# Reasonable Time, $\lambda$-Calculus, and Linear Logic

**MPRI course**
**Logique Linéaire et Paradigmes Logiques du Calcul,**
**Year 2023-24, part 4**
**Lecture 2**

Beniamino Accattoli

March 8, 2025

## 1 Introducing Reasonable Cost Models

Turing machines (shortened to TMs) are the standard computational theory[1] for computational complexity because their cost models are self-evident:

*Time*: the number of transitions performed during execution;

*Space*: the maximum number of cells of the tape used during execution.

On TMs, space cannot be greater than time, because using space requires time—we shall then say that space and time are *locked*.

In order to study complexity in a different theory $T$ one has to first fix cost models for $T$. The basic requirement for cost models is to be *reasonable*, that is, equivalent to those of Turing machines. Precisely, a cost model for a computational theory $T$ is *reasonable* if there are mutual simulations of $T$ and TMs (or another reasonable theory) working within:

- for *time*, a polynomial overhead;
- for *space*, a linear overhead.

In many cases, the two bounds hold simultaneously for the same simulation, but this is not a strict requirement. The aim is to ensure that the basic hierarchy of complexity classes

$$\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \tag{1}$$

can be equivalently defined on any reasonable theory, that is, that such classes are *robust*, or theory-independent. Note a slight asymmetry: while for time the complexity of the

[1]The literature usually refers to *computational models* rather than *computational theories*. We prefer to use *theory* as the word *model* is overloaded, in particular we want to avoid confusion with *cost model*, and also with *denotational model*.

required overhead (polynomial) coincides with the smallest robust time class (P), for space the smallest robust class is logarithmic (L) and not linear space. In particular, a simulation within a linear space overhead for logarithmic space implies that one needs to preserve logarithmic space.

Random access machines (RAMs), which are the theory of reference for modern computers, are reasonable. The notion of reasonable cost model was introduced by Slot and van Emde Boas in the 1980s [SvEB84, vEB90], while trying to obtain a reasonable space cost model for RAMs.

Reasonable cost models are used to refine the notion of *(in)effectively computable* via an extended *thesis*, known alternatively as the *strong*, *extended*, *efficient*, *modern*, or *complexity-theoretic* Church(-Turing) thesis, or also as the *invariance thesis* (reasonable cost models are also called *invariant* cost models, as they do not change complexity classes when changing theory):

*All computational theories are reasonable.*

The thesis has to be intended as a discrimination principle rather than as a proved fact, as it helps understanding, adjusting, or rejecting a computational theory depending on whether it is reasonable or not.

**Sub-Polynomial Overhead?** It is natural to wonder whether the invariance thesis can be strengthened, requiring for instance that all theories are correlated by a *linear* overhead in time. It turns out that such a requirement is too strong, because simulations between theories often require a non-linear time overhead. For instance, TMs simulate RAM with a *quadratic* time overhead, needed to simulate random access on a sequential tape. Said differently, sub-polynomial time classes are not robust, which is one of the reasons why the class P is so relevant.

**Cost Models for the $\lambda$-Calculus.** The theory of interest, here, is the $\lambda$-calculus. Since the inception of computer science Turing machines were considered more effective than the $\lambda$-calculus because the cost models of the $\lambda$-calculus are not really evident. More precisely, there are various natural candidates, but they all seem problematic.

First of all, since the $\lambda$-calculus is *non-deterministic*, one needs to fix a deterministic (or diamond) notion of evaluation, that is, to fix an evaluation strategy. For the moment, we want to be agnostic with respect to the strategy, so we assume given an evaluation strategy $\rightarrow_{\texttt{str}}$ (think of head, weak head, or leftmost reduction) with its corresponding notion of normal form $\texttt{nf}_X(t)$. Then, we have three candidates for the time cost of a $\rightarrow_{\texttt{str}}$ evaluation $e : t_0 \rightarrow_{\texttt{str}} t_1 \rightarrow_{\texttt{str}} t_2 \rightarrow_{\texttt{str}} \ldots \rightarrow_{\texttt{str}} t_n$ of length $n$:

*Ink time*: the time needed to print out all the terms $t_i$ for $i \in \{0, \ldots, n\}$;

*Abstract time*: the number $n$ of $\rightarrow_{\texttt{str}}$ steps;

*Low-level time*: the time taken by an abstract machine implementing $e$.

For space, it is not clear what abstract space would be, thus we have only ink space and low-level space:

*Ink space*: the size of the largest $\lambda$-term among the $t_i$.

*Low-level space*: the (max) space used by an abstract machine implementing $e$.

In this lecture we are mainly concerned with time. Ink time is locked with ink space and easily shown to be reasonable. The problem with it is twofold. On the one hand, it is a too generous notion, since the cost of functional programs is not usually estimated in this way (in functional programming practice). On the other hand, it is difficult to reason with such a notion, as it is not abstract enough, because different $\beta$-steps have different costs.

Low-level time is a more realistic measure, it is locked with low-level space (if they refer to the same implementation), and easily proved to be reasonable. The obvious drawback is that it rather is a family of cost models, as the cost depends on the implementation. Even when the evaluation strategy is fixed, there are many possible implementations, which may have very different asymptotic costs. Additionally, it is a notion of cost that depends very much on implementation details. It does not have the *distance-from-implementative-details* that is distinctive of the $\lambda$-calculus.

Abstract time is the best notion, since it does not depend on an implementation and it is close to the practice of cost estimates, which does count the number of *function calls*, that, roughly, is the number of $\beta$-steps. The puzzling point is that it is *not locked* with ink space, which seems the natural corresponding notion of space: ink space can indeed be *exponential* in abstract time (independently of the strategy), a degeneracy known as *size explosion*—we shall say that time and space are *explosive.* Size explosion seem to suggest that abstract time is not reasonable. Roughly, $\beta$-steps do not seem to be reasonable operations, as they are based on the complex operation of meta-level substitution.

Is the $\lambda$-calculus reasonable? It certainly is, with respect to unsatisfying cost models such as the ink or low-level ones. The interesting question rather is whether abstract time is a reasonable cost model. This was unclear for a while, because of the intuition that a reasonable time cost model has to be locked with its underlying notion of space, as it is the case for TMs.

**Key Points.** There are a few points that have to be better understood, and that we shall discuss in the following order (the following list provides sketched answers):

1. *Strategy*: which strategy should one consider as providing a notion of reasonable time for the $\lambda$-calculus? Roughly, the main evaluation strategies (weak head, head, leftmost, call-by-value, call-by-need) all do.

2. *Reasonable steps*: what is a reasonable computational step? Is $\beta$ a resonable step? This will lead us to isolate the *sub-term property.*

3. *Size explosion*: is size explosion inevitable? What is it due to? Size explosion is inevitable in the $\lambda$-calculus, and connected to the sub-term property.

4. *Abstract time*: is abstract time reasonable? Yes, but one has to disentangle it from ink space and reason *up to sharing.*

5. *Abstracting low-level*: is there a way of making low-level reasoning independent of the choice of an implementation? Can low-level time be expressed in an abstract way? Yes, it is possible, via a $\lambda$-calculus with sharing called *linear substitution calculus.*

We shall discuss these key points in this lecture, which is organized along the two natural halves of the study of cost models, namely finding:

1. A reasonable encoding of a reasonable theory (usually TMs) in the $\lambda$-calculus;

2. A reasonable encoding of the $\lambda$-calculus in a reasonable theory (usually RAMs).

**Space.** While *space* plays a key role in our understanding of the problem, we shall rather focus on time in this course. Results about reasonable space are more recent and more technical. They shall be discussed in another lecture.

## 2 From Turing Machines to the $\lambda$-Calculus

Most courses on the $\lambda$-calculus show how to represent partial recursive functions into it—see for instance the classic books by Barendregt [Bar84] or Krivine [Kri93]. The representation of TMs can be traced back to the appendix of Turing's 1936 paper [Tur36]. In contemporary literature, however this representation is hard to find, but in itself it is not difficult. It is also not hard to show that TMs can be *reasonably* simulated in the $\lambda$-calculus, that is, within a polynomial overhead. The reason is quite simple: TMs are a first-order system, while the $\lambda$-calculus is higher-order, so it is expected that higher-order can simulate first-order reasonably.

At first sight, then, this direction seems not to be particularly exciting. Yet, here it lies an interesting and counterintuitive aspect of the problem. Exactly because the higher-order world is much larger than the first-order one, it is possible to encode TMs in a very simple fragment of the $\lambda$-calculus, what we like to call the *deterministic $\lambda$-calculus* $\Lambda_{\mathtt{det}}$. Let us introduce it.

**The Deterministic $\lambda$-Calculus.** The language of terms of $\Lambda_{\mathtt{det}}$ is a strict subset of the $\lambda$-calculus and is defined by:

$$\textsc{Terms}\quad t,u,s,p \ ::= \ v \mid tv \qquad\qquad \textsc{Values}\quad v,v' \ ::= \ \lambda x.t \mid x$$

Note that the right subterm of an application has to be a value, in contrast to what happens in the ordinary $\lambda$-calculus. Evaluation in $\Lambda_{\mathtt{det}}$ is also strictly less general than in the ordinary $\lambda$-calculus, as evaluation contexts are *weak*, i.e. they do not enter inside abstractions bodies:

| Weak evaluation contexts | Root rule | Contextual closure |
|---|---|---|
| $E ::= \langle\cdot\rangle \mid Ev$ | $(\lambda x.t)u \mapsto_\beta t\{x{\leftarrow}u\}$ | $E\langle t\rangle \to_{det} E\langle u\rangle \quad \text{if } t \mapsto_\beta u$ |

The deterministic $\lambda$-calculus is essentially the intersection of a *continuation-passing style* (CPS) calculus, where arguments can only be values, and of a weak calculus. Being CPS, call-by-name and call-by-value coincide, simply because all arguments are values. CPS calculi however usually rely on strong evaluation, while here we adopt the weak one.

The combination of the CPS and weak restrictions provides the following determinism property, that justifies the name of the calculus.

**Lemma 2.1** (Determinism). *Let $t \in \Lambda_{\mathtt{det}}$. If $t \to_{det} u$ and $t \to_{det} s$ then $u = s$.*

*Proof.* By induction on $t$. If $t$ is a variable or an abstraction then it cannot reduce. If $t = pv$ then there are two cases:

- *$p$ is an abstraction $\lambda x.q$.* Then $t = (\lambda x.q)v \to_{det} q\{x{\leftarrow}v\}$ is the unique redex of $t$, that is, $u = s = q\{x{\leftarrow}v\}$.
- *$p$ is not an abstraction.* Then the two steps from $t$ come from two steps $p \to_{det} u'$ and $p \to_{det} s'$ with $u = u'v$ and $s = s'v$, because $\langle\cdot\rangle v$ is the only possible evaluation context. By *i.h.*, $u' = s'$, that is, $u = s$. □

**Encoding Turing Machines.** In the note [DLA17] available on Accattoli's webpage, there is an encoding of TMs in $\Lambda_{\mathtt{det}}$, due to Dal Lago, within a *linear* overhead in time, explained in all details. For this course, we only need the main result of that note, given right next. The interested reader, however, is invited to have a look at the note.

**Theorem 2.2** (Linear simulation, [DLA17]). *Let $\Sigma$ be an alphabet and $f : \Sigma^* \to \Sigma^*$ a function computed by a Turing machine $\mathcal{M}$ in time $g$. Then there is an encoding $\overline{\cdot}$ into $\Lambda_{\mathtt{det}}$ of $\Sigma$, strings, and Turing machines over $\Sigma$ such that for every $s \in \Sigma^*$, $\overline{\mathcal{M}}\,\overline{s} \to_{det}^{n} \overline{f(s)}$ where $n = \Theta(g(|s|) + |s|)$.*

Let's now see the consequence of the existence of such an encoding.

**Weak Strategies.** Essentially, all the weak strategies of the $\lambda$-calculus collapse when restricted to the deterministic $\lambda$-calculus: terms have at most one weak redex and so all weak strategies coincide. Then all weak strategies provide a reasonable simulation of TMs, and whenever a weak strategy (of the unrestricted $\lambda$-calculus) can be reasonably simulated on a reasonable model then it is reasonable, *independently of its efficiency.*

Let us now go back to the half-determinism of the $\lambda$-calculus. The difference in efficiency between strategies may be such that, on a given term, a strategy normalises while another one diverges. Despite its desperate inefficiency, if a diverging strategy can be implemented within a polynomial overhead then it is reasonable. And this is indeed possible: weak call-by-value is a reasonable strategy and yet it can diverge when weak call-by-name (aka weak head reduction) terminates. Typically, the following term $(\lambda x.y)((\lambda z.zz)(\lambda z.zz))$ normalises in one step in call-by-name but diverges in call-by-value (note that it does not belong to $\Lambda_{\mathtt{det}}$). Therefore, reasonable weak strategies need not being *terminating*, which is a quite counterintuitive fact. What they do need to verify is that they allow to simulate TMs (according to the encoding of Theorem 2.2), which is a weaker requirement. Namely, they only need to avoid stopping while there still are weak redexes.

**Strong Strategies and Perpetuality.** For strong strategies the question is subtler. First of all, let us be precise: we mean that one keeps the same encoding of Theorem 2.2, the image of which is in $\Lambda_{\mathtt{det}}$, but then one liberalizes the deterministic $\lambda$-calculus by adopting strong evaluation rather than the weak one. The change breaks the key determinism of $\Lambda_{\mathtt{det}}$, and then the choice of the strategy matters.

The encoding of Theorem 2.2 still provides a linear simulation of TMs for strategies such as strong call-by-name (aka leftmost reduction) and strong call-by-value. More generally, for every strong strategy reducing weak redexes or weak head redexes before any strong one.

For strong strategies, however, the behaviour with respect to termination is not irrelevant as in the weak case (because the relaxed $\Lambda_{\mathtt{det}}$ is not deterministic). The point is that the encoding relies on a fix-point operator. Fix-point operators always have, among the various possible evaluations, some diverging strong evaluations. Therefore, strong strategies selecting such diverging evaluations diverge on the encoding of every TM, thus failing to simulate them. Therefore, their number of steps cannot provide reasonable time cost models.

## 3 From $\lambda$-Calculus to Turing Machines

Encoding the $\lambda$-calculus into a reasonable theory is the difficult part of showing that abstract time is reasonable for the $\lambda$-calculus. The target theory is usually RAMs, as it is the theory usually used for studying the complexity of concrete algorithms. The simulation is studied in detail but the actual encoding on RAMs is never spelled out. It is considered enough to provide the complexity analysis of a semi-formal algorithm implementing the $\lambda$-calculus.

Given an evaluation sequence $t_0 \to_{\mathtt{str}}^{n} t_n$ in the $\lambda$-calculus, one wants to simulate it on RAMs in time polynomial in the following two key parameters:

1. *Size of the input*: the size $|t_0|$ of the initial term, that is, the number of constructors in $t_0$.

2. *Abstract time*: the number $n$ of $\beta$-steps.

We shall go through three steps: discussing reasonable steps, analyzing size explosion, and introducing sharing to circumvent it.

## 4 Reasonable Steps and the Sub-Term Property

On Turing machines, the time cost of a single transition is $\mathcal{O}(1)$: it is proportional to the size of the alphabet used by the machine, which is fixed and therefore constant. Given a computational theory $T$, its computational steps do not have to be constant-cost in order to provide a reasonable time cost model, as we now discuss.

Let us abstract away from the $\lambda$-calculus and $\beta$-reduction for a moment, and rather consider a generic deterministic rewriting system $(S, \to)$. For it to be reasonably simulated by RAMs, we introduce a notion of reasonable step.

**Definition 4.1** (Reasonable steps)**.** *A rewriting system $(S, \to)$ has* reasonable steps *if there exists a polynomial $p$ such that for all $t_0 \in S$ and for every evaluation sequence $t_0 \to^n t_n$ the cost of the $i$-th step $t_{i-1} \to t_i$ in the sequence is bound by $p(t_0, i)$ when implementing the sequence $e$ on RAMs, that is, it is polynomial in the size of the initial term and the number of preceding steps in the sequence.*

As expected, reasonable steps guarantee that the rewriting system can be reasonably simulated on RAMs.

**Proposition 4.2.** *If a rewriting system $(S, \to)$ has reasonable steps then every sequence $t_0 \to^n t_n$ can be implemented on RAMs in time polynomial in $n$ and $|t_0|$.*

*Proof.* The cost of the sequence is the sum of the costs of single steps, which by the reasonable steps hypothesis is a sum of polynomials, which is a polynomial. □

Usually, the reasonability of steps follows from the so-called sub-term property. The property has two formulations, a structural and a quantitative one.

**Definition 4.3** (Sub-term property)**.** *A rewriting system $(S, \to)$ has the sub-term property if in every sequence $t_0 \to^n t_n$ every step of the sequence either only involves a constant number of constructors or it duplicates or erases a term:*

- Structural sub-term property*: which is a sub-term of $t_0$ up to variables renaming;*
- Quantitative sub-term property*: of size bound by $|t_0|$,*

*In the case of duplications, the number of copies is also bound by $|t_0|$ (for both notions).*

The structural property provides the intuition and gives the name to the property. For cost analyses, however, one only needs the quantitative version. The two formulations are two sides of the same concept and most of the time we simply refer to the sub-term property, without further specification.

The sub-term property usually immediately implies the linear cost of steps with respect to $|t_0|$—steps are then reasonable. A priori, it might be that, even when the property holds and thus the size of duplicated terms is under control, *searching for redexes* might be expensive. Technically speaking, then, the sub-term property does not directly imply reasonability, even if in most cases it is usually the case. We shall discuss the search for redexes in the next lecture.

**$\lambda$-Calculus *vs* the Sub-Term Property.** In the $\lambda$-calculus, no evaluation strategy has the structural sub-term property, as we now show with an example. Let $\tau_t := \lambda y.ytt$ and $\mathtt{I} := \lambda x.x$. Then:

$$\begin{aligned} u := \quad & (\lambda x.x(\lambda z.\tau_z)\tau_x)\mathtt{I} && \rightarrow_\beta && \mathtt{I}(\lambda z.\tau_z)\tau_\mathtt{I} && \rightarrow_\beta \\ & (\lambda z.\tau_z)\tau_\mathtt{I} && \rightarrow_\beta && \tau_{\tau_\mathtt{I}} && = && \lambda y.y\tau_\mathtt{I}\tau_\mathtt{I} \end{aligned} \tag{2}$$

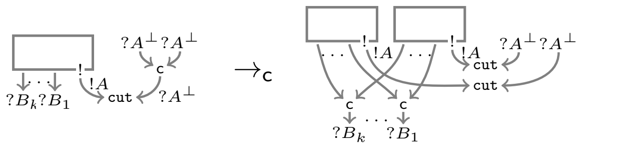
Note that the sub-term $\tau_\mathtt{I}$ duplicated by the third step is *not* a sub-term of $u$. Moreover, $u$ is closed and each term in the sequence as at most one $\beta$-redex, which is out of abstractions. Therefore, the example uses only weak evaluation and there are no alternative evaluation sequences. The example even belongs to the deterministic $\lambda$-calculus $\Lambda_{\mathtt{det}}$.

Roughly, the reason why the sub-term property breaks is meta-level substitution, which substitutes on all variable occurrences, thus affecting sub-terms that shall be duplicated later on.

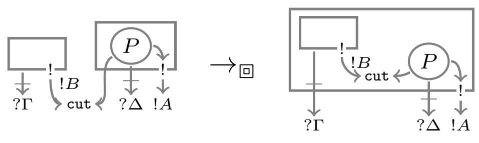
**Linear Logic *vs* the Sub-Term Property.** It is also easy to see that one of the main evaluation strategies for proof nets does not have the sub-term property. Such a strategy is *cut elimination at level 0* (the level of a link is the number of ! boxes in which it is contained), noted $\rightarrow_0$, which is, roughly, the analogous of head reduction for proof nets[2]. The level 0 strategy does not compute cut-free proofs. As head reduction can be iterated on arguments, obtaining leftmost reduction (which computes normal forms), level 0 reduction can be iterated *by levels*, obtaining the *least level* strategy $\rightarrow_{ll}$ that first reduces all cuts at level 0, then those at level 1, and so on. Least level reduction computes cut-free proofs.

We shall need only the following two reduction rules for proof nets:

Promotion / contraction

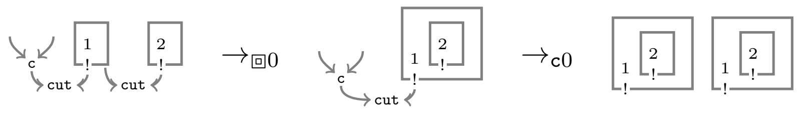


Promotion / auxiliary port



We shall use $\rightarrow_{\mathsf{c}0}$ and $\rightarrow_{\boxdot 0}$ for when these rules are applied to cuts at level 0.

Now, consider the following reduction sequence at level 0:



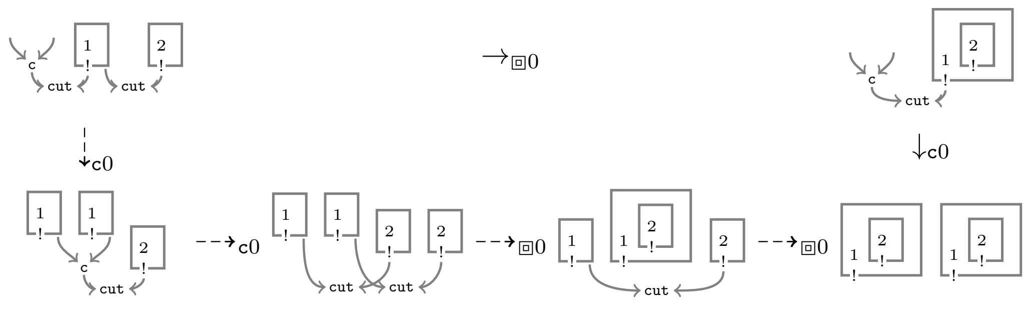
The second step duplicates a !-box which is not a sub-term of the initial proof net, breaking the sub-term property.

The example for linear logic is strictly weaker than for the $\lambda$-calculus, as it only shows that one specific strategy (which is however the one of reference in the literature) does

[2]All head reduction redexes are redexes at level 0, when $\lambda$-terms are translated to proof nets.

not have the sub-term property. It is interesting, however, because proof nets do not rely on meta-level substitution.

It is also instructive to see the diagram where one swaps the order of reduction of redexes of the given example:



The diagram can be closed by using only cut elimination at level 0—which is non-deterministic—and the alternative path does not break the sub-term property. Note however that the diagram is *not diamond* (in this respect, level 0 reduction is not as head reduction). This is problematic, because it means that the number of steps of cut elimination at level 0 is an ambiguous quantity, and then it cannot be used as a time cost model. One would then argue that only the sub-term preserving paths of the level 0 strategy should be considered. This is a possible approach, but the definition of such a sub-strategy is not evident, and it has never been studied.

## 5 Size Explosion

Here we show how the lack of the sub-term property leads to size explosion in the $\lambda$-calculus and in linear logic, that is to a family of terms (or proof nets) the size of which grows exponentially with the number of steps.

**(Open) Size Explosion in the $\lambda$-Calculus.** The simplest example of size explosion is a variation over the famous looping $\lambda$-term $\Omega := (\lambda x.xx)(\lambda x.xx) \to_\beta \Omega \to_\beta \ldots$. In $\Omega$ there is an infinite sequence of duplications. In the first size-exploding family that we see, there is a sequence of $n$ nested duplications. We define both the family $\{t_n\}_{n\in\mathbb{N}}$ of size-exploding terms and the family $\{u_n\}_{n\in\mathbb{N}}$ of results of the evaluation

$$\begin{array}{rcl@{\qquad}rcl} t_0 & := & y & u_0 & := & y \\ t_{n+1} & := & \delta t_n = (\lambda x.xx)t_n & u_{n+1} & := & u_n u_n \end{array}$$

We use $|t|$ for the size of a term, that is, its number of constructors, and say that a term is *neutral* if it is normal and it is not an abstraction. Moreover, we note $\to_{r\beta}$ the evaluation strategy that reduces the rightmost redex in a term, and use $\to_{r\beta}^n$ to denote $n$ steps of evaluation, with the convention that $\to_{r\beta}^0$ denotes the identity relation.

**Proposition 5.1** (Open and Rightmost Size Explosion). *Let $n \in \mathbb{N}$. Then $t_n \to_{r\beta}^n u_n$ and the i-th step makes two copies of $u_{i-1}$ (breaking the structural sub-term property from the second step on), moreover $|t_n| = \mathcal{O}(n)$, $|u_n| = \Omega(2^n)$, and $u_n$ is neutral.*

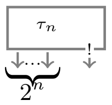
*Proof.* By induction on $n$. The base case $n = 0$ holds, because $t_0 = y = u_0$ and $\rightarrow^0_{r\beta}$ is the idenity. The inductive case: $t_{n+1} = (\lambda x.xx)t_n \rightarrow^n_{r\beta} (\lambda x.xx)u_n \rightarrow_{r\beta} u_n u_n = u_{n+1}$, where the first sequence is obtained by the *i.h.* and the last step is the rightmost one because $u_n$ is neutral. The bounds on the sizes are immediate. The fact that $u_{n+1} = u_n u_n$ is normal/neutral follows by the fact that $u_n$ is neutral. □

The statement of Proposition 5.1 stresses that, from $i \geq 2$, the $i$-th step of the sequence from $t_i$ duplicates $u_i$, which is not a sub-term of $t_i$, thus breaking the sub-term property.

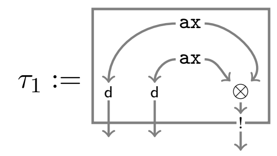
Let us clarify this important point. In the $\lambda$-calculus, a step $t \rightarrow_\beta u$ can only duplicate a sub-term of $t$, if it duplicates at all. Note that the sub-term property asks *something more*, namely that also *later* steps duplicate a sub-term of the initial one. This further requirement is not verified in the $\lambda$-calculus (as size explosion and counter-example (2) show), because by chaining duplications of what was duplicated at the previous step, one ends up duplicating terms of size *exponential* in the size of the initial term. Note that size explosion shows that, with sufficient iterations, the breaking of the structural sub-term property (duplicating a term that is not a sub-term of the initial term) leads to the breaking of the quantitative sub-term property (duplicating a term that is bigger than the initial term).

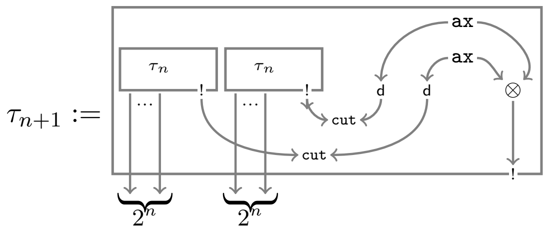
**Size Explosion in Linear Logic.** We are now going to show an example of size explosion for cut-elimination at level 0 in proof nets. It is an interesting fact by itself, but it also allows us to stress an important delicate point. Size explosion in the $\lambda$-calculus is partly due to the fact that $\beta$-reduction is based on meta-level substitution. We shall later see that decomposing $\beta$-reduction allows us to verify the sub-term property and avoid size explosion. The linear logic case is interesting because it does decompose $\beta$-reduction and yet still suffers of size explosion. Therefore, it proves that decomposing $\beta$ is not enough, one also needs to do it in the right way.

For the time being, let's consider untyped proof nets. We define various families of proof nets. The $n$-th net $\tau_n$ has the following shape:

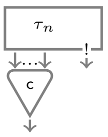
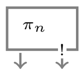


And it is defined as follows:

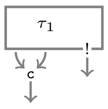
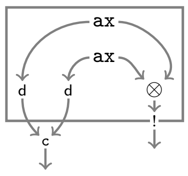




Then we define the family $\pi_n$, which is obtained from $\tau_n$ by contracting all its $2^n$ auxiliary conclusions via a tree of contractions:

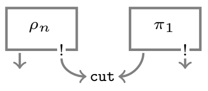
$\pi_n$ :=  also noted 

For instance, $\pi_1$ is the following net:

$\pi_1$ =  = 

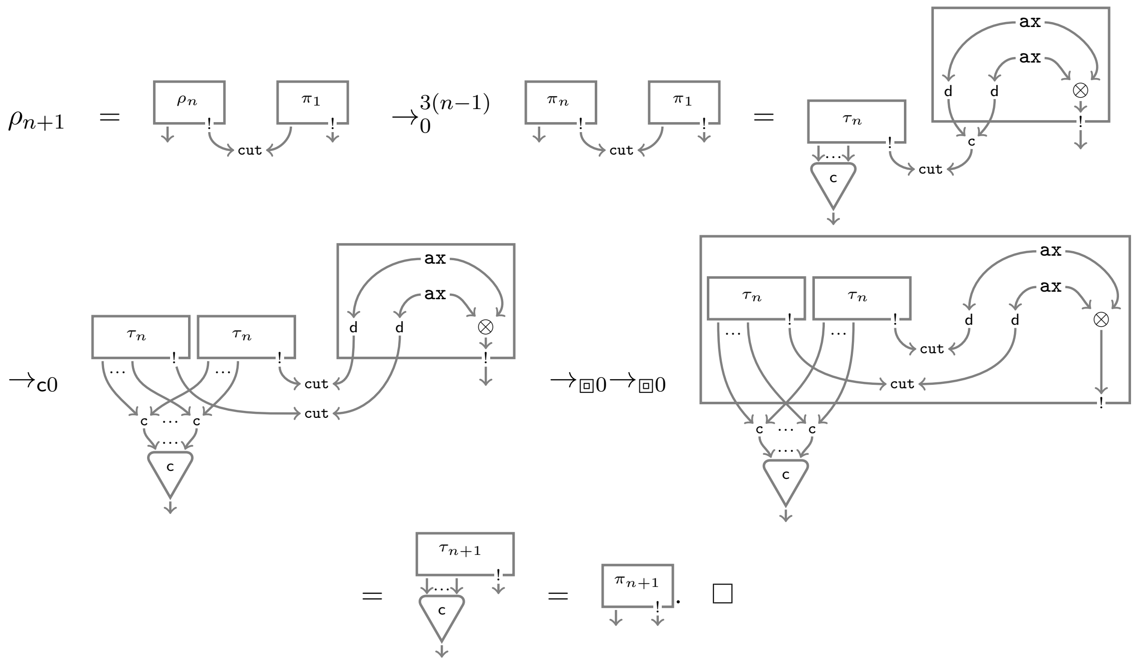
Finally, define the following family of nets:

$$\rho_1 \quad := \quad \pi_1$$

$\rho_{n+1}$ := 

**Lemma 5.2** (Size explosion). $\rho_n \rightarrow_0^{3(n-1)} \pi_n$ *and the* $3i$*-th step makes two copies of* $\tau_i$ *(thus breaking the sub-term property from the second step on),* $|\rho_n| = \mathcal{O}(n)$*, and* $|\pi_n| = \Omega(2^n)$*.*

*Proof.* By induction on $n$. The bounds on the sizes follow immediately from the definition and the *i.h.* For $n = 1$, we have $\rho_1 = \pi_1$, so the statement holds. For $n+1$, we have that by *i.h.* $\rho_n \rightarrow_0^{3(n-1)} \pi_n$. Therefore, we obtain:



. □

## 5.1 Size Explosion is Everywhere

Here we dissect size explosion in the $\lambda$-calculus. The aim is proving that all strategies suffer of size explosion.

**Explosiveness is not Stable Under Substitutions.** Let us starting by seeing what happens when the given open exploding family is closed by substituting the identity $\mathtt{I} := \lambda z.z$ on $y$. We consider a specific case, susbstituting $\mathtt{I}$ on $t_3$ and evaluate—as before—in a rightmost way:

$$\begin{array}{rcll} t_3\{y{\leftarrow}\mathtt{I}\} & = & \delta(\delta(\delta\mathtt{I})) & \\ & \to_{r\beta} & \delta(\delta(\mathtt{II})) & \\ & \to_{r\beta} & \delta(\delta(\mathtt{I})) & \\ & \to_{r\beta} & \delta(\mathtt{II}) & \\ & \to_{r\beta} & \delta\mathtt{I} & \\ & \to_{r\beta} & \mathtt{II} & \to_{r\beta} \ \mathtt{I} \end{array}$$

Note that the normal form is just the identity, that is, the size of the last term does not explode anymore (and the sub-term property is not broken). The reason is that replacing $y$ with the identity $\mathtt{I}$ turns applications of the form $yy$ into new $\beta$-redexes, which consume the size, removing the degeneracy. Of course, this fact holds for the whole family.

**Proposition 5.3.** *Let $n \in \mathbb{N}$. Then $t_n\{y{\leftarrow}\mathtt{I}\} \to_{r\beta}^{2n} \mathtt{I}$.*

*Proof.* By induction on $n$. The base case is immediate as $t_0\{y{\leftarrow}\mathtt{I}\} = u_0\{y{\leftarrow}\mathtt{I}\} = y\{y{\leftarrow}\mathtt{I}\} = \mathtt{I}$. The inductive case: $t_{n+1}\{y{\leftarrow}\mathtt{I}\} = \delta t_n\{y{\leftarrow}\mathtt{I}\} = \delta(t_n\{y{\leftarrow}\mathtt{I}\})$ then by *i.h.* $t_n\{y{\leftarrow}\mathtt{I}\} \to_{r\beta}^{2n} \mathtt{I}$, so that $\delta(t_n\{y{\leftarrow}\mathtt{I}\}) \to_{r\beta}^{2n} \delta\mathtt{I} \to_{r\beta} \mathtt{II} \to_{r\beta} \mathtt{I}$, that is, $t_{n+1}\{y{\leftarrow}\mathtt{I}\} \to_{r\beta}^{2(n+1)} \mathtt{I}$. □

**Closed Size Explosion.** The natural question then is: does size explosion depend on the use of open terms? The answer is no. Of course, it is enough to put the open family $\{t_n\}_{n\in\mathbb{N}}$ under the abstraction $\lambda y$ (that is considering the family $\{\lambda y.t_n\}_{n\in\mathbb{N}}$), that closes it, to obtain a closed size exploding family. But such a modified example explodes only under strong evaluation (that is, evaluation that goes under abstraction), while under weak evaluation the terms are normal. Let us then refine the question into: is there a family of *closed* terms that explodes under *weak* evaluation (using whatever weak evaluation strategy)? The answer is yes. The simplest case is obtained by tweaking the given open family, and still evaluate in a rightmost way.

Let $\pi := \lambda x.\lambda y.yxx$. Now, define:

$$\begin{array}{rclcrcl} p_0 & := & \mathtt{I} & \quad & q_0 & := & \mathtt{I} \\ p_{n+1} & := & \pi p_n & & q_{n+1} & := & \lambda y.yq_nq_n \end{array}$$

**Proposition 5.4** (Closed and Rightmost Size Explosion). *Let $n \in \mathbb{N}$. Then $p_n \to_{r\beta}^{n} q_n$, moreover $|p_n| = \mathcal{O}(n)$, $|q_n| = \Omega(2^n)$, and $q_n$ is normal.*

*Proof.* By induction on $n$. The base case $n = 0$ holds, because $p_0 = \mathtt{I} = q_0$ and $\rightarrow^0_{r\beta}$ is the idenity. The inductive case: $p_{n+1} = \pi p_n \rightarrow^n_{r\beta} \pi q_n = (\lambda x.\lambda y.yxx)q_n \rightarrow_{r\beta} \lambda y.yq_nq_n = q_{n+1}$, where the first sequence is obtained by the *i.h.* and the last step is the rightmost one because $q_n$ is normal by *i.h.* The bounds on the sizes are immediate. $\square$

Note that the family explodes with respect to rightmost weak evaluation also if one defines $\pi$ as $\lambda x.\lambda y.xx$. The difference is that this further family does no longer explode under strong evaluation.

**Strategy-Independent Size Explosion.** It is natural to wonder whether a different strategy would avoid size explosion. It is not difficult to see that if the open family $\{t_n\}_{n\in\mathbb{N}}$ is evaluated in a leftmost way, rather than according to the rightmost strategy, then evaluation takes an exponential number of steps. Let's see an example. Again we consider the term $t_3$ of the open exploding family above, but now we evaluate it using the leftmost strategy $\rightarrow_{lo}$. We obtain $t_3 \rightarrow^{2^3-1}_{lo} y^{2^3}$:

$$\begin{array}{rlrl} t_3 & = \ \delta(\delta(\delta y)) & \rightarrow_{lo} & \delta(\delta y)(\delta(\delta y)) \\ \rightarrow_{lo} & \delta y(\delta y)(\delta(\delta y)) & \rightarrow_{lo} & yy(\delta y)(\delta(\delta y)) \\ \rightarrow_{lo} & yy(yy)(\delta(\delta y)) & \rightarrow_{lo} & yy(yy)(\delta y(\delta y)) \\ \rightarrow_{lo} & yy(yy)(yy(\delta y)) & \rightarrow_{lo} & yy(yy)(yy(yy)) \end{array}$$

Note that size explosion disappeared, but for a different reason than when we substituted the identity $\mathtt{I}$ on $y$ above: now the result does have size exponential in $n$, but it is obtained in a number of steps *itself exponential* in $n$, and so there is no exponential gap between the number of $\beta$ steps and the size of the normal form.

Of course, this fact holds for the whole family $\{t_n\}_{n\in\mathbb{N}}$.

**Proposition 5.5.** *Let $n \in \mathbb{N}$. Then $t_n \rightarrow^{2^n-1}_{lo} u_n$.*

*Proof.* By induction on $n$. The base case is immediate as $t_0 = u_0 = y$ and $2^0-1 = 1-1 = 0$. The inductive case: $t_{n+1} = (\lambda x.xx)t_n \rightarrow_{lo} t_nt_n \rightarrow^{2^{n-1}-1}_{lo} u_nt_n \rightarrow^{2^{n-1}-1}_{lo} u_nu_n = u_{n+1}$, where the two $\rightarrow^{2^{n-1}-1}_{lo}$ sequences are obtained by the *i.h.* The number of steps for $t_n$ is then $2 \cdot (2^{n-1}-1)+1 = 2^n-2+1 = 2^n-1$, as required. $\square$

The new natural question is: is there a family that explodes with respect to leftmost evaluation? Again, the answer is yes. This time the tweak of the open family is a bit trickier, and the schema for defining the exploding family is slightly different. Define the following *pre-family*:

$$\begin{array}{rcl} s_1 & := & \delta = \lambda x_1.x_1x_1 \\ s_{n+1} & := & \lambda x_{n+1}.s_n(x_{n+1}x_{n+1}) \end{array}$$

The leftmost exploding family is actually given by $\{s_ny\}_{1\leq n\in\mathbb{N}}$ , that is, it is obtained by applying the term $s_n$ of the pre-family to the free variable $y$. The family of results is still $u_n$, as defined before for open size explosion (and such that $y = u_0$). The statement we are going to prove is also slightly different. It is more generally about $s_nu_m$ instead of just $s_ny$, in order to obtain a simple inductive proof.

**Proposition 5.6** (Open and Leftmost Size Explosion). *Let $n, m \in \mathbb{N}$ and $n > 0$. Then $s_n u_m \rightarrow_{lo}^n u_{n+m}$, moreover $|s_n| = \mathcal{O}(n)$.*

*Proof.* By induction on $n$. The base case $n = 1$ holds $s_1 u_m = (\lambda x_1.x_1 x_1)u_m \rightarrow_{lo} u_m u_m = u_{m+1}$. The inductive case: $s_{n+1} u_m = (\lambda x_{n+1}.s_n(x_{n+1}x_{n+1}))u_m \rightarrow_{lo} s_n(u_m u_m) = s_n u_{m+1} \rightarrow_{lo}^n u_{n+m+1}$, where the last sub-sequence is obtained by the *i.h.*, and the global number of $\rightarrow_{lo}$ steps is $n+1$, as required. The bound on the size is immediate. □

Note that the given family explodes also under strong evaluation that is not leftmost. Indeed, all evaluations of the family—independently of the strategy—have the same length, which is why we call it *strategy-independent.*

**Closed and Strategy-Independent Size Explosion.** The ideas for the closed family and for the strategy-independent family can be combined, obtaining that *size explosion affects every strategy of the $\lambda$-calculus*, in every setting.

Define the pre-family $\{t_n\}_{1 \leq n \in \mathbb{N}}$ and the family of results $\{s\}_{n \in \mathbb{N}}$ as follows:

$$t_1 := \lambda x.\lambda y.yxx \qquad t_{n+1} := \lambda x.t_n(\lambda y.yxx) \;\Big|\; s_0 := \mathtt{I} \qquad s_{n+1} := \lambda y.y s_n s_n$$

**Proposition 5.7** (Closed and strategy-independent size explosion). *Let $n > 0$. Then $t_n \mathtt{I} \rightarrow_{\beta}^n s_n$. Moreover, $|t_n \mathtt{I}| = \mathcal{O}(n)$, $|s_n| = \Omega(2^n)$, $t_n \mathtt{I}$ is closed, and $s_n$ is normal.*

It is also easily seen that this family is typable with simple types—do the exercise. What can you say about the size of the type?

**Number of Copies.** Let us show a final observation about size explosion. Consider the third element $t_3\mathtt{I}$ of the last family, and let's reduce it in a *innermost way*, exploiting the annotation $\tau_t := \lambda y.ytt$, so that $t_1 = \lambda x.\tau_x$ and $t_{n+1} = \lambda x.t_n \tau_x$:

$$\begin{array}{rcl} t_3\mathtt{I} & = & (\lambda x_3.(\lambda x_2.(\lambda x_1.\tau_{x_1})\tau_{x_2})\tau_{x_3})\mathtt{I} \\ & \rightarrow_{\beta} & (\lambda x_3.(\lambda x_2.\tau_{\tau_{x_2}}))\tau_{x_3})\mathtt{I} \\ & \rightarrow_{\beta} & (\lambda x_3.\tau_{\tau_{\tau_{x_3}}})\mathtt{I} \\ & \rightarrow_{\beta} & \tau_{\tau_{\tau_{\mathtt{I}}}} \end{array} \tag{3}$$

Note that the first two steps duplicate a renaming of $\tau_x$ and that the last step duplicates $\mathtt{I}$: it looks like the structural sub-term property is not broken. Here it plays a role the final clause in the definition of the property:

*In the case of duplications, the number of copies is also bound by $|t_0|$.*

Indeed in (3) it is the number of copies of the duplicated sub-term that grows exponentially ($\tau_{\tau_{\tau_{x_3}}}$ has eight occurrences of $x_3$), thus still breaking the sub-term property.

**Summing Up.** The last family we considered is such that:

- *Closed*: it is closed;
- *Strategy independent*: weak evaluation and all the possible strong evaluations of the family have the same length, and they all lead to the explosion;
- *Typable*: it is even typable with simple types.
- *Living in* $\Lambda_{\mathtt{det}}$: it is a term of the restricted deterministic $\lambda$-calculus used to encode TMs.

Therefore, every minimally expressive dialect of the $\lambda$-calculus suffers of size explosion. As a slogan, *size explosion is everywhere.*

Let us point out that, while $\Lambda_{\mathtt{det}}$ admits size explosion, the size of the terms of $\Lambda_{\mathtt{det}}$ encoding TMs never explode.

## 6 Circumventing Size Explosion Using Sharing

**Hidden Assumption about Space.** All dialects of $\lambda$-calculus suffer from size explosion. And size explosion seems to imply that the number of $\beta$-steps cannot be a reasonable time cost model: the number of steps does not even account for the time to write down the result, which is exponentially bigger. While this is the natural way to read size explosion, this is also somewhat the wrong conclusion. In fact, the number of $\beta$-steps *is* a reasonable time cost model, which means that there is a hidden wrong hypothesis in the natural reading. Let us focus on such a wrong hypothesis.

Essentially one is assuming that the underlying notion of space for the $\lambda$-calculus is given by *ink space*, that is, the maximum size of terms during evaluation. Since one is assuming that time and space are locked (that is, that you need time to use space), the time cost of the size exploding family must be at least exponential. The somewhat wrong hypothesis then is the adoption of ink space as the reference notion of space.

The number of $\beta$-steps of some strategies can be taken as a reasonable cost model by switching to *low-level space.* The idea is to evaluate *up to sharing*, where sharing is used to both retrieve the sub-term property, and thus have reasonable steps, and keep the representation of $\lambda$-terms compact, avoiding size explosion. The idea is to simulate small-step $\beta$-reduction in a refined system where $\beta$-reduction is decomposed in micro-steps and simulated using some form of sharing of sub-terms. The micro-step formalism can be a $\lambda$-calculus with explicit substitutions (as we shall do here), an abstract machine (as we shall do in the next lecture), or some graph-rewriting mechanism (such as proof nets). These approaches are all based on the following ideas:

1. *Smaller terms*: space is rather the maximum size of $\lambda$-terms *with sharing* during *micro-steps*, shared evaluation.
2. *Compact normal forms*: evaluation in the micro-step formalism stops before reaching the normal form, on a compact representation of it.

3. *Reasonable steps*: micro-step strategies can have the sub-term property, thus providing a notion of reasonable low-level cost model.

4. *From low-level to abstract time*: micro-step strategies simulate small-step strategies within a polynomial overhead.

As we said, the details of the micro-step formalism do not matter, as the same recipe can be adapted to explicit substitutions, abstract machine, and graph-based formalism. This is however only partly true, because it is a quite delicate recipe, and not every choice of micro-step formalism and micro-step strategy does work. For instance, we have shown that if proof nets are the formalism of choice, then the level 0 or least level strategies do not have the sub-term property, and thus cannot be used for the recipe. Therefore, some other evaluation strategies for proof nets are needed. We shall not have the time to discuss them, unfortunately.

**Micro(-Step) Weak Head Reduction.** To give substance to the discussion, we now give an example of such recipe by defining a sharing-based, micro-step version of weak head reduction. It is a strategy of the *linear substitution calculus*, a $\lambda$-calculus with explicit substitution (shortened to ES) which, as a calculus, shall be properly introduced and studied in a following lecture. For the moment, we only look at the strategy. We shall also state some properties that are going to be proved in the next lectures.

The language of terms is simply the one of terms with ES.

$$\textsc{Terms with sharing} \quad t, u, s, p \quad ::= \quad x \mid \lambda x.t \mid tu \mid t[x{\leftarrow}u]$$

Which can be related to $\lambda$-terms without ES via the notion of unfolding.

$$\begin{array}{c}\textsc{Unfolding}\\ \begin{array}{rclcrcl} x{\downarrow} & := & x & \qquad & (\lambda x.t){\downarrow} & := & \lambda x.t{\downarrow} \\ (t\,u){\downarrow} & := & t{\downarrow}\,u{\downarrow} & & t[x{\leftarrow}u]{\downarrow} & := & t{\downarrow}\{x{\leftarrow}u{\downarrow}\} \end{array}\end{array}$$

In order to define the strategy, we define two notions of contexts, weak head contexts and substitution contexts. Moreover, we redefine general contexts in presence of ES.

$$\begin{array}{rrcl} \textsc{General Contexts} & C & ::= & \langle\cdot\rangle \mid \lambda x.C \mid Ct \mid tC \mid C[x{\leftarrow}u] \mid t[x{\leftarrow}C] \\ \textsc{Substitution Contexts} & S & ::= & \langle\cdot\rangle \mid S[x{\leftarrow}u] \\ \textsc{Weak Head Contexts} & E & ::= & \langle\cdot\rangle \mid Et \mid E[x{\leftarrow}u] \end{array}$$

The rewriting rules use contexts to define both the root rules and their context closures.

$$\begin{array}{c}\textsc{Root rules}\\ \begin{array}{rccc} \textsc{Micro(-step)}\ \beta & S\langle\lambda x.t\rangle u & \mapsto_{m\beta} & S\langle t[x{\leftarrow}u]\rangle \\ \textsc{Micro(-step) substitution} & E\langle x\rangle[x{\leftarrow}u] & \mapsto_{msub} & E\langle u\rangle[x{\leftarrow}u] \end{array}\end{array}$$

$$\begin{array}{c}\textsc{Contextual closure}\\ E\langle t\rangle \rightarrow_{m\beta} E\langle u\rangle \quad \text{if } t \mapsto_{m\beta} u \\ E\langle t\rangle \rightarrow_{msub} E\langle u\rangle \quad \text{if } t \mapsto_{msub} u \end{array}$$

$$\textsc{Notation} \quad \rightarrow_{mwh} \quad := \quad \rightarrow_{m\beta} \cup \rightarrow_{msub}$$

The use of the substitution context $S$ in the micro $\beta$ rule is a compact notation for steps such as the following one:

$$(\lambda x.t)[x_1{\leftarrow}s_1]\dots[x_n{\leftarrow}s_n]u \quad\to\quad t[x{\leftarrow}u][x_1{\leftarrow}s_1]\dots[x_n{\leftarrow}s_n]$$

where the list of substitutions $[x_1{\leftarrow}s_1]\dots[x_n{\leftarrow}s_n]$ is seen as a substitution context $S = \langle\cdot\rangle[x_1{\leftarrow}s_1]\dots[x_n{\leftarrow}s_n]$.

Similarly, the micro substitution rule replaces a variable occurrence which is possibly far, in the term structure, from the acting ES. For instance:

$$(x(\lambda y.yx))[z{\leftarrow}t]u[x{\leftarrow}s] \quad\to_{msub}\quad (s(\lambda y.yx))[z{\leftarrow}t]u[x{\leftarrow}s]$$

Note also that it replaces only the weak head occurrence of $x$ leaving the other potential occurrences untouched.

There is no rule for erasing ESs. The linear substitution calculus has such a rule, but here it is not needed—we shall see it in a following lecture.

Micro weak head reduction work modulo $\alpha$-renaming, which is performed on-the-fly in the $\to_{msub}$ rule, to avoid capture. For instance.

$$(\lambda x.yx)[y{\leftarrow}xz] \quad\to_{msub}\quad (\lambda w.xzw)[y{\leftarrow}xz]$$

**Proposition 6.1.** *Micro weak head reduction $\to_{mwh}$ is deterministic.*

*Proof.* Let $t \to_{mwh} t_1$ and $t \to_{mwh} t_2$. We prove $t_1 = t_2$ by induction on $t$. Cases of $t$:

- *Variable* or *abstraction*. Impossible, because then $t$ cannot reduce.
- *Application*, that is, $t = us$. Sub-cases:
  - $u$ has shape $S\langle\lambda x.u'\rangle$. Then there is only one redex from $t$, namely $t = S\langle\lambda x.u'\rangle s \to_{m\beta} S\langle u'[x{\leftarrow}s]\rangle$.
  - $u \neq S\langle\lambda x.u'\rangle$. Then the two steps from $t$ come from two steps $u \to_{mwh} u_1$ and $u \to_{mwh} u_2$ with $t_1 = u_1 s$ and $t_2 = u_2 s$. By *i.h.*, $u_1 = u_2$, and so $t_1 = t_2$.
- *Substitution*, that is, $t = u[x{\leftarrow}s]$. Sub-cases:
  - $u$ has shape $E\langle x\rangle$. Then there is only one redex from $t$, namely $t = E\langle x\rangle[x{\leftarrow}s] \to_{msub} E\langle s\rangle[x{\leftarrow}s]$, because $t$ can be written as $E\langle x\rangle$ in one way only.
  - $u \neq E\langle x\rangle$. Then the steps from $t$ come from two steps $u \to_{mwh} u_1$ and $u \to_{mwh} u_2$ with $t_1 = u_1[x{\leftarrow}s]$ and $t_2 = u_2[x{\leftarrow}s]$. By *i.h.*, $u_1 = u_2$, and so $t_1 = t_2$. □

The next theorem says that micro weak head reduction implements weak head reduction up to unfolding, and that the number of $\to_{wh}$ head steps (noted $|e|$ for a $\to_{wh}$ evaluation sequence $e$) is exactly the number of $\to_{m\beta}$ steps (noted $|e|_{m\beta}$ for a $\to_{mwh}$ evaluation sequence $e$). The proof is omitted. We shall see in the next lecture how to prove the theorem.

**Theorem 6.2** (Implementation)**.** *Let $t$ be a $\lambda$-term without ES.*

1. *If $e : t \to_{mwh}^* u$ then $e{\downarrow} : t{\downarrow} \to_{wh}^* u{\downarrow}$ with $|e|_{m\beta} = |e{\downarrow}|$.*
2. *If $e : t \to_{wh}^* u$ then there exists $e' : t \to_{mwh}^* s$ with $s{\downarrow} = u$ and $|e| = |e'|_{m\beta}$.*

**Sub-Term Property.** Weak head reduction does not have the sub-term property. The main reason to study its micro-step refinement is that it has the sub-term property. It can be proved via a simple invariant. We need a definition.

**Definition 6.3** (Box sub-terms)**.** *Box contexts are given by the following grammar:*

$$\text{Box Contexts} \quad B \quad ::= \quad t\langle\cdot\rangle \mid t[x{\leftarrow}\langle\cdot\rangle] \mid C\langle B\rangle$$

*The box sub-terms of a term with ES $t$ are the sub-terms $u$ of $t$ in a box contexts, that is, such that $t = B\langle u\rangle$ for a box context $B$.*

Note that the box sub-terms of a term $t$ are exactly those sub-terms that end up in a !-box when $t$ is translated to linear logic proof nets, and so are exactly the sub-terms that can be duplicated.

**Lemma 6.4** (Local sub-term invariant)**.** *Let $t \rightarrow_{mwh} u$. Then the box sub-terms of $u$ are box sub-terms of $t$ up to $\alpha$-renaming.*

*Proof.* Cases of $t \rightarrow_{mwh} u$:

- *Micro $\beta$*, that is, $t = E\langle S\langle \lambda x.s\rangle p\rangle \rightarrow_{m\beta} E\langle S\langle s[x{\leftarrow}p]\rangle\rangle = u$. Note that $p$ is a box sub-term of $t$. All others box sub-terms of $u$ are in $E$, $S$, $s$ or $p$, and thus are also box sub-terms of $t$.
- *Micro substitution*, that is, $t = E\langle F\langle x\rangle[x{\leftarrow}s]\rangle \mapsto_{msub} E\langle F\langle s\rangle[x{\leftarrow}s]\rangle = u$. The box sub-terms inside the newly made copy of $s$ are traced back to the corresponding one in the ES copy of $s$ which is also in $u$, and all the others box sub-terms are evidently also box sub-terms in $u$. Here it is where some $\alpha$-renaming might take place in $E$. □

**Proposition 6.5** (Sub-term property)**.** *Let $t \rightarrow^*_{mwh} u$.*

1. *Let $s$ be a box sub-term of $u$. Then $|s| \leq |t|$.*
2. *Micro weak head reduction has the sub-term property.*

*Proof.*

1. By induction on the length $n$ of the sequence $t \rightarrow^*_{mwh} u$. If $n = 0$ then the statement holds. If $n > 0$ then let $p \rightarrow_{mwh} u$ be the last step of the sequence. By the local sub-term invariant (Lemma 6.4), the size of every box sub-term $s$ of $u$ is bound by the size of a box sub-term of $p$, which by *i.h.* satisfy the statement.
2. Only rule $\rightarrow_{msub}$ duplicates, and it does duplicate only box sub-terms, making one copy at a time. □

It is interesting to see how the counter-example (2) (page 8) to the sub-term property in the $\lambda$-calculus is evaluated with $\rightarrow_{mwh}$ (we recall that $\tau_t := \lambda y.ytt$):

$$\begin{array}{lll}(\lambda x.x(\lambda z.\tau_z)\tau_x)\mathtt{I} & \to_{m\beta} & (x(\lambda z.\tau_z)\tau_x)[x{\leftarrow}\mathtt{I}] \\ & \to_{msub} & (\mathtt{I}(\lambda z.\tau_z)\tau_x)[x{\leftarrow}\mathtt{I}] \\ & \to_{m\beta} & (y[y{\leftarrow}\lambda z.\tau_z]\tau_x)[x{\leftarrow}\mathtt{I}] \\ & \to_{msub} & ((\lambda z.\tau_z)[y{\leftarrow}\lambda z.\tau_z]\tau_x)[x{\leftarrow}\mathtt{I}] \\ & \to_{m\beta} & \tau_z[z{\leftarrow}\tau_x][y{\leftarrow}\lambda z.\tau_z][x{\leftarrow}\mathtt{I}] \\ & = & (\lambda y.yzz)[z{\leftarrow}\tau_x][y{\leftarrow}\lambda z.\tau_z][x{\leftarrow}\mathtt{I}]\end{array}$$

The last term is $\to_{mwh}$ normal and unfolds to the $\to_{wh}$ result $\lambda y.y\tau_{\mathtt{I}}\tau_{\mathtt{I}}$. Note that the sub-term $\tau_{\mathtt{I}}$ which breaks the sub-term property for $\to_{wh}$ is never duplicated by $\to_{mwh}$. The example also shows that $\to_{mwh}$ reductions are longer than $\to_{wh}$ reductions.

**Proposition 6.6** ($\to_{mwh}$ is reasonably implementable)**.** *Micro weak head reduction $\to_{mwh}$ has reasonable steps, and so it can be reasonably implemented on RAMs.*

*Proof.* Because of the sub-term property of $\to_{mwh}$, we are left to show that, in a sequence $t_0 \to^*_{mwh} t_n$, searching for the $\to_{mwh}$ redex in $t_i$ can be done in time polynomial in $|t_0|$ and $i$. This is fairly obvious because of the next two observations:

- Searching for the $\to_{mwh}$ redex in term $t_i$ can be done in time polynomial in $|t_i|$: it is enough to always go left on applications and ES until one finds a variable $x$ or an abstraction, and then go back up towards to root constructor to find the argument or the ES on $x$, if any.
- The size of $t_i$ is polynomial in $|t_0|$ and $i$ because of the sub-term property. □

**Back to Size Explosion.** Let's evaluate with micro weak head reduction $\to_{mwh}$ the term $s_3y$ of the left-to-right open size exploding family of Proposition 5.6.

$$s_1 \;:=\; \delta = \lambda x_1.x_1x_1 \qquad s_{n+1} \;:=\; \lambda x_{n+1}.s_n(x_{n+1}x_{n+1})$$

$$\begin{array}{lll}s_3y & = & (\lambda x_3.s_2(x_3x_3))\,y \\ & \to_{m\beta} & (s_2(x_3x_3))[x_3{\leftarrow}y] \\ & = & ((\lambda x_2.s_1(x_2x_2))(x_3x_3))[x_3{\leftarrow}y] \\ & \to_{m\beta} & (s_1(x_2x_2))[x_2{\leftarrow}x_3x_3][x_3{\leftarrow}y] \\ & = & (\lambda x_1.x_1x_1)(x_2x_2)[x_2{\leftarrow}x_3x_3][x_3{\leftarrow}y] \\ & \to_{m\beta} & (x_1x_1)[x_1{\leftarrow}x_2x_2][x_2{\leftarrow}x_3x_3][x_3{\leftarrow}y] \\ & \to_{msub} & (x_2x_2x_1)[x_1{\leftarrow}x_2x_2][x_2{\leftarrow}x_3x_3][x_3{\leftarrow}y] \\ & \to_{msub} & (x_3x_3x_2x_1)[x_1{\leftarrow}x_2x_2][x_2{\leftarrow}x_3x_3][x_3{\leftarrow}y] \\ & \to_{msub} & (yx_3x_2x_1)[x_1{\leftarrow}x_2x_2][x_2{\leftarrow}x_3x_3][x_3{\leftarrow}y]\end{array}$$

It is easily seen then that the shape of the result of executing $s_ny$ with $\to_{mwh}$ is:

$$yx_n\ldots x_1[x_1{\leftarrow}x_2x_2]\ldots[x_{n-1}{\leftarrow}x_nx_n][x_n{\leftarrow}y] \tag{4}$$

Note that the size of this final term is *linear* in $n$, that is, the evaluation of $s_ny$ does not explode, as expected, thanks to the sub-term property. On the other hand, unfolding the final term recursively duplicates all the $x_ix_i$ in the ESs, producing the final terms $u_n$ of size *exponential* in $n$ of the size exploding family. Therefore, the terms in (4) are compact representations of the terms $u_n$.

## 7 From Low-Level Time to Abstract Time.

Proposition 6.6 proves that the low-level time cost model given by micro weak head reduction is simulated reasonably by RAMs, and it is an instance of how sharing allows to circumvent size explosion. Therefore, until now we have developed the following diagram (the RAM/TM arrow is a basic fact from the literature, and the TM/$\Lambda$ arrow is given by Theorem 2.2, $\Lambda$ is the set of $\lambda$-terms without ES, and $\Lambda_{\mathtt{sh}}$ is the set of $\lambda$-terms with ES):

$$\begin{tikzcd}
(\Lambda, \to_{wh}) \arrow[d, "?"'] & & \mathrm{TM} \arrow[ll, "linear"'] \\
(\Lambda_{\mathtt{sh}}, \to_{mwh}) \arrow[rr, "polynomial"'] & & \mathrm{RAM} \arrow[u, "quadratic"']
\end{tikzcd} \tag{5}$$

The implementation theorem (Theorem 6.2) tells us that $\to_{mwh}$ implements $\to_{wh}$, but it does not tell us the overhead. It does tell us, however, that the number of non-shared step coincides with the number of $\to_{m\beta}$ steps. Therefore, $\to_{mwh}$ reductions are always at least as long as $\to_{wh}$ reductions. How longer can they be? To close the diagram we need to bound how many linear substitution steps there can be in simulating $\to_{wh}$ with $\to_{mwh}$. We shall see in the next lecture that the bound is quadratic. One then obtains the following diagram.

$$\begin{tikzcd}
(\Lambda, \to_{wh}) \arrow[d, "quadratic"'] & & \mathrm{TM} \arrow[ll, "linear"'] \\
(\Lambda_{\mathtt{sh}}, \to_{mwh}) \arrow[rr, "polynomial"'] & & \mathrm{RAM} \arrow[u, "quadratic"']
\end{tikzcd} \tag{6}$$

Summing up, we provided a simple instance of how to prove that *abstract time* is reasonable by means of a reasonable *low-level time* cost model. All instances in the literature follow a similar schema. Given a strategy $\to_{\mathtt{str}}$ on $\lambda$-terms and its associated low-level refinement $\to_{\mathtt{sh}X}$ in a system with sharing $S$, the proof that $\to_{\mathtt{str}}$ provides a reasonable notion of abstract time is obtained by studying the arrows with the question mark in the following diagram, and proving that they are polynomial in the size of the initial term $t$ and of the number $n$ of $\beta$-steps taken by $\to_{\mathtt{str}}$.

$$\begin{tikzcd}
(\Lambda, \to_{\mathtt{str}}) \arrow[d, "?"'] & & \mathrm{TM} \arrow[ll, "linear"'] \\
(S, \to_{\mathtt{sh}X}) \arrow[rr, "?"'] & & \mathrm{RAM} \arrow[u, "quadratic"']
\end{tikzcd} \tag{7}$$

The polynomiality of the bottom arrow is usually obtained via the sub-term property of $\to_{\mathtt{sh}X}$. For the left arrow, it is obtained via a detailed study of how $\to_{\mathtt{sh}X}$ implements $\to_{\mathtt{str}}$. If both simulations have polynomial overhead, one says that $\to_{\mathtt{str}}$ is *reasonably implementable*, despite it suffering from size explosion (every strategy of the $\lambda$-calculus does suffer from it), as it is meant to be implemented via $\to_{\mathtt{sh}X}$, which circumvents size explosion because of the sub-term property. The main consequence of such a schema is that the number $\to_{\mathtt{str}}$ steps then becomes a reasonable complexity measure—essentially the complexity class P defined via $\to_{\mathtt{str}}$ coincides with the one defined by RAM or Turing machines.

Let us stress the subtle key point here: $\to_{\mathtt{str}}$ is taken as a *specification* framework, the *reasonable execution* of which is actually done via $\to_{\mathtt{sh}X}$. The important aspect is that, once it is proved that $\to_{\mathtt{str}}$ is reasonably implementable, which requires $\to_{\mathtt{sh}X}$, then one can reason about time directly with $\to_{\mathtt{str}}$, by just counting $\to_{\mathtt{str}}$ steps, and forgetting about implementation details, which are encapsulated in $\to_{\mathtt{sh}X}$ and needed only for the proof of reasonability.

**Reasonable Sharing.** The key point underlying the diagram in (7), is the fact that the potential exponential growth of terms is isolated in the process of *unsharing*—also called *sharing unfolding*—of normal forms up to sharing. To be sure that one is not just sweeping the problem under the carpet, the adopted notion of sharing has to be reasonable. Typically, terms with sharing must be comparable for equality of their unfolding—a problem called *sharing equality*—in reasonable (that is, polynomial) time, without having to unfold the sharing (that would re-introduce an exponential blow-up). This fact is independent of the strategy and it has to be proven only once (for sharing as explicit substitutions or let expressions)—it has been done for the first time by Accattoli and Dal Lago [ADL12], where sharing equality is shown to be testable in time *quadratic* in the sizes of the terms with sharing. The test can actually be done in *linear* time, as shown by Condoluci, Accattoli, and Sacerdoti Coen [CASC19].

Let us be very clear about this key point. If the unfolded normal form of a term has to be printed then there is no way out: unfolding—and therefore printing—may re-introduce an exponential cost, even if the evaluation process has been carefully done with sharing as to implement it reasonably. Printing normal forms however is not how most evaluations end. If a normal form is used at all, it is often in an equality test, and the results about sharing equality show that it can be tested efficiently without unfolding. Then, one can claim that size explosion has finally been tamed.

## 8 Complements

**Sharing and Reasonable Strategies.** The kind of sharing at work in diagram (7), and therefore the definitions of $\to_{\mathtt{sh}X}$, depends very much on the strategy $\to_{\mathtt{str}}$.

The first result for weak strategies is due to Blelloch and Greiner in 1995 [BG95] and concerns weak call-by-value (shortened to CbV) evaluation. Similar results were then proved for weak call-by-name (CbN)—which is just another name for weak head reduction—and weak call-by-need (CbNeed). These results actually deal with weak evaluation *plus* the restriction that terms are closed. We call such a setting the *closed* $\lambda$-calculus. For the closed $\lambda$-calculus adopting sub-term sharing, as shown, and as it is done is most abstract machines in the literature via *environments*, is enough to obtain a reasonable implementation.

The strong variants of CbN (which is leftmost reduction), CbV, and CbNeed are also reasonable strategies. The first result is due to Accattoli and Dal Lago [ADL14] (2014), for strong CbN. They showed that the strong case is inherently harder than the weak case, as a more sophisticated notion of sharing, deemed *useful sharing*, is required—it is sketched in the next paragraph.

In the literature, there also is an example of an unreasonable strategy. Asperti and Mairson in [AM98] (1998) proved that Lévy's parallel optimal evaluation [Lév78] is unreasonable.

**Useful Sharing.** Going beyond the closed $\lambda$-calculus, can be done in two increasingly liberal ways: admitting open terms (open case) and evaluating under abstraction[3] (strong case). In these more liberal cases, sub-term sharing is no longer enough, as size explosion comes back with some more nasty tricks, typically disguised as *length explosion*: even when the refinement $\to_{\mathtt{sh}X}$ has reasonable steps, it may now happen that the simulation of $\to_{\mathtt{str}}$ by $\to_{\mathtt{sh}X}$ requires an *exponential* number of $\to_{\mathtt{sh}X}$ steps, reintroducing an exponential cost.

A close inspection of the phenomenon shows that this can be avoided by dividing shared substitution steps in two categories, *useful* and *useless* ones, and retaining useful steps while forbidding the useless ones.

The idea is to restrict to *minimal* unsharing work. We now sketch it using an intuitive approach similar to micro weak head reduction. A micro substitution steps is *useful* if it contributes somehow to the creation of $\beta$-redexes, as in the following case:

$$(xy)[x{\leftarrow}\mathtt{I}] \to (\mathtt{I}y)[x{\leftarrow}\mathtt{I}]$$

The $\beta$-redex $\mathtt{I}y$ has been created by the substitution of $\mathtt{I}$, which is then useful. A micro substitution step is instead *useless* if it only makes the term size *grow*, without creating $\beta$-redexes, as in this case:

$$(xy)[y{\leftarrow}\mathtt{I}] \to (x\mathtt{I})[x{\leftarrow}\mathtt{I}]$$

Useful sharing amounts to forbid such useless steps. While the idea is simple, its formalization is not so simple. First of all, note that being useful or useless is *not* a property of the ES, rather it is a property of *single variable replacements*. For instance, $(xx)[x{\leftarrow}\mathtt{I}]$ has both a *useful* and a *useless* micro step. Therefore, usefulness cannot be defined at the level of the $\lambda$-calculus but only at the micro level of refinements with sharing.

Additionally, some micro substitution steps are useful only in a *relative* way. Consider for instance the following micro step:

$$(xy)[x{\leftarrow}z][z{\leftarrow}\mathtt{I}] \to (zy)[x{\leftarrow}z][z{\leftarrow}\mathtt{I}] \qquad (8)$$

Is it a useful step? One would say no, as it does not create a $\beta$-redex. It is a necessary step, however, in order to perform the following step

$$(zy)[x{\leftarrow}z][z{\leftarrow}\mathtt{I}] \to (\mathtt{I}y)[x{\leftarrow}z][z{\leftarrow}\mathtt{I}]$$

[3]Evaluating under abstraction subsumes the open case. Indeed, if $\lambda x.t$ is closed, evaluating strongly means evaluating under $\lambda x$, that is, evaluating $t$, which may be open.

That does create a $\beta$-redex. Therefore, one has to consider the step in (8) as useful, even if the $\beta$-steps that it contributes to create is in fact created only *later on*.

These observations are meant to give a taste of why useful sharing is needed, of what it is, and why it is involved. We are not going to see useful sharing in detail, however, as it is too technical for the this short course.

**Reasonable *vs* Efficient.** Let us stress that *reasonable* does not mean *efficient*: CbN and CbV are incomparable for efficiency, and CbNeed is more efficient than CbN, and yet they are all reasonable. *Reasonable* and *efficient* are indeed unrelated properties of strategies. Roughly, efficiency is a *comparative* property, it makes sense only if there are many strategies and one aims at comparing them. Being reasonable instead is a property of the strategy *itself*, independently of any other strategy, and it boils down to the fact that the strategy can be implemented with a negligible overhead. Actually, if a strategy is too smart, then it risks to be unreasonable. It is essentially what happens with Lévy's parallel optimal evaluation. It is unreasonable, because implementing optimal steps requires an overhead which is more than polynomial—namely, a tower of exponentials in the worst case. Be careful again: unreasonable does not mean inefficient (but it does not mean efficient either).

**Being Reasonable Helps with Efficiency.** Despite the orthogonality of the properties of *being reasonable* and *being efficient*, the study of reasonable cost models does help in the study of efficiency. Given two strategies, if they are reasonable then it possible to compare them for efficiency by simply comparing the number of steps that they take on the same term. This is possible because, roughly, being reasonable means that one can *count 1* for each $\beta$-steps, since the overhead of implementing that $\beta$-step is negligible. If one or both the given strategies are not reasonable, instead, there is no easy way of comparing them for efficiency. In particular, comparing the number of steps can be completely misleading: just think of Lévy's optimal strategy, for which 1 step may actually count as a tower of exponentials.

**An Anecdote.** To give an idea of the subtlety but also of how much these questions are neglected by the community, let us report an anecdote. In 2011, we attended a talk where the speaker started by motivating the study of strong evaluation as follows. There exists a family $\{u_n\}_{n\in\mathbb{N}}$ of terms such that $u_n$ evaluates in $\Omega(2^n)$ steps with any weak strategy while it takes $\mathcal{O}(n)$ steps with a certain exotic strong strategy, namely rightmost-innermost. Thus, the speaker concluded, strong evaluation can be *more efficient* than weak evaluation, and it is worth studying it.

Such a reasoning is *wrong* (but the conclusion is correct, it is worth studying strong evaluation!). It is based on the hidden assumption that it makes sense to count the number of $\beta$-steps and compare strategies accordingly. As just explained, such an assumption is valid *only if* the compared strategies are reasonable, that is, if it is proved that their number of steps is a reasonable time measure. In the talk, the speaker was comparing weak strategies, of which we have various reasonable examples, together with a strong strategy that is not known to be reasonable. In particular, rightmost-innermost evaluation is probably unreasonable, given that, even when decomposed with sharing, it lacks the sub-term property, which is used in all proofs of reasonability in the literature.

That talk was given in front of an impressive audience, including many big names and historical figures of the $\lambda$-calculus. And yet no one noticed that identifying the number of $\beta$ steps with the actual cost was naïve and improper.

## References

[ADL12] Beniamino Accattoli and Ugo Dal Lago. On the invariance of the unitary cost model for head reduction. In Ashish Tiwari, editor, *23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*, volume 15 of *LIPIcs*, pages 22–37. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[ADL14] Beniamino Accattoli and Ugo Dal Lago. Beta Reduction is Invariant, Indeed. In *Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (CSL-LICS '14)*, pages 8:1–8:10, 2014.

[AM98] Andrea Asperti and Harry G. Mairson. Parallel beta reduction is not elementary recursive. In David B. MacQueen and Luca Cardelli, editors, *POPL '98, Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, CA, USA, January 19-21, 1998*, pages 303–315. ACM, 1998.

[Bar84] Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1984.

[BG95] Guy E. Blelloch and John Greiner. Parallelism in sequential functional languages. In John Williams, editor, *Proceedings of the seventh international conference on Functional programming languages and computer architecture, FPCA 1995, La Jolla, California, USA, June 25-28, 1995*, pages 226–237. ACM, 1995.

[CASC19] Andrea Condoluci, Beniamino Accattoli, and Claudio Sacerdoti Coen. Sharing equality is linear. In *Proceedings of the 21st International Symposium on Principles and Practice of Declarative Programming (PPDP 2019)*, pages 9:1–9:14, 2019.

[DLA17] Ugo Dal Lago and Beniamino Accattoli. Encoding Turing Machines into the Deterministic Lambda-Calculus. *CoRR*, abs/1711.10078, 2017.

[Kri93] Jean-Louis Krivine. *Lambda-calculus, types and models*. Ellis Horwood series in computers and their applications. Masson, 1993.

[Lév78] Jean-Jacques Lévy. Réductions correctes et optimales dans le lambda-calcul. Thése d'Etat, Univ. Paris VII, France, 1978.

[SvEB84] Cees F. Slot and Peter van Emde Boas. On tape versus core; an application of space efficient perfect hash functions to the invariance of space. In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 391–400. ACM, 1984.

[Tur36] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.