i$ for $i \in \{1, \ldots, n\}$, and when we write $\Gamma, x : A$ (as in the hypothesis of the rule for abstractions) we implicitly assume that $x \notin \mathtt{dom}(\Gamma)$.

The guarantee provided by simple types is a strong form of termination, namely strong normalization (a typable term has no diverging evaluation sequences).

Theorem 1.1. *If M is typable with simple types then M is strongly normalizing.*

This theorem can be proved in many different ways. Some of the proofs are remarkably concise, but they tend to not scale up to richer type systems, as they exploit strong properties of the system, due to its simplicity. We are not going to prove the theorem, as it should be background knowledge for the students of this course, and its proofs can easily be found in the literature.

Guarantees, Characterizations, and Decidability. Usually type systems do not characterize the property that they guarantee, that is, there are terms satisfying the property that are not typable in the system. In our case, there are strong normalizing terms that are not typable with simple types. For instance, the duplicator δ is not (δ is typable with polymorphic types, but there are more complex terms that are strongly normalizing and not typable with polymorphic types).

Type systems are often studied in order to be useful in practice for programming. Given the Turing completeness of the λ-calculus, most interesting properties of programs, typically termination, are undecidable by Rice's theorem. Therefore, every type system that would characterize them (i.e. such that a term has that property if and only if it is typable) would not allow automatic type inference, for instance.

2 Intersection Types

Characterizations of terminating λ-terms can be given by switching from simple to *intersection types*, introduced by Coppo and Dezani-Ciancaglini in 1978.

The basic idea is to allow a term to have more than one type at the same time. Concretely, one introduces a new type constructor $\sigma \cap \tau$, called *intersection*, and the following introduction and elimination rules:

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash M : B}{\Gamma \vdash M : A \cap B}\,(\cap i) \qquad \frac{\Gamma \vdash M : A \cap B}{\Gamma \vdash M : A}\,(\cap e_1) \qquad \frac{\Gamma \vdash M : A \cap B}{\Gamma \vdash M : B}\,(\cap e_2)$$

Intersection types are a flexible tool for characterizing termination. By carefully tuning the obtained type system (by adding further rules or side conditions), one obtains characterizations of terminating term with respect to various notions of termination, typically strongly or weakly normalizing terms, and head or weak head normalizing terms.

Undecidable. Since they characterize termination, whether a term is typable with intersection types is undecidable. Therefore, they are not a type system that is useful in practice. Some of their restrictions—that no longer characterize termination—do however find some applications in the theory of programming languages, see the survey by Bono and Dezani-Ciancaglini [BD20].

Intersections *vs* Conjuctions. As a logic, they do not really fit the Curry-Howard correspondence. At the type level, the rules for intersections look like those for conjuctions. When one looks at their decoration with terms, however, it becomes clear that intersections are *not* conjunctions. Conjuctions are decorated as follows:

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A \wedge B}\ (\wedge i)$$

where (M, N) is a constructors for pairs of terms. Note that the introduction of conjuctions involves two arbitrary terms M and N, while the rule for intersections requires twice the *same* term M, imposing a structural constraint that is absent for conjuctions.

Intersection types are not useful in practice and do not provide a Curry-Howard logic. What are they interesting for then? They provide a syntactic approach to define and study denotational semantics, as we shall see.

3 Set Types

In their original formulation, intersections satisfy the following three natural properties:

Associativity: $(\sigma \cap \tau) \cap \tau' = \sigma \cap (\tau \cap \tau')$;

Commutativity: $\sigma \cap \tau = \tau \cap \sigma$;

Idempotency: $\sigma \cap \sigma = \sigma$.

Essentially, intersections can be seen as *sets*, and the introduction of intersection showed before can be seen in set notation as follows:

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash M : B}{\Gamma \vdash M : \{A, B\}}\ (\cap i)$$

In the following, we adopt a modern approach and rename *intersection types* into *set types*, also removing other aspects of the theory of intersection types that are not needed for our study, such as subtyping and intersections on the right of arrows.

The Set Types System. Types at work in the type system are organized in two mutually defined layers:

$$\begin{array}{rrcl} \textsc{Basic Types} & \sigma, \tau & ::= & \alpha \mid S \multimap \sigma \\ \textsc{Set Types} & S & ::= & \{\sigma_i\}_{i \in I} \quad I \text{ a finite set} \end{array}$$

The definition suggests a connection with linear logic, where $A \to B$ is represented as $!A \multimap B$, while here we have $\{\sigma_1, \ldots, \sigma_n\} \multimap \tau$.

The empty set type $\emptyset$ is a valid type, that shall play a crucial role. A type context Γ now assigns a *set* type to every variable, but a non-empty set type only to a finite number of variables, forming its domain. That is, $\Gamma = x_1 : S_1, \ldots, x_n : S_n$ with $\mathtt{dom}(\Gamma) \subseteq \{x_1, \ldots, x_n\}$, where $\subseteq$ expresses the fact that when we specify the type of each variable we may also write some of the infinitely many one that are assigned to $\emptyset$. Given two type contexts Γ and Γ' we use $\Gamma \cup \Gamma'$ for the type contexts such that $(\Gamma \cup \Gamma')(x) := \Gamma(x) \cup \Gamma'(x)$.

The system has two kinds of judgements, $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M : S$, depending on whether the right-hand type is a basic or a set type.

The typing rules are:

$$\frac{}{\Gamma, x : \{\sigma\} \vdash x : \sigma}\ \mathtt{ax} \qquad \frac{(\Gamma'_i \vdash M : \sigma_i)_{i \in I}}{\cup_{i \in I} \Gamma'_i \vdash M : \{\sigma_i\}_{i \in I}}\ \mathtt{many}$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \setminus\!\!\setminus x \vdash \lambda x.M : \Gamma(x) \to \sigma}\ \mathtt{fun} \qquad \frac{\Gamma \vdash M : S \to \tau \quad \Gamma' \vdash N : S}{\Gamma \cup \Gamma' \vdash MN : \tau}\ \mathtt{app}$$

Some comments:

- *Weakening*: note the presence of Γ in the axiom rule, which expresses that the system includes weakening on the left.
- *Many*: the `many` rule is the only one introducing sets on the right of $\vdash$. The set of indices I can be empty, that is, the rule may have 0 premises. In such a case, it assigns the empty set $\emptyset$ type, and note that it can assign it to *whatever* term, because of the absence of premises. Namely, with 0 premises the rule takes the following form:

$$\frac{}{\vdash M : \emptyset}\ \mathtt{many}$$

- *Abstractions*: rule `fun` is a compact notation for the following two rules:

$$\frac{\Gamma, x : S \vdash M : \sigma}{\Gamma \vdash \lambda x.M : S \to \sigma}\ \mathtt{fun} \qquad \frac{\Gamma \vdash M : \sigma \quad x \notin \mathtt{dom}(\Gamma)}{\Gamma \vdash \lambda x.M : \emptyset \to \sigma}\ \mathtt{fun}$$

  where the second one handles the case of a variable not explicitly assigned in the typing context of the premise, and it is based on the fact that $\Gamma(x) = \emptyset$ if $x \notin \mathtt{dom}(\Gamma)$.

- *Applications and arguments*: note that the only place where set types appear on the right-hand side of $\vdash$ in the premises of a rule is for the argument of applications. This fact mimics the use of the promotion rule of LL for arguments in the translation of λ-calculus to LL.

  Note also that the application rule is formulated multiplicatively. It can also be formulated additively, this is not essential.

The notation $\Pi \triangleright \Gamma \vdash M : \sigma$ is used to stress that Π is a type derivation of the judgement $\Gamma \vdash M : \sigma$.

Examples. In contrast to simple or polymorphic types, set types can type the duplicating term $\delta = \lambda y.yy$ as follows:

$$\dfrac{\dfrac{\dfrac{}{y:\{\{\alpha\}\multimap\alpha\}\vdash y:\{\alpha\}\to\alpha}\;\texttt{ax} \qquad \dfrac{\dfrac{}{y:\{\alpha\}\vdash y:\alpha}\;\texttt{ax}}{y:\{\alpha\}\vdash y:\{\alpha\}}\;\texttt{many}}{y:\{\{\alpha\}\multimap\alpha,\alpha\}\vdash yy:\alpha}\;\texttt{app}}{\vdash \lambda y.yy:\{\{\alpha\}\multimap\alpha,\alpha\}\multimap\alpha}\;\texttt{fun}$$

The empty set type can be used to type erasing β-steps, as follows:

$$\dfrac{\dfrac{z:\alpha\vdash z:\alpha}{z:\alpha\vdash\lambda y.z:\emptyset\multimap\alpha}\;\texttt{fun} \qquad \dfrac{}{\vdash M:\emptyset}\;\texttt{many}}{z:\alpha\vdash(\lambda y.z)M:\alpha}\;\texttt{app}$$

Note indeed that $(\lambda y.z)M \to_\beta z$ erases M. The use of $\emptyset$ is actually quite subtle, as the next exemple shows:

$$\dfrac{\dfrac{}{y:\{\emptyset\multimap\alpha\}\vdash y:\emptyset\multimap\alpha}\;\texttt{ax} \qquad \dfrac{}{\vdash\Omega:\emptyset}\;\texttt{many}}{y:\{\emptyset\multimap\alpha\}\vdash y\Omega:\alpha}\;\texttt{app}$$

Here y is assigned an erasing type without y being actually able to erase its argument, and then it is applied to the diverging term Ω, obtaining that the diverging term $y\Omega$ is typable. What is the notion of termination that is characterized by set types then?

Characterization Theorem. The system of set types that we presented characterizes termination with respect to head reduction $\to_h$.

Theorem 3.1 (Characterization)**.** *A λ-term M is typable with set types if and only if M is $\to_h$-normalizing.*

In the first lecture, we saw that head normalizable terms can be taken as a notion of *defined* term when representing Godel's partial recursive functions (while normalizable terms cannot), and we saw that head normalization can also be justified via rewriting theory by means of a factorization theorem. Set types provide a type theoretic characterization, that, as we shall show in a moment, leads to a denotational characterization.

The proof of the characterization theorem is non-trivial. Direction $\Rightarrow$ is usually called *correctness*, and direction $\Leftarrow$ *completeness*. Completeness is easy, while correctness is involved. Let's start with completeness.

Completeness. The proof is in two steps, both peculiar to set types and not usually verified by other type systems. The first one is the fact that every head normal form is typable.

Proposition 3.2. *Every head normal form is typable with set types.*

Remember that head normal forms have shape $\lambda x_1.\ldots\lambda x_n.(yM_1\ldots M_k)$, where y may be one of the x_i, and $n,k\geq 0$.

Proof. $\boxed{\text{Exercise}}$: prove that head nfs are typable. Hint: consider first terms of the form $yM_1 \dots M_k$. □

The second step is a subject *expansion* property, stating that typability can be pulled back along a head β-step.

Proposition 3.3 (Subject expansion). *If $M \to_h N$ and $\Pi \triangleright \Gamma \vdash N : \sigma$ then there exists a type derivation $\Pi' \triangleright \Gamma \vdash M : \sigma$.*

For the proof, we need a removal lemma, whose proof is omitted (it is by induction on Π, and it is not difficult, simply a bit tedious and not really interesting).

Lemma 3.4 (Removal). *If $\Pi \triangleright \Gamma \vdash r\{x := p\} : \sigma$ then there exists a set type S and two derivations Π_r and Π_p such that*

1. $\Pi_r \triangleright \Gamma', x : S \vdash r : \sigma$,
2. $\Pi_p \triangleright \Gamma'' \vdash p : S$.

with $\Gamma = \Gamma' \cup \Gamma''$.

We can now prove subject expansion.

Proof. Proof by induction on $M \to_h N$.

- *Base case, step at top level*: $M = (\lambda x.r)p \to_h r\{x := p\} = N$. The derivation in the hypothesis is $\Pi \triangleright \Gamma \vdash r\{x := p\} : A$. Then by the removal lemma (Lemma 3.4) we obtain a set type S and two derivations $\Pi_r \triangleright \Gamma', x : S \vdash r : A$ and $\Pi_p \triangleright \Gamma'' \vdash p : S$ such that $\Gamma \cup \Gamma'$. The derivation Π' of the statement is then defined as follows:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Pi_r \triangleright \Gamma', x : S \vdash r : A\end{matrix}}{\Gamma' \vdash \lambda x.r : S \multimap A}\text{T-}\lambda \qquad \begin{matrix}\vdots\\ \Pi_p \triangleright \Gamma'' \vdash p : S\end{matrix}}{\Gamma' \cup \Gamma'' \vdash (\lambda x.r)p : A}\text{T-@}$$

- *Inductive case, step on the left of the root application*: $M = rp \to_h r'p = N$ with $r \to_h r'$. Then Π has the following shape.

$$\dfrac{\begin{matrix}\vdots\\ \Pi_{r'} \triangleright \Gamma' \vdash r' : S \multimap A\end{matrix} \quad \begin{matrix}\vdots\\ \Pi_p \triangleright \Gamma'' \vdash p : S\end{matrix}}{\Gamma' \cup \Gamma'' \vdash r'p : A}\text{T-@}$$

  with $\Gamma = \Gamma' \cup \Gamma''$. Applying the *i.h.* to the left sub-derivation $\Pi_{r'}$, we obtain $\Pi'_r \triangleright \Gamma' \vdash r : S \multimap \sigma$. Then Π' is defined as follows.

$$\Pi' := \dfrac{\begin{matrix}\vdots\\ \Pi'_r \triangleright \Gamma' \vdash r : S \multimap \sigma\end{matrix} \quad \begin{matrix}\vdots\\ \Pi_p \triangleright \Gamma'' \vdash p : S\end{matrix}}{\Gamma' \cup \Gamma'' \vdash rp : A}\text{T-@}$$

- *Inductive case, step under abstraction*: $M = \lambda x.r \to_h \lambda x.r' = N$ with $r \to_h r'$. Similar to the previous one. $\square$

The two steps together give us completeness.

Theorem 3.5 (Completeness). *If M is head normalizable then it is typable.*

Proof. Let $M \to_h^k N$ the reduction to head normal form. By induction on k. Case:

- $k = 0$) Then $M = N$ and M is typable by Proposition 3.2, that is, $\Gamma \vdash N : \sigma$.
- $k > 0$) Then $M \to_h r \to_h^{k-1} N$. By *i.h.*, r is typable, that is, $\Gamma \vdash r : \sigma$. By subject expansion, $\Gamma \vdash M : \sigma$. $\square$

Correctness. Correctness of set types with respect to head normalization is difficult to prove. The reason is that one needs to extract a termination argument from the typability hypothesis, but it is unclear what decreases.

The typical technique for showing termination from typability is the *reducibility* method, also called the *logical relation* technique. We are not going to see it here, because it is heavy and because in the next section we shall develop an alternative lighter approach.

Subject Reduction. As most type systems, set types verify *subject reduction*, which is the property dual to subject expansion: typibality is preserved along β-steps. The proof is also dual. It relies on a substitution lemma, stating that typability is stable by substitution, which is dual to the removal lemma seen above.

Lemma 3.6 (Substitution). *If $\Pi_r \triangleright \Gamma, x : S \vdash r : \sigma$ and $\Pi_p \triangleright \Gamma' \vdash p : S$ then there exists $\Pi_{r\{x:=p\}} \triangleright \Gamma \cup \Gamma' \vdash r\{x := p\} : \sigma$*

The proof of the lemma is omitted (for the same reasons as for the removal lemma, it is by induction on Π_r).

Proposition 3.7 (Subject reduction). *If $M \to_h N$ and $\Pi \triangleright \Gamma \vdash M : \sigma$ then there exists $\Pi' \triangleright \Gamma \vdash N : \sigma$.*

Proof. $\boxed{\text{Exercise}}$. $\square$

The Induced Denotational Model. Subject reduction and expansion taken together give the fact that set type judgements are stable by head reduction, that is, they are invariant. A denotational model of the λ-calculus is given by an invariant of β-reduction satisfying some additional properties—the technical notion is the one of *λ-model*. For the sake of simplicity, we slightly abuse terminologies and call *model* every interpretation that is invariant by β-reduction, forgetting about the additional properties[1].

The set semantics of a term is defined as:

[1]One way of defining λ-models is to consider the semantic interpretation $[\![\cdot]\!]$ from λ-terms to some mathematical objects and the induced equality $=_{[\![\cdot]\!]}$ on terms (defined as $M =_{[\![\cdot]\!]} N$ if $[\![M]\!] = [\![N]\!]$) and say that the co-domain of $[\![\cdot]\!]$ is a λ-model if $=_{[\![\cdot]\!]}$ is a λ-theory (discussed in the first lecture), that is, if it contains β-conversion (thus the interpretation is in particular stable by β-reduction) and it is stable by context closure and α-equivalence.

$$[\![M]\!]^{\texttt{set}} := \{((S_1, .., S_k), \sigma) \mid x_1 : S_1, .., x_k : S_k \vdash M : \sigma\}$$

By subject reduction and expansion $[\![M]\!]^{\texttt{set}}$ is invariant by head evaluation, but it can be easily shown that it is invariant for every β-step. By correctness (that we did not prove but it holds) and completeness, it is an *adequate* semantics with respect to head reduction $\to_h$, where *adequate* means that it is non-empty if and only if $\to_h$ terminates.

The Weak Head Case. We mentioned that set types are a flexible tool, that via minor variations characterize different notions of termination. A variant that we shall use in Sect. 7 is the one characterizing termination with respect to *weak* head reduction. To that aim, it is enough to add the following rule to the type system:

$$\frac{}{\vdash \lambda x.M : \alpha}\ \texttt{fun-ax}$$

Stating that *every* abstraction is typable, corresponding to the fact that every abstraction is a normal form in the weak head case. Note that now $\lambda x.\Omega$ is typable, while it is *not* typable in the system without rule `fun-ax`, because it $\to_h$-diverges.

4 Multi Types

Traditionally, intersection is idempotent, that is, one has $A \cap A = A$, which in terms of sets becomes $\{A, A\} = \{A\}$. Dropping idempotentcy does not change the set of typable terms, it only changes the assigned types and the structure of type derivations. The key point is that if $A \cap A \neq A$ then intersections can be seen as *multi-sets* rather than sets. Therefore, $A \cap A$ is rather written as $[A, A]$, and we use *multi (set) types* for concisely refer to non-idempotent intersection types.

Switching from set to multi sets strengthens the linear logic flavor as it is analogous to switching from $A \wedge A$, which is logically equivalent to A, to $A \otimes A$, which is instead *not* logically equivalent to A. To strengthen even more the linear flavor, we also remove weakening from axioms. The multi type system then becomes:

$$\begin{array}{rrcl} \text{Linear types} & \sigma, \tau & ::= & \alpha \mid \mu \multimap \sigma \\ \text{Multi-sets} & \mu & ::= & [\sigma_i]_{i \in I} \quad I \text{ a finite set} \end{array}$$

The empty multi-set is noted $[\,]$, and we use $\uplus$ for multi-set *sum*, for instance $[A, A, B] \uplus [A, C] = [A, B, A, A, C]$ and $M \uplus [\ ] = M$. The type contexts $\Gamma \uplus \Gamma'$ is defined as $(\Gamma \uplus \Gamma')(x) := \Gamma(x) \uplus \Gamma'(x)$.

The typing rules are:

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ \texttt{ax} \qquad \frac{(\Gamma'_i \vdash M : \sigma_i)_{i \in I}}{\uplus_{i \in I} \Gamma'_i \vdash M : [\sigma_i]_{i \in I}}\ \texttt{many}$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \setminus\!\!\setminus x \vdash \lambda x.M : \Gamma(x) \multimap \sigma}\ \texttt{fun} \qquad \frac{\Gamma \vdash M : \mu \multimap \tau \quad \Gamma' \vdash N : \mu}{\Gamma \uplus \Gamma' \vdash MN : \tau}\ \texttt{app}$$

Multi Types Semantics. The semantical interpretation of set types lifts to a *multi-set* semantics, as follows:

$$[\![M]\!]^{\texttt{mset}} := \{((\mu_1, .., \mu_k), \sigma) \mid x_1 : \mu_1, .., x_k : \mu_k \vdash M : \sigma\}$$

We shall see that multi types and set types do type the *same* terms. Therefore, from the adequacy of the set semantics with respect to head reduction $\rightarrow_h$ is follows the adequacy of the multi set semantics $[\![\cdot]\!]^{\texttt{mset}}$.

Multi Types and Linear Logic. The linear logic character of multi types is more than just a flavor. The multi set semantics is actually deeply related to the *relational* semantics of LL, the simplest and somewhat canonical model of LL (living in the category of sets and relations, where $\otimes$ is interpreted by the cartesian product of sets, and ! by the multi-set comonad). The multi set semantics is in fact exactly the relational semantics of LL *restricted* to the image of the (CbN) encoding of λ-terms.

Literature on Multi Types. In the literature, multi types are more often called *non-idempotent intersection types*. They make their first appearance as a technical tool to study intersection types in 1980 in a work by Coppo, Dezani-Ciancaglini, and Venneri [CDCV80]. They were however first considered by themselves only in 1994 by Gardner [Gar94], and then by various works [Kfo00, NM04, dC07, dC18]—a survey is [BKV17].

Multi types were also implicitly used in the large literature on relational semantics. In particular, in Ehrhard's extensive work on models of LL, that focussed on relational semantics and on its variations and enrichments, see for instance [Ehr93, Ehr05, BEM07, Ehr12].

Why Multi Types Are Interesting. The difference between set and multi set types is that the latter induce bigger types and bigger type derivations. Next section gives examples of this fact using Church numerals. The difference in size has two consequences that we analyze in detail after the examples. First, it allows for a simple proof of the correctness part of the characterization theorem for multi types. Second, it allows to extract bounds from type derivations and to unveil the quantitative character of relational semantics.

5 Examples of Multi Types Derivations

The typical example of bigger types and bigger derivations for multi types is given by Church numerals. Let's recall Church's encoding of natural numbers in the λ-calculus:

$$\begin{array}{lcl}
\underline{0} & := & \lambda z.\lambda y.y \\
\underline{1} & := & \lambda z.\lambda y.zy \\
\underline{2} & := & \lambda z.\lambda y.z(zy) \\
\dots & & \\
\underline{n} & := & \lambda z.\lambda y.z^n y
\end{array}$$

where $z^n y$ denotes $z(z(\ldots(zy)\ldots))$ where z occurs n times.

With simple types it is possible to type all numerals with the same type:

$$\underbrace{(\alpha \multimap \alpha)}_{\text{for } z} \multimap \underbrace{\alpha}_{y} \multimap \alpha$$

For instance, the derivation for $\underline{2}$ is (we write α^α for $\alpha \multimap \alpha$):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{z : \alpha^\alpha \vdash z : \alpha^\alpha}\,\texttt{ax} \qquad \dfrac{\dfrac{}{z : \alpha^\alpha \vdash z : \alpha^\alpha}\,\texttt{ax} \quad \dfrac{}{y : \alpha \vdash y : \alpha}\,\texttt{ax}}{z : \alpha^\alpha, y : \alpha \vdash zy : \alpha}\,\texttt{app}}{z : \alpha^\alpha, y : \alpha \vdash z(zy) : \alpha}\,\texttt{app}}{z : \alpha^\alpha \vdash \lambda y.z(zy) : \alpha \multimap \alpha}\,\texttt{fun}}{\vdash \lambda z.\lambda y.z(zy) : \alpha^\alpha \multimap \alpha \multimap \alpha}\,\texttt{fun}$$

Similarly, with set types there is a single set type for numerals:

$$\underbrace{\{\{\alpha\} \multimap \alpha\}}_{\text{for } z} \multimap \underbrace{\{\alpha\}}_{y} \multimap \alpha$$

Whose derivation for $\underline{2}$, for instance, is a simple decoration with sets of the derivation with simple types (writing $\alpha^{\{\alpha\}}$ for $\{\alpha\} \multimap \alpha$):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{z : \{\alpha^{\{\alpha\}}\} \vdash z : \alpha^{\{\alpha\}}}\,\texttt{ax} \qquad \dfrac{\dfrac{\dfrac{}{z : \{\alpha^{\{\alpha\}}\} \vdash z : \alpha^{\{\alpha\}}}\,\texttt{ax} \quad \dfrac{\dfrac{}{y : \{\alpha\} \vdash y : \alpha}\,\texttt{ax}}{y : \{\alpha\} \vdash y : \{\alpha\}}\,\texttt{many}}{z : \{\alpha^{\{\alpha\}}\}, y : \{\alpha\} \vdash zy : \alpha}\,\texttt{app}}{z : \{\alpha^{\{\alpha\}}\}, y : \{\alpha\} \vdash zy : \{\alpha\}}\,\texttt{many}}{z : \{\alpha^{\{\alpha\}}\}, y : \{\alpha\} \vdash z(zy) : \alpha}\,\texttt{app}}{z : \{\alpha^{\{\alpha\}}\} \vdash \lambda y.z(zy) : \{\alpha\} \multimap \alpha}\,\texttt{fun}}{\vdash \lambda z.\lambda y.z(zy) : \{\alpha^{\{\alpha\}}\} \multimap \{\alpha\} \multimap \alpha}\,\texttt{fun}$$

Now, with multi types things are different. It is indeed impossible to give the same multi type to all Church numerals. The typings for the first three are:

$\underline{0}$: $\underbrace{[\,\cdot\,]}_{\text{for } z} \multimap \underbrace{[\alpha]}_{y} \multimap \alpha$

$\underline{1}$: $\underbrace{[[\alpha] \multimap \alpha]}_{\text{for } z} \multimap \underbrace{[\alpha]}_{y} \multimap \alpha$

$\underline{2}$: $\underbrace{[[\alpha] \multimap \alpha, [\alpha] \multimap \alpha]}_{\text{for } z} \multimap \underbrace{[\alpha]}_{y} \multimap \alpha$

Where it is clear that the type has size proportional to the represented number. The n-th numeral $\underline{n} \;:=\; \lambda z.\lambda y.z^n y$ has type:

$$[\underbrace{[\alpha] \multimap \alpha, \ldots, [\alpha] \multimap \alpha}_{n \text{ times, for } z}] \multimap \underbrace{[\alpha]}_{y} \multimap \alpha$$

The derivation for $\underline{2}$ is as follows (writing $\alpha^{[\alpha]}$ for $[\alpha] \multimap \alpha$):

$$\begin{prooftree}
\AxiomC{}
\RightLabel{$\mathtt{ax}$}
\UnaryInfC{$z : [\alpha^{[\alpha]}] \vdash z : \alpha^{[\alpha]}$}
\AxiomC{}
\RightLabel{$\mathtt{ax}$}
\UnaryInfC{$z : [\alpha^{[\alpha]}] \vdash z : \alpha^{[\alpha]}$}
\AxiomC{}
\RightLabel{$\mathtt{ax}$}
\UnaryInfC{$y : [\alpha] \vdash y : \alpha$}
\RightLabel{$\mathtt{many}$}
\UnaryInfC{$y : [\alpha] \vdash y : [\alpha]$}
\RightLabel{$\mathtt{app}$}
\BinaryInfC{$z : [\alpha^{[\alpha]}], y : [\alpha] \vdash zy : \alpha$}
\RightLabel{$\mathtt{many}$}
\UnaryInfC{$z : [\alpha^{[\alpha]}], y : [\alpha] \vdash zy : [\alpha]$}
\RightLabel{$\mathtt{app}$}
\BinaryInfC{$z : [\alpha^{[\alpha]}, \alpha^{[\alpha]}], y : [\alpha] \vdash z(zy) : \alpha$}
\RightLabel{$\mathtt{fun}$}
\UnaryInfC{$z : [\alpha^{[\alpha]}, \alpha^{[\alpha]}] \vdash \lambda y.z(zy) : [\alpha] \multimap \alpha$}
\RightLabel{$\mathtt{fun}$}
\UnaryInfC{$\vdash \lambda z.\lambda y.z(zy) : [\alpha^{[\alpha]}, \alpha^{[\alpha]}] \multimap [\alpha] \multimap \alpha$}
\end{prooftree}$$

Note the lower app rule, where the multi-set $[\alpha^{[\alpha]}, \alpha^{[\alpha]}]$ is formed.

Bigger types induce bigger derivations. Let's consider as an example the application $\underline{2}\mathtt{I}$ of $\underline{2}$ to the identity $\mathtt{I}$. With set types, we compose the obtained derivation for $\underline{2}$ with a derivation for $\mathtt{I} = \lambda w.w$, as follows:

$$\begin{prooftree}
\AxiomC{$\vdots$}
\RightLabel{$\mathtt{fun}$}
\UnaryInfC{$\vdash \lambda z.\lambda y.z(zy) : \{\alpha^{\{\alpha\}}\} \multimap \{\alpha\} \multimap \alpha$}
\AxiomC{}
\RightLabel{$\mathtt{ax}$}
\UnaryInfC{$w : \{\alpha\} \vdash w : \alpha$}
\RightLabel{$\mathtt{fun}$}
\UnaryInfC{$\vdash \lambda w.w : \alpha^{\{\alpha\}}$}
\RightLabel{$\mathtt{app}$}
\BinaryInfC{$\vdash (\lambda z.\lambda y.z(zy))(\lambda w.w) : \{\alpha\} \multimap \alpha$}
\end{prooftree}$$

With multi types, instead, when typing $\underline{2}\mathtt{I}$, the bigger type for z requires *two* derivations for the identity:

$$\begin{prooftree}
\AxiomC{$\vdots$}
\RightLabel{$\mathtt{fun}$}
\UnaryInfC{$\vdash \lambda z.\lambda y.z(zy) : [\alpha^{[\alpha]}, \alpha^{[\alpha]}] \multimap [\alpha] \multimap \alpha$}
\AxiomC{}
\RightLabel{$\mathtt{ax}$}
\UnaryInfC{$w : [\alpha] \vdash w : \alpha$}
\RightLabel{$\mathtt{fun}$}
\UnaryInfC{$\vdash \lambda w.w : \alpha^{[\alpha]}$}
\AxiomC{}
\RightLabel{$\mathtt{ax}$}
\UnaryInfC{$w : [\alpha] \vdash w : \alpha$}
\RightLabel{$\mathtt{fun}$}
\UnaryInfC{$\vdash \lambda w.w : \alpha^{[\alpha]}$}
\RightLabel{$\mathtt{many}$}
\BinaryInfC{$\vdash \lambda w.w : [\alpha^{[\alpha]}, \alpha^{[\alpha]}]$}
\RightLabel{$\mathtt{app}$}
\BinaryInfC{$\vdash (\lambda z.\lambda y.z(zy))(\lambda w.w) : [\alpha] \multimap \alpha$}
\end{prooftree}$$

Since the two derivations for $\mathtt{I}$ are identical, it may seem that the increment in size brought by multi types is quite useless, also because it comes with the price of not having a uniform type for Church numerals. It turns out, however, that such a verbosity has very nice consequences.

A Question. Once that I was presenting this lecture, a student asked: *how is it possible that all Church numerals have the same set type? Does this imply that all Church numerals have the same set semantics?*

Careful: the fact that two terms are typable with the same type does not say anything about their set semantics, because the semantics is given by the set of *all* set types assignable to a term, and a term in general has an infinity of types. For instance, we can give a type to $\underline{2}$ that cannot be given to $\underline{1}$. Let Y and Z be two different set types. Note that $\underline{2}$ is typable with $\{\{Y\} \multimap \alpha, \{Y\} \multimap Z\} \multimap \{\alpha\} \multimap Z$ while $\underline{1}$ is not, showing that they do not have the same interpretation into set semantics.

6 Easy Proof of Correctness

Here we show that multi types enable easy proofs for correctness, restoring the symmetry between correctness and completeness (completeness of multi types with respect to head reduction $\to_h$ can be proved exactly as for set types, via subject expansion).

When we discussed correctness of set types (that is, typability implies termination of head reduction) we said that the proof is non-trivial because it requires a termination argument but it is not clear what is decreasing. With multi types, instead, it becomes very clear what is decreasing: it is the size of the type derivation that decreases at each head reduction step. We need a definition.

Definition 6.1 (Derivations size). *Let $\Pi \triangleright \Gamma \vdash M : \sigma$ be a multi types derivation. The* size $|\Pi|$ *of Π is the number of* `ax`, `fun`, `app` *rules in Π (all rules but* `many`*).*

The key point is that one obtains a refinement of subject reduction where the size of derivations decreases with head β steps.

Proposition 6.2 (Quantitative subject reduction). *If $M \to_h N$ and $\Pi \triangleright \Gamma \vdash M : \sigma$ then there exists $\Pi' \triangleright \Gamma \vdash N : \sigma$ such that $|\Pi| > |\Pi'|$.*

As plain subject reduction rests on the substitution lemma, quantitative subject reduction rests on a quantitative substitution lemma.

Lemma 6.3 (Quantitative substitution). *Let $\Pi_M \triangleright \Gamma, x : \mu \vdash M : \sigma$ and $\Pi_N \triangleright \vdash N : \mu$. Then there exists $\Pi_{M\{x:=N\}} \triangleright \Gamma \vdash M\{x := N\} : A$ such that $|\Pi_{M\{x:=N\}}| = |\Pi_M| + |\Pi_N| - |\mu|$, where $|\mu|$ is the number of elements in μ.*

As before, we omit the tedious proof of the lemma, which is by induction on Π_M. We only point out that it is the proof of the substitution lemma—and precisely the case in which Π_M is an axiom and so M a variable—that requires to exclude occurrences of rule `many` from the size of type derivations, because otherwise the quantitative relation $|\Pi_{M\{x:=N\}}| = |\Pi_M| + |\Pi_N| - |\mu|$ does not hold.

$\boxed{\text{Exercise}}$: prove quantitative subject reduction (Proposition 6.2) using the lemma (solution in the Appendix).

Now, we can easily prove correctness for multi types.

Theorem 6.4 (Correctness). *Let $\Pi \triangleright \Gamma \vdash M : \sigma$ be a multi type derivation. Then M is head normalizable.*

Proof. By induction on $|\Pi|$ and case analysis on whether M is head normal. If M is head normal then M is head normalizable. If $M \to_h N$ then by quantitative subject reduction (Proposition 6.2) there is $\Pi' \triangleright \Gamma \vdash N : \sigma$ such that $|\Pi| > |\Pi'|$. By *i.h.*, N is head normalizable. Then so is M. □

No Easy Proofs with Set Types. It is essential to stress that the reasoning used for correctness is *impossible* with set types. We can explain why, by resorting to our example with Church numerals.

Consider the step $\underline{n}\mathtt{I} \to_h \lambda y.\mathtt{I}^n y$. With set types, the derivation Π^n for $\underline{n}\mathtt{I}$ has only one sub derivation $\Pi^n_{\mathtt{I}}$ for $\mathtt{I}$ while the derivation Π'^n for the reduct $\lambda y.\mathtt{I}^n y$ has n copies of $\Pi_{\mathtt{I}}$. Now, Π'^n has also something less than Π^n: the `app` and `fun` rule typing the reduced redex are removed, and also the copies of $\mathtt{I}$ replace the n axioms for z (in the notation used before to define $\underline{n}$), which have then been removed. It is easy to see, however, that starting from $n \geq 5$ one has $|\Pi'^n| > |\Pi^n|$, that is, the copies of $\mathtt{I}$ add more rules to the derivation than those that are removed by the reduction step, showing that quantitative subject reduction does *not* hold with set types.

7 Exact Bounds

The proof of correctness given in the previous section actually proves a finer statement of *quantitative* correctness:

> *Quantitative correctness*: let $\Pi \ \triangleright \Gamma \vdash M : \sigma$ be a multi type derivation. Then $M \to_h^n N$ with N head normal and $n \leq \Pi$.

We have seen in previous lectures that the number of *weak* head β-steps is a reasonable cost model, and along the same lines it is possible to show that also the number of head β-steps is a reasonable cost model.

Quantitative correctness then states that multi type derivations bound the cost model, therefore providing complexity bounds. It is natural to wonder whether the inequality in the bound can become an equality. Said otherwise, *do multi type derivations capture time?* The answer is both *yes* and *no*: they do capture time, but to see it we have to refine 3 aspects of our study, because with the current definitions the inequality cannot be an equality.

Weak Evaluation. For the sake of simplicity, from now one we put ourselves in the slightly simpler setting of *weak* head reduction $\to_{wh}$, forbidding evaluation under abstraction. At the level of types, it means that we consider also the following additional typing rule:

$$\frac{}{\vdash \lambda x.M : \alpha}\ \mathtt{fun\text{-}ax}$$

which is also counted for the size $|\Pi|$ of a type derivation. With this additional rule, the multi type system characterizes termination of $\to_{wh}$.

Refining the Measures. A first reason for the gap between the size of the derivation and the number of steps is that the derivation accounts also for the size of the normal form, not just the number of steps. To be precise, the derivation accounts for part of the normal form. The type system being geared towards weak head reduction, its type derivations can measure only the parts of normal forms that are inspected or produced by weak head reduction, that is, it ignores abstractions.

Therefore, we introduce the following notion of weak head size, counting the number of applications of the left branch of syntax tree of the term. Define:

$$|x|_{wh} := 0 \qquad |\lambda x.M|_{wh} := 0 \qquad |MN|_{wh} := |M|_{wh} + 1$$

For now, we can prove that every type derivation for a weak head normal form bounds its weak head size.

Proposition 7.1. *Let M be a weak head normal form. Then $|M|_{wh} \leq |\Pi|$, for all derivations $\Pi \triangleright \Gamma \vdash M : \sigma$.*

Proof. $\boxed{\text{Exercise}}$. ☐

Now, one can prove that, if $\Pi \triangleright \Gamma \vdash M : \sigma$, and $M \to_{wh}^{n} N$ with N weak head normal, then $n + |N|_{wh} \leq |\Pi|$. Bounds are however still not exact, as the quantity $|\Pi|$ is too generous, we have to refine it.

We define as $|\Pi|_{@}$ the number of `app` rules in Π. With this definition, we have that at every $\to_{wh}$ step the size of the derivation decreases of exactly 1, that is, we have exact subject reduction.

Proposition 7.2 (Exact subject reduction)**.** *If $M \to_h N$ and $\Pi \triangleright \Gamma \vdash M : \sigma$ then there exists $\Pi' \triangleright \Gamma \vdash N : \sigma$ such that $|\Pi|_{@} = |\Pi'|_{@} + 1$.*

The difference with quantitative subject reduction is that the quantitative version states $|\Pi| > |\Pi'|$, while here we have $|\Pi|_{@} = |\Pi'|_{@} + 1$. The proof is as before, the only thing that changes is the consideration about the measures and the fact that a slightly different substitution lemma is used, namely the following one (whose proof is omitted).

Lemma 7.3 (Exact substitution)**.** *Let $\Pi_M \triangleright \Gamma, x : \mu \vdash M : \sigma$ and $\Pi_N \triangleright \vdash N : \mu$. Then there exists $\Pi_{M\{x:=N\}} \triangleright \Gamma \vdash M\{x := N\} : A$ such that $|\Pi_{M\{x:=N\}}|_{@} = |\Pi_M|_{@} + |\Pi_N|_{@}$.*

With the given measures, we can still show that type derivations bound the size of weak head normal forms. Moreover, we can show that every normal form has a derivation capturing exactly its weak head size.

Proposition 7.4. *Let M be a weak head normal form.*

1. *Then $|M|_{wh} \leq |\Pi|_{@}$, for all derivations $\Pi \triangleright \Gamma \vdash M : \sigma$.*

2. *Then there exists $\Pi \triangleright \Gamma \vdash M : \sigma$ such that $|M|_{wh} = |\Pi|_{@}$.*

Proof. $\boxed{\text{Exercise}}$. Hint for point 2: check the proof of Proposition 3.2. ☐

It is now possible to refine subject expansion along the lines of exact subject reduction, obtaining *exact* subject expansion, and a proof of exact completness, that is, the existence of derivation giving exact bounds for every $\to_{wh}$ terminating term, that is, for every typable term.

Tight Derivations. We proved the existence of a derivation giving exact bounds, but do all derivation give exact bounds? If not, can we characterize which ones? We now answer these questions.

First let us show two examples of derivations *not* giving exact bounds. Consider the following derivation Π:

$$\dfrac{\dfrac{\dfrac{}{x : [[\,] \to \sigma] \vdash x : [\,] \to \sigma}\,\texttt{ax} \qquad \dfrac{}{\vdash y : [\,]}\,\texttt{many}}{x : [[\,] \to \sigma] \vdash xy : \sigma}\,\texttt{app}}{\vdash \lambda x.xy : [[\,] \to \sigma] \multimap \sigma}\,\texttt{fun}$$

Here we have $|\Pi|_@ = 1$ while $|\lambda x.xy|_{wh} = 0$. The gap is caused by the fact that Π has typing rules under the normal abstraction, that is, the derivation giving the exact measure is the smaller one formed by `fun-ax` only:

$$\dfrac{}{\vdash \lambda x.xy : \alpha}\,\texttt{fun-ax}$$

Another example is the following derivation Π':

$$\dfrac{\dfrac{}{x : [[\alpha] \multimap \alpha] \vdash x : [\alpha] \multimap \alpha}\,\texttt{ax} \qquad \dfrac{\dfrac{}{y : [[\,] \to \alpha] \vdash y : [\,] \to \alpha}\,\texttt{ax} \qquad \dfrac{}{\vdash z : [\,]}\,\texttt{many}}{y : [[\,] \to \alpha] \vdash yz : \alpha}\,\texttt{app}}{x : [[\alpha] \multimap \alpha], y : [[\,] \to \alpha] \vdash x(yz) : \alpha}\,\texttt{app}$$

Here we have $|\Pi|_@ = 2$ while $|x(yz)|_{wh} = 1$. The gap is now caused by the fact that there is a smaller derivation which avoids typing the argument, namely the following one:

$$\dfrac{\dfrac{}{x : [[\,] \multimap \alpha] \vdash x : [\,] \multimap \alpha}\,\texttt{ax} \qquad \dfrac{}{\vdash yz : [\,]}\,\texttt{many}}{x : [[\,] \multimap \alpha] \vdash x(yz) : \alpha}\,\texttt{app}$$

whose size is 1, catching $|x(yz)|_{wh}$.

These examples suggest the following definition, whose aim is to catch minimal derivations.

Definition 7.5 (Tight types and derivations). *A linear type σ is* tight *if*

$\sigma = \alpha$, *or*

$\sigma = [\,] \multimap \tau$ *and τ is tight.*

A multi set type μ is tight *if*

$\mu = [\,]$, *or*

$\mu = [\sigma]$ *and σ is tight.*

A derivation $\Pi \triangleright \Gamma \vdash M : \sigma$ is tight *if $\sigma = \alpha$ and the types in Γ are tight.*

We can finally prove that tight derivations give exact bounds. First, we prove that tight derivations of normal forms give exact bounds.

Proposition 7.6. *Let M be a weak head normal form. Then $|M|_{wh} = |\Pi|_{@}$, for all tight derivations $\Pi \triangleright \Gamma \vdash M : \alpha$. Moreover, there is a unique tight derivation for M.*

Proof. The shape of M is either $\lambda x.N$ or $xN_1 \ldots N_k$. Then there are two cases:

1. $M = \lambda x.N$: since $\lambda x.N$ has type α, it cannot be typed with a `fun` rule. Then the derivation Π typing it is made out of a single `fun-ax` rule, satisfying $|M|_{wh} = 0 = |\Pi|_{@}$. Uniqueness holds because there are no choices.

2. $M = yM_1 \ldots M_k$: then $\Pi \triangleright \Gamma \vdash M : \alpha$ has the structure of k `app` rules having hereditary on the left an axiom for y, implying that $y \in \mathtt{dom}(\Gamma)$. Since Π is tight, the type $\Gamma(y)$ of y in Γ is tight, that is, of the form $[[\,]^n \multimap \alpha]$ for some n. Since the type of M is α, we have $n = k$. Then Π has the following shape:

$$\dfrac{\dfrac{\dfrac{y : [[\,]^k \multimap \alpha] \vdash y : [\,]^k \multimap \alpha \qquad \overline{\vdash M_1 : [\,]}\;\texttt{many}}{y : [[\,]^k \multimap \alpha] \vdash yM_1 : [\,]^{k-1} \multimap \alpha}\;\texttt{app}}{\vdots} \qquad}{}$$

$$\dfrac{y : [[\,]^k \multimap \alpha] \vdash yM_1 \ldots M_{k-1} : [\,] \multimap \alpha \qquad \overline{\vdash M_k : [\,]}\;\texttt{many}}{y : [[\,]^k \multimap \alpha] \vdash yM_1 \ldots M_k : \alpha}\;\texttt{app}$$

   which satisfies $|M|_{wh} = k = |\Pi|_{@}$. Uniqueness holds because there are no choices. □

We then use exact subject reduction to extend the result to every typable term, that is, we obtain exact correctness.

Theorem 7.7 (Exact correctness)**.** *Let $\Pi \triangleright \Gamma \vdash M : \sigma$ be a tight multi type derivation. Then $M \to_{wh}^{n} N$ with N weak head normal. Moreover, $n + |N|_{wh} = |\Pi|_{@}$.*

Proof. By induction on $|\Pi|_{@}$ and case analysis on whether M is weak head normal.

- If M is weak head normal then by $|M|_{wh} = |\Pi|_{@}$ by Proposition 7.6.
- If $M \to_{wh} M'$ then by exact subject reduction (Proposition 7.2) there is $\Pi' \triangleright \Gamma \vdash M' : \sigma$ such that $|\Pi|_{@} = |\Pi'|_{@} + 1$. By *i.h.*, $M' \to_{wh}^{n} N$ and $n + |N|_{wh} = |\Pi'|_{@}$. Then $M \to_{wh}^{n+1} N$ and $n + 1 + |N|_{wh} = |\Pi'|_{@} + 1 = |\Pi|_{@}$. □

We have already discussed exact completeness. Then we finally obtain an exact operational characterization via multi types.

Theorem 7.8 (Exact characterization)**.** *A term M is typable with a tight derivation Π if and only if $M \to_{wh}^{n} N$ with N weak head normal. Moreover, $n + |N|_{wh} = |\Pi|_{@}$.*

8 Understanding Quantitative Bounds

We now try to put the study of bounds of the previous section into larger perspective.

Minimality and Composability. We strived to characterize minimal derivations, obtaining exact bounds. What about non-minimal derivations? Are they of any use?

Types are devices for providing guarantees *compositionally*. Non-minimal derivations for a term M provide non-exact bounds for M because they anticipate about further evaluation that shall be triggered by composing M with other terms, that is, with its environment (careful, here we are *not* using the word *environment* with respect to the data structure of abstract machines).

On the other hand, minimal derivations achieve precise measurements at the cost of sacrificing essentially all possibilities of composability. From the examples in the previous section, it is clear that exact bounds are possible only when one types normal abstraction with α, which is not an arrow type, thus blocking the possibility of the abstraction to be applied. Similarly, weak head normal terms of the form $xM_1 \ldots M_k$ allow x to have only the type of a function erasing its arguments.

Therefore, there is a sort of *indeterminacy principle*: multi types that give exact bounds prevent interesting compositions, and those that allow interesting compositions do not give exact bound.

Quantitative Semantics. We saw that type derivations can give exact bounds, and that their final judgements provide the relational semantics. It is natural to ask whether bounds can be extracted from the semantics as well, that is, not from type derivations but from (the types in) judgements.

At first sight, this is impossible, because the semantics is invariant by evaluation and so it cannot count the number of steps. Being invariant by evaluation, however, means that essentially it is a description of the normal form. And indeed from the type judgement one can extract bounds on the size of the normal form only. For, one needs to define a notion of size $|J|$ for a type judgement J, which is simply given by the number of occurrences of the arrow $\to$ in all the types in J. It is possible to prove the following bound (proof omitted).

Proposition 8.1 (Judgements bound their own derivations)**.** *Let M be a weak head normal form and $\Pi \triangleright \Gamma \vdash M : \sigma$. Then $|M|_{wh} \leq |\Gamma \vdash M : \sigma|$.*

Moreover, for tight derivations we have equality. From the bound, provide exact bounds for $|M|_{wh}$, when M is weak head normal.

It is also possible to obtain bounds about evaluation lengths from judgement if one considers *two* terms M and N instead of one. The idea is to bound the length of evaluating MN using the judgments for M and N. This is possible, as shown by de Carvalho [dC07, dC18], but it requires a further level of technicalities and it is therefore omitted.

The important point here is the fact that there exists a way of lifting the bounding power of multi types derivations to multi type judgements, that is, to the relational semantics. This is important because relational semantics can also be defined independently of multi types. Therefore, the semantics itself has bounding power, or, equivalently, is *quantitative*. Multi types are only a handy tool to show it.

Semantical Time. Summing up, we can bound evaluation lengths and size of normal forms with both type derivations and the relational semantics. Are these quantitative aspects related to the time cost model?

We know that the number of weak head steps is a reasonable cost model (we proved it in lecture 3, via the MAM). But we also know that the gap between the evaluation length and the size of the normal form can be *exponential*, this is size explosion: there are families of terms $\{M_n\}_{n\in\mathbb{N}}$ such that $M_n \to_{wh}^{n} N_n$ with N_n normal and with $|M_n| = \mathcal{O}(n)$ and $|N_n| = \Omega(2^n)$. This is potentially harmful: it may mean that multi types give bounds that are too lax from a complexity point of view, even when they do provide exact bounds, because they bound *both* the number of steps (which is the cost model) and the size of the normal form (which may be exponentially bigger).

Fortunately, for the easy case that we studied here this is not possible. Remember that, to obtain exact bounds, we restricted to the weak head size $|M|_{wh}$ rather than considering the full size $|M|$ of terms. It turns out that, with the notations above, $|N_n|_{wh} = \mathcal{O}(n)$, that is, the weak head size does not explose. Thus, the measures given by multi types for the weak head case are faithful to the cost model. Similar results can be obtained for the head case.

The case of leftmost reduction, which is strong and computes full normal forms, however, is subtler. There the notion of size of normal form is the plain one, and size explosion strikes back. It is possible to obtain exact quantitative characterizations for leftmost reduction, but the bounds are not informative because of size explosion. It then becomes necessary to bound *separately* evaluation lengths and size of normal forms, following the alternative approach to tight derivations showed by Kesner. Unfortunately, that approach kills the semantical interpretation.

Understanding semantical time is a topic of active research and about which very little is known.

References

[BD20] Viviana Bono and Mariangiola Dezani-Ciancaglini. A tale of intersection types. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 7–20. ACM, 2020.

[BEM07] Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. Not enough points is enough. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 2007.

[BKV17] Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017.

[CDCV80] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Principal type schemes and lambda-calculus semantics. In *To H.B.Curry: Essays on Combinatory Logic, Lambda-calculus and Formalism*, pages 535–560. Academic Press, 1980.

[dC07] Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul.* Thèse de doctorat, Université Aix-Marseille II, 2007.

[dC18] Daniel de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *Math. Str. in Comput. Sci.*, 28(7):1169–1203, 2018.

[Ehr93] Thomas Ehrhard. Hypercoherences: A strongly stable model of linear logic. *Math. Struct. Comput. Sci.*, 3(4):365–385, 1993.

[Ehr05] Thomas Ehrhard. Finiteness spaces. *Math. Struct. Comput. Sci.*, 15(4):615–646, 2005.

[Ehr12] Thomas Ehrhard. Collapsing non-idempotent intersection types. In *CSL*, pages 259–273, 2012.

[Gar94] Philippa Gardner. Discovering needed reductions using type theory. In *Theoretical Aspects of Computer Software (TACS '94)*, volume 789 of *Lecture Notes in Computer Science*, pages 555–574, 1994.

[Kfo00] Assaf J. Kfoury. A linearization of the lambda-calculus and consequences. *Journal of Logic and Computation*, 10(3):411–436, 2000.

[NM04] Peter Møller Neergaard and Harry G. Mairson. Types, potency, and idempotency: why nonlinearity and amnesia make a type system work. In Chris Okasaki and Kathleen Fisher, editors, *Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming, ICFP 2004, Snow Bird, UT, USA, September 19-21, 2004*, pages 138–149. ACM, 2004.