



Lambda Calculi through the Lens of Linear Logic — Delia Kesner

Lecture 3, Quantitative Types - July 2, 2025

Contents

1	Motivations	2
2	From Simple Types to Quantitative Types	2
2.1	Simple Types	2
2.2	Intersection Types	2
2.3	Intersection constructor	3
3	Quantitative Types for Lambda Calculus	3
3.1	Gardner's System $\mathcal{H}$	3
3.2	System H with a Single Counter	4
3.3	Example: Church Numerals	4
4	Quantitative Types and Inhabitation	5
5	Quantitative Types for Measuring	7
5.1	Benefits of counters	7
5.2	Split measures	7
5.3	Head Normalization	7
5.4	System SH: Split Measures	8
5.5	Termination in System $\mathcal{SH}$	9
6	Conclusion and future works	9

1 Motivations

Quantitative information plays a crucial role in computer science, encompassing aspects such as time, space, probability, and cost. This type of analysis is becoming increasingly important across a variety of domains, including automata theory, graph algorithms, logic, and general algorithm design. It also finds applications in areas such as verification, model-checking, programming, and theorem proving, as well as in performance measurement, network analysis, and data mining. Within the theory of programming languages, quantitative properties of programs can be rigorously captured using type systems or relational models. This leads to the study of quantitative type systems, which aim to formalize and analyze the principles, properties, and applications of such systems to better understand and manage quantitative aspects of program behavior.

2 From Simple Types to Quantitative Types

These are two approaches to typing systems. Simple types assign one type per term, making type checking decidable. Quantitative types, such as the intersection type, allows a term to have several types simultaneously, increasing expressiveness but making the typing system undecidable.

2.1 Simple Types

Grammar: $A, B ::= \iota \mid A \rightarrow B$

Typing rules:

$$\frac{}{\Gamma, x : A \vdash x : A} \text{VAR} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x. t : A \rightarrow B} \text{ABS} \qquad \frac{\Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B} \text{APP}$$

Notes: Simple types are monomorphic and decidable. Polymorphic extensions retain decidability.

2.2 Intersection Types

An intersection $A \cap B$ is possible if we have A and B , with this we can capture a form of finite polymorphism. The typing system becomes more expressive compared to simple types. However, the typing system becomes undecidable, hence, it is not a typing system that is supposed to be used in a concrete programming language, but to make semantic interpretations for computation.

Grammar: $A, B ::= \iota \mid A \rightarrow B \mid A \cap B$

Key rule:

$$\frac{t : A \quad t : B}{t : A \cap B} \text{ INTER}$$

2.3 Intersection constructor

There are several ways to define the intersection constructor. The one we use is both associative and commutative.

$$\begin{array}{ll} (A \cap B) \cap C \sim A \cap (B \cap C) & \text{Associativity} \\ A \cap B \sim B \cap A & \text{Commutativity} \end{array}$$

There is also the notion of idempotent and non-idempotent intersection types. The idempotent have *unbounded* resources and uses qualitative properties (e.g. yes or no). Whilst, the non-idempotent type have *finite* resources and quantitative properties (e.g. bounds and measures), we can think of the non-idempotent as bounded linear logic.

$$\begin{array}{lll} \textbf{Idempotent:} & A \cap A \sim A & (\text{qualitative properties}) \\ \textbf{Non-idempotent:} & A \cap A \approx A & (\text{quantitative properties via multi-sets}) \end{array}$$

3 Quantitative Types for Lambda Calculus

3.1 Gardner's System $\mathcal{H}$

We use Gardner's system $\mathcal{H}$, where the multi-set are the intersections. From a quantitative linear logic intuition, we can think about a multi-set $[A_1, \dots, A_n]$ as the tensor $A_1 \otimes \dots \otimes A_n$ and we can think of $M \rightarrow A$ as $M^\perp \wp A$

Grammar:

$$\begin{array}{l} A ::= \iota \mid M \rightarrow A \\ M ::= [A_i]_{i \in I} \end{array}$$

Judgment form: Note that M_i denote the multi-type for each variable x_i and A is the type for the term t

$$x_1 : M_1, \dots, x_n : M_n \vdash t : A$$

Rules:

$$\begin{array}{c}
 \frac{}{x : [A] \vdash x : A} \text{Ax} \qquad \frac{\Gamma \vdash t : A}{\Gamma \setminus x \vdash \lambda x. t : \Gamma(x) \rightarrow A} \text{FUN} \qquad \frac{(\Gamma_i \vdash t : A_i)_{i \in I}}{\sqcup_{i \in I} \Gamma_i \vdash t : [A_i]} \text{MANY} \\
 \\
 \frac{\Gamma \vdash t : M \rightarrow B \quad \Delta \vdash u : M}{\Gamma \sqcup \Delta \vdash tu : B} \text{APP}
 \end{array}$$

3.2 System H with a Single Counter

The System $\mathcal{H}$ can be extended to encompass counters, that counts the size of the proof derivation. The counter increments on all rules except (many).

Notation

$$\Pi \triangleright_S \Gamma \vdash^{C_1, \dots, C_n} t : A$$

Where: Π is a (tree) derivation, S is a type system, Γ is a set of type declarations, $(C_1, \dots, C_n)$ are counters, t is a term and A is a type. Note that Π , S and $(C_1, \dots, C_n)$ are sometimes omitted when their meaning is trivial.

Rules

$$\begin{array}{c}
 \frac{}{x : [A] \vdash^{(1)} x : A} \text{ax} \qquad \frac{\Gamma \vdash^{(C)} t : A}{\Gamma \setminus x \vdash^{(C+1)} \lambda x. t : \Gamma(x) \rightarrow A} \rightarrow_i \\
 \\
 \frac{(\Gamma_i \vdash^{(C_i)} t : A_i)_{i \in I}}{\sqcup_{i \in I} \Gamma_i \vdash^{(+i \in I C_i)} t : [A_i]_{i \in I}} \text{many} \qquad \frac{\Gamma \vdash^{(C_1)} t : M \rightarrow B \quad \Delta \vdash^{(C_2)} u : M}{\Gamma \sqcup \Delta \vdash^{(C_1+C_2+1)} tu : B} \rightarrow_e
 \end{array}$$

3.3 Example: Church Numerals

Consider that we let $3 := \lambda f. \lambda x. f(f(fx))$ and let $B := [A] \rightarrow A$ Then we have the following derivation:

$$\begin{array}{c}
 \frac{}{x : [A] \vdash x : A} \\
 \frac{}{f : B \vdash f : [A] \rightarrow A} \quad \frac{}{x : [A] \vdash x : [A]} \\
 \hline
 f : [B], x : [A] \vdash fx : A \\
 \hline
 \frac{}{f : B \vdash f : [A] \rightarrow A} \quad \frac{}{f : [B], x : [A] \vdash fx : [A]} \\
 \hline
 f : [B, B], x : [A] \vdash f(fx) : A \\
 \hline
 \frac{}{f : B \vdash f : [A] \rightarrow A} \quad \frac{}{f : [B, B], x : [A] \vdash f(fx) : [A]} \\
 \hline
 f : [B, B, B], x : [A] \vdash f(f(fx)) : A \\
 \hline
 f : [B, B, B] \vdash \lambda x. f(f(fx)) : [A] \rightarrow A \\
 \hline
 \vdash \underline{3} : [B, B, B] \rightarrow [A] \rightarrow A
 \end{array}$$

The conclusion $\vdash 3 : [B, B, B] \rightarrow [A] \rightarrow A$ is the case when we have a non-idempotent/quantitative typing by using the multi-sets, where 3 is typed with an intersection type $[B, B, B]$. If we used idempotent/qualitative types using sets then the conclusion would have been $\vdash 3 : \{B\} \rightarrow \{A\} \rightarrow A$

4 Quantitative Types and Inhabitation

Dual Problems There is a conceptual duality between the **Typing Problem** and the **Inhabitation Problem**, by emphasizing which elements of the typing judgment

$$\Gamma \vdash t : A$$

are given and which must be synthesized, as well as the common foundational components shared by both problems.

Typing Problem.

$$\Gamma ? \vdash t : A ?$$

In this setting, the term t is given, and we are tasked with finding both a context Γ and a type A such that the judgment holds. This process requires two ingredients:

- The **term language** determines the syntactic structure and grammar of valid terms (e.g., variables, abstractions, applications).
- The **typing system** specifies the rules used to assign types to terms — for instance, function types, intersection types, or quantitative types.

Only by knowing both the term language and the typing system can we determine whether there exists a suitable typing derivation for t .

Inhabitation Problem.

$$\Gamma \vdash t? : A$$

Here, the context Γ and the type A are given, and the goal is to construct or search for a term t that inhabits the type A under the assumptions in Γ . Again, this depends on:

- The **term language**, which dictates what terms may be constructed.
- The **typing system**, which verifies whether a candidate term truly inhabits the target type.

These problems are closely related to:

- *Proof Search* in logical systems.
- *Program Synthesis* from types.

Decidability Landscape The decidability of typing and inhabitation depends on the type system and lambda calculus evaluation strategy:

Call-by-name λ -Calculus	Typing ($? \vdash t : ?$)	Inhabitation ($\Gamma \vdash ? : A$)
Simple Types	Decidable	Decidable
Unrestricted	Undecidable	Undecidable
Idempotent Types (∞ Resources, restricted)	Undecidable	Decidable
Idempotent Types (∞ Resources, unrestricted)	Undecidable	Decidable
Non-idempotent Types (Finite Resources)	Decidable	Decidable

Decidability prefers (finite) searches on finite resources.

Inhabitation Algorithm Properties

Let (Γ, σ) be an inhabitation goal. Then the inhabitation algorithm has the following properties:

- **Termination:** Every call generates a finite set of recursive calls.
- **Soundness:** If T is generated by the algorithm, then $\Gamma \vdash T : \sigma$.
- **Completeness:** If $\Gamma \vdash T : \sigma$, then T can be generated by the algorithm.

5 Quantitative Types for Measuring

Usually type systems provide us a way to express qualitative properties of programs, that is we can only say whether or not some property (like termination) holds, but we cannot specify quantities attached to it (in case of termination: we do not know the number of necessary reduction steps). For this we need to improve our type system with **counters** that will somehow encode our quantities.

5.1 Benefits of counters

Proper design of a qualitative type system allows us to easily prove certain properties of the language. For example termination is equivalent to term's typability in non-idempotent system $\mathcal{H}$ defined above. Below we present a sketch of the proof.

Proof. We define a relational model of programs from our language: $\llbracket t \rrbracket := \{\triangleright \Gamma \vdash t : A\}$ Then we prove that equivalent (with respect to reduction relation) programs are mapped to equal sets, i.e.: if $t \rightarrow u$ then $\llbracket t \rrbracket = \llbracket u \rrbracket$. This gives us an equivalence of derivations: $\triangleright \Gamma \vdash^{(C)} t : A \iff \triangleright \Gamma \vdash^{(C')} u : A$. From properties of system $\mathcal{H}$ follows that $C > C'$. Counters are natural numbers, so termination follows from well-foundedness of ' $<$ ' relation (we cannot have infinite decreasing sequences). $\square$

5.2 Split measures

With more complicated structure of counters we can give more detailed specifications of terms. Gardner's System $\mathcal{H}$ gives only an upper bound on the sum of term's size and the number of its reduction steps. We can improve this system with **exact measures**, that track the exact value of that sum, but better solution is provided with **split measures**. With this approach we introduce a separate counter for each quantity we want to keep track of and we make the tuple consisting of those counters our new counter.

5.3 Head Normalization

To give an example of the use of our system, we formalize the notion of **head normal forms**.

Head Normal Forms (HNF): $\lambda x_1 \dots \lambda x_n. y t_1 \dots t_m$

Head Evaluation: It is like ordinary evaluation, but we do not evaluate the right sides (the arguments) of the application.

Head Normalization: is head evaluation of a term to its head normal form.

Example:

$$I := \lambda y.y, \quad \Omega := (\lambda y.yy)(\lambda y.yy)$$

$$\lambda x.Ix\Omega \text{ is HNF, } \Omega \text{ is not.}$$

Gardner's system $\mathcal{H}$ gives the following formulation of termination:

Theorem 5.1. $\triangleright_{\mathcal{H}} \Gamma \vdash^{(C)} t : A$ iff t head normalizes in L steps to a HNF of size S and $L + S \leq G$

5.4 System SH: Split Measures

We now introduce system $\mathcal{SH}$ implementing split measures.

Grammar of types is following:

(Tight Constants)	$\mathbf{tt} ::= \mathbf{n} \mathbf{a}$
(Types)	$\mathbf{A} ::= \mathbf{tt} \mathbf{M} \rightarrow \mathbf{A}$
(Multi-types)	$\mathbf{M} ::= [\mathbf{A}_i]_{i \in I}$

Judgements have the form: $x_1 : \mathbf{M}_1, \dots, x_n : \mathbf{M}_n \vdash^{(L,S)} t : \mathbf{A}$ Here L captures the length of reduction sequence ending in HNF and S captures the size of that HNF.

We say that a derivation is **tight** if $\mathbf{A}$ and all types in Γ are tight constants.

Typing Rules:

$$\frac{}{x : [A] \vdash^{(0,0)} x : A} \text{Ax} \qquad \frac{(\Gamma_i \vdash^{(L_i, S_i)} t : A_i)_{i \in I}}{\bigsqcup_{i \in I} \Gamma_i \vdash^{(\sum_{i \in I} L_i, \sum_{i \in I} S_i)} t : [A_i]} \text{MANY}$$

$$\frac{\Gamma; x : M \vdash^{(L,S)} t : A}{\Gamma \vdash^{(L+1,S)} \lambda x.t : M \rightarrow A} \text{FUN}_C \qquad \frac{\Gamma; x : M \vdash^{(L,S)} t : a \quad \mathbf{IsTight}(\mathbf{M})}{\Gamma \vdash^{(L,S+1)} \lambda x.t : a} \text{FUN}_P$$

$$\frac{\Gamma \vdash^{(L,S)} t : M \rightarrow A \quad \Delta \vdash^{(L',S')} u : M}{\Gamma \cup \Delta \vdash^{(L+L',S+S')} tu : A} \text{APP}_C \qquad \frac{\Gamma \vdash^{(L,S)} t : n \quad \Delta \vdash^{(L',S')} u : a}{\Gamma \cup \Delta \vdash^{(L+L',S+S'+1)} tu : n} \text{APP}_P$$

Example:

$$\begin{array}{c}
 \frac{z : \mathbf{n} \vdash^{(0,0)} z : \mathbf{n}}{\vdash^{(0,1)} \mathbf{I} : \mathbf{a}} \\
 \frac{x : [[\mathbf{a}] \rightarrow \mathbf{a}] \vdash^{(0,0)} x : [\mathbf{a}] \rightarrow \mathbf{a} \quad \vdash^{(0,1)} \mathbf{I} : [\mathbf{a}]}{\vdash^{(0,1)} \lambda x.x\mathbf{I} : [[\mathbf{a}] \rightarrow \mathbf{a}] \rightarrow \mathbf{a}} \quad \frac{z : [\mathbf{a}] \vdash^{(0,0)} z : \mathbf{a}}{\vdash^{(1,0)} \mathbf{I} : [\mathbf{a}] \rightarrow \mathbf{a}} \\
 \hline
 \vdash^{(2,1)} (\lambda x.x\mathbf{I})\mathbf{I} : \mathbf{a}
 \end{array}$$

Term $(\lambda x.x\mathbf{I})\mathbf{I}$ evaluates in 2 steps

$$(\lambda x.x\mathbf{I})\mathbf{I} \rightarrow_{\beta} \mathbf{I}\mathbf{I} \rightarrow_{\beta} \mathbf{I}$$

to a normal form $\mathbf{I}$ of size 1.

5.5 Termination in System $\mathcal{SH}$

Now we can give a more precise formulation of termination in system $\mathcal{SH}$

Theorem 5.2. (*Split Measures*)

$$\triangleright_{\mathcal{SH}} \Gamma \vdash^{(L,S)} t : A \Leftrightarrow t \text{ head normalizes in } L \text{ steps to HNF of size } S$$

6 Conclusion and future works

Quantitative type systems offer a powerful framework for understanding and analyzing various computational phenomena. They enable a fine-grained characterization of termination behaviors—such as head, weak, strong, or infinitary termination—and help tackle classical problems like size explosion by distinguishing between upper bounds and exact measures. These systems provide a quantitative perspective on traditional semantic properties like solvability and genericity and benefit from the expressive strength of relational models. They can render previously undecidable problems, such as inhabitation, decidable, and offer a structured approach for proving observational equivalence, delineating complexity classes, and establishing completeness of reduction strategies.

Looking forward, several challenging areas remain to be explored. These include the development of quantitative type theories for effectful models of computation, useful and strong evaluation strategies, and techniques such as deep inference and general rewriting. Further refinement is needed to provide a more comprehensive quantitative account of traditional properties and to assess the efficiency of various programming strategies and implementations. Finally, a key objective is to identify decidable fragments of these theories that are both expressive and practical for use in real-world programming languages.