Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

THE FINITE LAMBDA CALCULUS

Relatore:
Chiar.mo Prof.
Andrea Asperti

Presentata da:
Filippo Cuti

III Sessione
Anno Accademico 2013-2014

Sommario

Il lambda calcolo è stato introdotto da Church negli anni '30 nell'ambito dello studio delle basi formali della matematica. È un sistema che permette di definire funzioni di ordine superiore. *Astraendo* una variabile x in un termine M si costruisce il nuovo termine $\lambda x.M$ che rappresenta quindi una funzione con parametro formale x. La valutazione della funzione su di un argomento è espressa mediante l'*applicazione* di un termine M ad un termine N.

L'interazione tra questi due meccanismi è data dalla β-regola:

$$\lambda x.M\ N \to M[N/x]$$

ovvero dalla sostituzione dell'argomento N al posto del parametro formale x. Come si può notare il meccanismo è molto intuitivo ed esprime chiaramente il comportamento dei termini nel calcolo.

È possibile dotare il lambda calcolo di un sistema di regole che assegnano ad ogni termine un *tipo*. Se si considerano i tipi come insiemi di termini, si può dire che un termine appartiene ad un tipo.

In questo lavoro di tesi si vuole iniziare lo studio del lambda calcolo finito, un'istanziazione del lambda calcolo su tipi finiti, ovvero tipi che hanno solo un numero finito di elementi *canonici*. Gli elementi canonici di un tipo T sono semplicemente gli elementi in forma normale di T, ma nel nostro caso vengono definiti sintatticamente assieme al tipo.

Il fatto di avere dei tipi con un numero finito di elementi canonici ci consente di aggiungere al calcolo la seguente regola, che trasforma una funzione f nel vettore che contiene i valori di f calcolati per ogni elemento del dominio:

$$\lambda x{:}\sigma.M \to \langle M[a_i/x]\rangle_\sigma$$

Con questa regola si vuole implementare un sorta un meccanismo di *memoization*.

La memoization consiste nel conservare in memoria il risultato di chiamate di funzione per evitare di ricalcolarne i valori in un secondo momento. Visto che lavoriamo con tipi finiti, invece di chiamare la memoization sono in alcuni casi, la applichiamo in parallelo su tutti i punti su cui è definita una funzione.

Il meccanismo della memoization, benché ampiamente utilizzato come tecnica di ottimizzazione, al meglio delle nostre conoscenze non è stato ancora studiato dal punto di vista teorico.

Il lambda calcolo finito è dunque una base semplice per iniziare l'indagine teorica della memoization.

Principale fonte di ispirazione dello studio è il lavoro di Goerdt [12, 11] in cui l'autore, estendendo dei risultati dovuti a Gurevich [17] e comunque seguendo una consolidata tradizione nell'ambito della teoria dei modelli [16], interpreta il sistema T di Gödel in un dominio finito.

Com'è noto, i termini del sistema T possono essere organizzati in una gerarchia basata sul *livello* dei tipi che occorrono nelle definizioni: il *rank* dei termini. Interpretando i termini del sistema in strutture finite gli autori ottengono una gerarchia di classi di complessità in cui si alternano spazio e tempo: i termini di rank 1 caratterizzano PTIME, rank 2 PSPACE, rank 3 EXPTIME ecc.

L'idea di introdurre i tipi finiti si riallaccia dunque alla tradizione di incorporare nella sintassi le informazioni semantiche di rilievo e che permettono di ottenere calcoli efficienti.

Nei prossimi capitoli viene introdotta e studiata la metateoria del calcolo: si dimostrano normalizzazione, subject reduction e confluenza; quest'ultima in particolare risulta molto interessante a causa di una coppia critica generata dalla regola di memoizazion. Infine, si dimostrano i limiti di complessità

analoghi a quelli dimostrati in [12].

Proprio la teoria della complessità è uno dei principali campi di applicazione del calcolo: le sue caratteristiche infatti lo rendono una base teorica adatta alla dimostrazione formale di risultati di complessità.

I risultati presentati in questa tesi sono descritti nell'articolo [5], in esame per la pubblicazione.

Uno dei personali contributi a quest'articolo è stato lo studio del problema della confluenza, in particolare una prima versione del teorema di confluenza locale; tuttavia quella presentata qui è diversa a causa di alcuni problemi tecnici discussi in seguito. La dimostrazione di normalizzazione si basa su quella per il lambda calcolo tipato semplice, descritta in [10].

In ogni caso, l'idea originale del lavoro ed alcune soluzioni tecniche sono del prof. Asperti.

Nel lavoro di tesi sono stati infine aggiunti alcuni esempi per chiarire l'intuizione dietro le strategie di riduzione e alcune dimostrazioni di semplici proprietà che per brevità non sono state incluse nell'articolo.

Contents

Chapter 1

Introduction

1.1 Background

The lambda calculus, introduced by Church in the thirties in his investigation of the foundations of mathematics, is the synthetic description of a higher order computational system based on function abstraction and application: a new function $\lambda x.M$ may be defined by *abstracting* a formal parameter x in the function body M, and the function may be evaluated by *applying* it to a specific argument N. The interaction between abstraction and application is simply described by the so called β-rule

$$\lambda x.M\ N \to M[N/x]$$

reducing the computation to a simple process of *substitution* of the argument N in place of the formal parameter x. The smallest congruence on lambda terms containing the β-rule is known as β-equivalence.

Though β-rule is an an extremely simple conceptual tool to express the the intended *behaviour* of abstraction and application, it is stated at the meta-level of the calculus. As consequence, we cannot look at β-rule as a truly computational device, as this formulation somewhat obscures many syntactic phenomena hidden under his definition. Such kind of phenomena were brought to light in particular when was necessary to fill the gap between

the (meta) theoretical definition of beta-rule and his real implementations. In this context, it was (slowly) realized that a less naive description of substitution (avoiding a bruteforce duplication of arguments) is required to obtain an efficient implementation of the calculus.

One simple and clear way to express an algorithm implementing the β-reduction is the definition of abstract machines, of which the simplest example is probably the well-known Krivine's machine [22]. With environments and closures it is possible to provide a mechanism for a lazy management of substitution that has no clear counterpart in pure lambda calculus, naturally suggesting to address substitution as an *explicit* component of the calculus. In particular in [9] is developed a calculus of explicit substitutions in which, simply removing some rules, one may express lazy or eager reduction strategy. From those rules it is possible to derive *almost deterministically* [9] the corresponding abstract machine.

Many other kind of machines expressing different evaluation strategies have been developed (see e.g. [9, 8]); however all the machines were obtained as a combination of *ingenuity* and skill to use the same words of [8], but without a complete theoretical investigation. In [8] is thus provided an algorithm that derives from an abstract specification of a strategy in a calculus with explicit substitution an abstract machine implementing the specified strategy.

An alternative syntactical framework where the complex issue of duplication can be better taken into account is provided by Linear Logic and proof nets [19]. Here, the resources are thoroughly managed, not by means of some kind of syntactical construct, but by the *logical nature* of the framework itself, which does not allow the use of the structural rules of weakening and contraction, forbidding the arbitrary reuse of an hypothesis.

The recent *structural lambda calculus* [2] is a nice synthesis between explicit substitutions and linear logic. In this work the author add to a calculus with explicit substitution some *structural rules* stemming from linear logic's sequent calculus. In this way duplication of arguments is carefully avoided

(as in linear logic is forbidden arbitrary use of hypothesis). From this rules can be derived (*distilled*, using the author's words) many kinds of abstract machines.

More sophisticated forms of reduction, pursuing sharing under λ-abstractions [23], require even more complex syntactical representations and metatheoretical frameworks, such as the so called *sharing graphs* [3].

An alternative way of sharing evaluation of functions is provided by *memoization*, that consists in progressively caching function calls, avoiding to re-compute a result on an already occurred input. While memoization is a largely used optimization technique, up to our knowledge, it has never been the object of a serious metatheoretical investigation. Here, we start this study, in the extremely simple case of finite data types. Since we work in a finite setting, instead of performing memoization "on demand", we work in parallel on all possible inputs, unfolding a function into a finite vector of cases (that is, essentially, its graph).

The finite setting is particularly interesting, since it helps to put in evidence the limitations of β-reduction as a computational mechanism: using Terui's colorful expression [27], β-reduction is in fact "desperately inefficient" for computing finite-valued function. Different "semantic" techniques may be profitably used for evaluation in the finite case, better exploiting the typing information.

The finite lambda calculus has its root in the mainstream of characterizations of complexity classes via the interpretation of formal languages into finite structures (see e.g. [16, 18] for recent surveys.) While most of the results in this area have a model-theoretic and descriptive nature, Gurevich [17] opened a more algebraic research direction, by observing that interpreting primitive recursive functions (resp. recursive functions) over finite structures one precisely get the log-space (resp. polynomial time) computable functions. The result was extended by Goerdt [11, 12] to higher complexity classes, using higher type primitive recursive definitions (i.e. Gödel System T). The terms of this language can be organized in a hierarchical structure, according

to the level of the types occurring in their definitions (the so called *rank* of a term). By *interpreting* these terms on finite structures, one obtains, as rank raises, an alternating hierarchy of time and space complexity classes: terms of rank-1 characterize LOGSPACE, rank-2 PTIME, rank-3 PSPACE, rank-4 EXPTIME=DTIME(2^{poly}), and so on. Similar results have been also obtained by Jones [20] and Kristiansen [21], who investigated the effect of removing successor-like functions in primitive recursive frameworks.

In Goerdt's generalization is expressed, though in a different way, the idea behind memoization. In fact in [12] the author use what he calls an *infinite term*, that is a term expressed by a family of related terms and which - in our case - is nothing but the identification of a function with his graph. However, the original idea should be credited to Schwichtenberg [25] who used it in the context of proof theory.

The simply typed lambda calculus that we address in this work is, essentially, the purely functional subcalculus of the system considered by Goerdt. The main difference is that while Goerdt gives a semantic description of the calculus, as a result of *interpreting* primitive recursion on a finite subrange of natural numbers, we are interested to look at the formal system as a rewriting system, with its own reduction rules and computational strategies, following the above mentioned tradition of incorporating into the syntax the relevant "semantic" techniques (in this case, memoization).

As a final source of inspiration of our work, we would like to mention some recent achievement in interactive theorem proving, such as the proof of four color theorem [13], or the formal verification of the Odd-order Theorem [14], where finite types play an essential role. In fact, in modern, dependently typed systems (see e.g. [15, 24]), the collection of all finite types is itself a type, allowing their *uniform* and *parametric* management: we can pass finite types as input to programs, and write polymorphic functions that work on "generic" finite types, with no knowledge of their cardinality or concrete definition. Programming with finite types is not only feasible, but also quite natural, suggesting a better theoretical investigation of this underestimated

and neglected topic.

The results presented here are described in the paper [5] which is submitted for publication . In this paper is addressed the generalization of the results described in [4] and studied the meta-theory of a subcalculs introduced in [4]. One of the personal contributions to that work was in the study of the confluence problem: in particular a first version of the confluence proof, as the one presented here is different due to a some technical problem. The normalization proof is almost identical to the classical proof based on the reducibility notion [10]. However, the original idea of the work and many technical solutions are due to prof. Asperti.

In the thesis were also added some examples to explain the intuition behind the reduction strategy and some proofs of simple properties not included in the paper.

The structure of the work is the following: in the first part (sections 2.1 to 3.3) we investigate the metatheory of reduction (subject reduction, strong normalization, and confluence); the structure of this part of the work is a consequence of some interesting peculiarities of the calculus briefly discussed in the next section. In the second part, we address the *complexity* of normalization (section 4.1), proving that any term of the finite lambda calculus of rank $2n+2$ can be normalized in *time* $exp_n(p)$ (a tower of exponential of length n, ending in a polynomial p, see section 4.1), and similarly any term of rank $2n+3$ can be normalized with *space* $exp_n(p)$.

In section 4.6, we use this fact to provide an alternative and simpler proof of a recent result by Terui [27].

1.2 Preliminary discussion

The main feature of the finite lambda calculus is a *memoization* rule unfolding a function into a *finite vector* of cases, one for each *canonical element* of the type σ:

$$(\mu)\quad \lambda x{:}\sigma.M \to \langle M[a_i/x]\rangle_\sigma \quad \text{for } a_i \in \mathcal{C}(\sigma)$$

The canonical elements $\mathcal{C}(\sigma)$ of type σ are, essentially, the terms in normal forms of the corresponding type, but they are defined syntactically along with types, since we need them for the definition of both typing and reduction rules. The other operation on vectors is field selection (case switch):

$$(\iota)\ \langle M_a \rangle_A\ a \to M_a$$

The β-rule and the μ-rule generate an interesting *critical pair*:

$$\begin{array}{ccc} & \lambda x\!:\!\sigma.M\ N & \\ {\scriptstyle\mu}\swarrow & & \searrow{\scriptstyle\beta} \\ \langle M[a_i/x]\rangle_\sigma\ N & & M[N/x] \end{array}$$

If N is an open term (e.g. a variable), we have no hope to close the diamond.

Instead of restricting the rules of the calculus, that is both unnatural and problematic, especially in view of substitution lemmata, we shall content ourselves to prove confluence in case of *closed terms*.

If $N\!:\!\sigma$ is closed, then it will eventually reduce to a canonical element a_j in $\mathcal{C}(\sigma)$, and we can close the diagram with the ι-rule:

$$\begin{array}{ccc} & \lambda x\!:\!\sigma.M\ N & \\ {\scriptstyle\mu}\swarrow & & \searrow{\scriptstyle\beta} \\ \langle M[a_i/x]\rangle_\sigma\ N & & M[N/x] \\ \downarrow{\scriptstyle *} & & \downarrow{\scriptstyle *} \\ \langle M[a_i/x]\rangle_\sigma\ a_j & \xrightarrow{\iota} & M[a_j/x] \end{array}$$

This fact relies however on two important properties:

1. *normalization*, since the term N must reduce to a canonical form;
2. *subject reduction*, since the canonical form must be of the same type of the term N.

Note also, by the way, that confluence may only hold for *well typed terms*: if the type of N is not σ (and hence $\lambda x\!:\!\sigma.M\ N$ is ill typed), we cannot close the critical pair above.

Let us finally remark that, in order to close the diagram, we need to reduce the argument N to the normal form, so we cannot hope to prove confluence neither by a parallel reduction technique [26], nor by tracing residuals (see e.g. [7]).

Since in any case we need strong normalization, the most obvious approach to confluence consists in proving a *local confluence property*, that, as it is well known, in conjunction with strong normalization entails confluence[1].

The plan of our metatheoretical investigation is thus the following: in section 2.1 we give the formal definition of the calculus, comprising the type system and the reduction rules; then we address in turn the properties of subject reduction (section 3.1), strong normalization (section 3.2) and local confluence (section 3.3).

[1]The confluence problems of the finite lambda calculus have strong similarities with the calculus of singleton types of Abel et al. [1] that is essentially a particular case of our calculus.

Chapter 2

The calculus

2.1 Definitions and notation

In this chapter we give the syntax of the calculus, the typing rules and the reduction rules together with the proofs of substitution lemmata.

2.1.1 Types and Terms

Definition 2.1.1. *(Types)*

1. *a* finite type *A is a type defined by a finite collection of elements (constructors) $a_1, \ldots, a_n$*

2. *a type is either a finite type or an* arrow type *$\sigma \to \tau$ between two types.*

Definition 2.1.2. *The terms of the language are defined according to the following syntax, where τ is a type:*

$$\begin{array}{lcll} M,N & := & x & \textit{variable} \\ & | & a_i & \textit{constructor} \\ & | & \lambda x{:}\tau.M & \textit{abstraction} \\ & | & (M\ N) & \textit{application} \\ & | & \langle M_1, \ldots, M_n \rangle_\tau & \textit{vector} \end{array}$$

Free variables are those not bound by a lambda; more formally, le M be a term, the set FV of the free variables of M is recursively defined by:

Definition 2.1.3. *(free variables)*

- $FV(x) = \{x\}$
- $FV(\lambda x.M) = FV(M) - \{x\}$
- $FV(P\ Q) = FV(P) \cup FV(Q)$
- $FV(\langle M_1, \ldots, M_n \rangle_\tau) = \bigcup_i FV(M_i)$

A term M is closed if $FV(M) = \emptyset$.

Definition 2.1.4. *(substitution)*

- $x[N/x] = N$
- $y[N/x] = y \quad if \quad y \neq x$
- $(\lambda y.M)[N/x] = \lambda y.M[N/x] \quad if \quad y \notin FV(N) \quad and \quad y \neq x$
- $P\ Q[N/x] = P[N/x]\ Q[N/x]$
- $\langle M_1, \ldots, M_n \rangle_\tau [N/x] = \langle M_1[N/x], \ldots, M_n[N/x] \rangle_\tau$

Definition 2.1.5. *(Canonical elements)*
For each type τ we define a set $\mathcal{C}(\tau)$ of canonical elements *of τ, together with total ordering $<_\tau$ on them; two canonical elements are equal if and only if they are syntactically identical.*

- *if $A = \{a_1, \ldots a_n\}$ each a_i is a canonical element; $a_i <_A a_j$ if and only if $i < j$.*
- *a canonical element of $\sigma \to \tau$ is a vector $\langle b_a \rangle_\sigma$ of canonical elements of τ indexed over canonical elements $a \in \sigma$; vectors are ordered lexicographically: $\langle b_a \rangle <_{\sigma\to\tau} \langle b'_a \rangle$ if and only if and only if there exists a canonical element $a : \sigma$ such that forall $a' <_\sigma a, b_{a'} = b'_{a'}$ and $b_a <_\tau b'_a$.*

Definition 2.1.6. *(Cardinality) The cardinality $\|\tau\|$ of a type τ is the number of canonical elements of the type.*

2.1.2 Typing

Definition 2.1.7. *(Typing rules) The type system is defined by the following rules where, as usual,* Γ *is a* context *associating types with free variables:*

$$\textit{(variable)} \qquad \Gamma \vdash x : \tau \quad \textit{if } x : \tau \in \Gamma$$

$$\textit{(constructor)} \qquad \Gamma \vdash a_i : A \quad \textit{if } A = \{a_1, \ldots, a_n\}$$

$$\textit{(abstraction)} \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma . M : \sigma \to \tau}$$

$$\textit{(application)} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash (M\, N) : \tau}$$

$$\textit{(vector)} \qquad \frac{\Gamma \vdash M_a : \tau \quad \forall a \in \mathcal{C}(\sigma)}{\Gamma \vdash \langle M_a \rangle_\sigma : \sigma \to \tau}$$

Let us remark that, according to the vector rule, the vector $\langle M_a \rangle_\sigma$ is well typed if it contains a component for each *canonical element* of type σ, that is, if it has the expected length.

It is easy to prove, by structural induction on the type σ, that all elements in $\mathcal{C}(\sigma)$ are well typed terms of type σ:

Lemma 2.1.8. *For each $c \in \mathcal{C}(\sigma)$ we have $\vdash c : \sigma$.*

By inversion on the definition of the typing rules, we easily get the following generation lemma:

Lemma 2.1.9. *(generation lemma)*

1. *if $\Gamma \vdash \lambda x : \sigma . M : \rho$, then there exists τ such that $\rho = \sigma \to \tau$ and $\Gamma, x : \sigma \vdash M : \tau$;*
2. *if $\Gamma \vdash M\, N : \tau$, then there exists σ s.t. $\Gamma \vdash M : \sigma \to \tau$ and $\Gamma \vdash N : \sigma$;*

3. *if* $\Gamma \vdash \langle M_a \rangle_\sigma : \rho$, *then there exists* τ *such that* $\rho = \sigma \to \tau$ *and, for any* $a \in \mathcal{C}(\sigma)$, $\Gamma \vdash M_a : \tau$.

We also need the following substitution lemma:

Lemma 2.1.10. *If* $\Gamma, x:\sigma, \Delta \vdash M : \tau$ *and* $\Gamma \vdash N : \sigma$, *then* $\Gamma, \Delta \vdash M[N/x] : \tau$.

Proof. By induction on the type-derivation $\Gamma, x:\sigma, \Delta \vdash M : \tau$.

□

2.2 Reduction

Let us come to the reduction rules.

Definition 2.2.1. *(one step reduction) One step reduction is the smallest relation containing the following rules*

$$
\begin{array}{lll}
(\beta)\ \lambda x.M\ N \to M[N/x] & & \\
(\iota)\ \langle M_a \rangle_\sigma\ a_i \to M_{a_i} & & \text{for } a_i \in \mathcal{C}(\sigma) \\
(\mu)\ \lambda x{:}\sigma.M \to \langle M[a_i/x] \rangle_\sigma & & \text{for } a_i \in \mathcal{C}(\sigma)
\end{array}
$$

and closed with respect to the following congruence rules:

$$
(@_l)\ \frac{M \to M'}{M\,N \to M'\,N} \qquad (@_r)\ \frac{N \to N'}{M\,N \to M\,N'}
$$

$$
(\lambda)\ \frac{M \to M'}{\lambda x{:}\tau.M \to \lambda x{:}\tau.M'} \qquad (v)\ \frac{N \to N'}{\langle v, N, v_1 \rangle_\tau \to \langle v, N', v_1 \rangle_\tau}
$$

In the (μ)-rule, vector components are ordered according to $<_\sigma$. In the (v)-rule, v_1 and v_2 are arbitrary lists of terms.

As usual, we shall use the notation $M \xrightarrow{*} N$ to denote the reflexive and transitive closure of the one step reduction relation.

2.2.1 Basic Properties

Here we prove the following properties, that are required to address the local confluence of the calculus:

Lemma 2.2.2. *For all terms* M, M', N, N'

1. *if* $M \xrightarrow{*} M'$ *then* $M[N/x] \xrightarrow{*} M'[N/x]$;
2. *if* $N \xrightarrow{*} N'$ *then* $M[N/x] \xrightarrow{*} M[N'/x]$.

Proof. (1) The proof is by structural induction on the term M.

- M is of the form $\lambda y.P$. By hypothesis it reduces (in many steps) to the term M'. The reduction must take place under the λ ending in an term of the form $\lambda y.P'$. By definition of substitution $(\lambda y.P)[N/x] = \lambda y.P[N/x]$ $(y \neq x)$. So by induction hypothesis to $P[N/x] \xrightarrow{*} P'[N/x]$. Then $\lambda y.P[N/x] \xrightarrow{*} \lambda y.P'[N/x]$ (by lemma 2.2.3), and by definition of substitution we close the case.

- M is of the form $P\,Q$. Reduction may take place inside P,Q or both. However, applying the induction hypothesis and the definition of substitution we get the claim.

- M is a vector: $\langle v, N, v_1 \rangle_\tau$. We then reduce inside a component- say N- getting N'. Then we can apply the induction hypothesis to N and, similarly to the previous cases we get the claim.

- If the term is a variable the case is trivial.

□

Proof. (2) The proof is by structural induction on the term M, and is similar to the previous one.

- M is of the form $\lambda y.P$.
  Then, by definition of substitution: $(\lambda y.P)[N/x] = \lambda y.P[N/x]$. Applying the induction hypothesis to P we have that: $P[N/x] \xrightarrow{*} P[N'/x]$. Then we close the case in a way similar to the previous lemma.

- M is of the form $P\,Q$. So, $P\,Q[N/x] = P[N/x]\,Q[N/x]$. Now we can apply the induction hypothesis to P and Q, obtaining the claim.

- M is a vector: $\langle v, N', v_1\rangle_\tau$. By definition of substitution we have $\langle v, N', v_1\rangle_\tau[N/x] = \langle v[N/x], N'[N/x], v_1[N/x]\rangle_\tau$. In a way similar to the previous, cases we get the claim by induction hypothesis.

- If the term is a variable the case is trivial.

□

Lemma 2.2.3. *For all terms P, P', Q, Q'*

1. *if $P \xrightarrow{*} P'$ then $P\ Q \xrightarrow{*} P'\ Q$*
2. *if $Q \xrightarrow{*} Q'$ then $P\ Q \xrightarrow{*} P\ Q'$*
3. *if $P \xrightarrow{*} P'$ then $\lambda x\!:\!\sigma.P \xrightarrow{*} \lambda x\!:\!\sigma.P'$*
4. *if $P \xrightarrow{*} P'$ then $\langle v, P, v_1\rangle_\sigma \xrightarrow{*} \langle v, P', v_1\rangle_\sigma$ for all lists of terms v, v_1.*

The proofs are by induction on the number of steps, and have the same structure: we first prove the base case by means of the relative congruence rule, and then, assuming the statement true for $(n-1)$-step reductions, we prove for n-step reductions applying again the right congruence rule.

In the following we shall use $P \xrightarrow{n} Q$ to indicate that M reduces to N in n steps.

Proof. **Base**: $n = 1$, $P \to P'$ $(Q \to Q')$.
We can prove that:

- $P\,Q \to P'\,Q$ applying the $@_l$ rule.

- $P\,Q \to P\,Q'$ applying the $@_r$ rule.

- $\lambda x.P \to \lambda x.P'$ applying the λ rule.

- $\langle v, P, v_1\rangle_\tau \to \langle v, P', v_1\rangle_\tau$ applying the v rule.

Induction. We suppose hypothesis true for reduction of length $n-1$. By hypothesis, we have that $P \xrightarrow{n} P'$ ($Q \xrightarrow{n} Q'$). The must exists P'' (Q'') such that

$$P \xrightarrow{n-1} P'' \to P' \quad (Q \xrightarrow{n-1} Q'' \to Q')$$

Now we can use the induction hypothesis to derive that:

- $P\,Q \xrightarrow{n-1} P''\,Q$.
- $P\,Q \xrightarrow{n-1} P\,Q''$.
- $\lambda x.P \xrightarrow{n-1} \lambda x.P''$.
- $\langle v, P, v_1 \rangle_\tau \to \langle v, P'', v_1 \rangle_\tau$.

As $P'' \to P'$ ($Q'' \to Q'$) we close the proof applying respectively $@_l$, $@_r$, λ, v rules. □

Chapter 3

Meta-theory

In this chapter we shall provide the main results concerning the meta-theory of the calculus. As anticipated in the introduction, the structure of the chapter is due to a problem with confluence induced by the critical pair discussed in chapter 1.1. In the following section we prove subject reduction lemma, that is the fact that reduction preserves typing. In section 3.2 we prove strong normalisation theorem and we conclude the chapter in section 3.3 proving the weak Church-Rosser property from which follows global confluence.

3.1 Subject Reduction

Lemma 3.1.1. *If* $\Gamma \vdash M : \sigma$ *and* $M \to N$, *then* $\Gamma \vdash N : \sigma$.

Proof. The proof is by induction on the definition of the one step relation.

- Case $M \xrightarrow{\beta} N$. We can only apply the β-rule when $M \equiv \lambda x{:}\sigma.M_1\ M_2$. So, it holds that

$$\Gamma \vdash \lambda x{:}\sigma.M_1\ M_2 : \tau$$

and by lemma 2.1.9.(2) there exists σ' such that

$$\Gamma \vdash \lambda x{:}\sigma.M_1 : \sigma' \to \tau \quad \text{and} \quad M_2 : \sigma.$$

Moreover, by lemma 2.1.9.(1) we must have $\sigma = \sigma'$ and

$$\Gamma, x:\sigma \vdash M_1 : \tau$$

By lemma 2.1.10

$$\Gamma \vdash M_1[M_2/x] : \tau.$$

- Case $M \xrightarrow{\mu} N$. We can only apply μ-rule when $M \equiv \lambda x:\sigma.M_1$. So it holds that

$$\Gamma \vdash \lambda x:\sigma.M_1 : \sigma \to \tau$$

By lemma 2.1.9.(1)

$$\Gamma, x:\sigma \vdash M_1 : \tau$$

By lemma 2.1.10

$$\Gamma \vdash M[a_i/x] : \tau \quad \forall a_i : \mathcal{C}(\sigma)$$

and, by the typing rule for vectors, we get

$$\Gamma \vdash \langle M[a_i/x]\rangle_\sigma : \sigma \to \tau$$

- Case $M \xrightarrow{\iota} N$. We can only apply ι-rule when $M \equiv \langle M_a\rangle\, a_i$ and $a_i \in \mathcal{C}(\sigma)$. By lemma 2.1.9.(2), there exists σ' such that

$$\Gamma \vdash \langle M_a\rangle : \sigma' \to \tau \;\text{ and }\; \Gamma \vdash a_i : \sigma'$$

Moreover, by lemma 2.1.9.(3) we must have $\sigma = \sigma'$ and, for any $a_i \in \mathcal{C}(\sigma)$, M_{a_i} has type τ.

- Case $M\,N \to M'N$, where $M \to M'$. If $\Gamma \vdash M\,N:\tau$, by lemma 2.1.9.(2), $\Gamma \vdash M : \sigma \to \tau$ and $\Gamma \vdash N : \sigma$ for some σ. By induction hypothesis $\Gamma \vdash M':\sigma \to \tau$, so $\Gamma \vdash M'\,N:\tau$.

- Case $M\,N \to M\,N'$, where $N \to N'$. If $\Gamma \vdash M\,N:\tau$, by lemma 2.1.9.(2), $\Gamma \vdash M : \sigma \to \tau$ and $\Gamma \vdash N : \sigma$ for some σ. By induction hypothesis $\Gamma \vdash N' : \sigma$, so $\Gamma \vdash M\,N : \tau$

- Case $\lambda x : \sigma.M \to \lambda x : \sigma.M'$, where $M \to M'$. If $\Gamma \vdash \lambda x : \sigma.M : \rho$, by lemma 2.1.9.(1), there exists τ such that $\rho = \sigma \to \tau$ and $\Gamma, x : \sigma \vdash M : \tau$. By induction hypothesis $\Gamma, x : \sigma \vdash M' : \tau$, and hence $\Gamma \vdash \lambda x : \sigma.M' : \sigma \to \tau$.

- Case $\langle v, M, v_1 \rangle \to \langle v, M', v_1 \rangle\_\sigma$, where $M \to M'$. If $\Gamma \vdash \langle v, M, v_1 \rangle_\sigma : \rho$, then by lemma 2.1.9.(1), there exists τ such that $\rho = \sigma \to \tau$ and all components of the vector have type τ. By induction hypothesis $\Gamma, \vdash M' : \tau$, and hence $\Gamma \vdash \langle v, M', v_1 \rangle_\sigma : \sigma \to \tau$.

□

3.2 Strong normalisation

In this section we prove the Strong Normalisation Theorem for the finite lambda calculus, that is the fact that the reduction relation is well founded. We follow Tait's approach based on the notion of reducibility (see e.g. [10]). Reducibility is an abstract property on terms entailing, among other things, their strong normalisation. So, proving that all terms are reducible implies strong normalisation as a corollary.

The proof is quite traditional, and we only include it for the sake of completeness. The main novelties with respect to the simply typed case is that we exclude vectors from neutral elements, and add a suitable lemma for them (Lemma 3.2.5):

Definition 3.2.1. *(Reducibility)*

- *If A is an atomic type, a term M is reducible of type A ($M \in Red_A$) if and only if M is strongly normalizable*
- *A term M is reducible of type $\sigma \to \tau$ ($M \in Red_{\sigma\to\tau}$) if for any $N \in Red_\sigma$, $M\,N \in Red_\tau$.*

Definition 3.2.2. *If M is a strongly normalizable term, we shall denote with $\nu(M)$ the length of the longest reduction sequence rooted in M.*

3.2.1 Reducibility properties

We shall use s.n. as an abbreviation for "strongly normalizable".

Definition 3.2.3. *A term is called neutral if it is not of the form* $\lambda x.P$ *or* $\langle P_a \rangle_\sigma$

A neutral term is a term that cannot create a redex when applied to an argument.

The property we are interested in are the following:

- (**CR 1**) If M is reducible of type T, then M is s.n.

- (**CR 2**) If M is reducible of type T and $M \xrightarrow{*} M'$, then M' is reducible of type T.

- (**CR 3**) If M is neutral, and whenever we convert a redex of M we obtain a term M' reducible of type T, then M is reducible of type T.

Now we prove, by induction on types, that **CR1-3** hold for all the types of the calculus.

3.2.2 Reducibility Theorem

Atomic types

A term of atomic type is reducible iff it is s.n. So we must show that the set of s.n terms of type T satisfies the three reducibility conditions:

- (**CR 1**) is a tautology.

- (**CR 2**) If M is s.n. then every term M' to which M reduces is also.

- (**CR 3**) A reduction path leaving M must pass through one of the terms M' , which are s.n., and so is finite.

Arrow type

A term of arrow type is reducible if an only if all its applications to reducible terms are reducible.

- (**CR 1**) Assume $M \in Red_{\sigma\to\tau}$, and let x be a variable of type σ; the induction hypothesis (**CR 3**) for σ says that the term x, which is neutral and normal, is reducible. So $M\,x$ is reducible, and the induction hypothesis (**CR 1**) for τ tell us that $M\,x$ is s.n. Since any subterm of a s.n. term is s.n.too, M is s.n.

- (**CR 2**) Assume $M \xrightarrow{*} M'$ and $M \in Red_{\sigma\to\tau}$. For any $N \in Red_\sigma$, $M\,N \in Red_\tau$ by definition of reducibility; moreover, $M\,N \xrightarrow{*} M'\,N$. The induction hypothesis (**CR 2**) for τ gives that $M'\,N$ is reducible. So M' is reducible by definition.

- (**CR 3**) Let M be neutral and suppose all the M' one step from M belongs to $Red_{\sigma\to\tau}$. Let N be a reducible term of type σ; we want to show that $M\,N$ is reducible. By induction hypothesis (**CR 1**) for σ , we know that N is strongly normalisable; so we can reason by induction on $\nu(N)$. Since M is neutral, in one step $M\,N$ converts to

  - $M'\,N$ with M' one step from M; but M' is reducible, so $M'\,N$ is.
  - $M\,N'$, with N' one step from N. N' is reducible by induction hypothesis (**CR 2**) for σ , and $\nu(N') < \nu(N)$; so, by the induction hypothesis for N, $M\,N'$ is reducible.

  Since all terms reachable in one step from $M\,N$ are reducible, we can apply the induction hypothesis (**CR 3**) for τ and conclude that $M\,N$ is reducible. Hence, M is reducible by definition.

We need a couple of lemmas relative to not neutral terms.

Lemma 3.2.4. *If for all $N \in red_\sigma$, $M[N/x] \in Red_\tau$, then $\lambda x : \sigma.P \in Red_{\sigma\to\tau}$.*

Proof. We need to prove that $\lambda x:\sigma.P\ Q$ is reducible for all $Q \in red_\sigma$. We reason by induction on $\nu(P)+\nu(Q)$. The term $\lambda x:\sigma.P\ Q$ may reduce in one step to

- $P[Q/x]$, which is reducible by hypothesis;
- $\lambda x:\sigma.P\ Q$ where $P \to P'$; so P' is reducible, $\nu(P') < \nu(P)$, and by induction hypothesis this term is reducible.
- $\lambda x:\sigma.P\ Q$ where $Q \to Q'$; so Q' is reducible, $\nu(Q') < \nu(Q)$, again, by induction hypothesis we have the claim.

In every case the neutral term $\lambda x:\sigma.P\ Q$ converts to reducible terms only, and by (**CR 3**) it is reducible. So $\lambda x:\sigma.P$ is reducible by definition. □

Lemma 3.2.5. *If for all $a \in \mathcal{C}(\sigma)$, P_a is a reducible term of type τ, then $\langle P_a\rangle_A$ is a reducible term of type $\sigma \to \tau$.*

Proof. We need to prove that for any $Q \in Red_\sigma$ the term $\langle P_a\rangle\, Q$ is a reducible term of type τ. The proof is by induction on $\sum_{a\in\mathcal{C}(\sigma)} \nu(P_a)+\nu(Q)$. The term $\langle P_a\rangle\, Q$ may convert in one step to

- $\langle P'_a\rangle\, Q$ where $\langle P'_a\rangle$ is a one-step reduct of $\langle P_a\rangle$; then $\sum_{a\in\mathcal{C}(\sigma)} \nu(P'_a) < \sum_{a\in\mathcal{C}(\sigma)} \nu(P_a)$ (at least of component has decreased), and we may apply the induction hypothesis to conclude that $\langle P'_a\rangle\, Q$ is reducible;
- $\langle P_a\rangle\, Q'$ where $Q \to Q'$; so Q' is reducible, $\nu(Q') < \nu(Q)$, and hence by induction hypothesis $\langle P_a\rangle\, Q'$ is reducible;
- P_{a_j} if $Q \equiv a_j$ that is reducible by assumption.

In all cases the neutral term $\langle P_a\rangle\, Q$ converts to reducible terms only, and by (**CR 3**) it is reducible. So $\langle P_a\rangle$ is reducible. □

Theorem 3.2.6. *Let M be a term with free variables in $\vec{x}$, and let $\vec{N}$ be a vector of reducible terms. Then the term $M[\vec{N}/\vec{x}]$ is reducible.*

Proof. By induction on t.

- M is x_i. Then $x_i[\vec{N}/\vec{x}] = N_i$ that is a reducible term by assumption.

- M is a constructor a of some atomic type A. This is a normal form, hence a strongly normalizable term, hence a reducible term by definition (since A is atomic).

- M is an application $P\,Q$. We want to prove that

$$(P\,Q)[\vec{N}/\vec{x}] = P[\vec{N}/\vec{x}]\,Q[\vec{N}/\vec{x}]$$

  is reducible. By induction hypothesis $P[\vec{N}/\vec{x}]$ and $Q[\vec{N}/\vec{x}]$ are reducible, and so is $P[\vec{N}/\vec{x}]\,(Q[\vec{N}/\vec{x}])$, by definition of reducibility.

- M is $\lambda y{:}\sigma.P$. We want to prove that $(\lambda y{:}\sigma.P)[\vec{N}/\vec{x}]) = \lambda y{:}\sigma.(P[\vec{N}/\vec{x}])$ is reducible. By lemma 3.2.4, it suffices to prove that for any $Q \in Red_\sigma$, the term $P[\vec{N}/\vec{x}][Q/y] = P[\vec{N}, P/x, \vec{y}]$ is reducible. But this follows from the induction hypothesis for P.

- The term M is of the form $\langle P_a \rangle$. By induction hypothesis $P_a[\vec{N}/\vec{x}]$ is reducible $\forall a \in \mathcal{C}(\sigma)$. So, by lemma 3.2.5, $\langle P_a \rangle[\vec{N}/\vec{x}] = \langle P_a[\vec{N}/\vec{x}] \rangle$ is reducible.

□

Corollary 3.2.7. *All terms are reducible.*

Proof. Obvious, since variables are reducible terms, and for any term M we have $M = M[\vec{x}/\vec{x}]$. □

Now, by **CR 1** we have the following

Corollary 3.2.8. *All terms are strongly normalisable.*

3.3 Local confluence

In this section we prove the local confluence lemma. Global confluence follows by strong normalisation and local confluence. As we anticipated in the introduction, we shall prove it only for *closed* and *well typed terms* since it doesn't hold, in general, neither for open nor ill typed terms.

The proof is by induction on the term, and then by cases on the reduction rules. Induction on the term is required to handle the subcases of two redexes deep inside a *same* subterm of the given term. A problem related to this induction is that, while subterms of a well typed term are still well typed, subterms of a closed terms are not necessarily closed (hence we could not use the induction hypothesis). In particular, we are faced with this situation when we are considering a term $\lambda x\!:\!\sigma.M$ and two redexes $R_1 : M \to M'$ and $R_2 : M \to M''$. To solve this situation, we adopt the following approach. Instead of proceeding by structural induction on the term, we proceed by induction on a suitable notion of *dimension* for terms, where the dimension of a variable of type σ is defined as the dimension of a canonical element in $\mathcal{C}(\sigma)$.
By memoization,

$$\begin{aligned}\lambda x\!:\!\sigma.M &\to \langle M[a/x]\rangle_\sigma\\ \lambda x\!:\!\sigma.M' &\to \langle M'[a/x]\rangle_\sigma\\ \lambda x\!:\!\sigma.M'' &\to \langle M''[a/x]\rangle_\sigma\end{aligned}$$

Consider a given component of the vector. We know that

$$M[a/x] \to M'[a/x] \quad \text{and} \quad M[a/x] \to M''[a/x]$$

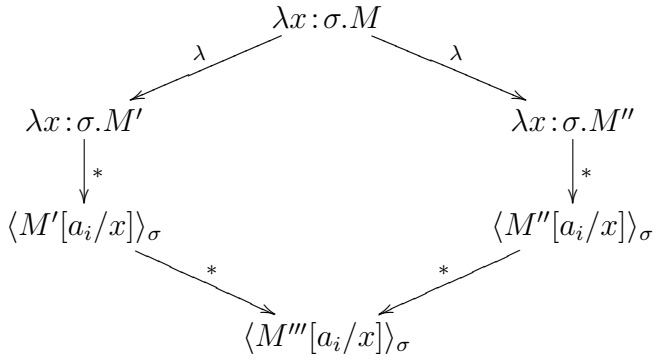
(in one step!) Moreover, the dimension of $M[a/x]$ is equal to the dimension of M and hence strictly less than the dimension of $\lambda x\!:\!\sigma.M$. We can hence apply the induction hypothesis, to conclude that there must exists, for each $a \in \mathcal{C}(\sigma)$ a term M'''_a such that

$$M'[a/x] \xrightarrow{*} M'''_a \quad \text{and} \quad M''[a/x] \xrightarrow{*} M'''_a$$

But then

$$\langle M'[a/x]\rangle_\sigma \xrightarrow{*} \langle M'''_a\rangle_\sigma \quad \text{and} \quad \langle M''[a/x]\rangle_\sigma \xrightarrow{*} \langle M'''_a\rangle_\sigma$$

that closes the diagram.



The notion of dimension that we need for our purposes is the following (that is slightly different from the more traditional notion of size of Definition 4.2.2.)

Definition 3.3.1. *(dimension)*

1. *The dimension $\mathcal{D}(\tau)$ of a type τ is defined as follows:*

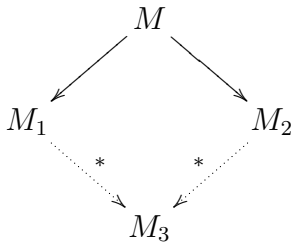
   - $\mathcal{D}(A) = 1$*; for A atomic*
   - $\mathcal{D}(\sigma \to \rho) = \|\sigma\| \times \mathcal{D}(\rho)$

2. *Let M be a well typed terms (so we know the type of each variable). Its* dimension $\mathcal{D}(M)$ *is defined as follows*

   - $\mathcal{D}(a) = 1$
   - $\mathcal{D}(x) = \mathcal{D}(\sigma)$ *if* $x:\sigma$
   - $\mathcal{D}(M\,N) = \mathcal{D}(M) + \mathcal{D}(N) + 1$*;*
   - $\mathcal{D}(\lambda x{:}\sigma.M) = \mathcal{D}(M) + 1$;
   - $\mathcal{D}(\langle M_a\rangle_\sigma) = \sum_{a\in\mathcal{C}(\sigma)} \mathcal{D}(M_a)$

It is easy to prove (by induction on σ) that, for any $c \in \mathcal{C}(\sigma)$, $\mathcal{D}(c) = \mathcal{D}(\sigma)$. As a consequence, if $x{:}\sigma$, $\mathcal{D}(x) = \mathcal{D}(c)$ for any $c \in \mathcal{C}(\sigma)$.

Theorem 3.3.2. *Let M be a well typed term. If $M \to M_1$ and $M \to M_2$, then exists a term M_3 such that $M_1 \xrightarrow{*} M_3$ and $M_2 \xrightarrow{*} M_3$; in a diagram*



Proof. Let R_1 and R_2 be the two redexes in M. The proof is by general recursion on $\mathcal{D}(M)$, and then by cases (inversion) on R_1 and R_2. We exploit the obvious symmetry of the problem to reduce the number of cases; in particular, the cases where $R_1 = R_2$ (at top level) are trivial and will not be discussed.

Base Case. If $\mathcal{D}(M) = 1$, M is either a canonical element or a variable of some atomic type A. In both cases it is in normal form, so the case is vacuous.

Inductive Case We suppose the statement holds for all terms N such that $\mathcal{D}(N) < \mathcal{D}(M)$ and prove it for M.

- **Subcase** R_1 is a $\beta - rule$. In this case, M must be of the form $(\lambda x : \sigma.P)\ Q$. and

  $$(\lambda x{:}\sigma.P)\ Q \xrightarrow{\beta} P[Q/x]$$

  R_2 is either inside $(\lambda x{:}\sigma.P)$ or inside Q; in the former case, it is either a μ rule or it is internal to P. We consider the previous three possibilities:
  • R_2 is a μ-rule. Since M is well typed (this is the only case where this hypothesis is used) Q has type σ. By the strong normalisation property it must have a normal form, and by the subject reduction property, this must be a canonical element $a_j \in \mathcal{C}(\sigma)$. Then, by Lemma 2.2.2.(2)

  $$P[Q/x] \xrightarrow{*} P[a_j/x]$$

Now, by the μ-redex we have that:

$$(\lambda x{:}\sigma.P)\ Q \xrightarrow{\mu} \langle P[a_i/x]\rangle_\sigma Q$$

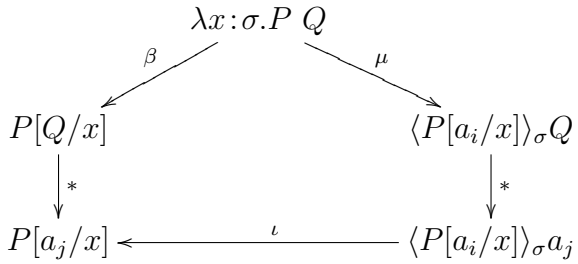
Since $Q \xrightarrow{*} a_j$ we have:

$$\langle P[a_i/x]\rangle_\sigma Q \xrightarrow{*} \langle P[a_i/x]\rangle_\sigma\ a_j$$

Applying the ι-rule we get:

$$\langle P[a_i/x]\rangle_A\ a_j \xrightarrow{\iota} P[a_j/x]$$

Summing up:



- $R_2 : P \to P'$, so that

$$(\lambda x{:}\sigma.P)\ Q \to (\lambda x{:}\sigma.P')\ Q.$$

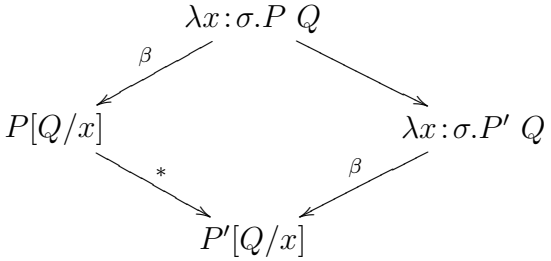
By Lemma 2.2.2.(1)

$$P[Q/x] \xrightarrow{*} P'[Q/x]$$

and by β-reduction:

$$(\lambda x{:}\sigma.P')\ Q \to P'[Q/x].$$

Summing up:



- $R_2 : Q \to Q'$, so that

$$(\lambda x{:}\sigma.P)\ Q \to (\lambda x : \sigma.P)\ Q'.$$

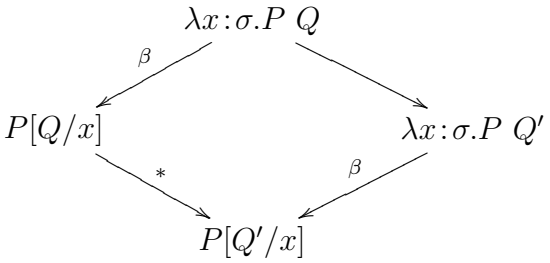
By Lemma 2.2.2.(2)

$$P[Q/x] \xrightarrow{*} P[Q'/x]$$

and by β-reduction:

$$(\lambda x{:}\sigma.P)\ Q \to P[Q'/x]$$

Summing up:



- **Subcase** R_1 is μ-rule, that is

$$\lambda x{:}\sigma.P \xrightarrow{\mu} \langle P[a_i/x]\rangle_\sigma$$

for $a_i \in \mathcal{C}(\sigma)$. The only not trivial case is when R_2 is λ-rule, that is

$$\lambda x{:}\sigma.P \to \lambda x{:}\sigma.P'$$

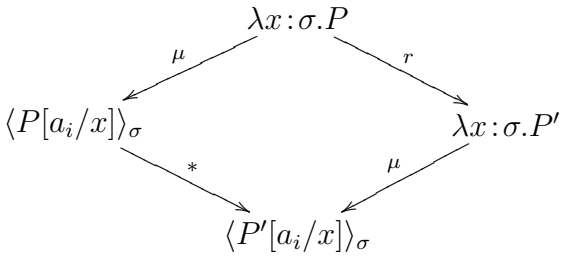
due to some redex $P \xrightarrow{r} P'$.
Since $P \to P'$, by Lemma 2.2.2.(1) we have:

$$\langle P[a_i/x]\rangle_{a_i \in A} \xrightarrow{*} \langle P'[a_i/x]\rangle_\sigma$$

On the other side, by memoization:

$$\lambda x{:}\sigma.P' \xrightarrow{\mu} \langle P'[a_i/x]\rangle_\sigma.$$

In a diagram:



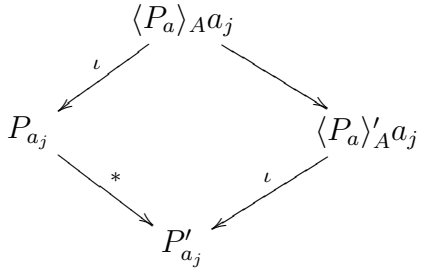
- **Subcase** R_1 is a ι-rule. In this case, M must be of the form: $\langle P_a\rangle_\sigma\ a_j$ with $a_j \in \sigma$. R_2 must eventually occur inside some component of the vector, that is $\langle P_a\rangle_\sigma = \langle v_1, P_{a_i}, v_2\rangle_\sigma$, where $P_{a_i} \to P'_{a_i}$, so that

$$\langle v_1, P_{a_i}, v_2\rangle_\sigma \to \langle v_1, P'_{a_i}, v_2\rangle_\sigma$$

We have two possibilities:

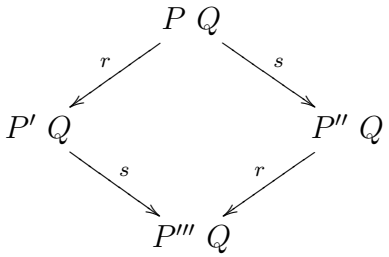
  - $i \neq j$. In this case $\langle v_1, P'_{a_i}, v_2\rangle_\sigma\ a_j \xrightarrow{\iota} P_{a_j}$.
  - $i = j$. Then $\langle v_1, P'_{a_i}, v_2\rangle_\sigma\ a_j \xrightarrow{\iota} P'_{a_j}$. Since P_{a_j} reduces to P'_{a_j} we have done.

In a diagram (latter case):

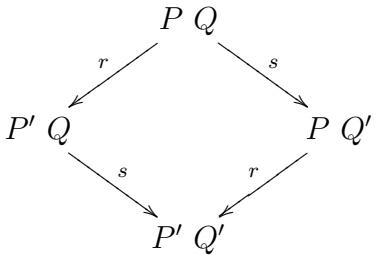


- **Subcase** R_1 is a $@_l$-rule, that is $P \xrightarrow{r} P'$ for some redex r. Omitting symmetric cases and considering that $M = P\,Q$ is an application, R_2 may only be a $@_l$-rule or a $@_r$-rule.
  • In the fist case, $P\,Q \to P''\,Q$, due to some redex $P \xrightarrow{s} P''$. Since $\mathcal{D}(P) < \mathcal{D}(P\,Q)$ we may apply the inductive hypothesis to conclude

that there exists a term P''' such that $P' \xrightarrow{*} P'''$ and $P'' \xrightarrow{*} P'''$. Then, by Lemma 2.2.3.(1) we can close the following diagram.



• In the second case, $P\,Q \xrightarrow{r} P\,Q'$ due to some redex $Q \xrightarrow{s} Q'$. Then we can reduce s in $P'\ Q$ and r in $P\ Q'$ obtaining $P'\ Q'$ which close the diagram.



- **Subcase** R_1 is a $@_r$-rule, that is $Q \xrightarrow{r} Q'$ for some redex r. This case is analogous to the previous one.

- **Subcase** R_1 is λ-rule, that is

$$\lambda x{:}\sigma.P \xrightarrow{\lambda} \lambda x{:}\sigma.P'$$

due to some redex $P \xrightarrow{r} P'$. The only possibility not explored yet (up to symmetries) is that R_2 is itself a λ-rule, that is

$$\lambda x{:}\sigma.P \xrightarrow{\lambda} \lambda x{:}\sigma.P''$$

due to some redex $P \xrightarrow{s} P''$. This is precisely the case discussed in the introduction of this section, so we avoid to repeat the proof.

- **Subcase** R_1 is v-rule. In this case: $M \equiv \langle P_{a_i}\rangle_{a_i\in A}$, and R_2 must be a v-rule as well: that is, both redexes occur inside some component of the vector. If both reductions are relative to the same component P_{a_i} we must use the induction hypothesis for P_{a_i}, together with

Lemma 2.2.3.(4).
If we have a redex $P_{a_i} \xrightarrow{r} P'_{a_i}$ and another redex $P_{a_j} \xrightarrow{s} P'_{a_j}$. with $i \neq j$, the two redexes are disjoint and we close the diagram in the obvious way, by firing the other redex.

□

Chapter 4

Complexity

4.1 Informal discussion

In this chapter, we provide some complexity bounds for the reduction of terms of the finite lambda calculus, in terms of their type. We first prove that we can reduce rank-2 terms in polynomial time and rank-3 terms in polynomial space. Next, we generalize the statement for higher ranks.

However, in the general case, we won't proof directly the bound on terms of rank $2n+2$ (resp. $2n+3$), but we first lower the rank to $2n+2$ ($2n+3$) by means of the **LR** transformation, introduced in the following sections together with definition of rank and level and the reduction strategy.

Our analysis essentially follows the approach in [12], that was in turn inspired by [25].

We shall use the notation $exp_n(p)$ for a tower of exponential of height n, that is, formally:

- $exp_0(p) = p$
- $exp_{n+1}(p) = 2^{exp_n(p)}$

First of all let us give with an informal description of the reduction strategy, together with an example that will help to understand its motivations.

What we call k-strategy is a balance between eager and lazy evaluation. Now, the memoization mechanism is intended to pre-calculate a function f for all its arguments. We can do it as we work with finite types and the complexity of this operation is bounded by the complexity of the function f times the cardinality of its type. Let us show an example of how exploit memoization to speed-up the computation.

Let n be a fixed natural number and $\mathbf{n}$ the corresponding λ-term according to the Church encoding. We define $M \circ N \equiv \lambda z.M\,(N\,z)$ with z fresh. (Terui uses these terms as a base to encode turing machines in [27].) If we reduce weakly the term $(\mathbf{n} \circ \mathbf{2})\ f$, we will get n iteration of $\mathbf{2}\,f$, and eventually 2^n application of f. So if t_f is the complexity of f, the overall complexity of the computation over an input x will be bounded by $t_f(|x|)^{2^n}$. But if we perform memoization of $\mathbf{2}\ f$, as we said above, we obtain a complexity that is bounded by the cardinality of the type of f times the complexity of f. The iteration of this operation will result in a polynomial complexity.

It is worth to note that in this case the number n is *fixed*. We could not define an iteration depending on n; however, simply adding recursion we gain this possibility. See [4] for an analogous example with an iterator term. Moreover, lacking recursion, we cannot encode turing machine with a single term. Indeed, in Terui's work [27], a turing machine with bounded resources is encoded with a family of terms, one term for each input length.

We have seen how memoization may help us to speed-up the computation. However, we must carefully use μ-rule, as if we start to perform memoization on higher order functions, the complexity may grow very fast.

Let us consider, for a base type o, the terms

$$True \equiv \lambda x, y.x : (o \to o \to o)$$

$$False \equiv \lambda x, y.y : (o \to o \to o)$$

which are the usual terms encoding booleans in simply-typed λ-calculus.

Now, if we consider the finite set $\mathcal{B} = \{0, 1\}$, we can obtain terms of our finite λ-calculus instantiating the type o to the set $\mathcal{B}$. (We will come back on this kind of terms in section 4.6, where are provided formal definitions).

Now we can reduce $\lambda x, y.x$ with our rules, especially memoization:

$$\lambda x, y.x \xrightarrow{\mu} \lambda x.\langle x, x\rangle \xrightarrow{\mu} \langle\langle 0, 0\rangle, \langle 1, 1\rangle\rangle$$

In a similar way we can reduce $False$ to $\langle\langle 0, 1\rangle, \langle 0, 1\rangle\rangle$. $True$ and $False$ are terms of level 1, but if we start to work with boolean operators the level and the rank raise.

For example, if we reduce by memoization the rank-2-term

$$\lambda p, q.p\, q\, p$$

which encodes the AND operator, we will get:

$$\begin{array}{l}
\langle \\
\quad \langle(\langle\langle 0, 1\rangle, \langle 0, 1\rangle\rangle)(\langle\langle 0, 1\rangle, \langle 0, 1\rangle\rangle)(\langle\langle 0, 1\rangle, \langle 0, 1\rangle\rangle), \\
\qquad (\langle\langle 0, 1\rangle, \langle 0, 1\rangle\rangle)(\langle\langle 0, 0\rangle, \langle 1, 1\rangle\rangle)(\langle\langle 0, 1\rangle, \langle 0, 1\rangle\rangle)\rangle, \\
\quad \langle(\langle\langle 0, 0\rangle, \langle 1, 1\rangle\rangle)(\langle\langle 0, 1\rangle, \langle 0, 1\rangle\rangle)(\langle\langle 0, 0\rangle, \langle 1, 1\rangle\rangle), \\
\qquad (\langle\langle 0, 0\rangle, \langle 1, 1\rangle\rangle)(\langle\langle 0, 0\rangle, \langle 1, 1\rangle\rangle)(\langle\langle 0, 0\rangle, \langle 1, 1\rangle\rangle)\rangle \\
\rangle
\end{array}$$

Which suggest us that memoization of higher order argument may not be a good idea! If we had an higher order functional with the term AND as argument, reducing it call by value would force us to cope with the duplication of such a huge term, but lazy evaluation avoid this kind of explosion of the argument.

The last two examples should had cleared the reason why we need to mix up lazy and eager evaluation. In a more general view, consider a rank n+3 term applied to a term of level n+1. This term should be reduced eagerly: in this way we avoid duplication of redexes in the argument, propagating only a value trough the term. Even if the argument is reduced by memoization his complexity stays in the required bounds (a tower of exponential of height n).

But if the argument has level greater than n+1, memoizing it will break the complexity bound: this is the reason behind lazy evaluation of higher order arguments.

A similar reason lies behind the lowering of the rank, that allow to work with a rank-$(n+2)$-term; such a term can have argument of level at most $n+1$; so our k-strategy is an eager strategy and evaluates a term in the required bounds. If we had a rank-$(2n+2)$-term, we would have to work with argument of level at most $2n+1$ that, reduced lazily, would be evaluated in an unknown context in a way that make difficult to keep track of reductions above level $n+1$.

Thus, as lowering the rank does not break the complexity bounds (we prove it in section 4.4.3), we use it to work with rank-$(n+2)$-terms.

In the rest of the discussion, we shall use capital letters $T, U, V, \ldots$ to range over types, reserving greek letters for substitutions.

4.2 Preliminaries

We need some definitions about finite lambda terms (level, rank, size, depth) that will be used for expressing our complexity bounds.

Definition 4.2.1. *(Level and Rank) The* level $\ell(T)$ *of a type* T *is defined in the following way:*

- $\ell(A) = 0$ *if* A *is atomic*
- $\ell(T_1 \to T_2) = max\{\ell(T_1)+1, \ell(T_2)\}$

A term $M : T$ *is of* level $\ell(M) = n$ *if* $\ell(T) = n$.
A term M *is of* rank n *if all its subterms are of level* $\leq n$.

Definition 4.2.2. *(Size)*
The size $|M|$ *of a term* M *is defined as follows (where* c *is an arbitrary constant):*

- $|x| = |c| = 1$;

- $|(M\ N)| = |M| + |N| + 1$;
- $|\lambda x{:}T.M| = |M| + 1$;
- $|\langle M_a \rangle_T| = \sum_{a \in \mathcal{C}(T)} |M_a|$

Note that size and dimension (cfr. Definition 3.3.1) coincide on canonical elements.

Definition 4.2.3. *(Depth)*
The depth $Dp(M)$ *of a term* M *is defined as follows (where* c *is an arbitrary constant):*

- $Dp(x) = Dp(c) = 1$;
- $Dp(M\,N) = max\{Dp(M), Dp(N)\} + 1$;
- $Dp(\lambda x{:}T.M) = Dp(M) + 1$;
- $Dp(\langle M_a \rangle_T) = max_{a \in \mathcal{C}(T)} Dp(M_a) + 1$

Definition 4.2.4. *(maxarg) Given a term* M*, we define the maximal dimension of arguments in* M *at level* n*:*

$$Maxarg_{M,n} = max\{\mathcal{D}(\sigma) \mid \sigma \text{ is the type of a subterm of } M \text{ and } \ell(\sigma) \leq n\ \}$$

If $t{:}\sigma \to \tau$ is a subterm of M, and $\ell(\sigma) \leq n$ then

$$Maxarg_{M,n} \geq \mathcal{D}(\sigma \to \tau) = \|\sigma\| \times \mathcal{D}(\tau) \geq \|\sigma\|$$

Moreover, if C is the maximal cardinality of atomic types, then $Maxarg_{M,n+1} \leq exp_n(p_M(C))$ for a suitable polynomial p_M.

4.3 Complexity measures

We assume that the cost in time of each reduction rule is bound by the size of the terms involved in it:

Definition 4.3.1. *(redex cost) Let $r = M \to N$ be single step reduction. Then, $Time(r) = max\{|M|, |N|\}$.*

Definition 4.3.2. *(Time and Space) Let M be closed term and let $\rho : M \xrightarrow{*} M'$ be a normalizing reduction sequence; then*

- *$Time(\rho)$ is the sum of $Time(r)$ for each redex r in ρ;*
- *$Space(\rho)$ is the maximal dimension of terms along ρ.*

4.4 Reduction strategy

The reduction strategy is composed of two phases. In a first phase we reduce the rank of a term from level $2n+2$ ($2n+3$) to level $n+2$ (resp. $n+3$). Then we apply, for $k = n + 1$ a k-strategy, that is a strategy that reduces lazily arguments of degree $> k$ and eagerly those with degree $\leq k$ (see 4.4.4 below for the precise definition). Note that applying a $n + 1$-strategy to a term of rank $n+2$ and level $\leq n+1$ means that all arguments will be reduced eagerly.

Definition 4.4.1. *(rank lowering)*
We define the following function $\mathbf{LR}^{n+1}$:

- $\mathbf{LR}^{n+1}(x) = x$;
- $\mathbf{LR}^{n+1}(\lambda x{:}T.M) = \lambda x{:}A.\mathbf{LR}^{n+1}(M)$;
- $\mathbf{LR}^{n+1}(\langle M_a \rangle_T) = \langle \mathbf{LR}^{n+1}(M_a) \rangle_T$
- $\mathbf{LR}^{n+1}((M\ N)) = M'[\mathbf{LR}^{n+1}(N)/x]$
  if $\mathbf{LR}^{n+1}(M) = \lambda x.M'$ *and* M *has level* $\geq n + 1$
- $\mathbf{LR}^{n+1}((M\ N)) = \mathbf{LR}^{n+1}(M)\ \mathbf{LR}^{n+1}(N)$ *otherwise*

First we state three facts which can easily proved by structural induction on the term.

Fact 4.4.2. *Let M be a term, we have that:*

$Dp(M) \leq |M|$

$Dp([M/N]) \leq Dp(M) + Dp(N)$

$|[M/N]| \leq |M| \cdot |N|$

These facts are useful to prove the following:

Proposition 4.4.3. *Let M be a term of rank $n+1$, without variables of level $n+1$. Then:*

1. $|\mathbf{LR}^{n+1}(M)| \leq 2^{|M|}$
2. $Dp(\mathbf{LR}^{n+1}(M)) \leq |M|$
3. $\mathbf{LR}^{n+1}$ *M does not contain subterms of the form $P\,Q$ with $\ell(P) = n+1$*
4. *if $\ell(M) \leq n$, then rank$(\mathbf{LR}^{n+1}(M)) = n$*

Proof. (Property 1) We proceed by structural induction on the term M.

- The term is of the form $P\ Q$. We have two subcases, by the definition of **LR**.

  • In the first case $\ell(P) \geq n+1$ and $\mathbf{LR}(P) = \lambda x.P'$. (From now we omit the superscript $n+1$). By the definition of **LR**, we have:

  $$|\mathbf{LR}(P\ Q)| = |P'[\mathbf{LR}(Q)/x]|$$

  Recalling that, for each M and N, $|M[N/x]| \leq |M| \cdot |N|$, we get:

  $$\begin{aligned} &|P'[\mathbf{LR}(Q)/x]| \\ &\leq |P'| \cdot |\mathbf{LR}(Q)| \\ &\leq |P'| \cdot 2^{|Q|} \leq 2^{|P\ Q|} \end{aligned}$$

  By induction hypothesis.

  • In the second case $\mathbf{LR}(P) \neq \lambda x.P'$. By the definition of **LR**, we have:

  $$|\mathbf{LR}(P\ Q)| = |\mathbf{LR}(P)\ \mathbf{LR}(Q)|$$

By definition of size and by induction hypothesis we have:

$$|\mathbf{LR}(P)\ \mathbf{LR}(Q)| \leq 1 + 2^{|P|} + 2^{|Q|}$$

and we get the claim by easy calculations.

- The term is of the form $\langle M_a \rangle_T$. By definition of **LR**:

$$|\mathbf{LR}(\langle M_a \rangle_T)| = |\langle \mathbf{LR}(M_a) \rangle_T|$$

By definition, the size is:

$$\sum_{a \in \mathcal{C}(T)} |\mathbf{LR}(M_a)|$$

And by induction hypothesis we get:

$$\sum_{a \in \mathcal{C}(T)} 2^{|M_a|}$$

Doing the calculation we close the case.

- The term is of the form $\lambda x : T.P$.

$$\begin{aligned} &|\mathbf{LR}(\lambda x : T.P)| \\ &\quad = |\lambda x : T.\mathbf{LR}(P)| \\ &\quad \leq 1 + 2^{|P|} \end{aligned}$$

by induction hypothesis and by definition if size. Again, we close doing the calculations.

- If the term is a variable the case is trivial.

□

Proof. (Property 2) We proceed by structural induction on the term M. The proof is similar to the previous one.

- The term is of the form $P\ Q$. We have two subcases.

  • In the first case $\ell(P) \geq n+1$ and $\mathbf{LR}(P) = \lambda x.P'$. By the definition of $\mathbf{LR}$, we have:

  $$Dp(\mathbf{LR}(P\ Q)) = Dp(P'[\mathbf{LR}(Q)/x])$$

  Recalling that, for each M and N, $Dp(M[N/x]) \leq Dp(M) + Dp(N)$, we get:

  $$\begin{aligned} &Dp(P'[\mathbf{LR}(Q)/x]) \\ &\leq Dp(P') + Dp(\mathbf{LR}(Q)) \leq Dp(P') + |Q| \end{aligned}$$

  By induction hypothesis. So, recalling that $Dp(P') \leq |P'|$, we get the claim.

  • In the second case $\mathbf{LR}(P) \neq \lambda x.P'$. By the definition of $\mathbf{LR}$, we have:

  $$Dp(\mathbf{LR}(P\ Q)) = Dp(\mathbf{LR}(P)\ \mathbf{LR}(Q))$$

  By definition of depth and by induction hypothesis we have:

  $$Dp(\mathbf{LR}(P)\ \mathbf{LR}(Q)) \leq 1 + max\{|P|, |Q|\}$$

  we have done.

- The term is of the form $\langle M_a \rangle_T$. By definition of $\mathbf{LR}$:

  $$Dp(\mathbf{LR}(\langle M_a \rangle_T)) = Dp(\langle \mathbf{LR}(M_a) \rangle_T)$$

  By definition, the depth is:

  $$max_{a \in \mathcal{C}(T)} Dp(\mathbf{LR}(M_a)) + 1$$

  And by induction hypothesis we get:

  $$max_{a \in \mathcal{C}(T)} |M_a| + 1$$

  Doing the calculation we close the case.

- The term is of the form $\lambda x : T.P$.

$$\begin{aligned} &Dp(\mathbf{LR}(\lambda x : T.P)) \\ &\quad = Dp(\lambda x : T.\mathbf{LR}(P)) \\ &\quad \leq 1 + |P| \end{aligned}$$

by induction hypothesis and by definition of depth. Again, we close doing the calculations.

- If the term is a variable the case is trivial.

□

Proof. (Property 3) The proof is by induction on the structure of M.

- The term is of the form $P\ Q$. We have two subcases.
  • In the first case $\ell(P) \geq n+1$ and $\mathbf{LR}(P) = \lambda x.P'$. By the definition of **LR**, we have:

$$\mathbf{LR}(P\ Q) = P'[\mathbf{LR}(Q)/x]$$

By induction hypothesis we know that neither $\lambda x.P'$, nor $\mathbf{LR}(Q)$ contain subterms of the form $M\ N$ with $\ell(M) = n+1$. From this we can conclude that the term:

$$\mathbf{LR}(P\ Q) = P'[\mathbf{LR}(Q)/x]$$

does not contain subterms of the form $M\ N$ with $\ell(M) = n+1$.

  • If $\mathbf{LR}(P) \neq \lambda x.P'$ we have that:

$$\mathbf{LR}(P\ Q) = \mathbf{LR}(P)\ \mathbf{LR}(Q)$$

The claim follows immediately from induction hypothesis.

- The term is of the form $\langle M_a \rangle_T$. By definition of **LR**:

$$\mathbf{LR}(\langle M_a \rangle_T) = \langle \mathbf{LR}(M_a) \rangle_T$$

By induction hypothesis no one of the components of the vector has subterms of the form $M\ N$ with $\ell(M) = n+1$. So the whole term cannot have such subterms.

- The term is of the form $\lambda x : T.P$.

$$\mathbf{LR}(\lambda x : T.P) = \lambda x : T.\mathbf{LR}(P)$$

We close the case easily by induction hypothesis.

- If the term is a variable the case is vacuous.

□

Proof. (Property 4) The proof is by induction on the structure of M.

- The term is of the form $P\ Q$. We have two subcases.
  • In the first case $\ell(P) \geq n+1$ and $\mathbf{LR}(P) = \lambda x.P'$. By the definition of **LR**, we have:

$$\mathbf{LR}(P\ Q) = P'[\mathbf{LR}(Q)/x]$$

By hypothesis we know that $rank(P\ Q) = n+1$, so $\ell(P) \leq n+1$ and as in this case $\ell(P) \geq n+1$ we must have that $\ell(P) = n+1$. Even the rank of P is $\leq n+1$ and as **LR** does not increase the rank, $\mathbf{LR}(P) = \lambda x.P'$ must have rank $\leq n+1$. So P' cannot contain subterms of level $n+1$. We must have that $\ell(Q) \leq n$ and $rank(Q) \leq n+1$ we can apply the induction hypothesis obtaining that $\mathbf{LR}(Q)$ has rank n. So, the term

$$\mathbf{LR}(P\ Q) = P'[\mathbf{LR}(Q)/x]$$

can only contain subterms of level $\leq n$, that is, has rank n.

• We fall in this case if $\mathbf{LR}(P) \neq \lambda x.P'$ or P has level $< n+1$ and we have that:

$$\mathbf{LR}(P\ Q) = \mathbf{LR}(P)\ \mathbf{LR}(Q)$$

In the former case we know that $\mathbf{LR}(P) \neq \lambda x.P'$ and, by hypothesis, that $\ell(P\ Q) \leq n$. So, P can have level $n+1$ only if is of the form $\lambda x.P'$. But this is not the case, so $\ell(P) \leq n$. Then we can apply the induction hypothesis to P. Applying the induction hypothesis to Q (which has level $< n$) we get the claim.

In the latter case we have that $\ell(P) \leq n$ and $rank(P) \leq n+1$. We can thus apply the induction hypothesis to P. Again, applying the induction hypothesis to Q too (which has level $< n$) we get the claim.

- The term is of the form $\langle M_a \rangle_T$. By definition of **LR**:

$$\mathbf{LR}(\langle M_a \rangle_T) = \langle \mathbf{LR}(M_a) \rangle_T$$

As the term has rank $n+1$ the single component cannot contain subterms of level $n+1$. So they all have level $\leq n$ and, by induction hypothesis, $rank(\mathbf{LR}(M_a)) = n$ for each $a \in \mathcal{C}(T)$. The the whole term has rank n.

- The term is of the form $\lambda x : T.P$.

$$\mathbf{LR}(\lambda x : T.P) = \lambda x : T.\mathbf{LR}(P)$$

By hypothesis $\lambda x : T.P$ has rank $n+1$ and level $\leq n$, so P must have rank $\leq n$ and level $< n$. So we can apply induction hypothesis to P and close the case.

- If the term is a variable, cannot have level $n+1$ by hypothesis. So has trivially rank n.

□

We give an operational description of the k-strategy by means of a recursive $Eval_k$ function, defined as follows:

Definition 4.4.4. *(Eval) Let $\vec{A}$ be a list of terms either in normal form or*

of level $> k$.

$$\begin{aligned}
Eval_k(M\ N\ \vec{A}) &= \mathit{let}\ v := Eval_k(N) \\
&\quad \mathit{in}\ Eval_k(M\ v\ \vec{A}) \\
&\quad\quad \mathit{if}\ \ell(N) \leq k \\
Eval_k(\lambda x{:}T.M\ N\ \vec{A}) &= Eval_k(M[N/x]\ \vec{A}) \\
&\quad\quad \mathit{if}\ \ell(N) > k\ \mathit{or}\ N = v \\
Eval_k(\langle M_a \rangle_\sigma\ a_i\ \vec{A}) &= Eval_k(M_{a_i}\ \vec{A}) \\
Eval_k(\lambda x{:}T.M) &= Eval_k(\langle M[a_i/x] \rangle_T) \\
Eval_k(\langle \vec{v}, N, \vec{P} \rangle_\tau) &= \mathit{let}\ v := Eval_k(N) \\
&\quad \mathit{in}\ Eval_k(\langle \vec{v}, v, \vec{P} \rangle_\tau)
\end{aligned}$$

In most cases, $Eval_k$ is tail recursive (that means that there is no additional space consumption due to recursion). The only exceptions are the eager evaluation of arguments (occurring in some pending context) and the evaluation of a vector component (occurring inside the vector). For the study of space complexity, we shall count the maximal number of *nested* (not tail recursive) calls of the $Eval_k$ function, together with the size of input arguments. The actual space consumption is a polynomial of the product of the size of the arguments and the number of nested calls.

4.5 Complexity Theorems

Here we finally prove the complexity bounds. We start proving that rank-2 terms can be reduced in polynomial time, rank-3 terms in polynomial space. Next, we prove the generalization both for time and space.

4.5.1 Applicative contexts and closing substitutions

We introduce now two notions that play an essential role in in the complexity analysis.

Definition 4.5.1. *(applicative context)*
A context is a term with a (typed) hole $\mathbf{C}[\_]$. *An* applicative context *is a con-*

text of the kind $(\_\ A_1 \dots A_n)$ *with the hole in head position of the application. It is* k-normal *if*

1. *its level is* $\leq k$ *and*

2. *any argument of level* $\leq k$ *is in normal form.*

Definition 4.5.2. *(Closing substitution)*
Given a term M *with* $FV(M) \subseteq \{x_1, \dots, x_n\}$ *a* closing substitution σ *is a substitution* $[\vec{M_i}/\vec{x_i}]$ *where all* M_i *are closed (hence* $M\sigma$ *is closed, too). We say that the substitution is* k-normal *if any term* M_i *of level* $\leq k$ *is in normal form.*

Suppose to have a term M of rank $\leq k+1$, and free variables of level $\leq k$. Then, all arguments in a closing substitution σ and in an applicative context $\mathbf{C}[\_]$ have level $\leq k$. So, if they are $k-normal$ all arguments are in normal form (and there is only a finite number of them).

4.5.2 Polynomial Bounds

In this section we shall prove that any rank-2 terms can be reduced in polynomial time and rank-3 terms in polynomial space. The proof are almost identical of those provided in [4] were the same problems are addressed for a similar calculus extended with primitive recursion. However we include the proofs for completeness.

Polynomial Time

Theorem 4.5.3. *Let* M *be a term of rank at most* 2*, with free variables of level at most* 1*. Let* $\mathcal{C}$ *be the maximum cardinality of sets in* M*. Then there exist a polynomial* p_M *such that*

- *for any 1-normal closing substitution* σ*,*
- *for any 1-normal applicative context* $\mathbf{C}[\_]$*,*

$$Time(\mathbf{C}[M\sigma]) \leq p_M(C)$$

Proof. Note that we can associate with M a polynomial $q(x)$ such that for all subterms M' of M and for any 1-normal closing substitution σ

$$|M\sigma| \leq q(C)$$

Such a polynomial $q(x)$ will provide a bound to the cost of 1-step reductions relative to subterms of M.

Let $x_i\sigma = B_i$ and $\mathbf{C}[\_] = \_\ A_1 \dots A_n$. All terms B_i, A_j have level ≤ 1, and hence are in normal form.
The proof proceeds by induction on the structure of M.

- If $M = x_i$, the level of x_i must be either 0 or 1:
  - if x_i has level 0, the context must be empty, and $x_i\sigma = B_i$ is already in normal form;
  - if x_i has level 1 then B_i has level 1 and $A_1 \dots A_n$ have level 0. The term $\mathbf{C}[x_i\sigma] = B_i\ A_1 \dots A_n$ is hence a vector applied to some selectors, and can be computed in a number of steps proportional to the size of B;
- If M is a constant, the result is trivial;
- M is an application $(P\ Q)$. Then

  $$\mathbf{C}[(P\ Q)\sigma] = \mathbf{C}[(P\sigma\ Q\sigma)]$$

  We proceed by cases on the level of Q.
  - If $level(Q) = 0$, the reduction starts normalizing $Q\sigma$ to some value v; by induction hypothesis we may assume that

    $$Time(Q\sigma) \leq p_Q(C)$$

    Then, the normalization proceeds with the reduction of $\mathbf{C}[Q\sigma\ v])$; the context $\mathbf{C}'[\_] = \mathbf{C}[\_\ v]$ is still a 1-normal context, so we can apply the induction hypothesis for P, obtaining that

    $$Time(\mathbf{C}[P\sigma\ v]) = Time(\mathbf{C}'[P\sigma] \leq p_P(C)$$

Summing up

$$\begin{aligned} &Time(\mathbf{C}[P\sigma\ Q\sigma]) \\ &\quad \leq Time(Q\sigma) + Time(\mathbf{C}[P\sigma\ v])\} \\ &\quad \leq p_Q(C) + p_P(C) = p_M(C) \end{aligned}$$

– If level of $Q = 1$, $Q\sigma$ will reduce to a vector v by means of memoization. The cost of memoization is bound by the sum of $\mathbf{C}'[Q\sigma]$ for all closing 1-normal substitutions $\mathbf{C}'[\_]$, whose number is bound by some polynomial $p(C)$. So we have:

$$\begin{aligned} &Time(\mathbf{C}[P\sigma\ Q\sigma]) \\ &\quad \leq Time(Q\sigma) + Time(\mathbf{C}[P\sigma\ v])\} \\ &\quad \leq p(C) \cdot p_Q(C) + p_P(C) = p_M(C) \end{aligned}$$

- M is a vector $\langle M_a \rangle_A$. Then, $\mathbf{C}[\langle M_a \rangle_A \sigma] = \mathbf{C}'[\langle M_a \rangle_A \sigma\ a_i]$ that reduces to $\mathbf{C}'[M_{a_i}\sigma\ a_i]$. We use the inductive hypothesis for M_{a_i}.

- M is an abstraction $\lambda x.P$. Let $\mathbf{C}[(\lambda x.P)\sigma] = \mathbf{C}'[(\lambda x.P)\sigma\ A_1]$. The evaluation proceeds reducing the β-redex:

$$\mathbf{C}'[(\lambda x.P)\sigma\ A_1 \rightarrow \mathbf{C}'[P(\sigma[x := A_1])]$$

The cost of this step is bound by $q(C)$.
$Co'[\_]$ and $\sigma' = \sigma[x := A_1]$ are still 1-normal, so we can apply the induction hypothesis to P concluding that there exists a polynomial $p_P(x)$ such that

$$Time(\mathbf{C}'[P(\sigma[x := A_1])]) \leq p_P(C)$$

Hence,

$$Time(\mathbf{C}'[(\lambda x.P)\sigma\ A_1]) \leq q(C) + P_p(C)$$

□

Polynomial Space

We want to prove that any term and rank at most 3 can be reduced to the normal form in polynomial space.

As anticipated in the discussion at the beginning of the chapter,in this case, an eager strategy would not work, since the memoization of a rank-2 function would require exponential space. So, we consider a 2-strategy: we reduce eagerly arguments of level ≤ 1 but lazily arguments of level 2.

Since arguments of level 2 are treated lazily, we need to take into account their space complexity in the inductive statement. We do not know in which context they will be evaluated, but for the fact that it will be 2-normal.

Definition 4.5.4.

for any term M of level 2 with free variables of level ≤ 1 we define $\widehat{Space}(M)$ as the maximum among $Space(\mathbf{C}[M\sigma])$ for any (2-)normal substitution σ and any (2-)normal applicative context $\mathbf{C}[\_]$.

Let us observe that, since M is of level 2 and only contains variables of type ≤ 1, all arguments in σ and C are of level ≤ 1, and hence 2-normal is the same as normal. The notion is well defined since we only have a finite number of normal substitutions and applicative contexts.

Theorem 4.5.5. *Let M be a term of rank at most 3, with free variables of level at most 2. Let $\mathcal{C}$ be the maximum cardinality of sets in M. Then there exist a polynomial p_M such that the following statement is true:*

- *let σ be an arbitrary 2-normal closing substitution,*
- *let $\mathbf{C}[\_]$ be an arbitrary 2-normal applicative context,*
- *let $m, l \geq 1$ be two constants such that for all terms U of level 2 in σ or $\mathbf{C}[\_]$*

$$m \geq \widehat{Space}(U) \quad \text{and} \quad l \geq |U|$$

- *then*

$$Space(\mathbf{C}[M\sigma]) \leq p_M(C) \cdot l + m$$

Corollary 4.5.6. *Let M be a closed term of rank at most 3. Let $\mathcal{C}$ be the maximum cardinality of sets in M. Then there exist a polynomial p_M such*

$$\mathit{Space}(M) \leq p_M(C)$$

Proof of Theorem 4.5.5

The proof is by induction on the structure of M.

- If $M = x_i$, the level of x_i must be ≤ 2.
  Suppose $x_i\sigma = B_i$ and $\mathbf{C}[\_] = \_\ A_1 \dots A_n$. We proceed by cases on the level of x_i:
  - if x_i has level 0, the context must be empty, and $x_i\sigma = B_i$ is already in normal form;
  - if x_i has level 1 then B_i has level 1 and $A_1 \dots A_n$ have level 0, so they are all in normal form by definition of 2-normality. $\mathbf{C}[x_i\sigma] = B_i\ A_1 \dots A_n$ is hence a vector applied to some selectors, and get normalized by a sequence of σ-rules;
  - if x_i has level 2, then $\mathbf{C}[x_i\sigma] = \mathbf{C}[B]$ and

    $$\mathit{Space}(\mathbf{C}[B]) \leq \widehat{\mathit{Space}}(B) \leq m$$

- If M is a constant, the result is trivial;
- M is an application $(P\ Q)$. Then

  $$\mathbf{C}[(P\ Q)\sigma] = \mathbf{C}[(P\sigma\ Q\sigma)]$$

  We proceed by cases on the level of Q.
  - If $\mathit{level}(Q) = 0$, the reduction starts normalizing $Q\sigma$ to some value v (while the context is pending); by induction hypothesis we may assume that

    $$\mathit{Space}(Q\sigma) \leq p_Q(C) \cdot l + m$$

Then, the normalization proceeds with the reduction of $\mathbf{C}[Q\sigma\ v]$); the context $\mathbf{C}'[\_] = \mathbf{C}[\_\ v]$ is still a 2-normal context, so we can apply the induction hypothesis for P, and assume that

$$\mathit{Space}(\mathbf{C}[P\sigma\ v]) = \mathit{Space}(\mathbf{C}'[P\sigma] \leq p_P(C) \cdot l + m$$

Summing up

$$\begin{aligned}
&\mathit{Space}(\mathbf{C}[P\sigma\ Q\sigma]) \\
&\quad \leq max\{|\mathbf{C}[P\sigma\ Q]| + \mathit{Space}(Q\sigma), \mathit{Space}(\mathbf{C}[P\sigma\ v])\} \\
&\quad \leq max\{q(C) \cdot l + p_Q(C) \cdot l + m, p_P(C) \cdot l + m\} \\
&\quad \leq p_M(C) \cdot l + m
\end{aligned}$$

for a suitable polynomial $p_m(x)$.

– If level of $Q = 1$ the situation is similar to the previous one. Suppose that Q has type $A \to B$ where both A and B are atomic; the only difference is that the normalization of $Q\sigma$ to a vector v will require the computation of $(Q\sigma\ a_i)$ for all possible canonical elements of $a_i \in A$. By induction hypothesis, there exists a polynomial $p_Q(x)$ such that for any a_i

$$\mathit{Space}(Q\sigma\ a_i) \leq p_Q(C) \cdot l + m$$

and so the computation of v can be done in polynomial space too.

– Suppose $\mathit{level}(Q) = 2$; in this case, $Q\sigma$ is not reduced but is passed by name to $P\sigma$. The induction hypothesis for Q implies that, for a suitable polynomial p_Q, and any applicative context $\mathbf{C}'[\_]$ (whose arguments must eventually be of level 0)

$$\mathit{Space}(\mathbf{C}'[Q\sigma]) \leq p_Q(C) \cdot l + m$$

that is equivalent to say that

$$\widehat{\mathit{Space}}(Q\sigma) \leq p_Q(C) \cdot l + m$$

Moreover,

$$|Q\sigma| \leq q(C) \cdot L$$

Hence, the context $\mathbf{C}''[\_] = \mathbf{C}[(\_\ Q\sigma)]$, the new size bound $l' = q(C) \cdot l$ and the new space bound $m' = p_Q(C) \cdot l + m$ satisfy the hypothesis of the inductive statement for P. Applying it, we obtain

$$\begin{aligned} &\mathit{Space}(\mathbf{C}[(P\sigma\ Q\sigma)]) \\ &\qquad \leq p_P(C) \cdot (q(C) \cdot l) + p_Q(C) \cdot l + m \\ &\qquad \leq p_M(C) \cdot l + m \end{aligned}$$

for a suitable polynomial $p_M(x)$.

- M is an abstraction $\lambda x.P$. Let $\mathbf{C}[(\lambda x.P)\sigma] = \mathbf{C}'[(\lambda x.P)\sigma\ A_1]$. The evaluation proceeds reducing the β-redex:

$$\mathbf{C}'[(\lambda x.P)\sigma\ A_1 \to \mathbf{C}'[P(\sigma[x := A_1])]$$

$Co'[\_]$ and $\sigma' = \sigma[x := A_1]$ are still 2-normal, so we can apply the induction hypothesis to P concluding that there exists a polynomial $p_P(x)$ such that

$$\mathit{Space}(\mathbf{C}'[P(\sigma[x := A_1])]) \leq p_P(C) \cdot l + m$$

Hence,

$$\begin{aligned} &\mathit{Space}(\mathbf{C}'[(\lambda x.P)\sigma\ A_1]) \\ &\qquad = max\{|\mathbf{C}'[(\lambda x.P)\sigma\ A_1]|, \mathit{Space}(\mathbf{C}'[P\sigma[x := A_1])\} \\ &\qquad \leq p_M(C) \cdot l + m \end{aligned}$$

for a suitable polynomial $P_M(x)$.

- M is a vector $\langle M_a \rangle_A$. Then, $\mathbf{C}[\langle M_a \rangle_A \sigma] = \mathbf{C}'[\langle M_a \rangle_A \sigma\ a_i]$ that reduces to $\mathbf{C}'[M_{a_i}\sigma\ a_i]$. We use the inductive hypothesis for M_{a_i}.

4.5.3 Higher ranks

In this section we generalize the previous results to higher ranks. As explained in the introduction to the chapter, the strategy is different, as we use an auxiliary transformation to lower the rank of the terms.

Time

In this section we prove that any rank-$(2n+2)$ term of level $\leq n+1$, can be normalized in time $O(exp_n(p))$ for some polynomial p.

First we prove an upper-bound on reduction time for a generic term of rank-$(n+2)$. The main result follows as a corollary, thanks to rank lowering.

Theorem 4.5.7. *Let M be a term of rank at most $n+2$, with free variable of level at most $n+1$. Let $Maxarg_M = Maxarg_{M,n+1}$. Then*

- *for any $n+1$-normal closing substitution σ,*
- *for any $n+1$-normal applicative context $\mathbf{C}[\_]$,*

$$Time(\mathbf{C}[M\sigma]) \leq |M| \cdot Maxarg_M^{Dp(M)}$$

Proof. Note that since the context is $n+1$-normal, $\ell(\mathbf{C}[M\sigma]) \leq n+1$, so, if $\ell(M) = n+2$, then $\mathbf{C}[\ ]$ contains at least one argument.

The proof proceeds by induction on the structure of M; we use a $n+1$-strategy (that is, an eager strategy, in this case).

- $M = x_i : T$. Suppose $x_i\sigma = B_i$ and $\mathbf{C}[\_] = \_\ A_1 \dots A_n$. As the level of x_i is at most $n+1$, $\ell(B_i) \leq n+1$ and hence for each A_i, $\ell(A_i) < n+1$. So, all the terms in $\mathbf{C}[x_i\sigma] = B_i\ A_1 \dots A_n$ are in normal form, and the term reduces in a sequence of ι-steps, bound by $Maxarg_M$.
- M is an application $(P\ Q)$. Then

$$\mathbf{C}[(P\ Q)\sigma] = \mathbf{C}[(P\sigma\ Q\sigma)]$$

Q must be of level $\leq n+1$ so, due to the $n+1$-strategy, it will be normalized first in an empty context, resulting in some value v. By induction hypothesis we have:

$$Time(Q\sigma) \leq |Q| \cdot Maxarg^{Dp(Q)}$$

Then the reduction proceeds with reducing P in the context $C[\_\, v]$ that is still n+2-normal, hence by induction hypothesis we get:

$$Time(\mathbf{C}[P\sigma\, v]) \leq |P| \cdot Maxarg^{Dp(P)}$$

Summing up,

$$\begin{aligned} &Time(\mathbf{C}[(P\sigma\ Q\sigma)]) \\ &\quad \leq Time(Q\sigma) + Time(\mathbf{C}[P\sigma\, v]) \\ &\quad \leq |Q| \cdot Maxarg^{Dp(Q)} + |P| \cdot Maxarg^{Dp(P)} \\ &\quad \leq (|P| + |Q|) \cdot Maxarg^{1+max\{Dp(P),Dp(Q)\}} \\ &\quad \leq |P\ Q| \cdot Maxarg^{Dp(PQ)}. \end{aligned}$$

- M is an abstraction $\lambda x : T.P$. There are two subcases: either the context is empty or not.
  If the context is empty, then $\ell(M) \leq n+1$. The evaluation proceeds by memoization, computing for each $a \in \mathcal{C}(T)$, the component $P\sigma[a/x]$ in an empty context. The substitution $\sigma' = \sigma[a/x]$ is still a closing substitution, so by induction hypothesis,

$$Time(P\sigma[a/x]) \leq |P| \cdot Maxarg_P^{Dp(P)}$$

and hence

$$\begin{aligned} &Time((\lambda x : T.P)\sigma) \\ &\quad \leq \|T\| \cdot |P| \cdot Maxarg_P^{Dp(P)} \\ &\quad \leq |P| \cdot Maxarg_{\lambda x:T.P\sigma} \cdot Maxarg_P^{Dp(P)} \\ &\quad \leq |M| \cdot Maxarg_M^{Dp(M)} \end{aligned}$$

If the context is not empty, let

$$\mathbf{C}[(\lambda x{:}T.P)\sigma] = \mathbf{C}'[(\lambda x{:}T.P)\sigma\ a]$$

The evaluation proceeds reducing the β-redex:

$$\mathbf{C}'[(\lambda x{:}T.P)\sigma\ a] \to \mathbf{C}'[P(\sigma[a/x])]$$

The time required by the β-redex is $\leq |M| \cdot Maxarg_M$. Moreover, $\mathbf{C}'[\_]$ and $\sigma' = \sigma[a/x]$ are $n+1$ normal. So by the induction hypothesis:

$$Time(\mathbf{C}'[P\sigma']) \leq |P| \cdot Maxarg_P^{Dp(P)}$$

Then, the total time is given by:

$$\begin{aligned} &Time(\mathbf{C}[(\lambda x{:}T.P)\sigma]) \\ &\quad \leq |M| \cdot Maxarg_M + Time(\mathbf{C}'[P\sigma']) \\ &\quad \leq |M| \cdot Maxarg_M + |P| \cdot Maxarg_P^{Dp(P)} \\ &\quad \leq |M| \cdot (Maxarg_M \cdot Maxarg_M^{Dp(P)}) \\ &\quad \leq |M| \cdot Maxarg_M^{Dp(M)} \end{aligned}$$

- M is a vector $\langle P_{a_i} \rangle_T$ where $a_i \in \mathcal{C}(T)$. Again, either the context is empty or not.
  In the first case, $\ell(M) \leq n+1$. To reduce M to a normal form, we have to normalize (in an empty context) every component of the vector. For each $P_{a_i}\sigma$ we can apply the induction hypothesis:

$$Time(P_{a_i}\sigma) \leq |P_{a_i}| \cdot Maxarg_{P_{a_i}}^{Dp(P_{a_i})}$$

So the total time needed to compute M is:

$$\begin{aligned} &Time(\langle P_{a_i} \rangle_T) \\ &\quad \leq \textstyle\sum_{a_i \in \mathcal{C}(T)} (|P_{a_i}| \cdot Maxarg_{P_{a_i}}^{Dp(P_{a_i})}) \\ &\quad \leq (\textstyle\sum_{a_i \in \mathcal{C}(T)} |P_{a_i}|) \cdot Maxarg_M^{Dp(M)} \\ &\quad \leq |M| \cdot Maxarg_M^{Dp(M)} \end{aligned}$$

If the context is not empty, let

$$\mathbf{C}[\langle P_{a_i} \rangle_T \sigma] = \mathbf{C}'[\langle P_{a_i} \rangle_T \sigma\ a]$$

By the ι-rule we have:

$$\mathbf{C}'[\langle P_{a_i}\rangle_T\sigma\ a] \to \mathbf{C}'[P_a\sigma]$$

This step is bounded by: $\|T\| \leq Maxarg_M$. By the induction hypothesis for $P_a\sigma$ we have:

$$Time(P_a\,\sigma) \leq |P_a| \cdot Maxarg_{P_a}^{Dp(P_a)}$$

So

$$\begin{aligned} &Time(M\sigma)\\ &\quad \leq Maxarg_M + Time(P_a\sigma)\\ &\quad \leq Maxarg_M + |P_a| \cdot Maxarg_{P_a}^{Dp(P_a)}\\ &\quad \leq |M| \cdot Maxarg_M^{Dp(M)} \end{aligned}$$

□

Corollary 4.5.8. *Let M be a term of level $\leq n+1$, rank $\leq 2n+2$ with free variables of level $\leq n+1$. Let C be the maximal cardinality of atomic types. Then, for some polynomial p:*

$$Time(M) \leq exp_n(p(C))$$

Proof. Iterating n time **LR** over the term M of rank $2n+2$ and level $\leq n+1$, we obtain a term M' of rank $n+2$:

$$M' = \mathbf{LR}^{n+3}(\ldots \mathbf{LR}^{2n+2}(M))$$

By properties 1) and 2) in 4.4.3, M' can be computed in time $exp_n(r(C))$ for a suitable polynomial r where moreover $|M'| \leq exp_n(r(C))$ and $Dp(M) \leq exp_{n-1}(r(C))$. By Theorem 4.5.7, M' can be normalized in time

$$|M'| \cdot Maxarg_{M'}^{Dp(M')} \leq exp_n(r(C)) \cdot Maxarg_{M'}^{exp_{n-1}(r(C))}$$

and since $Maxarg_{M'} \leq exp_n(q)$ the claim follows by easy computations. □

Space

Again, we first prove an upper-bound on reduction space of a generic rank-$(n+3)$-term of level $\leq n+2$. As we anticipated, we split the proof in two steps: first we provide an upper bound to the number $Ncalls(M)$ of *nested* (not tail-recursive) calls to the *Eval* during the computation of $Eval(M)$; next we give an upper bound $Asize(M)$ to the size of the input arguments of *all* recursive call to *Eval*.

As we treat lazily arguments of level $n+2$, we do not know in which context they will be evaluated, that motivates the following definition:

Definition 4.5.9. *for any term M of level $n+2$ with free variables of level $\leq n+1$ we define*

$$\widehat{Ncalls}(M) \quad (resp. \quad \widehat{Asize}(M))$$

as the maximum among $Ncalls(\mathbf{C}[M\sigma])$ (resp. $Asize(\mathbf{C}[M\sigma])$) for any (n+1)-normal substitution σ and any (n+1)-normal applicative context $\mathbf{C}[\_]$.

Theorem 4.5.10. *Let M be a term of rank $\leq n+3$, with free variables of level $\leq n+2$.*

- *for any n+1-normal closing substitution σ,*
- *for any n+1-normal applicative context $\mathbf{C}[\_]$,*
- *let $l, k, m \geq 1$ be three constants such that for all terms U of level $n+2$ in σ or $\mathbf{C}[\_]$*

$$l \geq |U|, Maxarg_M \quad k \geq \widehat{Ncalls}(U) \quad m \geq \widehat{Asize}(U)$$

then

$$Ncalls(\mathbf{C}[M\sigma]) \leq |M|^{Dp(M)} + k$$

$$Asize(\mathbf{C}[M\sigma]) \leq \max\{l \cdot |M|^{Dp(M)}, m\}$$

Proof. The proof is an induction on the structure of M.
We start proving the bound on nested calls (the two proofs are not mutually recursive).

- $M = x_i : T$.
  Suppose $x_i\sigma = B_i$ and $\mathbf{C}[\_] = \_\ A_1 \dots A_n$. The level of x_i is at most n+2. We distinguish two cases:

  - The level of x_i is at most n+1. In this case, all terms in $\mathbf{C}[x_i\sigma] = B_i\ A_1 \dots A_n$ are in normal form and the term normalizes via a sequence of ι-steps. We have no recursive calls.
  - If the level of x_i is $n+2$, then $\mathbf{C}[x_i\sigma] = \mathbf{C}[B]$ and

$$Ncalls(\mathbf{C}[B]) \leq \widehat{Ncalls}(M) \leq k.$$

- $M = (P\ Q)$. Then

$$\mathbf{C}[(P\ Q)\sigma] = \mathbf{C}[(P\sigma\ Q\sigma)].$$

  We proceed by cases on the level of Q.

  - If $Level(Q) \leq n+1$, the strategy starts reducing, in an empty context, Q to a value v. By the induction hypothesis for Q:

$$Ncalls(Q\sigma) \leq |Q|^{Dp(Q)} + k$$

    Now, the context $\mathbf{C}'[\_] = \mathbf{C}[\_v]$ is still $n+2$ normal, so we can apply the induction hypothesis to P:

$$Ncalls(\mathbf{C}'[P\sigma]) \leq |P|^{Dp(P)} + k$$

    The total number of nested calls is given by:

$$\begin{aligned} &Ncalls(\mathbf{C}[P\ Q]\sigma) \\ &\quad \leq max\{1 + Ncalls(Q\sigma), Ncalls(\mathbf{C}'[P\sigma])\} \\ &\quad \leq max\{1 + |Q|^{Dp(Q)} + k, |P|^{Dp(P)} + k\} \\ &\quad \leq |M|^{Dp(M)} + k \end{aligned}$$

- $Level(Q) = n+2$. The strategy acts lazily. By induction hypothesis it holds that, for each (n+1-normal) context $\mathbf{C}'[\_]$:

$$Ncalls(\mathbf{C}'[Q\sigma]) \leq |Q|^{Dp(Q)} + k$$

that is to say,

$$\widehat{Ncalls}(Q) \leq |Q|^{Dp(Q)} + k$$

The new context $\mathbf{C}''[\_] = \mathbf{C}[\_Q\sigma]$ and the constant $k' = |Q|^{Dp(Q)} + k$ satisfy the induction hypothesis for P. So we have that:

$$\begin{aligned} Ncalls(\mathbf{C}[(P\ Q)\sigma]) &= Ncalls(\mathbf{C}''[P\sigma]) \\ &\leq |P|^{Dp(P)} + k' \\ &\leq |P|^{Dp(P)} + |Q|^{Dp(Q)} + k \\ &\leq |M|^{Dp(M)} + k \end{aligned}$$

- $M = (\lambda x{:}T.P)$. We have two cases as in the time case.

  - The context is empty: the term reduces by the μ-rule

$$\mathbf{C}[(\lambda x{:}T.P)\sigma] \to \mathbf{C}[\langle P(\sigma[a/x])\rangle_T]$$

The substitution $\sigma' = \sigma[a/x]$ is still $n+1$-normal, hence we can apply the induction hypothesis for P:

$$Ncalls(\mathbf{C}[P\sigma']) \leq |P|^{Dp(P)} + k$$

The total number of nested calls is:

$$\begin{aligned} &Ncalls(\mathbf{C}[M\sigma]) \\ &\quad\leq \max_{a\in\mathcal{C}(T)} Ncalls(\mathbf{C}[P\sigma']) \\ &\quad\leq |P|^{Dp(P)} + k \\ &\quad\leq |M|^{Dp(M)} + k \end{aligned}$$

  - If the context is not empty, let

$$\mathbf{C}'[(\lambda x{:}T.P)\sigma] = \mathbf{C}[(\lambda x{:}T.P)\sigma\ a]$$

Applying the β-rule we get:

$$\mathbf{C}[(\lambda x{:}T.P)\sigma\, a] \to \mathbf{C}[P(\sigma[a/x])]$$

Applying the induction hypothesis to $\mathbf{C}[P(\sigma[a/x])]$:

$$\mathit{Ncalls}(\mathbf{C}'[P\sigma']) \leq |P|^{\mathit{Dp}(P)} + k$$

Hence we have:

$$\begin{aligned}\mathit{Ncalls}(\mathbf{C}[M\sigma]) &\leq \mathit{Ncalls}(\mathbf{C}'[P\sigma'])\\ &\leq |P|^{\mathit{Dp}(P)} + k \leq |M|^{\mathit{Dp}(M)} + k\end{aligned}$$

- M is a vector $\langle P_a\rangle_T$, where P_a is indexed over $a \in \mathcal{C}(T)$. Again we have to distinguish two cases. If the context $\mathbf{C}[\_]$ is empty, we normalize the term by normalizing, in an empty context, the single components. We can apply the induction hypothesis for the single components:

$$\mathit{Ncalls}(P_a\sigma) \leq (|P_a|)^{\mathit{Dp}(P_a)} + k$$

We have:

$$\begin{aligned}&\mathit{Ncalls}(\mathbf{C}[M\sigma])\\ &\quad\leq 1 + \max_{a\in\mathcal{C}(T)} \mathit{Ncalls}(\mathbf{C}[P\sigma'])\\ &\quad\leq 1 + |P_a|^{\mathit{Dp}(P_a)} + k\\ &\quad\leq |M|^{\mathit{Dp}(M)} + k\end{aligned}$$

If the context is not empty, let $\mathbf{C}'[\_] = \mathbf{C}[\_\, a_i]$ the ι-rule is triggered

$$\mathbf{C}[\langle P_a\rangle_T\, \sigma\, a_i] \to \mathbf{C}[P_{a_i}\sigma]$$

By the induction hypothesis for P_{a_i},

$$\mathit{Ncalls}(\mathbf{C}[P_{a_i}\sigma]) \leq |P_{a_i}|^{\mathit{Dp}(P_{a_i})} + k$$

Then we get

$$\begin{aligned}&\mathit{Ncalls}(\mathbf{C}[M\sigma]) \leq\\ &\quad |P_{a_i}|^{\mathit{Dp}(P_{a_i})} + k \leq |M|^{\mathit{Dp}(M)} + k\end{aligned}$$

Now we prove the bound for the size of the arguments.

- $M = x_i : T$.
  Suppose $x_i\sigma = B_i$ and $\mathbf{C}[\_] = \_\ A_1 \ldots A_n$. The level of x_i is at most $n{+}2$. We distinguish two cases:
  - The level of x_i is $< n{+}2$. In this case, $Asize(\mathbf{C}[B]) \leq Maxarg_M \leq m$.
  - The level of x_i is $n+2$. Then, $Asize(\mathbf{C}[B]) \leq \widehat{Asize}(M) \leq m$
- $M = (P\ Q)$. Then $\mathbf{C}[(P\ Q)\sigma] = \mathbf{C}[(P\sigma\ Q\sigma)]$.
  We proceed by cases on the level of Q.
  - If $Level(Q) \leq n+1$, the strategy starts reducing, in an empty context, Q to a value v. By the induction hypothesis for Q:

    $$Asize(Q\sigma) \leq \max\{l \cdot (|Q|)^{Dp(Q)}, m\}$$

    Now, the context $\mathbf{C}'[\_] = \mathbf{C}[\_v]$ is still $n+1$ normal, so we can apply the induction hypothesis to P:

    $$Asize(\mathbf{C}'[P\sigma]) \leq \max\{l \cdot (|P|)^{Dp(P)}, m\}$$

    So, the total size is given by:

    $$\begin{aligned} &Asize(\mathbf{C}[P\ Q\sigma]) \\ &\quad \leq \max\{|M|, Asize(Q\sigma), Asize(\mathbf{C}'[P\sigma])\} \\ &\quad \leq \max\{|M|, l \cdot (|P|)^{Dp(P)}, l \cdot (|Q|)^{Dp(Q)}, m\} \\ &\quad \leq \max\{l \cdot (|M|)^{\max\{Dp(P), Dp(Q\})+1}, m\} \\ &\quad \leq \max\{l \cdot (|M|)^{Dp(M)}, m\} \end{aligned}$$

  - $Level(Q) = n+2$. The strategy acts lazily, then we need to take into account the size of the parameters of the argument Q. By induction hypothesis it holds that, for each ($n{+}1$-normal) context $\mathbf{C}'[\_]$:

    $$Asize(\mathbf{C}'[Q\sigma]) \leq \max\{l \cdot (|Q|)^{Dp(Q)}, m\}$$

that is to say,

$$\widehat{Asize}(Q) \leq \max\{l \cdot (|Q|)^{Dp(Q)}, m\}$$

The context $\mathbf{C}''[\_] = \mathbf{C}[\_Q\sigma]$ and the new constants $m' = \max\{l \cdot (|Q|)^{Dp(Q)}, m\}$, and $l' = l \cdot |Q|$ satisfy the induction hypothesis for P. So we have that:

$$\begin{aligned} Asize(\mathbf{C}[(P\ Q)\sigma]) &= Asize(\mathbf{C}''[P\sigma]) \\ &\leq \max\{l' \cdot (|P|)^{Dp(P)}, m'\} \\ &\leq \max\{l \cdot |Q| \cdot (|P|)^{Dp(P)}, l \cdot (|Q|)^{Dp(Q)}, m\} \\ &\leq \max\{l \cdot (|M|)^{\max\{Dp(P),Dp(Q)\}+1}, m\} \\ &\leq \max\{l \cdot (|M|)^{Dp(M)}, m\} \end{aligned}$$

- $M = (\lambda x{:}T.P)$. We have two cases.

  - The context is empty; the term reduces by a μ-rule

    $$\mathbf{C}[(\lambda x{:}T.P)\sigma] \to \mathbf{C}[\langle P(\sigma[a/x])\rangle_T]$$

    The size of arguments in this step is bounded by:

    $$\begin{aligned} \textstyle\sum_{a\in\mathcal{C}(T)} |P[a/x]| &\leq \|T\| \cdot |P| \\ &\leq Maxarg_M \cdot |P\downarrow| \leq l \cdot |P| \end{aligned}$$

    The substitution $\sigma' = \sigma[a/x]$ is still $n+2$-normal, hence we can apply the induction hypothesis for P:

    $$Asize(\mathbf{C}[P\sigma']) \leq \max\{l \cdot (|P|)^{Dp(P)}, m\}$$

    The total size of the arguments is given by:

    $$\begin{aligned} Asize(\mathbf{C}[M\sigma]) &\leq \max\{l \cdot |P|, l \cdot (|P|)^{Dp(P)}, m\} \\ &\leq \max\{l \cdot (|M|)^{Dp(M)}, m\} \end{aligned}$$

  - If the context is not empty, let

    $$\mathbf{C}'[(\lambda x{:}T.P)\sigma] = \mathbf{C}[(\lambda x{:}T.P)\sigma\, a]$$

Applying the β-rule we get:

$$\mathbf{C}[(\lambda x{:}T.P)\sigma\, a] \to \mathbf{C}[P(\sigma[a/x])]$$

The size of arguments in this step is bounded by:

$$|P[a/x]| \leq l \cdot |P|$$

$\mathbf{C}'[\_]$ and $\sigma' = \sigma[a/x]$ are still $n+2$-normal, so by induction hypothesis:

$$Asize(\mathbf{C}'[P\sigma']) \leq \max\{l \cdot (|P|)^{Dp(P)}, m\}$$

Hence:

$$\begin{aligned} Asize(\mathbf{C}[M\sigma]) &\leq \max\{l \cdot |P|, Asize(\mathbf{C}'[P\sigma'])\} \\ &\leq \max\{l \cdot (|M|)^{Dp(M)}, m\} \end{aligned}$$

- M is a vector $\langle P_a \rangle_T$. where P_a is indexed over $a \in \mathcal{C}(T)$. Again we have to distinguish two cases. If the context $\mathbf{C}[\_]$ is empty, we normalize the term by normalizing, in an empty context, the single components. We can apply the induction hypothesis for the single components:

$$Asize(P_a\sigma) \leq \max\{l \cdot |P_a|^{Dp(P_a)}, m\}$$

The size of the vector is given by

$$\sum_{a \in \mathcal{C}(T)} |P_a| \leq ||T|| \cdot |M| \leq l \cdot |M|$$

So, the overall size of parameters is:

$$\begin{aligned} &Asize(\mathbf{C}[M\sigma]) \\ &\quad \leq \max\{l \cdot |M|, \\ &\qquad \max_{a \in \mathcal{C}(T)}\{l \cdot |P_a|^{Dp(P_a)}, m\}\} \\ &\quad \leq \max\{l \cdot |M|^{Dp(M)}, m\} \end{aligned}$$

Now we consider the case of a non-empty context. Let $\mathbf{C}'[\_] = \mathbf{C}[\_\, a_i]$. In this context will be triggered the ι-rule. So

$$\mathbf{C}[\langle P_a \rangle_T \, \sigma \, a_i] \to \mathbf{C}[P_{a_i}\sigma]$$

Hence, the space needed is the maximum among the size of the vector, which is again:

$$\sum_{a \in \mathcal{C}(T)} |P_a| \leq ||T|| \cdot |M| \leq l \cdot |M|$$

and the space needed to reduce the term P_{a_i}. By the induction hypothesis for P_{a_i},

$$Asize(\mathbf{C}[P_{a_i}\sigma]) \leq \max\{l \cdot |P_{a_i}|^{Dp(P_{a_i})}, m\}$$

Then we get

$$\begin{aligned} Asize(\mathbf{C}[M\sigma]) \leq & \\ & \max\{l \cdot |M|, Asize(\mathbf{C}[P_{a_i}\sigma])\} \\ & \max\{l \cdot |M|, l \cdot |P_{a_i}|^{Dp(P_{a_i})}, m\} \\ & \max\{l \cdot |M|^{Dp(M)}, m\} \end{aligned}$$

□

Corollary 4.5.11. *Let M be a term of level $\leq n+1$, rank $\leq 2n+3$, with free variables of level at most $n+2$, for a polynomial p:*

$$Space(M) \leq exp_n(p)$$

Proof. The proof is analogous to the time case. By property 4), $\mathbf{LR}^{n+1}(M)$ has rank n if the term has level $< n$. Again, iterating n time $\mathbf{LR}$ over the term M of rank $2n+3$ and level $\leq n+2$, we obtain a term M' of rank $n+3$:

$$M' = \mathbf{LR}^{n+4}(\ldots \mathbf{LR}^{2n+3}(M))$$

By properties 1) and 2) in 4.4.3, $|M'|$ is bounded by $exp_n(r(C))$ and $Dp(M) \leq exp_{n-1}(r(C))$. By Theorem 4.5.10, M' can be normalized in a space that is polynomially related to the product of:

$$Ncalls(\mathbf{C}[M\sigma]) \leq |M|^{Dp(M)} + k$$

and

$$Asize(\mathbf{C}[M\sigma]) \leq \max\{l \cdot |M|^{Dp(M)}, m\}$$

So, the claim follows by easy calculations.

□

4.6 The boolean test problem

The boolean test problem $BTP(r)$ consists in deciding if a simply typed lambda term of rank r and Boolean type is convertible with $True = \lambda x, y : o.x$. In a recent paper [27], Terui has proved that this problem is n-EXPTIME[1] complete for $r = 2n + 2$, and n-EXPSPACE complete for $r = 2n + 3$.

The hardness part is relatively easy (see [27] for details). For orders 2 and 3, it can proved by encoding, respectively, Boolean circuits and quantified Boolean circuits. Beyond order 3, it is possible to encode Turing machines with bounded time and space; in particular, each Turing machine is encoded by a *family* of λ-terms, one for each input length (see also [6] for an encoding similar to Terui's one). It is worth to observe that adding primitive recursion to the language we get that bit of parametricity that is enough to treat such a family as a single term [11, 21].

To prove that we may compute the normal form of a simply typed boolean term of rank r within the above mentioned complexity bounds, Terui deploys a complex machinery involving Krivine abstract machine, linear logic, Scott models and intersection types. The results obtained in this work, allow us to give a much more direct and syntactical proof. We just need a simple lemma.

Definition 4.6.1. *Let t : Bool be a closed term of the simply typed lambda calculus of type Bool. We define $t^{\mathcal{B}}$ as the term of the finite lambda calculus obtained by instantiating o with the finite set $\mathcal{B} = \{0, 1\}$.*

Lemma 4.6.2. *For any closed term t : Bool of the simply typed λ-calculus, $t \xrightarrow{*} True$ if and only if $t^{\mathcal{B}} \xrightarrow{*} True^{\mathcal{B}}$.*

Proof.

- ($\Rightarrow$) This direction is obvious since the finite λ-calculus is a supersystem of the simply typed lambda calculus.
- ($\Leftarrow$) If $t \not\xrightarrow{*} True$ then $t \xrightarrow{*} False$, and for the same reasons of the previous point, $t^{\mathcal{B}} \xrightarrow{*} False^{\mathcal{B}}$; since the finite lambda calculus is confluent

[1] The class n-EXPTIME is the union of DTIME($exp_n(p)$) for all polynomials p, and similarly n-EXPSPACE is the union of DSPACE($exp_n(p)$).

for closed, well typed terms, we have $t^{\mathcal{B}} \not\xrightarrow{*} \mathit{True}^{\mathcal{B}}$.

□

Corollary 4.6.3. *The boolean test problem $BTP(r)$ is in n-EXPTIME for $r = 2n + 2$, and n-EXPSPACE for $r = 2n + 3$.*

Proof. The term t is convertible with *True* if and only if $t \xrightarrow{*} \mathit{True}$, if and only if, by the previous lemma, $t^{\mathcal{B}} \xrightarrow{*} \mathit{True}^{\mathcal{B}}$. Clearly, $\mathit{rank}(t) = \mathit{rank}(t^{\mathcal{B}})$. By Theorem 4.5.7, if $\mathit{rank}(t^{\mathcal{B}}) = 2n + 2$, $t^{\mathcal{B}}$ can be normalized with complexity n-EXPTIME; by Theorem 4.5.10, if $\mathit{rank}(t^{\mathcal{B}}) = 2n + 3$, $t^{\mathcal{B}}$ can be normalized with complexity n-EXPSPACE. □

Chapter 5

Conclusions

In this thesis we introduced the theory of the *finite lambda calculus*, that is a syntactical instantiation of the simply typed lambda calculus to finite types, exploiting the possibility to unfold a function into a finite number of cases (memoization rule). We started the study of its meta-theory, proving a large number of metatheoretical results (subject reduction, strong normalization, confluence), as well as bounds on the complexity of reduction.

The finite lambda calculus is intended to be a good basis for a formal and clear way of study the reduction mechanisms. In fact, the framework we presented here has a simple syntactical nature and embeds in an abstract rewrite system many aspect which were present in preceding works, but in a latent way. For instance, the idea of infinite term, which may appear as a technical trick, here gains a clear computational meaning and an immediate intuitive interpretation as the graph of a function.

Many aspects of the calculus still deserve a better investigation; concerning the theory of reduction, it could be interesting to address *standardization*, and see how the calculus can be integrated with a mechanism of *explicit substitutions*. It would also be worth to explore the relation with different reduction techniques, such as environment machines and, especially, optimal reduction.

About complexity, the proofs presented in this work are essentially in-

spired by [12] and can be possibly refined, both in view of providing tighter bounds, as well as more elegant and sharper arguments. The main contribution of the finite lambda calculus is in fact that of providing a theoretical framework inside which one can comfortably address complexity proofs at an adequate formal level.

The meta-theory presented here was indeed formalized in the *matita* theorem prover, although we have not discussed the formalization here.

Finally, a natural extension of the calculus would consist in studying a lazy generalization of the memoization rule to inductive types.

Bibliography

[1] Andreas Abel. Irrelevance in type theory with a heterogeneous equality judgement. In *Foundations of Software Science and Computational Structures*, pages 57–71. Springer, 2011.

[2] Beniamino Accattoli and Delia Kesner. The structural λ-calculus. In *Computer Science Logic*, pages 381–395. Springer, 2010.

[3] Andrea Asperti. *The optimal implementation of functional programming languages*, volume 45. Cambridge University Press, 1998.

[4] Andrea Asperti. Complexity via finite types . *Manuscript submitted for pubblication*, 2014.

[5] Andrea Asperti and Filippo Cuti. The Finite lambda calculus. *Manuscript submitted for pubblication*, 2014.

[6] Andrea Asperti and Harry G Mairson. Parallel beta reduction is not elementary recursive. In *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 303–315. ACM, 1998.

[7] Hendrik Pieter Barendregt, Wil Dekkers, and Richard Statman. *Lambda calculus with types*. Cambridge University Press, 2013.

[8] Małgorzata Biernacka and Olivier Danvy. A concrete framework for environment machines. *ACM Transactions on Computational Logic (TOCL)*, 9(1):6, 2007.

[9] P-L Curien. An abstract framework for environment machines. *Theoretical Computer Science*, 82(2):389–402, 1991.

[10] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. Cambridge University Press, New York, NY, USA, 1989.

[11] Andreas Goerdt. Characterizing complexity classes by general recursive definitions in higher types. *Information and Computation*, 101(2):202–218, 1992.

[12] Andreas Goerdt and Helmut Seidl. Characterizing complexity classes by higher type primitive recursive definitions, part ii. In *Aspects and Prospects of Theoretical Computer Science: 6th International Meeting of Young Computer Scientists, Smolenice, Czechoslovakia, November 19-23, 1990. Proceedings*, volume 464, page 148. Springer Science & Business Media, 1990.

[13] Georges Gonthier. Formal proof – the four-color theorem, 2008.

[14] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O' Connor, Sidi Ould Biha, et al. A machine-checked proof of the odd order theorem. In *Interactive Theorem Proving*, pages 163–179. Springer, 2013.

[15] Georges Gonthier and Assia Mahboubi. An introduction to small scale reflection in coq. *Journal of Formalized Reasoning*, 3(2), 2010.

[16] Erich Grädel. *Finite Model Theory and its applications*, volume 2. Springer Science & Business Media, 2007.

[17] Yuri Gurevich. Algebras of feasible functions. In *in" Proc. 24th Annual IEEE Sympos. Found. Comput. Sci.* Citeseer, 1983.

[18] Neil Immerman. *Descriptive complexity*. Springer Science & Business Media, 1999.

[19] Girard Jean-Yves. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[20] Neil D. Jones. The expressive power of higher-order types or, life without cons. *Journal of Functional Programming*, 11:55–94, 1 2001.

[21] Lars Kristiansen. Complexity-theoretic hierarchies induced by fragments of *goedel's* t. *Theory of Computing Systems*, 43(3-4):516–541, 2008.

[22] Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.

[23] J-J Lévy. *Reduction corrcectes et optimales dans le lambda calcul*. PhD thesis, University of Paris, 1978.

[24] Norell. *Towards a practical programming language based on dependent types theory* . PhD thesis, Department of Computer Science and Engeneering, Chalmer university of tecnology, 2007.

[25] Helmut Schwichtenberg. Elimination of higher type levels in definitions of primitive recursive functionals by means of transfinite recursion. *Studies in Logic and the Foundations of Mathematics*, 80:279–303, 1975.

[26] M. Takahashi. Parallel reductions in lambda-calculus. *Information and Computation*, 118(1):120 – 127, 1995.

[27] Kazushige Terui. Semantic evaluation, intersection types and complexity of simply typed lambda calculus. In *RTA*, volume 15, pages 323–338, 2012.