Theoretical Computer Science 813 (2020) 70–99

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Combining linear logic and size types for implicit complexity

Patrick Baillot, Alexis Ghyselen [*]

Univ Lyon, CNRS, ENS de Lyon, Universite Claude-Bernard Lyon 1, LIP F-69342, Lyon Cedex 07, France

A R T I C L E   I N F O

Article history:
Received 26 October 2018
Received in revised form 12 September 2019
Accepted 19 September 2019
Available online 25 September 2019

Keywords:
Implicit computational complexity
λ-calculus
Linear logic
Type systems
Polynomial time complexity
Size types

A B S T R A C T

Several type systems have been proposed to statically control the time complexity of lambda-calculus programs and characterize complexity classes such as FPTIME or FEXPTIME. A first line of research stems from linear logic and restricted versions of its !-modality controlling duplication. An instance of this is light linear logic for polynomial time computation [5]. A second approach relies on the idea of tracking the size increase between input and output, and together with a restricted recursion scheme, to deduce time complexity bounds. This second approach is illustrated for instance by non-size-increasing types [8]. However, both approaches suffer from limitations. The first one, that of linear logic, has a limited intensional expressivity, that is to say some natural polynomial time programs are not typable. As to the second approach it is essentially linear, more precisely it does not allow for a non-linear use of functional arguments. In the present work we incorporate both approaches into a common type system, in order to overcome their respective constraints. The source language we consider is a lambda-calculus with data-types and iteration, that is to say a variant of Gödel's system T. Our goal is to design a system for this language allowing both to handle non-linear functional arguments and to keep a good intensional expressivity. We illustrate our methodology by choosing the system of elementary linear logic (ELL) and combining it with a system of linear size types. We discuss the expressivity of this new type system, called sEAL, and prove that it gives a characterization of the complexity classes FPTIME and 2k-FEXPTIME, for $k \geq 0$.

1. Introduction

Controlling the time complexity of programs is a crucial aspect of program development. Complexity analysis can be performed on the overall final program and some automatic techniques have been devised for this purpose. However, if the program does not meet our expected complexity bound it might not be easy to track which subprograms are responsible for the poor performance and how they should be rewritten in order to improve the global time bound. Can one instead investigate some methodologies to program while staying in a given complexity class? Can one carry such program construction without having to deal with explicit annotations for time bounds? These are some questions that have been explored by *implicit computational complexity*, a line of research which defines calculi and logical systems corresponding to various complexity classes, such as FP, FEXPTIME, FLOGSPACE …

[*] Corresponding author.
E-mail address: alexis.ghyselen@ens-lyon.fr (A. Ghyselen).

https://doi.org/10.1016/j.tcs.2019.09.032
0304-3975/© 2019 Elsevier B.V. All rights reserved.

State of the art. A first success in implicit complexity was the recursion theoretic characterization of FP [1]. This work on safe recursion leads to languages for polynomial time [2], for oracle functionals or for probabilistic computation [3,4]. Among the other different approaches of implicit complexity one can mention two important threads of work. The first one is issued from linear logic, which provides a decomposition of intuitionistic logic with a modality, !, accounting for duplication. By designing variants of linear logic with weak versions of the ! modality one obtains systems corresponding to different complexity classes, like light linear logic (LLL) for the class FP [5] and elementary linear logic (ELL) for the classes k-FEXPTIME, for $k \geq 0$. [5–7]. These logical systems can be seen as type systems for some variants of lambda-calculi. A key feature of these systems, and the main ingredient for proving their complexity properties, is that they induce a stratification of the typed program into levels. We will thus refer to them as *level-based systems*. Their advantage is that they deal with a higher-order language, and that they are also compatible with polymorphism. Unfortunately from a programming point of view they have a critical drawback: only few and very specific programs are actually typable, because the restrictions imposed to recursion by typing are in fact very strong. A second thread of work relies on the idea of tracking the size increase between the input and the output of a program. This approach is well illustrated by Hofmann's Non-size-increasing (NSI) type system [8]: here the types carry information about the input/output size difference, and the recursion is restricted in such a way that typed programs admit polynomial time complexity. An important advantage with respect to LLL is that the system is algorithmically more expressive, that is to say that far more programs are typable. This has triggered a fertile research line on type-based complexity analysis using ideas of amortized cost analysis [9–11]. Some aspects of higher-order have been addressed [12] but note that this approach deals with complexity analysis and not with the characterization of complexity classes. In particular it does not suggest disciplines to program within a given complexity class. A similar idea is also explored by the line of work on quasi-interpretations [13,14], with a slightly different angle: here the kind of dependence between input and output size can be more general but the analysis is more of a semantic nature and in particular no type system is provided to derive quasi-interpretations. The type system $\mathsf{d}\ell\mathsf{T}$ of [15,16] can be thought of as playing this role of describing the dependence between input and output size, and it allows us to derive time complexity bounds, even though these are not limited to polynomial bounds. Altogether we will refer to these approaches as *size-based systems*. However, they also have a limitation: characterizations of complexity classes have not been obtained for full-fledged higher-order languages, but only for linear higher-order languages, that is to say languages in which functional arguments have to be used at most once (as in [8,14]).

Problematic and methodology. So on the one hand level-based systems manage higher-order but have a poor expressivity, and on the other hand sized-based systems have a good expressivity but do not characterize complexity classes within a general higher-order language... On both sides some attempts have been made to repair these shortcomings but only with limited success: in [17] for instance LLL is extended to a language with recursive definitions, but the main expressivity problem remains; in [14] quasi-interpretations are defined for a higher-order language, but with a linearity condition on functional arguments. The goal of the present work is precisely to improve this situation by reconciling the level-based and the size-based approaches. From a practical point of view we want to design a system which would bring together the advantages of the two approaches. From a fundamental point of view we want to understand how the levels and the input/output size dependencies are correlated, and for instance if one of these two characteristics subsumes the other one.

One way to bridge these two approaches could be to start with a level-based system such as LLL, and try to extend it with more typing rules so as to integrate in it some size-based features. However, a technical difficulty for that is that the complexity bounds for LLL and variants of this system are usually obtained by following specific term reduction strategies such as the *level-by-level* strategy. Enriching the system while keeping the validity of such reduction strategies turns out to be very intricate. For instance this has been done in [17] for dealing with recursive definitions with pattern-matching, but at the price of technical and cumbersome reasonings on the reduction sequences. Our methodology to overcome this difficulty in the present work will be to choose a variant of linear logic for which we can prove the complexity bound by using a measure which decreases for *any* reduction step. So in this case there is no need for specific reduction strategy, and the system is more robust to extensions. For that purpose we use elementary linear logic (ELL), and more precisely the elementary lambda-calculus studied in [18].

Our language. Let us recall that ELL is essentially obtained from linear logic by dropping the two axioms $!A \multimap A$ and $!A \multimap !!A$ for the ! functor (the co-unit and co-multiplication of the comonad). Basically, if we consider the family of types $\mathsf{W} \multimap !^i\mathsf{W}$ (where W is a type for binary words), the larger the integer i, the more computational power we get... This results in a system that can characterize the classes k-FEXPTIME, for $k \geq 0$ [6]. The paper [18] gives a reformulation of the principles of ELL in an extended lambda-calculus with constructions for !. It also incorporates other features (references and multithreading) which we will not be interested in here. Our idea will be to enrich the elementary lambda-calculus by a kind of *bootstrapping*, consisting in adding more terms to the "basic" type $\mathsf{W} \multimap \mathsf{W}$. For instance, we can think of giving to this type enough terms for representing all polynomial time functions. The way we implement this idea is by using a second language. We believe that several equivalent choices could be made for this second language, and here we adopt for simplicity a variant of the language $\mathsf{d}\ell\mathsf{T}$ from [16], a descendant of previous work on linear dependent types [19]. This language is a linear version of system T, that is to say a lambda-calculus with recursion, with types annotated with size expressions. Note that our notion of linearity is more restrictive than some other works on linear system T [20], in particular it is strictly less expressive than non-linear system T. Actually, the type system of our language can be thought of as a linear cousin of sized types [21,22] and we call it sℓT. So on the whole our global language can be viewed as a kind of two-layer system, the lower one used for tuning first-order intensional expressivity, and the upper one for dealing with higher-order computation and non-linear use of functional arguments. We will call it sEAL, for *sized Elementary affine logic* typed λ-calculus.

This article is an extended and improved version of a previous conference paper [23]. Besides giving more intuitions and detailed explanations, the main novelties with respect to the conference article are the following ones:

- full proofs of the results (some lengthy or technical parts are left in an appendix),
- the generalization of the results on the sEAL type system of [23] to its extension with polymorphism,
- some concrete examples of terms typable in sEAL solving non-trivial problems (SAT, QBF_k, $SUBSET\ SUM$, see Section 4.2).

Roadmap. We will first define the language sℓT of sized linear types and investigate its properties (Section 2). Then we will recall the elementary lambda-calculus, define our enriched calculus sEAL and study the reduction properties of this calculus (Section 3). We will then give some examples in sEAL (Section 4). After that we will establish the complexity results (Section 5).

2. Presentation of sℓT and control of the reduction procedure

We present sℓT (for sized linear system T) which is a linear λ-calculus with constructors for base types and a constructor for high-order primitive recursion. Types are enriched with a polynomial index describing the size of the value represented by a term, and this index imposes a restriction on recursions. With this, we are able to derive a weight on terms in order to control the number of reduction steps.

2.1. Syntax of sℓT and type system

The set of *terms* and *values* of sℓT are defined by the following grammars:

$$\begin{aligned} t, u ::= {} & x \mid \lambda x.t \mid t\ u \mid t \otimes u \mid \mathtt{let}\ x \otimes y = t\ \mathtt{in}\ u \mid \mathtt{zero} \mid \mathtt{succ}(t) \mid \mathtt{ifn}(t, u) \\ & \mid \mathtt{itern}(V, t) \mid \epsilon \mid \mathtt{s}_i(t) \mid \mathtt{ifw}(t_0, t_1, u) \mid \mathtt{iterw}(V_0, V_1, t) \mid \mathtt{tt} \mid \mathtt{ff} \mid \mathtt{if}(t, u) \end{aligned}$$

$$\begin{aligned} V, W ::= {} & x \mid \lambda x.t \mid V \otimes W \mid \mathtt{zero} \mid \mathtt{succ}(V) \mid \mathtt{ifn}(V, W) \mid \mathtt{itern}(V, W) \mid \epsilon \\ & \mid \mathtt{s}_i(V) \mid \mathtt{ifw}(V_0, V_1, W) \mid \mathtt{iterw}(V_0, V_1, W) \mid \mathtt{tt} \mid \mathtt{ff} \mid \mathtt{if}(V, W) \end{aligned}$$

with $i \in \{0, 1\}$.

We define free variables and free occurrences as usual, and we work up to α-renaming. Here, we choose the alphabet $\{0, 1\}$ for simplification, but we could have taken any finite alphabet Σ and in this case, the constructors `ifw` and `iterw` would need a term for each letter.

The definitions of the constructors will be more explicit with their reduction rules and their types. For intuition, the constructor $\mathtt{ifn}(t, t')$ defines a function on integers that does a pattern matching on its input, and the constructor $\mathtt{itern}(V, t)$ is such that $\mathtt{itern}(V, t)\ \underline{n} \to^* V^n\ t$, if $\underline{n}$ is the coding of the integer n, that is $\mathtt{succ}^n(\mathtt{zero})$.

Definition 1 *(Substitution).* For an object t with a notion of free variable and substitution we write $t[t'/x]$ the term t in which free occurrences of x have been replaced by t'.

Base reductions in sℓT are given by the rules described in Fig. 1. Note that in the `iterw` rule, the order in which we apply the iterated functions is the reverse of the one for iterators we see usually. In particular, it does not correspond to the reduction defined in [16]. Those base reductions can be applied in contexts C defined by the following grammar:

$$\begin{aligned} C := {} & [] \mid C\ t \mid V\ C \mid C \otimes t \mid t \otimes C \mid \mathtt{let}\ x \otimes y = C\ \mathtt{in}\ t \mid \mathtt{succ}(C) \mid \mathtt{ifn}(C, t) \\ & \mid \mathtt{ifn}(V, C) \mid \mathtt{itern}(V, C) \mid \mathtt{s}_i(C) \mid \mathtt{ifw}(C, t, u) \mid \mathtt{ifw}(t, C, u) \mid \mathtt{ifw}(V, W, C) \\ & \mid \mathtt{iterw}(V_0, V_1, C) \mid \mathtt{if}(C, t) \mid \mathtt{if}(t, C). \end{aligned}$$

We introduce the system of linear types with sizes. First, base types are given by the following grammar:

$$U := \mathsf{W}^I \mid \mathsf{N}^I \mid \mathsf{B} \qquad\qquad I, J, \dots := a \mid n \in \mathbb{N}^* \mid I + J \mid I \cdot J$$

$\mathbb{N}^*$ is the set of non-zero integers. I represents an *index* and a represents an *index variable*. We define for indexes the notions of occurrences of a variable in the usual way, and we work up to renaming of variables. We also define the

$$\begin{array}{rl|rl}
\texttt{if}(V,W)\ \texttt{tt} & \to V & (\lambda x.t)\ V & \to t[V/x] \\
\texttt{if}(V,W)\ \texttt{ff} & \to W & \texttt{let}\ x \otimes y = V \otimes W\ \texttt{in}\ t & \to t[V/x][W/y] \\
\texttt{ifn}(V,W)\ \texttt{zero} & \to W & \texttt{ifn}(V,V')\ \texttt{succ}(W) & \to V\ W \\
\texttt{itern}(V,W)\ \texttt{zero} & \to W & \texttt{itern}(V,V')\ \texttt{succ}(W) & \to \texttt{itern}(V, V\ V')\ W \\
\texttt{ifw}(V_0,V_1,W)\ \epsilon & \to W & \texttt{ifw}(V_0,V_1,V')\ \texttt{s}_i(W) & \to V_i\ W \\
\texttt{iterw}(V_0,V_1,W)\ \epsilon & \to W & \texttt{iterw}(V_0,V_1,V')\ \texttt{s}_i(W) & \to \texttt{iterw}(V_0,V_1,V_i\ V')\ W
\end{array}$$

Fig. 1. Base rules for sℓT.

$$\frac{}{\mathsf{B} \sqsubset \mathsf{B}} \qquad \frac{I \leq J}{\mathsf{N}^I \sqsubset \mathsf{N}^J} \qquad \frac{I \leq J}{\mathsf{W}^I \sqsubset \mathsf{W}^I}$$

$$\frac{E \sqsubset D \qquad D' \sqsubset E'}{D \multimap D' \sqsubset E \multimap E'} \qquad \frac{D \sqsubset E \qquad D' \sqsubset E'}{D \otimes D' \sqsubset E \otimes E'}$$

Fig. 2. Subtyping Rules.

substitution of a variable in an index in the usual way. Then, we can generalize substitution to types, for example $\mathsf{N}^I[J/a] = \mathsf{N}^{I[J/a]}$. The intended meaning is that closed values of type N^I (resp. W^I) will be integers (resp. words) of size (resp. length) at most I.

Definition 2 *(Order on Indexes).* For two indexes I and J, we say that $I \leq J$ if for any valuation ϕ mapping all the variables of I and J to non-zero integers, we have $I_\phi \leq J_\phi$. I_ϕ is I where all variables have been replaced by their value in ϕ, thus I_ϕ is a non-zero integer. We also consider that if $I \leq J$ and $J \leq I$ then $I = J$ (i.e. we take the quotient set for the equivalence relation). Remark that by definition of indexes, we always have $1 \leq I$.

For two indexes I and J, we say that $I < J$ if for any valuation ϕ mapping all the variables of I and J to non-zero integers, we have $I_\phi < J_\phi$. This is not equivalent to $I \leq J$ and $I \neq J$, as we can see with $a \leq a \cdot b$.

For example, we have $a + 1 \leq 2 \cdot a$, $a + b \cdot a = (b+1) \cdot a$ and $a + 1 < a + b + c$. Here we only consider polynomial indexes. This is a severe restriction w.r.t. linear dependent types, used for example in [16,24], in which indexes can use any set of functions described by some rewrite rules. But in the present setting this is sufficient because we only want sℓT to characterize polynomial time computation.

Definition 3. Types are given by the grammar:

$$D, E, F := U \mid D \multimap E \mid D \otimes E.$$

The subtyping order $\sqsubset$ on those types is described in Fig. 2. This definition allows for example the subtyping $\mathsf{N}^{a+1} \multimap \mathsf{W}^{2a} \sqsubset \mathsf{N}^a \multimap \mathsf{W}^{3a}$, meaning that a function taking an integer of size smaller than $a+1$ and returning a word of size at most $2a$ can also be seen as a function taking an integer of size smaller than a and returning a word of size at most $3a$.

Definition 4 *(Variable Contexts). Variables contexts* are denoted $\Gamma, \dots$, with the shape $\Gamma = x_1 : D_1, \dots, x_n : D_n$. We say that $\Gamma \sqsubset \Gamma'$ when Γ and Γ' have exactly the same variables, and for $x : D$ in Γ and $x : D'$ in Γ' we have $D \sqsubset D'$. *Ground variables contexts*, denoted $d\Gamma$, are variables contexts in which all types are base types. We write $\Gamma = \Gamma', d\Gamma$ to denote the decomposition of Γ into a ground variable context $d\Gamma$ and a variable context Γ' in which types are non-base types. This allows us to decompose a context into his duplicable variables $d\Gamma$ and the non-duplicable ones. For a variable context without base type, we denote $\Gamma = \Gamma_1, \Gamma_2$ when Γ is the concatenation of Γ_1 and Γ_2, and Γ_1 and Γ_2 do not have any common variables.

We denote proofs as $\pi \lhd \Gamma \vdash t : D$ and we define an index $\omega(\pi)$ called the *weight* for such a proof. The idea is that the weight will be an upper-bound for the number of reduction steps of t. Note that since $\omega(\pi)$ is an index, this bound can depend on some index variables. The rules for those proofs are described by Fig. 3. Here are some remarks:

- Observe that this system enforces a linear usage of variables of non-base types, this can be seen for instance in binary rules, such as application, where non-base variables (those in Γ, Γ') do not occur both in t and u.
- In the rule for `itern` and `iterw` described in Fig. 3, the index variable a must be a fresh variable. Then $d\Gamma \vdash V : D \multimap D[a+1/a]$ means intuitively that for any index J, we have $d\Gamma \vdash V : D[J/a] \multimap D[J+1/a]$. This will be formalized in Lemma 2. Also, we need some monotonicity with respect to a (expressed here by the condition $E \sqsubset E[a+1/a]$), as it is essential for subtyping (see Lemma 5). Note that in this definition, D is not necessarily monotonous, but it must be a subtype of a monotonous type E. This gives us more freedom on the type D than directly asking for monotonicity. Finally, linearity in this rule is expressed by the impossibility to use higher-order variables in the iterated function, contrary to other notions of linearity for system T [20].

$$\pi \lhd \frac{D \sqsubset E}{\Gamma, x : D \vdash x : E} \qquad \omega(\pi) = 1$$

$$\pi \lhd \frac{\sigma \lhd \Gamma, x : D \vdash t : E}{\Gamma \vdash \lambda x.t : D \multimap E} \qquad \omega(\pi) = 1 + \omega(\sigma)$$

$$\pi \lhd \frac{\sigma \lhd \Gamma, d\Gamma \vdash t : E \multimap D \qquad \tau \lhd \Gamma', d\Gamma \vdash u : E}{\Gamma, \Gamma', d\Gamma \vdash t\ u : D} \qquad \omega(\pi) = \omega(\sigma) + \omega(\tau)$$

$$\pi \lhd \frac{\sigma \lhd \Gamma, d\Gamma \vdash t : D \qquad \tau \lhd \Gamma', d\Gamma \vdash u : E}{\Gamma, \Gamma', d\Gamma \vdash t \otimes u : D \otimes E} \qquad \omega(\pi) = \omega(\sigma) + \omega(\tau) + 1$$

$$\pi \lhd \frac{\sigma \lhd \Gamma, d\Gamma, x : D, y : E \vdash u : F \qquad \tau \lhd \Gamma', d\Gamma \vdash t : D \otimes E}{\Gamma, \Gamma', d\Gamma \vdash \mathtt{let}\ x \otimes y = t\ \mathtt{in}\ u : F} \qquad \omega(\pi) = \omega(\sigma) + \omega(\tau)$$

$$\pi \lhd \frac{}{\Gamma \vdash \mathtt{zero} : \mathsf{N}^I} \qquad \omega(\pi) = 0$$

$$\pi \lhd \frac{J + 1 \leq I \qquad \sigma \lhd \Gamma \vdash t : \mathsf{N}^J}{\Gamma \vdash \mathtt{succ}(t) : \mathsf{N}^I} \qquad \omega(\pi) = \omega(\sigma)$$

$$\pi \lhd \frac{\sigma \lhd \Gamma, d\Gamma \vdash t : \mathsf{N}^I \multimap D \qquad \tau \lhd \Gamma', d\Gamma \vdash u : D}{\Gamma, \Gamma', d\Gamma \vdash \mathtt{ifn}(t, u) : \mathsf{N}^I \multimap D} \qquad \omega(\pi) = \omega(\sigma) + \omega(\tau) + 1$$

$$\pi \lhd \frac{\begin{array}{ccc} D \sqsubset E & E[I/a] \sqsubset F & E \sqsubset E[a+1/a] \\ \multicolumn{2}{c}{\sigma \lhd d\Gamma \vdash V : D \multimap D[a+1/a]} & \tau \lhd \Gamma, d\Gamma \vdash t : D[1/a] \end{array}}{\Gamma, d\Gamma \vdash \mathtt{itern}(V, t) : \mathsf{N}^I \multimap F} \qquad \omega(\pi) = \omega(\tau) + I \cdot (\omega(\sigma) + 1)[I/a]$$

$$\pi \lhd \frac{}{\Gamma \vdash \epsilon : \mathsf{W}^I} \qquad \omega(\pi) = 0$$

$$\pi \lhd \frac{J + 1 \leq I \qquad \sigma \lhd \Gamma \vdash t : \mathsf{W}^J}{\Gamma \vdash \mathtt{s}_i(t) : \mathsf{W}^I} \qquad \omega(\pi) = \omega(\sigma)$$

$$\pi \lhd \frac{\forall i, \sigma_i \lhd \Gamma_i, d\Gamma \vdash t_i : \mathsf{W}^I \multimap D \qquad \tau \lhd \Gamma', d\Gamma \vdash u : D}{\Gamma_1, \Gamma_2, \Gamma', d\Gamma \vdash \mathtt{ifw}(t_1, t_2, u) : \mathsf{W}^I \multimap D} \qquad \omega(\pi) = \omega(\sigma_1) + \omega(\sigma_2) + \omega(\tau) + 1$$

$$\pi \lhd \frac{\begin{array}{ccc} D \sqsubset E & E[I/a] \sqsubset F & E \sqsubset E[a+1/a] \\ \multicolumn{2}{c}{\forall i, \sigma_i \lhd d\Gamma \vdash V_i : D \multimap D[a+1/a]} & \tau \lhd \Gamma, d\Gamma \vdash t : D[1/a] \end{array}}{\Gamma, d\Gamma \vdash \mathtt{iterw}(V_0, V_1, t) : \mathsf{W}^I \multimap F} \qquad \omega(\pi) = \omega(\tau) + I \cdot (\omega(\sigma_1) + \omega(\sigma_2) + 1)[I/a]$$

$$\pi \lhd \frac{}{\Gamma \vdash \mathtt{tt} : \mathsf{B}} \qquad \omega(\pi) = 0$$

$$\pi \lhd \frac{}{\Gamma \vdash \mathtt{ff} : \mathsf{B}} \qquad \omega(\pi) = 0$$

$$\pi \lhd \frac{\sigma \lhd \Gamma, d\Gamma \vdash t : D \qquad \tau \lhd \Gamma', d\Gamma \vdash u : D}{\Gamma, \Gamma', d\Gamma \vdash \mathtt{if}(t, u) : \mathsf{B} \multimap D} \qquad \omega(\pi) = \omega(\sigma) + \omega(\tau) + 1$$

Fig. 3. Type system for sℓT.

2.2. Examples in sℓT

For the sake of conciseness, we may write from now on $\lambda x, y, z.t$ instead of $\lambda x.\lambda y.\lambda z.t$.

Other iterators. Using a function reversing a word rev, that one could construct easily, we can define an iterator on words doing operations in the usual order. We call this iterator Riterw. Formally it is defined by:

$$\mathtt{Riterw}(V_0, V_1, t) := \lambda w.\mathtt{iterw}(V_0, V_1, t)\ (\mathtt{rev}\ w).$$

We have $\mathtt{Riterw}(V_0, V_1, W)\ \underline{w_0 w_1 \dots w_n} \to^* V_{w_0}\ (V_{w_1}\ (\cdots (V_{w_n}\ W) \cdots))$.

We also show that for integers we can construct an iterator $\mathtt{rec}(V, t)$ with:

$$\mathtt{rec}(V, t)\ \underline{n} \to^* V\ \underline{n-1}\ (V\ \underline{n-2}\ (\dots\ (V\ \mathtt{zero}\ t) \dots))$$

and such that the following rule is derivable:

$$\frac{\begin{array}{ccc} D \sqsubset E & E[I/a] \sqsubset F & E \sqsubset E[a+1/a] \\ \multicolumn{2}{c}{d\Gamma \vdash V : D \multimap \mathsf{N}^a \multimap D[a+1/a]} & \Gamma, d\Gamma \vdash t : D[1/a] \end{array}}{\Gamma, d\Gamma \vdash \mathtt{rec}(V, t) : \mathsf{N}^I \multimap F}$$

We first give a term that takes a pair with an object x of type D and an integer n and returns the pair $(V\ n\ x, n+1)$:

$$t_{step} := \lambda r.\mathtt{let}\ x \otimes n = r\ \mathtt{in}\ (V\ n\ x) \otimes \mathtt{succ}(n).$$

We have $d\Gamma \vdash t_{step} : (D \otimes \mathsf{N}^a) \multimap (D[a+1/a] \otimes \mathsf{N}^{a+1})$. Thus, we can iterate on this term using $\mathtt{itern}$ and this gives us the desired iterator:

$$\mathtt{rec}(V, t) := \lambda n.\mathtt{let}\ x \otimes m = (\mathtt{itern}(t_{step}, t \otimes \mathtt{zero})\ n)\ \mathtt{in}\ x.$$

This constructor can be defined likewise for words.

Addition for unary integers. In order to give an example of weight, we define the addition for unary integers in sℓT. It is represented by:

$$\mathtt{add} := \lambda x.\mathtt{itern}(\lambda y.\mathtt{succ}(y), x) : \mathsf{N}^I \multimap \mathsf{N}^J \multimap \mathsf{N}^{I+J}.$$

The typing derivation is:

$$\dfrac{(1) \qquad \dfrac{\dfrac{I+a+1 \leq I+a+1 \qquad \dfrac{\mathsf{N}^{I+a} \sqsubset \mathsf{N}^{I+a}}{x : \mathsf{N}^I, y : \mathsf{N}^{I+a} \vdash y : \mathsf{N}^{I+a}}}{x : \mathsf{N}^I, y : \mathsf{N}^{I+a} \vdash \mathtt{succ}(y) : \mathsf{N}^{I+a+1}}}{\pi_1 \lhd x : \mathsf{N}^I \vdash \lambda y.\mathtt{succ}(y) : \mathsf{N}^{I+a} \multimap \mathsf{N}^{I+a+1}} \qquad \dfrac{\mathsf{N}^I \sqsubset \mathsf{N}^{I+1}}{\pi_2 \lhd x : \mathsf{N}^I \vdash x : \mathsf{N}^{I+1}}}{\dfrac{x : \mathsf{N}^I \vdash \mathtt{itern}(\lambda y.\mathtt{succ}(y), x) : \mathsf{N}^J \multimap \mathsf{N}^{I+J}}{\pi_0 \lhd \cdot \vdash \mathtt{add} : \mathsf{N}^I \multimap \mathsf{N}^J \multimap \mathsf{N}^{I+J}}}$$

where (1) is $\mathsf{N}^{I+a} \sqsubset \mathsf{N}^{I+a}$, $\mathsf{N}^{I+a} \sqsubset \mathsf{N}^{I+J}$, $\mathsf{N}^{I+a} \sqsubset \mathsf{N}^{I+a+1}$, that is the conditions imposed by the iteration rule. We have $\omega(\pi_1) = 2$ and $\omega(\pi_2) = 1$. Thus, the final weight is $\omega(\pi_0) = 1 + (1 + J \cdot (2+1)) = 3J + 2$.

Multiplication for unary integers. We also sketch the multiplication in sℓT. The multiplication can be represented by:

$$\mathtt{mult} := \lambda x.\mathtt{itern}(\lambda y.\mathtt{add}\ x\ y, \mathtt{zero}) : \mathsf{N}^I \multimap \mathsf{N}^J \multimap \mathsf{N}^{I \cdot J}$$

and the typing derivation is:

$$\dfrac{\dfrac{\dfrac{x : \mathsf{N}^I, y : \mathsf{N}^{I \cdot a} \vdash \mathtt{add}\ x\ y : \mathsf{N}^{I \cdot a + I}}{\pi_1 \lhd x : \mathsf{N}^I \vdash \lambda y.\mathtt{add}\ x\ y : \mathsf{N}^{I \cdot a} \multimap \mathsf{N}^{I \cdot a}[a+1/a]} \qquad \dfrac{}{\pi_2 \lhd x : \mathsf{N}^I \vdash \mathtt{zero} : \mathsf{N}^I}}{x : \mathsf{N}^I \vdash \mathtt{itern}(\lambda y.\mathtt{add}\ x\ y, \mathtt{zero}) : \mathsf{N}^J \multimap \mathsf{N}^{I \cdot J}}}{\pi_0 \lhd \cdot \vdash \mathtt{mult} : \mathsf{N}^I \multimap \mathsf{N}^J \multimap \mathsf{N}^{I \cdot J}}$$

With the previous weight for $\mathtt{add}$, we obtain $\omega(\pi_1) = 3 \cdot I \cdot a + 5$. We also have $\omega(\pi_2) = 1$. Thus, we can compute the final weight:

$$\omega(\pi_0) = 1 + \omega(\pi_2) + J \cdot (\omega(\pi_1) + 1)[J/a] = 3IJ^2 + 6J + 2.$$

Note that this way of doing multiplication is not optimal as we iterate on the largest number during the addition. However, this is a good example for the weight, so we presented this version of multiplication instead of a better one.

Addition on binary integers. Now, we define some terms working on integers written in binary, with type W^I, as we will use them later in order to describe our term for SAT (see Section 4). First, we can define an addition on binary integers in sℓT with a control on the number of bits. More precisely, we can give a term $\mathtt{Cadd} : \mathsf{N}^I \multimap \mathsf{W}^{J_1} \multimap \mathsf{W}^{J_2} \multimap \mathsf{W}^I$ such that $\mathtt{Cadd}\ n\ w_1\ w_2$ outputs the least significant n bits of the sum $w_1 + w_2$. For example, $\mathtt{Cadd}$ 3 101 110 = 011, and $\mathtt{Cadd}$ 5 101 110 = 01011. This will usually be used with a n greater than the expected number of bits, the idea being that those extra 0 can be useful for some other programs. The term follows the usual idea for addition: the result is computed bit by bit starting from the right, and we keep track of the carry.

Unary integers to binary integers. We define a term $\mathtt{CUnToBi} : \mathsf{N}^I \multimap \mathsf{N}^J \multimap \mathsf{W}^I$ such that on the input n, n', this term computes the least n significant bits of the representation of n' in binary:

$$\mathtt{CUnToBi} = \lambda n.\mathtt{itern}(\lambda w.\mathtt{Cadd}\ n\ w\ (\mathtt{s_1}(\epsilon)), \mathtt{Cadd}\ n\ \epsilon\ \epsilon).$$

Binary integers to unary integers. We would like a way to compute the unary integer for a given binary integer. However, this function is exponential in the size of its input, so it should have intuitively the type $\mathsf{W}^J \multimap \mathsf{N}^{2^J}$. As we cannot do exponentiation on indexes, it is impossible to write such a function in $\mathtt{s}\ell\mathsf{T}$. Nevertheless, given an additional information bounding the size of this unary word, we can give a term $\mathtt{CBiToUn} : \mathsf{N}^I \multimap \mathsf{W}^J \multimap \mathsf{N}^I$ such that on an input n, w this term computes the minimum between n and the unary representation of w. What is important in this type is the discarding of J. In order to do that, we first define a term $\mathtt{min} : \mathsf{N}^I \multimap \mathsf{N}^J \multimap \mathsf{N}^I$. As N^I is the type of integers of size smaller than I, we have indeed that the minimum between an integer of size smaller than I and an integer of size smaller than J is smaller than I. It is not a precise bound but it is sufficient for its incoming use. The idea to construct $\mathtt{min}$ is to use the following term:

$$t_{step} := \lambda r.\mathtt{let}\; n \otimes m = r \;\mathtt{in}\; \mathtt{ifn}(\lambda m'.\mathtt{succ}(n) \otimes m', n \otimes \mathtt{zero})\; m.$$

The term t_{step} takes a pair of integers (n, m) and if $m = 0$, it does nothing, otherwise it returns $(n + 1, m - 1)$. We can derive the typing $\cdot \vdash t_{step} : (\mathsf{N}^a \otimes \mathsf{N}^J) \multimap (\mathsf{N}^{a+1} \otimes \mathsf{N}^J)$. Thus, we can iterate on this term, and if we iterate n times starting from the pair $(0, n')$ we indeed compute the minimum between n and n':

$$\mathtt{min} = \lambda n, n'.\mathtt{let}\; m \otimes m' = (\mathtt{itern}(t_{step}, \mathtt{zero} \otimes n')\; n) \;\mathtt{in}\; m.$$

Now that we have the term $\mathtt{min}$, we can define the following term:

$$\mathtt{CBiToUn} = \lambda n.\mathtt{iterw}(\lambda n'.\mathtt{min}\; n\; (\mathtt{mult}\; n'\; \underline{2}), \lambda n'.\mathtt{min}\; n\; (\mathtt{succ}(\mathtt{mult}\; n'\; \underline{2})), \mathtt{zero}).$$

2.3. Some intermediate lemmas for the subject reduction

Index variable substitution and subtyping. In order to prove the subject reduction for $\mathtt{s}\ell\mathsf{T}$ and that the weight is a bound on the number of reduction steps of a term, we give some intermediate lemmas.

First, we show that typed values are linked to normal forms. In particular, this theorem shows that a value of type N is indeed of the form $\mathtt{succ}(\mathtt{succ}(\ldots(\mathtt{succ}(\mathtt{zero}))\ldots))$. From this it follows that in this call-by-value calculus, when an argument is of type N, it is the encoding of an integer.

Theorem 1. *Let t be a term in $s\ell T$, if t is closed and has a typing derivation $\vdash t : D$ then t is normal if and only if t is a value V.*

The proof is very common and it can be found in Appendix A.1.1.

We now give a list of intermediate lemmas for which we do not always detail the proofs if they are immediate.

Lemma 1 *(Weakening). Let Δ, Γ be disjoint typing contexts, and $\pi \lhd \Gamma \vdash t : D$. Then, we have a proof $\pi' \lhd \Gamma, \Delta \vdash t : D$ with $\omega(\pi) = \omega(\pi')$.*

Lemma 2 *(Index substitution). Let I be an index.*

1. *Let J_1, J_2 be indexes such that $J_1 \leq J_2$ then $J_1[I/a] \leq J_2[I/a]$.*
2. *Let J_1, J_2 be indexes such that $J_1 < J_2$ then $J_1[I/a] < J_2[I/a]$.*
3. *Let D, D' be types such that $D \sqsubset D'$ then $D[I/a] \sqsubset D'[I/a]$.*
4. *If $\pi \lhd \Gamma \vdash t : D$ then $\pi[I/a] \lhd \Gamma[I/a] \vdash t : D[I/a]$.*
5. *$\omega(\pi[I/a]) = \omega(\pi)[I/a]$.*

Proof. Point 1 and Point 2 are by definition of $\leq$ and $<$, then Point 3 is a direct induction on types using Point 1 for base types. Point 4 and Point 5 are proved by induction on π:

- In the case of $\mathtt{succ}$ or $\mathtt{s}_i$, we use Point 1.
- In the axiom rule, we use Point 3.
- Then, the only interesting cases are iterations. We show here the iteration for integers. Suppose that we have the following proof:

$$\pi \lhd \dfrac{\dfrac{D \sqsubset E \qquad E \sqsubset E[b+1/b]}{\sigma \lhd d\Gamma \vdash V : D \multimap D[b+1/b]} \qquad \dfrac{E[J/b] \sqsubset F}{\tau \lhd \Gamma, d\Gamma \vdash t : D[1/b]}}{\Gamma, d\Gamma \vdash \mathtt{itern}(V, t) : \mathsf{N}^J \multimap F}$$

With $\omega(\pi) = \omega(\tau) + J \cdot (\omega(\sigma) + 1)[J/b]$. We want to prove that $\pi[I/a] \lhd \Gamma[I/a] \vdash \mathtt{itern}(V, t) : \mathsf{N}^{J[I/a]} \multimap F[I/a]$.

By induction hypothesis and Point 3 of Lemma 2 we have

$$D[I/a] \sqsubset E[I/a] \qquad E[I/a] \sqsubset E[b+1/b][I/a] \qquad\qquad E[J/b][I/a] \sqsubset F[I/a]$$
$$\sigma[I/a] \lhd d\Gamma[I/a] \vdash V : D[I/a] \multimap D[b+1/b][I/a] \qquad \tau[I/a] \lhd \Gamma[I/a], d\Gamma[I/a] \vdash t : D[1/b][I/a]$$

By using the fact that b must be a fresh variable in Γ, $d\Gamma$, J and F, we can suppose, by renaming, that b does not occur in I. Then, we obtain:

$$\pi[I/a] \lhd \frac{\begin{array}{ccc} D[I/a] \sqsubset E[I/a] \quad E[I/a] \sqsubset E[I/a][b+1/b] & & E[I/a][J[I/a]/b] \sqsubset F[I/a] \\ \sigma[I/a] \lhd d\Gamma[I/a] \vdash V : D[I/a] \multimap D[I/a][b+1/b] & & \tau[I/a] \lhd \Gamma[I/a], d\Gamma[I/a] \vdash t : D[I/a][1/b] \end{array}}{\Gamma[I/a], d\Gamma[I/a] \vdash \mathtt{itern}(V, t) : \mathsf{N}^{J[I/a]} \multimap F[I/a]}$$

with weight $\omega(\pi[I/a]) = J[I/a] + \omega(\tau)[I/a] + J[I/a] \cdot \omega(\sigma)[I/a][J[I/a]/b]$.
And so $\omega(\pi[I/a]) = \omega(\pi)[I/a]$. □

Lemma 3 *(Monotonic index substitution). Take J_1, J_2 such that $J_1 \leq J_2$.*

1. *Let I be an index, then $I[J_1/a] \leq I[J_2/a]$.*
2. *For any proof π, $\omega(\pi[J_1/a]) \leq \omega(\pi[J_2/a])$.*
3. *Let E be a type. If $E \sqsubset E[a+1/a]$ then $E[J_1/a] \sqsubset E[J_2/a]$ and if $E[a+1/a] \sqsubset E$ then $E[J_2/a] \sqsubset E[J_1/a]$.*

Proof. Point 1 can be proved by induction on indexes, and then Point 2 is just a particular case of Point 1, by Lemma 2.4. Point 3 is proved by induction on the type E, and the proof is detailed in Appendix A.1.2. □

Lemma 4 *(Typing Base Values). If $\pi \lhd \Gamma, d\Gamma \vdash V : U$ then we have a proof $\pi' \lhd d\Gamma \vdash V : U$ with $\omega(\pi) = \omega(\pi')$. Moreover, $\omega(\pi') \leq 1$.*

Indeed, the only rules we can use to type values of base type are the axiom rule with a variable in $d\Gamma$ or the rules for base constructors on integers, words or boolean such as `zero` or `succ`.

Another important lemma is the one for subtyping, it shows that we do not need an explicit rule for subtyping and subtyping does not harm the upper bound derived from typing. Moreover, this lemma is important in order to substitute variables, since the axiom rule allows subtyping.

Lemma 5 *(Subtyping). If $\pi \lhd \Gamma \vdash t : D$ then for all Γ', D' such that $D \sqsubset D'$ and $\Gamma' \sqsubset \Gamma$, we have a proof $\pi' \lhd \Gamma' \vdash t : D'$ with $\omega(\pi') \leq \omega(\pi)$.*

Proof. This can be proved by induction on π. The only interesting cases are for iterations, in which case the property directly follows from Point 2 and Point 3 of Lemma 3. The iteration for integers is detailed in the Appendix A.1.3 □

Term substitution lemma. In order to prove the subject reduction of the calculus, we examine what happens during a substitution of a value in a term. There are two cases, first the substitution of variables with base types, that is to say duplicable variables, and then the substitution of variables with a non-base type for which the type system imposes linearity.

Lemma 6 *(Value Substitution). Suppose that $\pi \lhd \Gamma_1, d\Gamma, x : E \vdash t : D$ and $\sigma \lhd \Gamma_2, d\Gamma \vdash V : E$, then we have a proof $\pi' \lhd \Gamma_1, \Gamma_2, d\Gamma \vdash t[V/x] : D$. Moreover, if E is a base type then $\omega(\pi') \leq \omega(\pi)$. Otherwise, $\omega(\pi') \leq \omega(\pi) + \omega(\sigma)$.*

Proof. This is proved by induction on π. For the base type case, we use Lemma 4 to show that Γ_2 can be ignored, and then as $d\Gamma$ is duplicable, the proof is rather direct. For the non-base case, in multiplicative rules such as application and `if`, the property holds by the fact that x only appears in one of the premises, and so $\omega(\sigma)$ appears only once in the total weight. □

2.4. Subject reduction and upper bound

We can now express the subject-reduction of the calculus and the fact that the weight of a proof strictly decreases during a reduction.

Theorem 2. *Suppose that $\tau \lhd \Gamma \vdash t_0 : D$ and $t_0 \to t_1$, then there is a proof $\tau' \lhd \Gamma \vdash t_1 : D$ such that $\omega(\tau') < \omega(\tau)$.*

The proof of this theorem can be found in Appendix A.1.4. The main difficulty is to prove the statement for base reductions. Base reductions that induce a substitution, like the usual β-reduction, can be proved with Lemma 6. The other interesting cases are the rules for iterators. For such a rule, the subject reduction is given by a good use of the fresh variable given in the typing rule.

As the indexes can only define polynomials, the weight of a sequent can only be a polynomial on the index variables. And so, in sℓT, we can only define terms that work in time polynomial in their inputs.

2.5. Polynomial indexes and degree

For the following section on the elementary affine logic, we need to define a notion of degree of indexes and make clear some properties of this notion.

Definition 5. The indexes can be seen as multi-variables polynomials, and we can define the *degree* of an index I by induction on I.

- $\forall n \in \mathbb{N}^*, d(n) = 0$.
- For an index variable a, $d(a) = 1$.
- $d(I + J) = max(d(I), d(J))$.
- $d(I \cdot J) = d(I) + d(J)$.

This definition of degree is essential for the control of reductions in sEAL, that we present in the following section. We obtain the following property for degree.

Theorem 3 *(Degree). For all indexes I and J, the following properties are verified:*

1. *For all non-zero integer k, we have $I[k/a] \le k^{d(I)} \cdot I[1/a]$.*
2. *If $I \le J$ then $d(I) \le d(J)$.*

Proof. The first point is proved by induction on I. For the second point, let us first show the following lemma.

Lemma 7. *Let I be an index with at most one index variable a. Then, we have $a^{d(I)} \le I \le I[1/a] \cdot a^{d(I)}$.*

This is proved directly by induction on indexes, and it uses the fact that the constant integers in indexes are non-zero, the image of a variable in a valuation is non-zero and an index is always positive.

Now, we prove our theorem by contraposition. Given I, J such that $d(I) > d(J)$, we construct two new indexes called I' and J' that are I and J in which we replaced all variables by a new fresh variable a. The degree stays the same, and we have, by Lemma 7:

$$a^{d(J)+1} \le a^{d(I)} \le I' \text{ and } J' \le a^{d(J)} \cdot J'[1/a].$$

If we replace a by $k = (J'[1/a] + 1)$ (which is a non-zero integer), we obtain:

$$I'[k/a] \ge k^{d(J)+1} \text{ and } J'[k/a] \le k^{d(J)} \cdot (k-1).$$

And so we have $I'[k/a] > J'[k/a]$. We deduce that we have a valuation ϕ that sends all variables of I and J to k such that $I_\phi > J_\phi$, so we do not have $I \le J$. By contraposition, we obtain Point 2 of Theorem 3. □

This second point shows that our notion of degree is well-defined w.r.t. the equivalence relation between indexes.

3. Elementary affine logic and sizes

We work on an elementary affine lambda calculus based on [18] without multithreading and side-effects, that we present here. In order to solve the problem of intensional expressivity of this calculus, we enrich it with constructors for integers, words and booleans, and some iterators on those types following the usual constraint on iteration in elementary affine logic (EAL). Then, based on the fact that the proof of correctness in [18] is robust enough to support functions computable in polynomial time with type $\mathsf{N} \multimap \mathsf{N}$ (see Section A.2 in the appendix for more details), we enrich EAL with the polynomial time calculus defined previously. We call this new language sEAL (EAL with sizes). More precisely, we add the possibility to use first-order sℓT terms in this calculus in order to work on those base types, particularly we can then do controlled iterations for those types. We then adapt the measure used in [18] to sEAL to find an upper-bound on the number of reductions for a term.

$$\frac{}{\Gamma, x : T \mid \Delta \vdash x : T}\ (\text{Lin Ax}) \qquad \frac{}{\Gamma \mid \Delta, x : T \vdash x : T}\ (\text{Glob Ax})$$

$$\frac{\Gamma, x : T \mid \Delta \vdash M : U}{\Gamma \mid \Delta \vdash \lambda x.M : T \multimap U}\ (\lambda) \qquad \frac{\Gamma \mid \Delta \vdash M : U \multimap T \qquad \Gamma' \mid \Delta \vdash N : U}{\Gamma, \Gamma' \mid \Delta \vdash M\ N : T}\ (\text{App})$$

$$\frac{\cdot \mid \Delta \vdash M : T}{\Gamma \mid \Delta', [\Delta] \vdash !M :\ !T}\ (!\ \text{Intro}) \qquad \frac{\Gamma \mid \Delta \vdash M :\ !T \qquad \Gamma' \mid \Delta, x : [T] \vdash N : U}{\Gamma, \Gamma' \mid \Delta \vdash \mathtt{let}\ !x = M\ \mathtt{in}\ N : U}\ (!\ \text{Elim})$$

$$\frac{\Gamma \mid \Delta \vdash M : T \qquad \alpha \text{ fresh in } \Gamma,\ \Delta}{\Gamma \mid \Delta \vdash M : \forall \alpha.T}\ (\forall\ \text{Intro}) \qquad \frac{\Gamma \mid \Delta \vdash M : \forall \alpha.T}{\Gamma \mid \Delta \vdash M : T[U/\alpha]}\ (\forall\ \text{Elim})$$

Fig. 4. Type system for the EAL-calculus.

3.1. An EAL-calculus

First, let us present a λ-calculus for the elementary affine logic. In this calculus, any sequence of reduction terminates in elementary time. The keystone of this proof is the use of the modality "!", called *bang*, inspired by linear logic. As in linear logic, there are restrictions for the duplication of variables in this calculus. Moreover, the bang has a more limited use that in linear logic, this limitation gives birth to the notion of *depth*. This notion is crucial to derive the elementary bound on this calculus. To describe formally this calculus, we follow the presentation from [18] and we encode the usual restrictions in a type system.

Definition 6. The set of *terms* is given by the grammar:

$$M, N := x \mid \lambda x.M \mid M\ N \mid !M \mid \mathtt{let}\ !x = M\ \mathtt{in}\ N.$$

The constructor $\mathtt{let}\ !x = M\ \mathtt{in}\ N$ binds the variable x in N. We define as usual the notion of free variables, free occurrences and substitution.

Definition 7. The *semantic* of this calculus is given by the rules:

$$(\lambda x.M)\ N \to M[N/x] \qquad \mathtt{let}\ !x = !M\ \mathtt{in}\ N \to N[M/x]$$

Those rules can be applied in any context.

We add to this calculus a polymorphic type system that also restrains the possible terms we can write. Types are given by the grammar:

$$T, U := \alpha \mid T \multimap U \mid !T \mid \forall \alpha.T.$$

Definition 8 *(Typing Contexts). Linear variables contexts* are denoted Γ, with the shape $\Gamma = x_1 : T_1, \dots, x_n : T_n$. We write Γ_1, Γ_2 the disjoint union between Γ_1 and Γ_2. *Global variables contexts* are denoted Δ, with the shape $\Delta = x_1 : T_1, \dots, x_n : T_n, y_1 : [T'_1], \dots, y_m : [T'_m]$. We say that $[T]$ is a *discharged type*, as we could see in light linear logic [5,25]. A global variable context $x_1 : T_1, \dots, x_n : T_n, y_1 : [T'_1], \dots, y_m : [T'_m]$ may sometimes be denoted by $\Delta, [\Delta']$. In this case, we consider that $\Delta = x_1 : T_1, \dots, x_n : T_n$ and $\Delta' = y_1 : T'_1, \dots, y_m : T'_m$.

Typing judgments have the shape $\Gamma \mid \Delta \vdash M : T$. The intended meaning is that variables in Γ are used linearly in M while variables in Δ can be used non-linearly in M.

The rules are given in Fig. 4. Observe that all the rules are multiplicative for Γ and, seen from bottom to top, the "! Intro" rule erases linear contexts, non-discharged types and transforms discharged types into usual types. With this, we can see that some restrictions appear in a typed term. First, in $\lambda x.M$, x occurs at most once in M, and moreover, there is no "! Intro" rule below the axiom rule for x. Then, in $\mathtt{let}\ !x = M\ \mathtt{in}\ M'$, x can be used several times in M', but there is exactly one "! Intro" rule below each axiom rule for x. For example, with this type system, we can not type terms like $\lambda x.!x$, $\lambda f, x.f\ (f\ x)$, $\mathtt{let}\ !x = M\ \mathtt{in}\ x$ or $\mathtt{let}\ !x = M\ \mathtt{in}\ !!x$.

With this type system, we obtain as a consequence of the results exposed in [18] that any sequence of reductions of a typed term terminates in elementary time. This proof relies on the notion of depth linked with the modality "!" and a measure on terms bounding the number of reductions for this term. We will adapt those two notions in the following part on sEAL, but for now, let us present some terms and encoding in this EAL-calculus.

3.1.1. Examples of terms in EAL and church integers

First, a useful term $\mathtt{fonct} : \forall \alpha, \alpha'.!(\alpha \multimap \alpha') \multimap !\alpha \multimap !\alpha'$:

$$\mathtt{fonct} := \lambda f, x.\mathtt{let}\ !g = f\ \mathtt{in}\ \mathtt{let}\ !y = x\ \mathtt{in}\ !(g\ y).$$

We sketch the typing derivation for this term.

$$\dfrac{\dfrac{\dfrac{}{f :!(\alpha \multimap \alpha') \mid \cdot \vdash f :!(\alpha \multimap \alpha')} \quad \dfrac{\dfrac{}{x :!\alpha \mid g : [(\alpha \multimap \alpha')] \vdash x :!\alpha} \quad \dfrac{\dfrac{\cdots}{\cdot \mid g : (\alpha \multimap \alpha'), y : \alpha \vdash g\ y : \alpha'}}{\cdot \mid g : [(\alpha \multimap \alpha')], y : [\alpha] \vdash !(g\ y) :!\alpha'}}{x :!\alpha \mid g : [(\alpha \multimap \alpha')] \vdash \texttt{let}\ !y = x\ \texttt{in}\ !(g\ y) :!\alpha'}}{\dfrac{f :!(\alpha \multimap \alpha'), x :!\alpha \mid \cdot \vdash \texttt{let}\ !g = f\ \texttt{in}\ \texttt{let}\ !y = x\ \texttt{in}\ !(g\ y) :!\alpha'}{\dfrac{f :!(\alpha \multimap \alpha') \mid \cdot \vdash \lambda x.\texttt{let}\ !g = f\ \texttt{in}\ \texttt{let}\ !y = x\ \texttt{in}\ !(g\ y) :!\alpha \multimap !\alpha'}{\cdot \mid \cdot \vdash \texttt{fonct} :!(\alpha \multimap \alpha') \multimap !\alpha \multimap !\alpha'}}}}{\dfrac{\cdot \mid \cdot \vdash \texttt{fonct} : \forall \alpha'.!(\alpha \multimap \alpha') \multimap !\alpha \multimap !\alpha'}{\cdot \mid \cdot \vdash \texttt{fonct} : \forall \alpha, \alpha'.!(\alpha \multimap \alpha') \multimap !\alpha \multimap !\alpha'}}$$

This term allows application under a bang. Indeed, the following reduction can be derived:

$$\texttt{fonct}\ !M\ !N \to^* \texttt{let}\ !g = !M\ \texttt{in}\ \texttt{let}\ !y = !N\ \texttt{in}\ !(g\ y) \to^* !(M\ N)$$

Unary integers can be encoded in this calculus as Church integers, with $\mathsf{N} = \forall \alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$. For example, 3 is represented by the term:

$$\underline{3} = \lambda f.\texttt{let}\ !g = f\ \texttt{in}\ !(\lambda x.g\ (g\ (g\ x))) : \mathsf{N}.$$

And we can represent addition and multiplication with type $\mathsf{N} \multimap \mathsf{N} \multimap \mathsf{N}$:

$$\texttt{add} = \lambda n, m, f.\texttt{let}\ !f' = f\ \texttt{in}\ \texttt{let}\ !g = n\ !f'\ \texttt{in}\ \texttt{let}\ !h = m\ !f'\ \texttt{in}\ !(\lambda x.h\ (g\ x)).$$

$$\texttt{mult} = \lambda n, m, f.\texttt{let}\ !g = f\ \texttt{in}\ n(m\ !g).$$

And finally, one can also define an iterator using integers:

$$\texttt{iter} = \lambda f, x, n.\texttt{fonct}\ (n\ f)\ x : \forall \alpha.!(\alpha \multimap \alpha) \multimap !\alpha \multimap \mathsf{N} \multimap !\alpha$$

with $\texttt{iter}\ !M\ !M'\ \underline{n} \to^* !(M^n\ M')$.

3.1.2. *Intensional expressivity*

Those examples show that this calculus suffers from limitation. First, we need to work with Church integers, because of a lack of data structures. Furthermore, we need to be careful with the modality, and this can be sometimes a bit tricky, as one can observe with the addition. And finally if we want to do an iteration, we are forced to work with types with bangs. This implies that each time we need to use an iteration, we are forced to add a bang in the final type. Typically, this prevents from iterating a function which has itself been defined by iteration. It has been proved that polynomial and exponential complexity classes can be characterized in a variant of this calculus with recursive types [26]. For example, with a type for words W and booleans B we have that $!\mathsf{W} \multimap !!\mathsf{B}$ characterizes polynomial time computation. However, because of the restrictions mentioned above, some natural polynomial time programs cannot be typed with the type $!\mathsf{W} \multimap !!\mathsf{B}$. We say that this calculus has a limited intensional expressivity. One goal of this paper is to try to lessen this problem, and for that, we now present an enriched version of this calculus, sEAL, using the language $\mathsf{s}\ell\mathsf{T}$.

3.2. *Syntax and type system for sEAL*

Let us first give some notations on vectors.

Definition 9 *(Vectors).* In the following we will work with vectors of $\mathbb{N}^{n+1}$, for $n \in \mathbb{N}$. We introduce here some notations on those vectors. We usually denote vectors by $\mu = (\mu(0), \dots, \mu(n))$.

When there is no ambiguity with the value of n, for $0 \leq k \leq n$, we note $\mathbb{1}_k$ for the vector μ with $\mu(k) = 1$ and $\forall i, 0 \leq i \leq n, i \neq k, \mu(i) = 0$. We extend this notation for $k > n$. In this case, $\mathbb{1}_k$ is the zero-vector.

Let $\mu_0, \mu_1 \in \mathbb{N}^{n+1}$. We write $\mu_0 \leq \mu_1$ when $\forall i, 0 \leq i \leq n, \mu_0(i) \leq \mu_1(i)$. We also write $\mu_0 \leq_{lex} \mu_1$ for the lexicographic order on vectors.

For $k \in \mathbb{N}$, when there is no ambiguity with the value of n, we write $\tilde{k}$ the vector μ such that $\forall i, 0 \leq i \leq n, \mu(i) = k$.

Then, the concatenation of two vectors is denoted by (μ_0, μ_1), the pointwise addition by $\mu_0 + \mu_1$ and the scalar product by $k \cdot \mu$.

$$\begin{array}{rl|rl}
(\lambda x.M)\ N & \to M[N/x] & \mathtt{let}\ !x = !M\ \mathtt{in}\ N & \to N[M/x] \\
\mathtt{let}\ x \otimes y = M \otimes M'\ \mathtt{in}\ N & \to N[M/x][M'/y] & \mathtt{ifn}(M,N)\ \mathtt{zero} & \to N \\
\mathtt{ifn}(M,M')\ \mathtt{succ}(N) & \to M\ N & \mathtt{iter}^!_{\mathsf{N}}(!M,!N)\ \underline{n} & \to !(M^n\ N) \\
\mathtt{ifw}(M_0,M_1,N)\ \epsilon & \to N & \mathtt{ifw}(M_0,M_1,M')\ \mathtt{s}_i(N) & \to M_i\ N \\
\mathtt{iter}^!_{\mathsf{W}}(!M_0,!M_1,!N)\ \underline{w} & \to !(M^w\ N) & \mathtt{if}(M,N)\ \mathtt{tt} & \to M \\
\mathtt{if}(M,N)\ \mathtt{ff} & \to N & \text{if } t \to t' \text{ in } \mathsf{s\ell T} \text{ then } [t]() & \to [t']()
\end{array}$$

$$\begin{array}{rl}
[\lambda x_n \ldots x_1.t](M_1,\ldots,M_{n-1},\underline{v}) & \to [\lambda x_{n-1}\ldots x_1.t[\underline{v}/x_n]](M_1,\ldots,M_{n-1}) \\
[\underline{v}]() & \to \underline{v}
\end{array}$$

Fig. 5. Base rules for sEAL.

3.2.1. Terms and reductions

Terms of sEAL are defined by the following grammar:

$$\begin{aligned}
M, N ::= {} & x \mid \lambda x.M \mid M\ N \mid !M \mid \mathtt{let}\ !x = M\ \mathtt{in}\ N \mid M \otimes N \mid \mathtt{let}\ x \otimes y = M\ \mathtt{in}\ N \\
& \mid \mathtt{zero} \mid \mathtt{succ}(M) \mid \mathtt{ifn}(M,N) \mid \mathtt{iter}^!_{\mathsf{N}}(M,N) \mid \mathtt{tt} \mid \mathtt{ff} \mid \mathtt{if}(M,N) \\
& \mid \epsilon \mid \mathtt{s}_i(M) \mid \mathtt{ifw}(M_0,M_1,N) \mid \mathtt{iter}^!_{\mathsf{W}}(M_0,M_1,N) \mid [\lambda x_n \ldots x_1.t](M_1,\ldots,M_n)
\end{aligned}$$

with $i \in \{0,1\}$.

Note that the t used in $[\lambda x_n \ldots x_1.t](M_1,\ldots,M_n)$ refers to terms defined in sℓT. This notation means that we call the function t defined in sℓT with arguments $M_1,\ldots,M_n$. Moreover, n can be any integer, even zero. Constructors for iterations directly follow from the ones we can usually define in EAL for Church integers or Church words, as we could see in the previous section on EAL. As usual, we work up to α-isomorphism and we do not explicit the renaming of variables. As before, for words, the choice of the alphabet $\Sigma = \{0,1\}$ is arbitrary, we could have chosen any finite alphabet.

Definition 10 *(Base type values).* We note $\underline{v}$ for base type values, defined by the grammar:

$$\underline{v} := \mathtt{zero} \mid \mathtt{succ}(\underline{v}) \mid \epsilon \mid \mathtt{s}_i(\underline{v}) \mid \mathtt{tt} \mid \mathtt{ff}$$

with $i \in \{0,1\}$.

In particular, if n is an integer and w is a binary word, we note $\underline{n}$ for the base value $\mathtt{succ}^n(\mathtt{zero})$, and $\underline{w} = \underline{w_1 \cdots w_n}$ for the base value $\mathtt{s}_{w_1}(\ldots \mathtt{s}_{w_n}(\epsilon)\ldots)$. We also define the *size* $|\underline{v}|$ of $\underline{v}$.

$$|\mathtt{zero}| = |\epsilon| = |\mathtt{tt}| = |\mathtt{ff}| = 1 \qquad |\mathtt{succ}(\underline{v})| = |\mathtt{s}_i(\underline{v})| = 1 + |\underline{v}|$$

We may use the following notation for terms.

Definition 11 *(Iterated Applications).* For terms M, M' and an integer n, we write $M^n M'$ to denote n applications of M to M'. In particular, $M^0 M' = M'$. We also define for a word w, given terms M_a for all letter a, $M^w M'$. This is defined by induction on words with $M^\epsilon M' = M'$ and $M^{aw'} M' = M_a\ (M^{w'} M')$.

Base reductions are defined by the rules given in Fig. 5. Note that for some of these rules, for example the last one, $\underline{v}$ can denote either the sℓT term or the sEAL term.

$$\begin{aligned}
C ::= {} & [] \mid \lambda x.C \mid C\ N \mid M\ C \mid !C \mid \mathtt{let}\ !x = C\ \mathtt{in}\ N \mid \mathtt{let}\ !x = M\ \mathtt{in}\ C \mid C \otimes N \mid M \otimes C \\
& \mid \mathtt{let}\ x \otimes y = C\ \mathtt{in}\ N \mid \mathtt{let}\ x \otimes y = M\ \mathtt{in}\ C \mid \mathtt{succ}(C) \mid \mathtt{ifn}(C,N) \mid \mathtt{ifn}(M,C) \\
& \mid \mathtt{iter}^!_{\mathsf{N}}(C,N) \mid \mathtt{iter}^!_{\mathsf{N}}(M,C) \mid \mathtt{if}(C,N) \mid \mathtt{if}(M,C) \mid \mathtt{s}_i(C) \mid \mathtt{ifw}(C,M_1,N) \\
& \mid \mathtt{ifw}(M_0,C,N) \mid \mathtt{ifw}(M_0,M_1,C) \mid \mathtt{iter}^!_{\mathsf{W}}(C,M_1,N) \mid \mathtt{iter}^!_{\mathsf{W}}(M_0,C,N) \\
& \mid \mathtt{iter}^!_{\mathsf{W}}(M_0,M_1,C) \mid [\lambda x_n \ldots x_1.t](M_1,\ldots,M_{j-1},C,M_{j+1},\ldots,M_n).
\end{aligned}$$

with $i \in \{0,1\}$ and $j \in \{1,\ldots,n\}$.

Those reductions can be extended to any context, and so we have $M \to M'$ if there is a context C and a base reduction $M_0 \to M_0'$ such that $M = C[M_0]$ and $M' = C[M_0']$. In order to work with sℓT, we use the three last rules and contexts: from the term $[\lambda x_n \ldots x_1.t](M_1,\ldots,M_n)$, we could start by reducing the term M_n to obtain $[\lambda x_n \ldots x_1.t](M_1,\ldots,M_{n-1},\underline{v})$, then use the second last reduction rule to obtain $[\lambda x_{n-1}\ldots x_1.t[\underline{v}/x_n]](M_1,\ldots,M_{n-1})$, and repeat n times to obtain a term of the form $[t']()$. We can then reduce this term t to a normal form $\underline{v}$ in sℓT and we obtain in sEAL the term $[\underline{v}]()$. Finally, with the last rule we obtain the value $\underline{v}$. Note that this order for the reduction is not mandatory as contexts do not impose to always start by reducing M_n.

$$\pi \lhd \frac{}{\Gamma, x:T \mid \Delta \vdash x:T} \qquad \mu_n(\pi) = \mathbb{1}_0$$

$$\pi \lhd \frac{}{\Gamma \mid \Delta, x:T \vdash x:T} \qquad \mu_n(\pi) = \mathbb{1}_0$$

$$\pi \lhd \frac{\sigma \lhd \Gamma, x:T \mid \Delta \vdash M:U}{\Gamma \mid \Delta \vdash \lambda x.M : T \multimap U} \qquad \mu_n(\pi) = \mu_n(\sigma) + \mathbb{1}_0$$

$$\pi \lhd \frac{\sigma \lhd \Gamma \mid \Delta \vdash M : U \multimap T \qquad \tau \lhd \Gamma' \mid \Delta \vdash N : U}{\Gamma, \Gamma' \mid \Delta \vdash M\ N : T} \qquad \mu_n(\pi) = \mu_n(\sigma) + \mu_n(\tau) + \mathbb{1}_0$$

$$\pi \lhd \frac{\sigma \lhd \cdot \mid \Delta \vdash M : T}{\Gamma \mid \Delta', [\Delta] \vdash !M :\ !T} \qquad \mu_n(\pi) = (1, \mu_{n-1}(\sigma))$$

$$\pi \lhd \frac{\sigma \lhd \Gamma \mid \Delta \vdash M :\ !T \qquad \tau \lhd \Gamma' \mid \Delta, x:[T] \vdash N : U}{\Gamma, \Gamma' \mid \Delta \vdash \mathtt{let}\ !x = M\ \mathtt{in}\ N : U} \qquad \mu_n(\pi) = \mu_n(\sigma) + \mu_n(\tau) + \mathbb{1}_0$$

$$\pi \lhd \frac{\sigma \lhd \Gamma \mid \Delta \vdash M : T \qquad \tau \lhd \Gamma' \mid \Delta \vdash N : U}{\Gamma, \Gamma' \mid \Delta \vdash M \otimes N : T \otimes U} \qquad \mu_n(\pi) = \mu_n(\sigma) + \mu_n(\tau) + \mathbb{1}_0$$

$$\pi \lhd \frac{\sigma \lhd \Gamma \mid \Delta \vdash M : S \otimes U \qquad \tau \lhd \Gamma', x:S, y:U \mid \Delta \vdash N : T}{\Gamma, \Gamma' \mid \Delta \vdash \mathtt{let}\ x \otimes y = M\ \mathtt{in}\ N : T} \qquad \mu_n(\pi) = \mu_n(\sigma) + \mu_n(\tau) + \mathbb{1}_0$$

$$\pi \lhd \frac{\sigma \lhd \Gamma \mid \Delta \vdash M : T \qquad \alpha \text{ fresh in } \Gamma, \Delta}{\Gamma \mid \Delta \vdash M : \forall \alpha.T} \qquad \mu_n(\pi) = \mu_n(\sigma)$$

$$\pi \lhd \frac{\sigma \lhd \Gamma \mid \Delta \vdash M : \forall \alpha.T}{\Gamma \mid \Delta \vdash M : T[U/\alpha]} \qquad \mu_n(\pi) = \mu_n(\sigma)$$

Fig. 6. Type and measure for generic constructors in sEAL.

3.2.2. *Types*

Types are usual types for intuitionistic linear logic enriched with some base types for booleans, integers and words.

$$A := \mathsf{B} \mid \mathsf{N} \mid \mathsf{W}$$

$$T, U, S := \alpha \mid A \mid T \multimap U \mid !T \mid T \otimes U \mid \forall \alpha.T$$

A type A is called a base type. The type for words W depends on the choice of the alphabet Σ.

Definition 12 *(Contexts and Type System). Linear variables contexts* are denoted Γ and *global variables contexts* are denoted Δ. They are defined in the same way as in the previous part on the EAL-calculus. Typing judgments have the usual shape of dual contexts judgments $\pi \lhd \Gamma \mid \Delta \vdash M : T$. For such a proof π, and $i \in \mathbb{N}$, we define a *weight* $\omega_i(\pi) \in \mathbb{N}$.

Definition 13 *(Measure and Depth).* For all integers k and n, we note $\mu_n^k(\pi) = (\omega_k(\pi), \dots, \omega_n(\pi))$, with the convention that if $k > n$, then $\mu_n^k(\pi)$ is the empty vector. We write $\mu_n(\pi)$ to denote the vector $\mu_n^0(\pi)$. In the definitions given in the type system, instead of defining $\omega_i(\pi)$ for all i, we define $\mu_n(\pi)$ for all n, from which one can recover the weights. We will often call $\mu_n(\pi)$ the *measure* of the proof π. The *depth* of a proof π (or a typed term), denoted *depth*(π), is the greatest integer i such that $\omega_i(\pi) \neq 0$. It is always defined for any proof.

The idea behind the definition of measure is to show that with a reduction step, this measure strictly decreases for the lexicographic order and we can control the growing of the weights. The rules are given on Fig. 6, Fig. 7 and Fig. 8.

The rules given in Fig. 6 represent the usual constructors in EAL. Those rules impose some restrictions on the use of variables similar to the ones described in the previous section on classical EAL. Observe that the constructors for base type values such as `zero` and `succ` given in Fig. 7 influence the weight only in position 1 and not 0 like the other constructors. As a consequence, if you take for example a proof of $\cdot \vdash \underline{v} : \mathsf{W}$, then this proof has depth 1.

For the rule given by Fig. 8, we first introduce some notations.

Definition 14 *(Base Types in sℓT and sEAL).* For a base type A of sEAL and an index I, we define a base type $A^{(I)}$ in sℓT:

$$\mathsf{B}^{(I)} := \mathsf{B} \qquad \mathsf{N}^{(I)} := \mathsf{N}^I \qquad \mathsf{W}^{(I)} := \mathsf{W}^I$$

Reciprocally, for a base type U in sℓT, we define a type in sEAL $\mathtt{type}(U)$ and an index $\mathtt{ind}(U)$.

- $\mathtt{type}(\mathsf{B}) := \mathsf{B}$ and $\mathtt{ind}(\mathsf{B}) = 1$.
- $\mathtt{type}(\mathsf{N}^I) := \mathsf{N}$ and $\mathtt{ind}(\mathsf{N}^I) = I$.
- $\mathtt{type}(\mathsf{W}^I) := \mathsf{W}$ and $\mathtt{ind}(\mathsf{W}^I) = I$.

$$\pi \lhd \frac{}{\Gamma \mid \Delta \vdash \mathtt{zero} : \mathsf{N}} \qquad \mu_n(\pi) = \mathbb{1}_1$$

$$\pi \lhd \frac{\sigma \lhd \Gamma \mid \Delta \vdash M : \mathsf{N}}{\Gamma \mid \Delta \vdash \mathtt{succ}(M) : \mathsf{N}} \qquad \mu_n(\pi) = \mu_n(\sigma) + \mathbb{1}_1$$

$$\pi \lhd \frac{\sigma \lhd \Gamma \mid \Delta \vdash M : \mathsf{N} \multimap T \qquad \tau \lhd \Gamma' \mid \Delta \vdash N : T}{\Gamma, \Gamma' \mid \Delta \vdash \mathtt{ifn}(M, N) : \mathsf{N} \multimap T} \qquad \mu_n(\pi) = \mu_n(\sigma) + \mu_n(\tau) + \mathbb{1}_0$$

$$\pi \lhd \frac{\sigma \lhd \Gamma \mid \Delta \vdash M : \ !(T \multimap T) \qquad \tau \lhd \Gamma' \mid \Delta \vdash N : \ !T}{\Gamma, \Gamma' \mid \Delta \vdash \mathtt{iter}^!_{\mathsf{N}}(M, N) : \mathsf{N} \multimap !T} \qquad \mu_n(\pi) = \mu_n(\sigma) + \mu_n(\tau) + \mathbb{1}_0$$

$$\pi \lhd \frac{}{\Gamma \mid \Delta \vdash \epsilon : \mathsf{W}} \qquad \mu_n(\pi) = \mathbb{1}_1$$

$$\pi \lhd \frac{\sigma \lhd \Gamma \mid \Delta \vdash M : \mathsf{W}}{\Gamma \mid \Delta \vdash \mathtt{s}_i(M) : \mathsf{W}} \qquad \mu_n(\pi) = \mu_n(\sigma) + \mathbb{1}_1$$

$$\pi \lhd \frac{\forall i, \sigma_i \lhd \Gamma_i \mid \Delta \vdash M_i : \mathsf{W} \multimap T \qquad \tau \lhd \Gamma' \mid \Delta \vdash N : T}{\Gamma_1, \Gamma_2, \Gamma' \mid \Delta \vdash \mathtt{ifw}(M_0, M_1, N) : \mathsf{W} \multimap T} \qquad \mu_n(\pi) = \mu_n(\sigma_1) + \mu_n(\sigma_2) + \mu_n(\tau) + \mathbb{1}_0$$

$$\pi \lhd \frac{\forall i, \sigma_i \lhd \Gamma_i \mid \Delta \vdash M_i : \ !(T \multimap T) \qquad \tau \lhd \Gamma' \mid \Delta \vdash N : \ !T}{\Gamma_1, \Gamma_2, \Gamma' \mid \Delta \vdash \mathtt{iter}^!_{\mathsf{W}}(M, N) : \mathsf{W} \multimap !T} \qquad \mu_n(\pi) = \mu_n(\sigma_1) + \mu_n(\sigma_2) + \mu_n(\tau) + \mathbb{1}_0$$

$$\pi \lhd \frac{}{\Gamma \mid \Delta \vdash \mathtt{tt} : \mathsf{B}} \qquad \mu_n(\pi) = \mathbb{1}_1$$

$$\pi \lhd \frac{}{\Gamma \mid \Delta \vdash \mathtt{ff} : \mathsf{B}} \qquad \mu_n(\pi) = \mathbb{1}_1$$

$$\pi \lhd \frac{\sigma \lhd \Gamma \mid \Delta \vdash M : T \qquad \tau \lhd \Gamma' \mid \Delta \vdash N : T}{\Gamma, \Gamma' \mid \Delta \vdash \mathtt{if}(M, N) : \mathsf{B} \multimap T} \qquad \mu_n(\pi) = \mu_n(\sigma) + \mu_n(\tau) + \mathbb{1}_0$$

Fig. 7. Type and measure for constructors on base types in sEAL.

$$\pi \lhd \frac{\forall i, (1 \le i \le k), \sigma_i \lhd \Gamma_i \mid \Delta \vdash \ M_i : A_i \qquad \tau \lhd x_1 : A_1^{(a_1)}, \ldots, x_k : A_k^{(a_k)} \vdash_{\mathsf{s}\ell\mathsf{T}} t : U}{\Gamma, \Gamma_1, \ldots, \Gamma_k \mid \Delta \vdash [\lambda x_k \ldots x_1.t](M_1, \ldots, M_k) : \mathtt{type}(U)}$$

$$\mu_n(\pi) = \sum_{i=1}^{k} \mu_n(\sigma_i) + k(d(\omega(\tau) + I) + 1) \cdot \mathbb{1}_0 + ((\omega(\tau) + I)[1/b_1] \cdots [1/b_l] + 1) \cdot \mathbb{1}_1$$
$$\text{where } I = \mathtt{ind}(U) \text{ and } \{b_1, \ldots, b_l\} = \mathtt{Var}(\omega(\tau)) \cup \mathtt{Var}(I).$$

Fig. 8. Typing rule and measure for the sℓT call in sEAL.

Note that we associate the index 1 to B since boolean values are either tt or ff, thus values of size 1.

The premise for t is a proof τ in sℓT. In this proof, we add on each non-boolean base type A_i an index variable a_i. This proof τ must yield a base type U, and this converts to a base type in sEAL. Moreover, the previous section gives us a weight $\omega(\tau)$ for this proof in sℓT.

Let us now comment on the definition of $\mu_n(\pi)$. First, $\mathtt{Var}(I)$ is a notation for the set of variables in I. Then, at position 0 in the weight, we put k times the degree of $\omega(\tau)$ and $\mathtt{ind}(U)$. Indeed, as one could see in the incoming definition of $\mathcal{R}ed$ (Definition 16) or more informally in Appendix A.2, having k times the degree at position 0 allows the future k substitution of $x_1, \ldots, x_k$ by their actual value. Then, as this term outputs a base type, and as base types have their size at position 1 in the weight, we add at position 1 an expression that will allow us to bound the number of reductions in sℓT and the size of the output. Furthermore, remark that when $k = 0$, the term $[t]()$ influences only the weight at position 1, such as constructors for base types.

3.3. Subject reduction and measure

In this section, we show that we can bound the number of reduction steps of a typed term using the measure. This is done first by showing that a reduction preserves some properties on the measure, and then by giving an explicit integer bound that will strictly decrease after a reduction. This proof uses the same method as the one from [18]. The relation $\mathcal{R}ed$ defined in the following is a generalization of the usual requirements exposed in elementary linear logic in order to control reductions.

Let us first express that type variables can be substituted.

Lemma 8 *(Substitution of Type Variables). Suppose that $\pi \lhd \Gamma \mid \Delta \vdash M : T$. Then, for every type variable α and for every type U, we can derive a proof $\pi[U/\alpha] \lhd \Gamma[U/\alpha] \mid \Delta[U/\alpha] \vdash M : T[U/\alpha]$ with $\forall n, \mu_n(\pi) = \mu_n(\pi[U/\alpha])$.*

Proof. By induction on π. All cases are straightforward, we just need to be a bit careful with the renaming of variables for the introduction of $\forall$ and choose the good instantiation for the elimination of $\forall$. $\square$

Let us then express substitution lemmas for sEAL. There are 3 cases to consider: linear variables, discharged global variables and non-discharged global variables.

Lemma 9 *(Linear Substitution). Suppose that $\pi \lhd \Gamma_1, x : T' \mid \Delta \vdash M : T$ and $\sigma \lhd \Gamma_2 \mid \Delta \vdash M' : T'$, then, we have a proof $\pi' \lhd \Gamma_1, \Gamma_2 \mid \Delta \vdash M[M'/x] : T$. Moreover, for all n, $\mu_n(\pi') \leq \mu_n(\pi) + \mu_n(\sigma)$.*

Proof. The proof comes from the fact that rules are multiplicative for Γ and so x only appears in one of the premises for each rule. Thus, the proof σ is used only once in the new proof π'. $\square$

Lemma 10 *(General Substitution). Suppose that $\pi \lhd \Gamma \mid \Delta, x : T' \vdash M : T$ and $\sigma \lhd \cdot \mid \Delta \vdash M' : T'$ and the number of occurrences of x in M is less than K, then we have a proof $\pi' \lhd \Gamma \mid \Delta \vdash M[M'/x] : T$. Moreover, for all n, $\mu_n(\pi') \leq \mu_n(\pi) + K \cdot \mu_n(\sigma)$.*

Proof. This time, the non-linearity of the variable x induces a duplication of the proof σ, that is why the measure $\mu_n(\sigma)$ is also duplicated. $\square$

Lemma 11 *(Discharged Substitution). If $\pi \lhd \Gamma \mid \Delta', [\Delta], x : [T'] \vdash M : T$ and $\sigma \lhd \cdot \mid \Delta \vdash M' : T'$ then we have a proof $\pi' \lhd \Gamma \mid \Delta', [\Delta] \vdash M[M'/x] : T$. Moreover, for all n, $\mu_n(\pi') \leq (\omega_0(\pi), (\mu_n^1(\pi) + \omega_1(\pi) \cdot \mu_{n-1}(\sigma)))$.*

Proof. The proof of this lemma relies directly on Lemma 10. Indeed, a variable with a discharged type can be used only after crossing a (!-Intro) rule and then the upper bound on $\mu_n(\pi')$ comes from the previous lemma since the number of occurrences of x in M is less than $\omega_1(\pi)$. $\square$

Next, let us give two important definitions, t_α and $\mathcal{R}ed$, in order to derive the upper bound on the number of reduction steps in sEAL.

Definition 15 *(Tower of Functions).* We define a family of tower functions $t_\alpha(x_1, \dots, x_n)$ on vectors of integers by induction on n, where we assume $\alpha \geq 1$ and $x_i \geq 2$ for all i: $t_\alpha() = 0$ and $t_\alpha(x_1, \dots, x_n) = (\alpha \cdot x_n)^{2^{t_\alpha(x_1,\dots,x_{n-1})}}$ for $n \geq 1$.

For example, $t_\alpha(3, 4, 5) = (5\alpha)^{2^{(4\alpha)^{2^{3\alpha}}}}$. Note that $t_\alpha(x_1, \dots, x_n)$ is a polynomial function in x_n (if $x_1, \dots, x_{n-1}$ are fixed) and a tower of exponential of height $2n$ for x_1 (if $x_2, \dots, x_n$ are fixed). This function gives us a bound on the measure of a given proof. However, it is not convenient to work with, so we give a sufficient condition on two vectors μ and μ' to have $t_\alpha(\mu') < t_\alpha(\mu)$.

Definition 16 *($\mathcal{R}ed$).* We define a relation on vectors denoted $\mathcal{R}ed$. Intuitively, we want $\mathcal{R}ed(\mu, \mu')$ to express that a proof of measure μ has been reduced to a proof of measure μ'. Let $\mu, \mu' \in \mathbb{N}^{n+1}$. We have $\mathcal{R}ed(\mu, \mu')$ if and only if the following conditions are satisfied:

1. $\mu \geq \tilde{2}$ and $\mu' \geq \tilde{2}$.
2. $\mu' <_{lex} \mu$, so formally there exists $0 \leq i_0 \leq n$, $\mu = (\omega_0, \dots, \omega_n)$ and $\mu' = (\omega_0, \dots, \omega_{i_0-1}, \omega'_{i_0}, \dots, \omega'_n)$, with $\omega_{i_0} > \omega'_{i_0}$.
3. There exists $d \in \mathbb{N}, 1 \leq d \leq (\omega_{i_0} - \omega'_{i_0})$ such that for all $j > i_0$, we have $\omega'_j \leq \omega_j \cdot (\omega_{i_0+1})^{d-1}$.

The first condition with $\tilde{2}$, that can also be seen in the definition of t_α, makes calculation easier, since with this condition, exponentials and multiplications conserve the strict order between integers. This does not harm the proof, since we can simply add $\tilde{2}$ to each vector we will consider. For an example of two vectors in relation, we have $\mathcal{R}ed((2, 5, 3, 2), (2, 2, 25, 15))$:

- $(2, 5, 3, 2) \geq (2, 2, 2, 2)$ and $(2, 2, 25, 15) \geq (2, 2, 2, 2)$.
- $(2, 2, 25, 15) <_{lex} (2, 5, 3, 2)$, with $i_0 = 1$.
- If we take $d = 3$, we have indeed $1 \leq d \leq 5 - 2$. Moreover, we have $25 \leq 3 \cdot 3^2 = 27$ and $15 \leq 2 \cdot 3^2 = 18$.

One can see on this example that $\mathcal{R}ed(\mu, \mu')$ indicates that $\mu' <_{lex} \mu$ and the components of the vector μ' are not "too big" compared to μ.

We can then connect those two definitions:

Theorem 4. *Let $\mu, \mu' \in \mathbb{N}^{n+1}$ and $\alpha \geq n$, $\alpha \geq 1$. If we have $\mathcal{R}ed(\mu, \mu')$ then $t_\alpha(\mu') < t_\alpha(\mu)$.*

The proof can be found in Appendix A.3.1. It shows that if we want to ensure that a certain integer defined with t_α strictly decreases for a reduction, it is sufficient to work with the relation $\mathcal{R}ed$.

Finally, we need to consider rules for polymorphism. Because of the $\forall$ elimination and introduction rules, our type system is not syntax directed. However, we can prove that for some terms, namely introduction terms, we can "recover" syntax directed rules.

Definition 17. A term M_{intro} is said to be an *introduction term* if it has one of those form:

$$\lambda x.M \mid !M \mid M \otimes N \mid \texttt{zero} \mid \texttt{succ}(M) \mid \texttt{ifn}(M,N) \mid \texttt{iter}^!_{\texttt{N}}(M,N) \mid \texttt{tt} \mid \texttt{ff}$$
$$\mid \texttt{if}(M,N) \mid \epsilon \mid \texttt{s}_i(M) \mid \texttt{ifw}(M_0,M_1,N) \mid \texttt{iter}^!_{\texttt{W}}(M_0,M_1,N).$$

To each such term we can associate a typing rule, for instance to $\lambda x.M$ the rule $(\multimap)$, to $!M$ the rule $(!)$ etc. Note that these rules correspond to introduction rules and rules for base type constructors.

For the sake of simplicity, we introduce a notation for list of objects. Let us write $\overline{T}$ to denote a sequence $T_1,\dots,T_n$ of types, and $\forall\overline{\alpha}.T$ to denote the type $\forall\alpha_1\dots\forall\alpha_n.T$ when T does not begin with a quantifier. We can now present the generation lemma.

Lemma 12 *(Generation lemma). Let M_{intro} be an introduction term. Let π be a typing $\pi \lhd \Gamma \mid \Delta \vdash M_{intro} : \forall\overline{\alpha}.T$, where type variables in $\overline{\alpha}$ are fresh in Γ and Δ. Let $\overline{T'}$ be an instantiation of $\overline{\alpha}$. Let $(\boldsymbol{R})$ denote the rule associated to M_{intro} and n its number of premises. Then there exist type derivations $\pi'_1,\dots,\pi'_n$ such that if π' is the derivation obtained by applying the rule $(\boldsymbol{R})$ to $\pi'_1,\dots,\pi'_n$ we have:*

- *π' is a typing of conclusion $\pi' \lhd \Gamma \mid \Delta \vdash M_{intro} : T[\overline{T'}/\overline{\alpha}]$.*
- *For all integer k, $\mu_k(\pi') = \mu_k(\pi)$.*

For example, if $\pi \lhd \Gamma \mid \Delta \vdash \lambda x.M : \forall\overline{\alpha}.(T \multimap U)$, then for any sequence of type $\overline{T'}$ with same length as $\overline{\alpha}$, such that variables in $\overline{\alpha}$ are not free in $\overline{T'}$, we have a proof $\pi'_1 \lhd \Gamma, x : T[\overline{T'}/\overline{\alpha}] \mid \Delta \vdash M : U[\overline{T'}/\overline{\alpha}]$ with for all k, $\mu_k(\pi) = \mu_k(\pi'_1) + \mathbb{1}_0$.

So for each case of form for M_{intro}, we can state this lemma more formally just by looking at the associated typing rule. Observe that for terms like `zero`, this lemma only states that the measure of the proof is exactly $\mathbb{1}_1$.

Proof. This is proved by induction on $\pi \lhd \Gamma \mid \Delta \vdash M_{intro} : \forall\overline{\alpha}.T$. Observe that for a given M_{intro} there are only 3 possibles rules: introduction and elimination of $\forall$ and the rule $(\mathbf{R})$ associated to the form of M_{intro}.

- **Rule $(\mathbf{R})$.** This case is trivial, this is exactly the definition of the generation lemma, with $\overline{\alpha} = \emptyset$. Observe that for this case, it is important to consider only introduction terms, otherwise there is no reason that $\overline{\alpha} = \emptyset$.
- **Elimination of $\forall$.** Suppose we have the proof

$$\pi \lhd \frac{\tau \lhd \Gamma \mid \Delta \vdash M_{intro} : \forall\alpha_0.\forall\overline{\alpha}.T}{\Gamma \mid \Delta \vdash M_{intro} : \forall\overline{\alpha}.T[T'_0/\alpha_0]}$$

  By definition, we have $\forall k, \mu_k(\pi) = \mu_k(\tau)$. By renaming, α_0 and variables in $\overline{\alpha}$ are not free in T'_0. Take a sequence of type $\overline{T'}$ with same length as $\overline{\alpha}$, then $(T'_0, \overline{T'})$ has the same length as $(\alpha_0, \overline{\alpha})$, thus we can conclude directly by induction hypothesis and using the fact that $\forall k, \mu_k(\pi) = \mu_k(\tau)$.
- **Introduction of $\forall$.** Suppose we have the proof

$$\pi \lhd \frac{\tau \lhd \Gamma \mid \Delta \vdash M_{intro} : \forall\overline{\alpha}.T \qquad \alpha_0 \text{ fresh in } \Gamma \text{ and } \Delta}{\Gamma \mid \Delta \vdash M_{intro} : \forall(\alpha_0,\overline{\alpha}).T}$$

  By definition, we have $\forall k, \mu_k(\pi) = \mu_k(\tau)$. Take a sequence of type $(T'_0, \overline{T'})$. Let $(\mathbf{R})$ denote the rule associated to M_{intro} and n its number of premises. By induction hypothesis, there exist type derivations $\pi'_1,\dots,\pi'_n$ such that if π' is the derivation obtained by applying the rule $(\mathbf{R})$ to $\pi'_1,\dots,\pi'_n$, we have $\pi' \lhd \Gamma \mid \Delta \vdash M_{intro} : T[\overline{T'}/\overline{\alpha}]$ and $\forall k, \mu_k(\pi') = \mu_k(\pi)$. By Lemma 8, we can instantiate α_0 by T'_0 in $\pi'_1,\dots,\pi'_n$ and we obtain proofs $\pi''_1,\dots,\pi''_n$ such that, if we denote π'' the derivation obtained by applying the rule $(\mathbf{R})$ to $\pi''_1,\dots,\pi''_n$, we have $\pi'' \lhd \Gamma \mid \Delta \vdash M_{intro} : T[(T'_0,\overline{T'})/(\alpha_0,\overline{\alpha})]$. Moreover, for all k, $\mu_k(\pi'') = \mu_k(\pi') = \mu_k(\pi)$. $\square$

We can now state the subject reduction of sEAL and we show that the measure allows us to construct a bound on the number of reductions.

Theorem 5. *Let $\tau \lhd \Gamma \mid \Delta \vdash M_0 : T$ and $M_0 \to M_1$. Let α be an integer equal or greater than the depth of τ. Then there is a proof $\tau' \lhd \Gamma \mid \Delta \vdash M_1 : T$ such that $\mathcal{R}ed(\mu_\alpha(\tau) + \tilde{2}, \mu_\alpha(\tau') + \tilde{2})$. Moreover, the depth of τ' is smaller than the depth of τ.*

The proof can be found in Appendix A.3.2. With Lemma 12, for some terms, we can do as if the typing rules were syntax directed. Then, the proof uses the substitution lemmas (Lemma 9 and Lemma 11) for reductions in which substitution appears. For the other constructors, one can see that the measure given in the type system for sEAL is following this idea of the relation $\mathcal{R}ed$. For example, in $[\lambda x_n \dots x_1.t](M_1, \dots, M_{n-1}, \underline{v}) \to [\lambda x_{n-1} \dots x_1.t[\underline{v}/x_n]](M_1, \dots, M_{n-1})$, the degree that appears at position 0 is here to compensate the growing of the measure at position 1. Now using the previous results, we can easily conclude our bound on the number of reductions.

Theorem 6. *Let $\pi \lhd \Gamma \mid \Delta \vdash M : T$. Denote $\alpha = max(depth(\pi), 1)$, then $t_\alpha(\mu_\alpha(\pi) + \tilde{2})$ is a bound on the number of reductions from M.*

4. Programming with sEAL

4.1. Simple examples in sEAL

We give some examples of terms in sEAL, first some terms we can usually see for the elementary affine logic, and then we give the term for computing tower of exponentials.

Some general results and notations on sEAL.

- For base types A we have the coercion $A \multimap !A$. For example, for words, we have $coerc_W\ \underline{w} \to^* \ !\underline{w}$, with:

$$coerc_W = \texttt{iter}^!_{\mathsf{W}}(!(\lambda w'.\mathtt{s}_0(w')), !(\lambda w'.\mathtt{s}_1(w')), !\epsilon).$$

- We write $\lambda x \otimes y.M$ for the term $\lambda z.\texttt{let}\ x \otimes y = z\ \texttt{in}\ M$.

Polynomials and tower of exponentials in sEAL. Recall that we defined polynomials in sℓT. With this we can define polynomials in sEAL with type $\mathsf{N} \multimap \mathsf{N}$ using the sℓT call. Moreover, using the iteration in sEAL, we can define a tower of exponential.

We can compute the function $k \mapsto 2^{2^k}$ in sEAL with type $\mathsf{N} \multimap !\mathsf{N}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{n : \mathsf{N} \mid \cdot \vdash n : \mathsf{N} \qquad \sigma \lhd x_1 : \mathsf{N}^{a_1} \vdash_{s\ell\mathsf{T}} \texttt{mult}\ x_1\ x_1 : \mathsf{N}^{a_1 \cdot a_1}}{n : \mathsf{N} \mid \cdot \vdash [\lambda x_1.\texttt{mult}\ x_1\ x_1](n) : \mathsf{N}}}{\pi \lhd \cdot \mid \cdot \vdash \lambda n.[\lambda x_1.\texttt{mult}\ x_1\ x_1](n) : \mathsf{N} \multimap \mathsf{N}}}{\cdot \mid \cdot \vdash\ !(\lambda n.[\lambda x_1.\texttt{mult}\ x_1\ x_1](n)) :\ !(\mathsf{N} \multimap \mathsf{N})} \qquad \cdot \mid \cdot \vdash\ !\underline{2} :\ !\mathsf{N}}{\cdot \mid \cdot \vdash \texttt{exp} = \texttt{iter}^!_{\mathsf{N}}(!\lambda n.[\lambda x_1.\texttt{mult}\ x_1\ x_1](n), !\underline{2}) : \mathsf{N} \multimap !\mathsf{N}}$$

$$\texttt{iter}^!_{\mathsf{N}}(!\lambda n.[\lambda x_1.\texttt{mult}\,x_1\,x_1](n), !\underline{2})\ \underline{k} \to^* !((\lambda n.[\lambda x_1.\texttt{mult}\,x_1\,x_1](n))^k\ \underline{2}) \to^* !(\underline{2^{2^k}}).$$

For an example of measure, for the subproof π, we have $depth(\pi) = 1$. From Section 2.2, we can deduce the weight for σ: $\omega(\sigma) = 4 + 6a_1 + 3a_1^3$. We can then deduce:

$$\mu(\pi) = (1 + 1 + 1 \cdot (d(\omega(\sigma) + a_1 \cdot a_1) + 1), 1 + (\omega(\sigma) + a_1 \cdot a_1)[1/a_1]) = (6, 15).$$

If we define $2^x_0 = x$ and $2^x_{k+1} = 2^{2^x_k}$, with the use of polynomials, we can represent the function $n \mapsto 2^{P(n)}_{2k}$ for all $k \geq 0$ and polynomial P with a term of type $\mathsf{N} \multimap !^k\mathsf{N}$.

4.2. Example: testing satisfiability of a propositional formula

We sketch here the construction of a term for deciding the SAT problem.

The term for SAT has type $\mathsf{N} \otimes \mathsf{W} \multimap !\mathsf{B}$ and given a formula on conjunctive normal form encoded in the type $\mathsf{N} \otimes \mathsf{W}$, it checks its satisfiability. The modality in front of the output $!\mathsf{B}$ shows that we used a non-polynomial computation, or more precisely an iteration in sEAL, as expected of a term for satisfiability.

We encode formula in conjunctive normal form in the type $\mathsf{N} \otimes \mathsf{W}$, representing the number of distinct variables in the formula and the encoding of the formula by a word on the alphabet $\Sigma = \{0, 1, \#, |\}$. A literal is represented by the number of the corresponding variable written in binary and the first bit determines if the literal is positive or negative. Then # and | are used as separator for literals and clauses.

For example, the formula $(x_1 \vee x_0 \vee x_2) \wedge (x_3 \vee \overline{x_0} \vee \overline{x_1}) \wedge (\overline{x_2} \vee x_0 \vee \overline{x_3})$ could be represented by $\underline{4} \otimes \underline{|\#101\#100\#110|\#111\#000\#001|\#010\#100\#011}$.

```
LittoBool≡ λn,v,l. ifw(
    λl'.nth v (CBiToUn n l')
    ,λl'. not (nth v (CBiToUn n l'))
    ,ff
) l
ClausetoBool≡ λn,v,c. let w ⊗ b =
    itern(
        λw' ⊗ b'. let w0 ⊗ w1 = Extract# w' in
            w0 ⊗ (or b' (LitttoBool n v w1))
        ,c ⊗ ff
    ) (occ# c)
in b
SAT≡ λn ⊗ w. let !r = iter!N(
    ! (λn0 ⊗ n1. succ(n0) ⊗ [double](n1))
    ,!(0 ⊗ 1)
) n in let !wf =coerc w in
    ! (let n ⊗ exp =r in
        [λn, exp, wf. rec(
            λv,b. or b (FormulatoBool n (CUnToBi n v) wf)
            ,ff
        ) exp](n, exp, wf)
    )
```

```
QBF0≡ λv ⊗ q ⊗ f. FormulatoBool (length v) v f
QBF1≡ λv ⊗ n ⊗ exp ⊗ q ⊗ f. rec(
    λv',b. (andor q) b (QBF0
        (conc v (CUnToBi n v')) ⊗ (not q) ⊗ f)
    ,q
) exp
```

```
SubSum≡ λwS. Riterw(
    λw ⊗ r. let w0 ⊗ w1 =Extract| w in w0 ⊗ r
    ,λw ⊗ r. let w0 ⊗ w1 =Extract| w in w0 ⊗ (Binaryadd r w1)
    ,wS ⊗ s0(ε))
SolvSubSum≡ λk ⊗ wS. let !r =
    (exp [occ|] ⊗ coerc)(wS) ⊗ (coerc k) in
    ![λn,w,k. rec(
        λv,b or b (equal k (SubSum w (CUnToBi (occ| w) v)))
        ,ff) n
    ](r)
```

Fig. 9. Examples in sEAL.

Intermediate terms in $s\ell T$. For the sake of simplicity, we sometimes omit to describe all terms in `ifw` or `iterw`, especially for the letters # and |, when they are not important. First, we can easily define a term $\mathtt{occ}_{\mathtt{a}} : \mathsf{W}^I \multimap \mathsf{N}^I$ that gives the number of occurrences of $a \in \Sigma$ in a word. We can also describe a term that gives the nth bit (from the right) of a binary word as a boolean $\mathtt{nth} : \mathsf{W}^I \multimap \mathsf{N}^I \multimap \mathsf{B}$. And finally, we have a term $\mathtt{Extract}_{\mathtt{a}} : \mathsf{W}^I \multimap \mathsf{W}^I \otimes \mathsf{W}^I$ that separates a word $w = w_0 a w_1$ in $w_0 \otimes w_1$ such that w_1 does not contain any a. This function will allow us to extract the last clause/literal of a word representing a formula.

A valuation is represented by a binary word with a length equal to the number of variables, such that the nth bit of the word represents the boolean associated to the nth variable. To begin with, we define the term $\mathtt{LittoBool} : \mathsf{N}^I \multimap \mathsf{W}^J \multimap \mathsf{W}^K \multimap \mathsf{B}$ such that given the number of variables, a valuation and the encoding of a literal, this term yields the boolean value that the valuation assigns to the literal. This term is described in Fig. 9.

We then define a term $\mathtt{ClausetoBool} : \mathsf{N}^I \multimap \mathsf{W}^J \multimap \mathsf{W}^K \multimap \mathsf{B}$ such that, given the number of variables, a valuation and a word representing a clause, this term outputs the truth value of this clause using the valuation. The definition is given in Fig. 9. With this we can check if a clause is true given a certain valuation. We can define in the same way a term for the truth value of a formula $\mathtt{FormulatoBool} : \mathsf{N}^I \multimap \mathsf{W}^J \multimap \mathsf{W}^K \multimap \mathsf{B}$. It is the same definition as `ClausetoBool`, where we replace “`or`” by “`and`” and “`LitttoBool`” by “`ClausetoBool`”.

Testing all different valuations. Now that we have `FormulatoBool`, all we have to do is to test this term with all possible valuations. If n is the number of variables, all possible valuations are described by all the binary integers from 0 to $2^n - 1$. Thus, intuitively, given the number of variables n and the formula w_f, we want to compute:

$$\bigvee_{v=0}^{2^n-1} \mathtt{FormulatoBool}\, n\, v\, w_f.$$

In order to do this, we use the constructor for iteration defined in Section 2.2: $\texttt{rec}(V, t)\ \underline{n} \to^* V\ \underline{n-1}\ (V\ \underline{n-2}\ (\dots\ (V\ \texttt{zero}\ t)\dots))$.

We can then give the term for SAT as described in Fig. 9.

The first iteration computes both 2^n and a copy of n. This technique is important as it shows that the linearity of sEAL for base variables is not too constraining for the iteration. Then, the second iteration in sℓT computes the big "or" given previously. Note that we need to be cautious about how to write integers, the number of variables n and exp (the integer representing 2^n) are given in unary, but we need the valuation in binary. And with that we have $\texttt{SAT} : \mathsf{N} \otimes \mathsf{W} \multimap !\mathsf{B}$.

Defining a sℓT term for QBF_k. Now we consider the following QBF_k problem, with k being a fixed non-negative integer. Take a formula:

$$Q_k x_n, x_{n-1}, \dots, x_{i_{k-1}+1}.\ Q_{k-1} x_{i_{k-1}}, x_{i_{k-1}-1}, \dots, x_{i_{k-2}+1} \dots,\ Q_1 x_{i_1}, x_{i_1-1}, \dots x_0.\phi.$$

The formula ϕ is a propositional formula in conjunctive normal form on the variables from x_0 to x_n, and $Q_i \in \{\forall, \exists\}$ are alternating quantifiers. That means that if Q_1 is $\forall$ then Q_2 must be $\exists$ and then Q_3 must be $\forall$ and so on. Here the variables are ordered for simplification. It can always be done by renaming. And now we have to answer if this formula is true. This can be solved in our enriched EAL calculus.

First, let us talk about the encoding of such a formula. With those ordered variables, a representation of such a formula can be a term of type $\mathsf{N}_k \otimes \mathsf{N}_{k-1} \otimes \dots \otimes \mathsf{N}_1 \otimes \mathsf{B} \otimes \mathsf{W}$. For all i with $1 \le i \le k$, N_i represents the number of variables between the quantifiers Q_i and Q_{i-1}. The boolean represents the quantifier Q_k, with the convention $\forall = \texttt{tt}$. And finally, the formula ϕ is encoded in a word as previously. This is not a canonical representation of a formula, but for any good encoding of a QBF_k formula we should be able to extract this information with a sℓT term, so for simplification, we directly take this encoding. For example, with $k = 2$, the formula:

$$\forall x_3, x_2.\exists x_1, x_0.(x_1 \vee x_0 \vee x_2) \wedge (x_3 \vee \overline{x_0} \vee \overline{x_1}) \wedge (\overline{x_2} \vee x_0 \vee \overline{x_3})$$

is represented by:

$$\underline{2} \otimes \underline{2} \otimes \texttt{tt} \otimes \underline{|\#101\#100\#110|\#111\#000\#001|\#010\#100\#011}.$$

Now we define by induction on k a sℓT term called $\texttt{QBF}_k$ for k a non-negative integer. We give to this term the type:

$$\texttt{QBF}_k : \mathsf{W}^{K_1} \otimes \mathsf{N}^{I_k} \otimes \mathsf{N}^{J_k} \otimes \dots \otimes \mathsf{N}^{I_1} \otimes \mathsf{N}^{J_1} \otimes \mathsf{B} \otimes \mathsf{W}^{K_2} \multimap \mathsf{B}.$$

One can see a similitude with the representation of a SAT formula. But we add some arguments. First, the argument w_v of type W^{K_1} is a valuation on free variables of the QBF_k formula. Then we are given for each quantifier two integers n_i and exp_i of type N^{I_i} and N^{J_i}, with n_i being the number of variables between the quantifiers Q_i and Q_{i-1}, and $exp_i = 2^{n_i}$. Finally, the boolean represents the quantifier Q_k and W^{K_2} is a formula on variables from x_0 to $x_{n_1+\dots+n_k+length(w_v)-1}$.

$\texttt{QBF}_0$ has almost already been defined. See Fig. 9 for the exact term.

Now, let us describe the term for $\texttt{QBF}_1$. One can observe that it is close to the sℓT term $\texttt{SAT}$. First, we have $\texttt{andor} : \mathsf{B} \multimap \mathsf{B} \multimap \mathsf{B} = \texttt{if}(\text{and}, \text{or})$. We also write $\texttt{conc} : \mathsf{W}^I \multimap \mathsf{W}^J \multimap \mathsf{W}^{I+J}$ the term for concatenation of words. With that we can define $\texttt{QBF}_1$ as explained in Fig. 9. So, contrary to $\texttt{SAT}$, we do not always do a big "or" on the results of $\texttt{QBF}_0$ but we do either a big "and" if the quantifier Q_k is $\forall$, or a big "or" if the quantifier is $\exists$, as one can observe with the use of $\texttt{andor}$. And when we call $\texttt{QBF}_0$, we have to update the current valuation w_v and we have to alternate the quantifier. Now with this intuition, we can deduce the general term for $\texttt{QBF}_{k+1}$ using $\texttt{QBF}_k$, and then we can also deduce the sEAL term that just computes the arguments of the sℓT term $\texttt{QBF}_k$ (with only one "!" as we only need to compute exponentials) and uses this function. And so, we obtain a term solving QBF_k with type $\mathsf{N}_k \otimes \mathsf{N}_{k-1} \otimes \dots \otimes \mathsf{N}_1 \otimes \mathsf{B} \otimes \mathsf{W} \multimap !\mathsf{B}$.

4.3. Solving the $SUBSET\ SUM$ problem

We give here another example of solving an NP-Complete problem. Given a goal integer $k \in \mathbb{N}$ and a set S of integers, is there a subset $S' \subset S$ such that $\sum\limits_{n \in S'} n = k$? We explain how we could solve this problem in our calculus. We represent the $SUBSET\ SUM$ problem by two words, k written as a binary integer and a word of the form $|\underline{n_1}|\underline{n_2}|\dots|\underline{n_m}$, with the integers written in binary, representing the set S. In order to solve this problem, we can first define a sℓT term $\texttt{equal} : \mathsf{W}^I \multimap \mathsf{W}^J \multimap \mathsf{B}$ that verifies if two binary integers are equal. Note that this is not exactly the equality on words because of the possible extra zeros at the beginning. Then, we can define a term $\texttt{SubSum} : \mathsf{W}^I \multimap \mathsf{W}^J \multimap \mathsf{W}^{I \cdot J}$ such that, given the word w_S representing the set S and a binary word w_{sub} with a length equal to the cardinality of S, this term computes the sum of all the elements of the subset represented by w_{sub}, since this word can be seen as a function from S to $\{0, 1\}$. See Fig. 9 for the term. We obtain a type $\mathsf{W}^{I \cdot J}$ for the output because we iterate at most J times a function for binary addition which can be given a type $\mathsf{W}^{a \cdot I} \multimap \mathsf{W}^I \multimap \mathsf{W}^{(a+1) \cdot I}$. Note that to define this function, we use $\texttt{Extract}_|$ defined previously. Then, we can solve the $SUBSET\ SUM$ problem in the same way as SAT in the term $\texttt{SolvSubSum}$. The notation $(f \otimes g)(x)$, when f and g are functions defined by iterators, stands for the function defined by iteration on a couple, where the first projection will compute f and the second one g, such as before in $\mathtt{SAT}$. And so, we obtain a term of type $\mathsf{W} \otimes \mathsf{W} \multimap !\mathsf{B}$. We could also construct a term that gives us the subset corresponding to the goal, by changing the type in the iteration $\mathtt{rec}$ from $\mathsf{N}^a \multimap \mathsf{B} \multimap \mathsf{B}$ to $\mathsf{N}^a \multimap (\mathsf{B} \otimes \mathsf{W}^l) \multimap (\mathsf{B} \otimes \mathsf{W}^l)$, W^l being the type of the argument w.

5. Complexity results: characterization of 2k-EXP and 2k-FEXP

Now that we have proved Theorem 5, we have obtained a bound on the number of reduction steps from a term in sEAL. More precisely, this bound shows that between two consecutive weights ω_{i+1} and ω_i, there is a difference of 2 in the height of the tower of exponentials. This will allow us to give a characterization of the classes $2k$-EXP for $k \geq 0$, and each modality "!" in the type of a term will induce a difference of 2 in the height of the tower of exponentials. With exactly the same method, we also have a characterization of the classes $2k$-FEXP for $k \geq 0$.

Restricted reductions and values. First, we show that the previous bound on the number of reductions steps in Theorem 6 can be improved. Indeed, if we restrict the possible reductions, we obtain a more precise bound.

Definition 18 *(Reductions up to a Certain Depth).* For $i \in \mathbb{N}$, we define the i-reductions, that we note $\to_i$:

- $\forall i \geq 1, [t]() \to_i [t'](\,)$ if $t \to t'$ in sℓT. Moreover, $[\underline{v}]() \to_i \underline{v}$.
- For the other base reductions $M \to M'$, we have $\forall i \in \mathbb{N}, M \to_i M'$.
- For all $i \in \mathbb{N}$, if $M \to_i M'$ then $!M \to_{i+1} !M'$.
- For all others constructors for contexts, the index i stays the same. For example with the application, we have for all $i \in \mathbb{N}$, if $M \to_i M'$ then $M\ N \to_i M'\ N$.

Now, we can find a more precise measure to bound the number of i-reductions.

Lemma 13. *Let $i \in \mathbb{N}$, $\tau \lhd \Gamma \mid \Delta \vdash M_0 : T$ and $M_0 \to_i M_1$. Then there is a proof $\tau' \lhd \Gamma \mid \Delta \vdash M_1 : T$ such that $\mathcal{R}ed(\mu_i(\tau) + \tilde{2}, \mu_i(\tau') + \tilde{2})$.*

The proof of this lemma is very similar to the proof of Theorem 5, the details are expressed in the proof of Theorem 5 in Appendix A.3.2. We can then deduce the following theorem using previous results on the relation $\mathcal{R}ed$.

Theorem 7. *Let $\pi \lhd \Gamma \mid \Delta \vdash M : T$ and $\alpha = max(i, 1)$. Then $t_\alpha(\mu_i(\pi) + \tilde{2})$ is a bound on the number of i-reductions from M.*

Let us now give an over approximation of the set of closed normal terms for i-reductions, that we call i-values.

Definition 19 *(Values Associated to Restricted Reductions).* We define for all $i \in \mathbb{N}$, *closed i-values V^i* by the following grammar.

$$\begin{aligned} V^0 &:= M \\ \forall i \geq 1, V^i &:= \lambda x.M \mid !V^{i-1} \mid V^i_0 \otimes V^i_1 \mid \mathtt{zero} \mid \mathtt{succ}(V^i) \mid \mathtt{ifn}(V^i_0, V^i_1) \mid \mathtt{iter}^!_{\mathsf{N}}(V^i_0, V^i_1) \\ &\quad \mid \mathtt{tt} \mid \mathtt{ff} \mid \mathtt{if}(V^i_0, V^i_1) \mid \epsilon \mid \mathtt{s}_i(V^i) \mid \mathtt{ifw}(V^i_0, V^i_1, V^i_2) \mid \mathtt{iter}^!_{\mathsf{W}}(V^i_0, V^i_1, V^i_2) \end{aligned}$$

We can then prove the following lemma:

Lemma 14. *Let M be a term. If M is closed and has a typing derivation then, for all $i \in \mathbb{N}$, if M is normal for i-reductions then M is an i-value V^i.*

The proof is very similar to the one of Theorem 1. Note that we do not have the converse, an i-value is not a normal form for i-reductions. However, we do not need the converse to obtain the complexity results, we are only interested in base type closed i-values with $i \geq 1$. That is why 0-values and non-closed term, such as M in $\lambda x.M$, are so generic.

From the previous results, we now have that, from a typed term M, we can reach the normal form for i-reductions for M in less than $t_i(\mu_i(\pi) + \tilde{2})$ reductions, and this normal form is an i-value.

A characterization of $2k$-EXP. Now, we sketch how the type $!\mathsf{W} \multimap !^{k+1}\mathsf{B}$ can characterize the class $2k$-EXP for $k \geq 0$. Recall that 2^x_k is defined by $2^x_0 = x$ and $2^x_{k+1} = 2^{2^x_k}$. The class k-EXP is the class of problems solvable by a Turing machine that works in time $2^{p(n)}_k$ on an entry of size n, where p is a polynomial. First we show that the number of reductions for such a term is bounded by a tower of exponentials of height $2k$.

Lemma 15. *Let $\pi \lhd \cdot \mid \cdot \vdash t : !\mathsf{W} \multimap !^{k+1}\mathsf{B}$. Let w be a word of size $|w|$. We can compute the result of $t\ !\underline{w}$ in less than a $2k$-exponential tower in $|w|$.*

Proof. Observe that the result of this computation is of type $!^{k+1}\mathsf{B}$, and a $(k+2)$-value of type $!^{k+1}\mathsf{B}$ is exactly of the form $!^{k+1}\mathtt{tt}$ or $!^{k+1}\mathtt{ff}$. So it is enough to only consider $(k+2)$-reductions to compute the result, by Lemma 14. The measure μ_n of $t\ !\underline{w}$ is $\mu_n = \mu_n(\pi) + 2 \cdot \mathbb{1}_0 + |\underline{w}| \cdot \mathbb{1}_2$. By Theorem 7, we can bound the number of reductions from $t\ !\underline{w}$ by $t_{k+2}(\mu_{k+2} + \tilde{2})$. By definition, in $t_{k+2}(\mu_{k+2} + \tilde{2})$, we can see that the weight at position 2, where the size of w appears, is at height $2k$. □

Now we have to prove that we can simulate a Turing-machine in our calculus. This proof is usual in implicit complexity [6,26]. A sketch of this proof can be found in the appendix, Section A.4. With this, using Lemma 15, we obtain the following theorem.

Theorem 8. *Terms of type* $!\mathsf{W} \multimap !^{k+1}\mathsf{B}$ *characterize the class* $2k$*-EXP.*

As explained previously, this theorem can be expanded for the classes $2k$-FEXP, that is the class of functions from words to words that can be computed by a one-tape Turing machine with a running time bounded by $2_{2k}^{p(|w|)}$ on a word w. For more precise definitions of such classes, see [26]. This characterization uses the same proof, replacing $!\mathsf{W} \multimap !^{k+1}\mathsf{B}$ with $!\mathsf{W} \multimap !^{k+1}\mathsf{W}$.

Let us now briefly compare this result for sEAL with the situation for EAL. Recall that in EAL with recursive types, we can characterize k-EXP with the type $!\mathsf{W} \multimap !^{k+2}\mathsf{B}$ [6]. The difference can be explained by the fact that in EAL, in the type $\mathsf{N} \multimap \mathsf{N}$ we only have polynomials of degree 1 (polynomials in general have the type $!\mathsf{N} \multimap !\mathsf{N}$), whereas in our case, polynomials have the type $\mathsf{N} \multimap \mathsf{N}$.

6. Conclusion

We believe that our main contribution in this paper is to define a new methodology to combine size-based and level-based type systems, which we have illustrated here with the example of $\mathsf{s}\ell\mathsf{T}$ and EAL, but we think it is of more general interest. In the present particular setting of sEAL we can wonder which enrichment we can add to EAL while keeping its properties, for instance: new data-types (lists, trees), the possibility to freely duplicate base types...We should also investigate type inference techniques, by drawing inspiration from linear dependent types [16,27] and EAL [28]. But more importantly we would like to explore to which other systems we could apply this methodology:

- First can we define a similar system in which we could move up one level of ! and stay in polynomial time? We conjecture that this could be obtained with EAL but replacing $\mathsf{s}\ell\mathsf{T}$ with a system of indexes of degree at most 1 and coefficients 1, instead of polynomial indexes. In this case we believe that the type $!\mathsf{W} \multimap !!\mathsf{B}$ would correspond to PTIME. An alternative choice could be to use a non-size-increasing types system [8] instead of $\mathsf{s}\ell\mathsf{T}$.
- Can we define a system in which all levels stay in FPTIME? Beside the condition on indexes (degree at most 1) we would also need for that purpose to replace EAL with another level-based system. Light linear logic [5] is a natural candidate, but we would need to find a measure-based argument for its complexity bound, which is a challenging objective.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to thank the anonymous referees for many insightful comments.

This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program “Investissements d’Avenir” (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR), and by the ANR Project ELICA ANR-14-CE25-0005.

Appendix A

A.1. Lemmas and theorems in $\mathsf{s}\ell\mathsf{T}$

A.1.1. Values and normal form

We prove Theorem 1:

Let t be a term in $\mathsf{s}\ell\mathsf{T}$*, if t is closed and has a typing derivation* $\vdash t : D$ *then t is normal if and only if t is a value* V.

Proof. First, we prove by induction on values V that if V is closed and has a typing derivation then V is normal. We treat only some cases and the others are easily deducible from those cases.

- If $V = \lambda x.t$ then V is normal since in the definition of contexts for reductions, we cannot reduce under a λ-abstraction.
- If $V = V_0 \otimes V_1$. V is closed so are V_0 and V_1. Moreover, V has a typing derivation, so it must end with the introduction of tensor rule, and we deduce that V_0 and V_1 have also a typing derivation. So by induction hypothesis, V_0 and V_1 are normal. Then V has no base reduction possible, and no contexts reductions since V_0 and V_1 are normals, so V is normal.
- If $V = \mathtt{zero}$ then V is normal.

Now for the other implication, we prove that if a closed typed term is normal then it is a value. We prove that by induction on terms, again we only detail some interesting cases.

- If $t := t_0\ t_1$. Suppose, by absurd, that t is a closed typed normal term. Since t has a typing derivation, we know that t_0 and t_1 are also closed typed terms. By definition of contexts in which we can apply reductions, t_0 is normal, and so by induction hypothesis, t_0 is a value. Again, by definition of contexts, t_1 is normal, and so by induction hypothesis, t_1 is a value. So t_0 is a value with an arrow type $D \multimap E$. By looking at the definition of values, either t_0 is a λ-abstraction, or it is one of the functional constructors like $\mathtt{ifn}$. If t_0 is a λ-abstraction, as t_1 is a value, we could apply the usual β-rule, so this is not possible because t is normal. If t_0 is $\mathtt{ifn}(V, V')$, as t_1 is a value of type N, it is the encoding of an integer, and so t is not normal since we could apply one of the $\mathtt{ifn}$ rules. All the other cases work in the same way, and we deduce that t cannot be normal.
- If $t := \mathtt{let}\ x \otimes y = t_0\ \mathtt{in}\ t_1$. Suppose that t is a closed typed normal term. Since t has a typing derivation, we know that t_0 has also a typing derivation, and t_0 is closed. By definition of contexts, t_0 is normal and so by induction hypothesis, t_0 is a value. t_0 has a tensor type $D \otimes E$, by definition of values, t_0 is of the form $V \otimes W$, this is absurd since in this case t would not be normal. And so, we deduce that t cannot be a normal term. □

A.1.2. Monotonic index substitution

We recall and prove Point 3 of Lemma 3.

Take J_1, J_2 such that $J_1 \le J_2$. Let E be a type. If $E \sqsubset E[a+1/a]$ then $E[J_1/a] \sqsubset E[J_2/a]$ and if $E[a+1/a] \sqsubset E$ then $E[J_2/a] \sqsubset E[J_1/a]$.

Proof. Let us first show an intermediate lemma:

Lemma 16. *Let I be an index. If an index variable a has at least one occurrence in I then $I < I[a+1/a]$.*

By induction on I. On the base case, $I = a$ so obviously $a < a+1$. In the inductive case $I = J_1 + J_2$, if a has at least one occurrence in I, then a has at least one occurrence in J_1 or in J_2. If a appears in J_1 and J_2, by induction hypothesis we have $J_1 < J_1[a+1/a]$ and $J_2 < J_2[a+1/a]$. So, we obtain directly $I < I[a+1/a]$. If a appears in J_1 and not in J_2, then we have $J_1 < J_1[a+1/a]$ and $J_2 = J_2[a+1/a]$. Thus, we obtain $I < I[a+1/a]$. The last case is symmetric. The case of the multiplication is similar to the one for addition, using the fact that all indexes are at least 1 so multiplication conserves the strict ordering.

Now we can prove Point 3 of Lemma 3 by induction on E.

- Suppose that E is a base type. The boolean case is direct, so we suppose that $E = \mathsf{N}^I$ (the case for words is similar). By Point 1 of Lemma 3, we have $I[J_1/a] \le I[J_2/a]$. This concludes the first case. Now, suppose that $E[a+1/a] \sqsubset E$. By definition, this means $I[a+1/a] \le I$. By Lemma 16, a have no occurrence in I. Thus, we have directly $I[J_2/a] \le I[J_1/a]$, and so $E[J_2/a] \sqsubset E[J_1/a]$.
- If $E = D \multimap D'$. Suppose that $E \sqsubset E[a+1/a]$. By definition, this means that $D' \sqsubset D'[a+1/a]$ and $D[a+1/a] \sqsubset D$. By induction hypothesis, we have $D'[J_1/a] \sqsubset D'[J_2/a]$ and $D[J_2/a] \sqsubset D[J_1/a]$. Thus, we obtain $E[J_1/a] \sqsubset E[J_2/a]$. The other case $E[a+1/a] \sqsubset E$ is similar.
- The case for $E = D \otimes D'$ is direct by induction hypothesis. □

A.1.3. Subtyping in proofs

We recall and show one case of Lemma 5.

If $\pi \lhd \Gamma \vdash t : D$ then for all Γ', D' such that $D \sqsubset D'$ and $\Gamma' \sqsubset \Gamma$, we have a proof $\pi' \lhd \Gamma' \vdash t : D'$ with $\omega(\pi') \le \omega(\pi)$.

Proof. We proceed by induction on the proof π. We only detail the case of the iteration for integers. Suppose that we have

$$\pi \lhd \frac{\begin{array}{c}D \sqsubset E \qquad E[I/a] \sqsubset F \\ \sigma \lhd d\Gamma \vdash V : D \multimap D[a+1/a]\end{array} \qquad \begin{array}{c}E \sqsubset E[a/a+1] \\ \tau \lhd \Gamma, d\Gamma \vdash t : D[1/a]\end{array}}{\Gamma, d\Gamma \vdash \mathtt{itern}(V, t) : \mathsf{N}^I \multimap F}$$

with $\omega(\pi) = \omega(\tau) + I \cdot (\omega(\sigma)+1)[I/a]$. Let $\Gamma', d\Gamma', I', F'$ be such that $\mathsf{N}^I \multimap F \sqsubset \mathsf{N}^{I'} \multimap F'$ and $\Gamma', d\Gamma' \sqsubset \Gamma, d\Gamma$. By definition, we have $F \sqsubset F'$ and $I' \leq I$. By induction hypothesis we have $\sigma' \lhd d\Gamma' \vdash V : D \multimap D[a+1/a]$ and $\tau' \lhd \Gamma', d\Gamma' \vdash t : D[1/a]$ with $\omega(\sigma') \leq \omega(\sigma)$ and $\omega(\tau') \leq \omega(\tau)$. We give then the following proof π':

$$\pi' \lhd \frac{\begin{array}{c} D \sqsubset E \qquad E[I'/a] \sqsubset F' \\ \sigma' \lhd d\Gamma' \vdash V : D \multimap D[a+1/a] \end{array} \qquad \begin{array}{c} E \sqsubset E[a/a+1] \\ \tau' \lhd \Gamma', d\Gamma' \vdash t : D[1/a] \end{array}}{\Gamma, d\Gamma \vdash \mathtt{itern}(V,t) : \mathsf{N}^{I'} \multimap F'}$$

with $\omega(\pi') = \omega(\tau') + I' \cdot (\omega(\sigma')+1)[I'/a]$. Indeed, by Point 3 of Lemma 3, we have $E[I'/a] \sqsubset E[I/a]$. Thus, by transitivity, we have:

$$E[I'/a] \sqsubset E[I/a] \sqsubset F \sqsubset F'.$$

Moreover, $\omega(\pi') \leq \omega(\pi)$ since $\omega(\tau') \leq \omega(\tau)$, $I' \leq I$, and

$$(\omega(\sigma')+1)[I'/a] \leq (\omega(\sigma')+1)[I/a] \leq (\omega(\sigma)+1)[I/a]$$

by Point 1 of Lemma 3 for the first inequality, and since $\omega(\sigma') \leq \omega(\sigma)$, we got the second inequality by Point 1 of Lemma 2. □

A.1.4. Main theorem of $s\ell T$

We recall and prove Theorem 2.

Suppose that $\tau \lhd \Gamma \vdash t_0 : D$ and $t_0 \to t_1$, then there is a proof $\tau' \lhd \Gamma \vdash t_1 : D$ such that $\omega(\tau') < \omega(\tau)$.

Proof. By induction. We first consider the base-reduction case. Some cases are trivial and we will not develop them. Indeed the if-rules can be proved with weakening (Lemma 1).

- If $t_0 = (\lambda x.t)V$, and $t_1 = t[V/x]$, we have a proof:

$$\tau \lhd \frac{\dfrac{\pi \lhd \Gamma_1, d\Gamma, x : E \vdash t : D}{\Gamma_1, d\Gamma \vdash \lambda x.t : E \multimap D} \qquad \sigma \lhd \Gamma_2, d\Gamma \vdash V : E}{\Gamma_1, \Gamma_2, d\Gamma \vdash (\lambda x.t)V : D}$$

  with $\omega(\tau) = \omega(\sigma) + 1 + \omega(\pi)$.
  Then by using the value substitution lemma (Lemma 6) with π and σ, we obtain a proof $\pi' \lhd \Gamma_1, \Gamma_2, d\Gamma \vdash t[V/x] : D$. Moreover, we have $\omega(\pi') \leq \omega(\pi) + \omega(\sigma) < \omega(\tau)$. This concludes this case.
- If $t_0 = \mathtt{let}\ x \otimes y = V_0 \otimes V_1\ \mathtt{in}\ t$ and $t_1 = t[V_0/x][V_1/y]$, we can conclude this case by using twice Lemma 6.
- If $t_0 = \mathtt{itern}(V,V')\ \mathtt{zero}$ and $t_1 = V'$. We have a proof:

$$\tau \lhd \frac{\dfrac{\begin{array}{c} D \sqsubset E \qquad E \sqsubset E[a+1/a] \\ \sigma_1 \lhd d\Gamma \vdash V : D \multimap D[a+1/a] \end{array} \qquad \begin{array}{c} E[I/a] \sqsubset F \\ \sigma_2 \lhd \Gamma, d\Gamma \vdash V' : D[1/a] \end{array}}{\Gamma, d\Gamma \vdash \mathtt{itern}(V,V') : \mathsf{N}^I \multimap F} \qquad \dfrac{}{\Gamma', d\Gamma \vdash \mathtt{zero} : \mathsf{N}^I}}{\Gamma, \Gamma', d\Gamma \vdash \mathtt{itern}(V,V')\ \mathtt{zero} : F}$$

  with $\omega(\tau) = I + \omega(\sigma_2) + I \cdot \omega(\sigma_1)[I/a] \geq 1 + \omega(\sigma_2)$.
  We have $D[1/a] \sqsubset E[1/a] \sqsubset E[I/a] \sqsubset F$ by Lemma 2 and Lemma 3 since $1 \leq I$. So, by subtyping and weakening (Lemma 5 and Lemma 1), we have a proof $\sigma_2' \lhd \Gamma, \Gamma', d\Gamma \vdash V' : F$ with $\omega(\sigma_2') \leq \omega(\sigma_2) < \omega(\tau)$.
  This concludes this case. The proof for the rule $\mathtt{iterw}$ with ϵ follows the same pattern.
- If $t_0 = \mathtt{itern}(V,V')\ \mathtt{succ}(W)$ and $t_1 = \mathtt{itern}(V, V\ V')\ W$. We have a proof:

$$\tau \lhd \frac{\dfrac{\begin{array}{c} D \sqsubset E \qquad E \sqsubset E[a+1/a] \\ \sigma_1 \lhd d\Gamma \vdash V : D \multimap D[a+1/a] \end{array} \qquad \begin{array}{c} E[I/a] \sqsubset F \\ \sigma_2 \lhd \Gamma, d\Gamma \vdash V' : D[1/a] \end{array}}{\Gamma, d\Gamma \vdash \mathtt{itern}(V,V') : \mathsf{N}^I \multimap F} \qquad \dfrac{\pi \lhd \Gamma', d\Gamma \vdash W : \mathsf{N}^J \qquad J+1 \leq I}{\Gamma', d\Gamma \vdash \mathtt{succ}(W) : \mathsf{N}^I}}{\Gamma, \Gamma', d\Gamma \vdash \mathtt{itern}(V,V')\ \mathtt{succ}(W) : F}$$

  with $\omega(\tau) = \omega(\pi) + I + \omega(\sigma_2) + I \cdot \omega(\sigma_1)[I/a]$.
  We can construct a proof τ_0' for $\mathtt{itern}(V, V\ V')$. The proof is described in Fig. 10. This gives us a proof for t_1.

$$\tau' \lhd \frac{\tau_0' \lhd \Gamma, d\Gamma \vdash \mathtt{itern}(V, V\ V') : \mathsf{N}^J \multimap F \qquad \pi \lhd \Gamma', d\Gamma \vdash W : \mathsf{N}^J}{\Gamma, \Gamma', d\Gamma \vdash \mathtt{itern}(V, V\ V')W : F}$$

$$\dfrac{(1)\quad \dfrac{}{\sigma_1[a+1/a] \lhd d\Gamma \vdash V : D[a+1/a] \multimap D[a+1/a][a+1/a]} \qquad \dfrac{\sigma_1[1/a] \lhd d\Gamma \vdash V : D[1/a] \multimap D[a+1/a][1/a] \qquad \sigma_2 \lhd \Gamma, d\Gamma \vdash V' : D[1/a]}{\Gamma, d\Gamma \vdash V\ V' : D[a+1/a][1/a]}}{\Gamma, d\Gamma \vdash \mathtt{itern}(V, V\ V') : \mathsf{N}^J \multimap F}$$

where (1) is: $D[a+1/a] \sqsubset E[a+1/a] \qquad E[a+1/a] \sqsubset E[a+1/a][a+1/a] \qquad E[a+1/a][J/a] = E[J+1/a] \sqsubset E[I/a] \sqsubset F$

Fig. 10. A derivation for $\mathtt{itern}(V, V\ V')$.

with $\omega(\tau') = \omega(\pi) + J + \omega(\sigma_2) + \omega(\sigma_1)[1/a] + J \cdot \omega(\sigma_1)[a+1/a][J/a]$.
And we have $\omega(\tau') \leq \omega(\pi) + J + \omega(\sigma_2) + (J+1) \cdot \omega(\sigma_1)[J+1/a]$ so, since $J+1 \leq I$, we have $\omega(\tau') < \omega(\pi) + I + \omega(\sigma_2) + I \cdot \omega(\sigma_1)[I/a] = \omega(\tau)$.
The rules for $\mathtt{iterw}$ in the cases $\mathtt{s}_0$ and $\mathtt{s}_1$ follow the same pattern.

Now we need to verify that a reduction under context strictly decreases the weight. This can be proved directly by structural induction on contexts. □

A.2. Adding polynomial time functions in EAL

As explained in the beginning of Section 3, we show informally that the proof of correctness in [18] is robust enough to support the addition of polynomial time functions in the type $\mathsf{N} \multimap \mathsf{N}$. This is a generic enrichment of EAL that does not describe the layer computing polynomial time function.

We work in the classical EAL calculus described in Section 3. For any function f from integers to integers, we define a new constructor f in the classical EAL-calculus, and a new reduction rule $f\ \underline{n} \to \underline{f(n)}$, saying that f applied to the Church encoding of the integer n is reduced to the Church encoding of the integer $f(n)$. We consider a cost to this reduction, depending on the integer n and the function f, that we call $C_f(n)$. We consider that this constructor f has type $\mathsf{N} \multimap \mathsf{N}$.

If this function f is a polynomial time computable function, we can bound the cost function $C_f(n)$ by a polynomial function $(n+2)^d$ for a certain d, and we can also bound the size of $f(n)$ by the cost, and so $f(n) \leq (n+2)^d$. The proof of correctness relies on a measure μ on terms, and as in the present work, this measure yields a bound by computing $t_\alpha(\mu)$ (see Section 3 or [18]).

Now if we look at the reduction rule, if we call $\mu(f)$ the measure for f, we go from $\mu(f) + (1, n+1)$ to $(0, (n+2)^d)$. In order to take in consideration the cost of the reduction, we add it in the measure. Thus, we consider that in the right part of the reduction, we have the measure $(0, 2(n+2)^d)$. If we define $\mu(f) = (d, 1)$, this reduction follows the relation $\mathcal{R}ed$ defined in Section 3. Thus, as in the present work, we can prove that EAL enriched with constructors for polynomial functions characterizes $2k$-EXP.

A.3. Proofs in sEAL

A.3.1. Link between t_α and $\mathcal{R}ed$

We first give some properties on t_α and $\mathcal{R}ed$. The proofs are often only calculation. Let us first recall the definitions.
When $\alpha \geq 1$ and $\forall i, x_i \geq 2$ then:

$$t_\alpha() = 0 \qquad t_\alpha(x_1, \dots, x_n) = (\alpha \cdot x_n)^{2^{t_\alpha(x_1, \dots, x_{n-1})}}.$$

We have $\mathcal{R}ed(\mu, \mu')$ if and only if the following conditions are satisfied:

1. $\mu \geq \tilde{2}$ and $\mu' \geq \tilde{2}$.
2. $\mu' <_{lex} \mu$, so formally there exists $0 \leq i_0 \leq n$, $\mu = (\omega_0, \dots, \omega_n)$ and $\mu' = (\omega_0, \dots, \omega_{i_0-1}, \omega'_{i_0}, \dots, \omega'_n)$, with $\omega_{i_0} > \omega'_{i_0}$.
3. There exists $d \in \mathbb{N}, 1 \leq d \leq (\omega_{i_0} - \omega'_{i_0})$ such that for all $j > i_0$, we have $\omega'_j \leq \omega_j \cdot (\omega_{i_0+1})^{d-1}$.

Lemma 17. *If $\mu \leq \mu'$ then $t_\alpha(\mu) \leq t_\alpha(\mu')$.*

This is just a simple consequence of the fact that the exponentiation is monotonic.

Lemma 18 *(Shift). Let $k \in \mathbb{N}^*$. Let $\mu = (\omega_0, \dots, k \cdot \omega_{i-1}, \omega_i, \dots \omega_n)$ and $\mu' = (\omega_0, \dots, \omega_{i-1}, k \cdot \omega_i, \dots \omega_n)$. Then $t_\alpha(\mu') \leq t_\alpha(\mu)$.*

Proof. Let us define $\mu_0 = (\omega_0, \dots, \omega_{i-2})$.

$k \geq 1$ so $k \leq 2^{2^{k-1}-1}$, then

$k \cdot \omega_i \leq \omega_i \cdot 2^{2^{k-1}-1} \leq (\omega_i)^{2^{k-1}}$ since $w_i \geq 2$. So,

$$\alpha \cdot k \cdot \omega_i \leq (\alpha \cdot \omega_i)^{2^{(\alpha \cdot (k-1) \cdot \omega_{i-1})^{2^{t_\alpha(\mu_0)}}}}.$$

$$(\alpha \cdot k \cdot \omega_i)^{2^{(\alpha\cdot\omega_{i-1})^{2^{t_\alpha(\mu_0)}}}} \leq (\alpha \cdot \omega_i)^{2^{(\alpha\cdot(k-1)\cdot\omega_{i-1})^{2^{t_\alpha(\mu_0)}}} \cdot 2^{(\alpha\cdot\omega_{i-1})^{2^{t_\alpha(\mu_0)}}}} \text{ and so,}$$

$$t_\alpha(\mu_0, (\omega_{i-1}, k\cdot\omega_i)) \leq (\alpha\cdot\omega_i)^{2^{(\alpha\cdot k\cdot\omega_{i-1})^{2^{t_\alpha(\mu_0)}}}} = t_\alpha(\mu_0, (k\cdot\omega_{i-1}, \omega_i)).$$

We can now obtain $t_\alpha(\mu') \leq t_\alpha(\mu)$ by monotonicity of exponential. □

Lemma 19. *If $\tilde{2} \leq \mu' < \mu$ then $\mathcal{R}ed(\mu, \mu')$.*

Proof. Take $d = 1$ and the proof is simple. □

Lemma 20. *If $\mathcal{R}ed(\mu, \mu')$ then for all μ_0, we have $\mathcal{R}ed(\mu + \mu_0, \mu' + \mu_0)$.*

Proof. The conditions 1 and 2 in the definition of $\mathcal{R}ed(\mu + \mu_0, \mu' + \mu_0)$ are given by the hypothesis $\mathcal{R}ed(\mu, \mu')$. We keep the notations $\omega_j, \omega'_j, i_0, d$.

$$1 \leq d \leq \omega_{i_0} - \omega'_{i_0} \text{ so } 1 \leq d \leq (\omega_{i_0} + \mu_0(i_0)) - (\omega'_{i_0} + \mu_0(i_0)).$$

Let $j > i_0$, we have:

$$\omega'_j + \mu_0(j) \leq \omega_j \cdot (\omega_{i_0+1})^{d-1} + \mu_0(j) \leq (\omega_j + \mu_0(j)) \cdot (\omega_{i_0+1} + \mu_0(i_0+1))^{d-1}$$

since $\omega_{i_0+1} \geq 1$. □

We now want to prove Theorem 4. First let us recall the statement of this theorem:
Let $\mu, \mu' \in \mathbb{N}^{n+1}$ and $\alpha \geq n$, $\alpha \geq 1$. If $\mathcal{R}ed(\mu, \mu')$ then $t_\alpha(\mu') < t_\alpha(\mu)$.

Proof. Suppose $\mathcal{R}ed(\mu, \mu')$. Using the notations from the definition of $\mathcal{R}ed$, we have:

$$\mu \geq (\omega_0, \ldots, \omega'_{i_0} + d, \omega_{i_0+1}, \ldots, \omega_n) \text{ and we have}$$
$$\mu' \leq (\omega_0, \ldots, \omega_{i_0-1}, \omega'_{i_0}, \omega_{i_0+1}\cdot(\omega_{i_0+1})^{d-1}, \ldots, \omega_n\cdot(\omega_{i_0+1})^{d-1}).$$

Let us call $\mu_0 = (\omega_0, \ldots, \omega_{i_0-1})$.

$$\alpha\cdot d \geq 1 \text{ so } \alpha\cdot d < 2^{\alpha\cdot d} \text{ then,}$$

$$\text{as } \omega_{i_0+1} \geq 2, \text{ we have } (\omega_{i_0+1})^{\alpha\cdot d} < (\omega_{i_0+1})^{2^{\alpha\cdot d}} \text{ so,}$$

$$\alpha\cdot(\omega_{i_0+1})^{\alpha\cdot d} < (\alpha\cdot\omega_{i_0+1})^{2^{(\alpha\cdot d)^{2^{t_\alpha(\mu_0)}}}} \text{ and so}$$

$$(\alpha\cdot(\omega_{i_0+1})^{\alpha\cdot d})^{2^{(\alpha\cdot\omega'_{i_0})^{2^{t_\alpha(\mu_0)}}}} < (\alpha\cdot\omega_{i_0+1})^{2^{(\alpha\cdot(d+\omega'_{i_0}))^{2^{t_\alpha(\mu_0)}}}}.$$

$$t_\alpha(\omega_0, \ldots, \omega_{i_0-1}, \omega'_{i_0}, (\omega_{i_0+1})^{\alpha\cdot d}) < t_\alpha(\omega_0, \ldots, \omega_{i_0-1}, \omega'_{i_0} + d, \omega_{i_0+1}).$$

By Lemma 17, since $\omega_{i_0+1}\cdot(\omega_{i_0+1})^{(n-i_0)(d-1)} \leq (\omega_{i_0+1})^{\alpha\cdot d}$, and by monotonicity of the exponential, we obtain:

$$t_\alpha(\omega_0, \ldots, \omega_{i_0-1}, \omega'_{i_0}, \omega_{i_0+1}\cdot(\omega_{i_0+1})^{(n-i_0)(d-1)}, \ldots, \omega_n) < t_\alpha(\omega_0, \ldots, \omega'_{i_0} + d, \omega_{i_0+1}, \ldots, \omega_n).$$

Using several times the shift lemma (Lemma 18), we obtain:

$$t_\alpha(\omega_0, \ldots, \omega_{i_0-1}, \omega'_{i_0}, \omega_{i_0+1}\cdot(\omega_{i_0+1})^{d-1}, \ldots, \omega_n\cdot(\omega_{i_0+1})^{d-1}) < t_\alpha(\omega_0, \ldots, \omega'_{i_0} + d, \omega_{i_0+1}, \ldots, \omega_n).$$

Again by Lemma 17, we obtain $t_\alpha(\mu') < t_\alpha(\mu)$. □

A.3.2. Main theorem of sEAL

In this section, we prove the main theorem for the enriched EAL calculus, Theorem 5:
Let $\tau \lhd \Gamma \mid \Delta \vdash M_0 : T$ and $M_0 \to M_1$. Let α be an integer equal or greater than the depth of τ. Then there is a proof $\tau' \lhd \Gamma \mid \Delta \vdash M_1 : T$ such that $\mathcal{R}ed(\mu_\alpha(\tau) + \tilde{2}, \mu_\alpha(\tau') + \tilde{2})$. Moreover, the depth of τ' is smaller than the depth of τ.

Proof. To begin with, we show that we can consider that the first rule of τ is not an elimination or an introduction of quantification.

Lemma 21. *Let $\tau \lhd \Gamma \mid \Delta \vdash M_0 : T$ be a proof that does not start with an introduction or elimination of quantifier, and $M_0 \to M_1$. Let α be an integer equal or greater than the depth of τ. Then there is a proof $\tau' \lhd \Gamma \mid \Delta \vdash M_1 : T$ such that $\mathcal{R}ed(\mu_\alpha(\tau) + \tilde{2}, \mu_\alpha(\tau') + \tilde{2})$. Moreover, the depth of τ' is smaller than the depth of τ.*

Suppose that we proved Lemma 21. We can then prove Theorem 5 by induction on τ.

- If τ does not start with an introduction or elimination of quantification, then by Lemma 21 we can conclude this case.
- If τ is:

$$\frac{\sigma \lhd \Gamma \mid \Delta \vdash M_0 : T \qquad \alpha \text{ fresh in } \Gamma, \Delta}{\Gamma \mid \Delta \vdash M_0 : \forall \alpha.T}$$

  with $\mu(\tau) = \mu(\sigma)$. By induction hypothesis with σ, there is a proof $\sigma' \lhd \Gamma \mid \Delta \vdash M_1 : T$ such that $\mathcal{R}ed(\mu_\alpha(\sigma) + \tilde{2}, \mu_\alpha(\sigma') + \tilde{2})$. Moreover, the depth of σ' is smaller than the depth of σ.
  We can construct the proof τ':

$$\frac{\sigma' \lhd \Gamma \mid \Delta \vdash M_1 : T \qquad \alpha \text{ fresh in } \Gamma, \Delta}{\Gamma \mid \Delta \vdash M_1 : \forall \alpha.T}$$

  And we have $\mu(\tau') = \mu(\sigma')$. We can conclude this case. The case of elimination of $\forall$ is similar.

Now, we prove Lemma 21. We first consider base reductions without contexts. With the generation lemma (Lemma 12), the case for the `if`-constructors are straightforward, it is a simple consequence of Lemma 19. We detail the other cases:

- If $M_0 = (\lambda x.M)M'$ and $M_1 = M[M'/x]$, we have a proof:

$$\tau \lhd \frac{\dfrac{\pi \lhd \Gamma_1, x : T' \mid \Delta \vdash M : T}{\Gamma_1 \mid \Delta \vdash \lambda x.M : T' \multimap T} \qquad \sigma \lhd \Gamma_2 \mid \Delta \vdash M' : T'}{\Gamma_1, \Gamma_2 \mid \Delta \vdash (\lambda x.M)M' : T}$$

  The double line corresponds to the generation lemma (Lemma 12). We will use this notation everywhere in the proof.

$$\forall n \in \mathbb{N}, \mu_n(\tau) = \mu_n(\sigma) + \mu_n(\pi) + 2 \cdot \mathbb{1}_0.$$

  The proof $\tau' \lhd \Gamma_1, \Gamma_2 \mid \Delta \vdash M[M'/x] : T$ is given by Lemma 9. As a consequence, we have:

$$\forall n \in \mathbb{N}, \mu_n(\tau') \le \mu_n(\pi) + \mu_n(\sigma) \text{ so, } \forall n \in N, \mu_n(\tau') < \mu_n(\tau).$$

  Then, it is still true for $n = \alpha \ge depth(\tau)$ and the depth of τ' is smaller than the depth of τ. Moreover, by Lemma 19, we obtain directly that $\mathcal{R}ed(\mu_\alpha(\tau) + \tilde{2}, \mu_\alpha(\tau') + \tilde{2})$.
- If $M_0 = \texttt{let}\ !x = !M'\ \texttt{in}\ M$ and $M_1 = M[M'/x]$ then we have a proof:

$$\tau \lhd \frac{\dfrac{\sigma \lhd \cdot \mid \Delta \vdash M' : T'}{\Gamma_1 \mid \Delta', [\Delta] \vdash !M' : !T'} \qquad \pi \lhd \Gamma_2 \mid \Delta', [\Delta], x : [T'] \vdash M : T}{\Gamma_1, \Gamma_2 \mid \Delta', [\Delta] \vdash \texttt{let}\ !x = !M'\ \texttt{in}\ M : T}$$

$$\forall n \in \mathbb{N}, \mu_n(\tau) = \mu_n(\pi) + (2, \mu_{n-1}(\sigma)).$$

  By Lemma 11, we obtain a proof $\pi' \lhd \Gamma_2 \mid \Delta', [\Delta] \vdash M[M'/x] : T$, with:

$$\forall n \in \mathbb{N}, \mu_n(\pi') \le (\omega_0(\pi), (\mu_n^1(\pi) + \omega_1(\pi) \cdot \mu_{n-1}(\sigma))).$$

  By weakening we have $\tau' \lhd \Gamma_1, \Gamma_2 \mid \Delta', [\Delta] \vdash M[M'/x] : T$. By the precedent upper-bound, we obtain $depth(\tau') \le depth(\tau)$. Moreover, $\omega_0(\tau) - \omega_0(\tau') \ge 2$, and so for $\alpha \ge depth(\tau) \ge 0$, we have $\mu_\alpha(\tau') <_{lex} \mu_\alpha(\tau)$. Finally, for $\alpha \ge j > 0$, we have:

$$\omega_j(\tau') + 2 \le \omega_j(\pi) + \omega_1(\pi) \cdot \omega_{j-1}(\sigma) + 2.$$
$$\omega_j(\tau') + 2 \le (\omega_j(\pi) + \omega_{j-1}(\sigma) + 2) \cdot (\omega_1(\pi) + \omega_0(\sigma) + 2).$$
$$\omega_j(\tau') + 2 \le (\omega_j(\tau) + 2) \cdot (\omega_1(\tau) + 2).$$

  And so we have indeed $\mathcal{R}ed(\mu_\alpha(\tau) + \tilde{2}, \mu_\alpha(\tau') + \tilde{2})$.
- If $M_0 = \texttt{let}\ x \otimes y = M \otimes M'\ \texttt{in}\ N$ and $M_1 = N[M/x][M'/y]$, we have a proof:

$$\tau \lhd \dfrac{\dfrac{\sigma \lhd \Gamma \mid \Delta \vdash M : T \qquad \sigma' \lhd \Gamma' \mid \Delta \vdash M' : T'}{\Gamma, \Gamma' \mid \Delta \vdash M \otimes M' : T \otimes T'} \qquad \pi \lhd \Gamma'', x : T, y : T' \mid \Delta \vdash N : T''}{\Gamma, \Gamma', \Gamma'' \mid \Delta \vdash \mathtt{let}\ x \otimes y = M \otimes M'\ \mathtt{in}\ N : T''}$$

$$\forall n \in \mathbb{N}, \mu_n(\tau) = \mu_n(\pi) + \mu_n(\sigma) + \mu_n(\sigma') + 2 \cdot \mathbb{1}_0.$$

Using twice Lemma 9, we obtain a proof $\tau' \lhd \Gamma, \Gamma', \Gamma'' \mid \Delta \vdash N[M/x][M'/y] : T''$ with:

$$\forall n \in \mathbb{N}, \mu_n(\tau') \leq \mu_n(\pi) + \mu_n(\sigma) + \mu_n(\sigma') < \mu_n(\tau).$$

So $depth(\tau') \leq depth(\tau)$ and for $\alpha \geq depth(\tau)$, $\mu_\alpha(\tau') < \mu_\alpha(\tau)$. By Lemma 19, we have $\mathcal{R}ed(\mu_\alpha(\tau) + \tilde{2}, \mu_\alpha(\tau') + \tilde{2})$.

- If $M_0 = \mathtt{iter}^!_{\mathsf{N}}(!M, !M')\ \underline{k}$ and $M_1 = !(M^k M')$, then we have a proof:

$$\tau \lhd \dfrac{\dfrac{\dfrac{\sigma_1 \lhd \cdot \mid \Delta \vdash M : T \multimap T}{\Gamma_1 \mid \Delta', [\Delta] \vdash !M : !(T \multimap T)} \qquad \dfrac{\sigma_2 \lhd \cdot \mid \Delta \vdash M' : T}{\Gamma_2 \mid \Delta', [\Delta] \vdash !M' : !T}}{\Gamma_1, \Gamma_2 \mid \Delta', [\Delta] \vdash \mathtt{iter}^!_{\mathsf{N}}(!M, !M') : \mathsf{N} \multimap !T} \qquad \sigma \lhd \Gamma_3 \mid \Delta', [\Delta] \vdash \underline{k} : \mathsf{N}}{\Gamma_1, \Gamma_2, \Gamma_3 \mid \Delta', [\Delta] \vdash \mathtt{iter}^!_{\mathsf{N}}(!M, !M')\underline{k} : !T}$$

$$\forall n \in \mathbb{N}, \mu_n(\tau) = \mu_n(\sigma) + (4, \mu_{n-1}(\sigma_1) + \mu_{n-1}(\sigma_2)).$$

Also note that $\forall n \in \mathbb{N}, \mu_n(\sigma) = (k+1) \cdot \mathbb{1}_1$. We can construct τ':

$$\tau' \lhd \dfrac{\dfrac{\sigma_1 \lhd \cdot \mid \Delta \vdash M : T \multimap T \qquad \begin{array}{c} \dfrac{\sigma_1 \lhd \cdot \mid \Delta \vdash M : T \multimap T \qquad \sigma_2 \lhd \cdot \mid \Delta \vdash M' : T}{\vdots} \\ \cdot \mid \Delta \vdash M^{k-1} M' : T \end{array}}{\cdot \mid \Delta \vdash M^k M' : T}}{\Gamma_1, \Gamma_2, \Gamma_3 \mid \Delta', [\Delta] \vdash !(M^k M') : !T}$$

$$\forall n \in \mathbb{N}, \mu_n(\tau') = k \cdot \mathbb{1}_1 + (1, k \cdot \mu_{n-1}(\sigma_1) + \mu_{n-1}(\sigma_2)).$$

We can see that $depth(\tau') \leq depth(\tau)$. Furthermore, $\omega_0(\tau) - \omega_0(\tau') \geq 2$, so for $\alpha \geq depth(\tau) \geq 0$, we have $\mu_\alpha(\tau') <_{lex} \mu_\alpha(\tau)$. We have $\omega_1(\tau') + 2 \leq (\omega_1(\tau) + 2)^2$, indeed:

$$k \cdot (1 + \omega_0(\sigma_1)) + \omega_0(\sigma_2) + 2 \leq (k + 1 + \omega_0(\sigma_1) + \omega_0(\sigma_2) + 2)^2.$$

For $1 < j \leq \alpha$, $\omega_j(\tau') + 2 \leq (\omega_j(\tau) + 2)(\omega_1(\tau) + 2)$. Indeed:

$$k \cdot \omega_{j-1}(\sigma_1) + \omega_{j-1}(\sigma_2) + 2 \leq (\omega_{j-1}(\sigma_1) + \omega_{j-1}(\sigma_2) + 2)(k + 1 + \omega_0(\sigma_1) + \omega_0(\sigma_2) + 2).$$

We can conclude $\mathcal{R}ed(\mu_\alpha(\tau) + \tilde{2}, \mu_\alpha(\tau') + \tilde{2})$. The proof for the rule $\mathtt{iter}^!_{\mathsf{W}}$ follows the same pattern.

- If $M_0 = [\lambda x_k \ldots x_1.t](M'_1, \ldots, M'_{k-1}, \underline{v})$ and $M_1 = [\lambda x_{k-1} \ldots x_1.t[\underline{v}/x_k]](M'_1, \ldots, M'_{k-1})$, then we have the following proof.

$$\tau \lhd \dfrac{\forall 1 \leq i \leq (k-1), \sigma_i \lhd \Gamma_i \mid \Delta \vdash M'_i : A_i \qquad \sigma \lhd \Gamma_k \mid \Delta \vdash \underline{v} : A_k \qquad \pi \lhd x_k : A_k^{(a_k)}, \ldots, x_1 : A_1^{(a_1)} \vdash_{\mathsf{s\ell T}} t : U}{\Gamma, \Gamma_1, \ldots \Gamma_k \mid \Delta \vdash [\lambda x_k \ldots x_1.t](M'_1, \ldots, M'_{k-1}, \underline{v}) : \mathtt{type}(U)}$$

Note that, with the generation lemma (Lemma 12), the proof σ induces that $\underline{v}$ is either an actual integer $\underline{m}$, an actual word $\underline{w}$ or an actual boolean $\mathtt{tt}$ or $\mathtt{ff}$. Moreover, $\forall n \in \mathbb{N}, \mu_n(\sigma) = |\underline{v}| \cdot \mathbb{1}_1$ and

$$\forall n \in \mathbb{N}, \mu_n(\tau) = \sum_{i=1}^{k-1} \mu_n(\sigma_i) + |v| \cdot \mathbb{1}_1 + k(d(\omega(\pi) + I) + 1) \cdot \mathbb{1}_0 + ((\omega(\pi) + I)[1/b_1] \cdots [1/b_l] + 1) \cdot \mathbb{1}_1$$

where $\mathtt{ind}(U) = I$ and $\{b_1, \ldots, b_l\} = \mathtt{Var}(I) \cup \mathtt{Var}(\omega(\pi))$. From the proof π, we can construct by Lemma 2 a proof

$$\pi[|\underline{v}|/a_k] \lhd x_k : A_k^{(|\underline{v}|)}, x_{k-1} : A_{k-1}^{(a_{k-1})}, \ldots, x_1 : A_1^{(a_1)} \vdash t : U[|\underline{v}|/a_k].$$

Furthermore, we can construct a proof $\sigma' \lhd \cdot \vdash_{\mathsf{s\ell T}} \underline{v} : A_k^{|\underline{v}|}$. By Lemma 6,

$$\pi' \lhd x_{k-1} : A_{k-1}^{(a_{k-1})}, \ldots, x_1 : A_1^{(a_1)} \vdash t[\underline{v}/x_k] : U[|\underline{v}|/a_k]$$

and $\omega(\pi') \leq \omega(\pi)[|\underline{v}|/a_k]$. We can now construct the proof τ':

$$\tau' \lhd \frac{\forall 1 \le i \le (k-1), \sigma_i \lhd \Gamma_i \mid \Delta \vdash M'_i : A_i \qquad \pi' \lhd x_{k-1} : A_{k-1}^{(a_{k-1})}, \dots, x_1 : A_1^{(a_1)} \vdash_{\mathsf{s}\ell\mathsf{T}} t[\underline{v}/x_k] : U[|\underline{v}|/a_k]}{\Gamma, \Gamma_k, \Gamma_1, \dots \Gamma_{k-1} \mid \Delta \vdash [\lambda x_{k-1} \dots x_1.t[\underline{v}/x_k]](M'_1, \dots, M'_{k-1}) : \mathtt{type}(U)}$$

Let us denote $\{b'_1, \dots, b'_{l'}\} = \mathtt{Var}(I) \cup \mathtt{Var}(\omega(\pi)) \cup \mathtt{Var}(\omega(\pi'))$.

$$\forall n \in \mathbb{N}, \mu_n(\tau') = \sum_{i=1}^{k-1} \mu_n(\sigma_i) +$$
$$(k-1)(d(\omega(\pi') + I[|\underline{v}|/a_k]) + 1) \cdot \mathbb{1}_0 + ((\omega(\pi') + I[|\underline{v}|/a_k])[1/b'_1] \cdots [1/b'_{l'}] + 1) \cdot \mathbb{1}_1.$$

With this, we can first see that $depth(\tau') \le depth(\tau)$. Moreover, by Theorem 3, since $\omega(\pi') + I[|\underline{v}|/a_k] \le (I + \omega(\pi))[|\underline{v}|/a_k]$, we have:

$$d(\omega(\pi') + I[|\underline{v}|/a_k]) \le d((I + \omega(\pi))[|\underline{v}|/a_k]) \le d(I + \omega(\pi)).$$

By Theorem 3, $(I + \omega(\pi))[|\underline{v}|/a_k] \le |\underline{v}|^{d(I+\omega(\pi))} \cdot (I + \omega(\pi))[1/a_k]$.

So, $(I + \omega(\pi))[|\underline{v}|/a_k][1/b'_1, \dots, b'_{l'}] \le |\underline{v}|^{d(I+\omega(\pi))} \cdot (I + \omega(\pi))[1/b'_1, \dots, b'_{l'}]$

by Lemma 2 (the substitution for a_k is either one of the b' by definition, or irrelevant if a_k does not appear in the indexes). Now from those results, we have:

$$\forall n \in \mathbb{N}, \mu_n(\tau') \le \sum_{i=1}^{k-1} \mu_n(\sigma_i) + (k-1)(d(\omega(\pi) + I) + 1) \cdot \mathbb{1}_0 + (|\underline{v}|^{d(I+\omega(\pi))} \cdot (I + \omega(\pi))[1/b'_1, \dots, b'_{l'}] + 1)\mathbb{1}_1.$$

Now we can prove $\mathcal{R}ed(\mu_\alpha(\tau) + \tilde{2}, \mu_\alpha(\tau') + \tilde{2})$:
By the precedent bound, $\omega_0(\tau) - \omega_0(\tau') \ge d(\omega(\pi) + I) + 1$. Then:

$$\omega_1(\tau') + 2 \le \sum_{i=1}^{k-1} \omega_1(\sigma_i) + |\underline{v}|^{d(I+\omega(\pi))} \cdot (I + \omega(\pi))[1/b'_1, \dots, b'_{l'}] + 3.$$

$$\omega_1(\tau') + 2 \le (\sum_{i=1}^{k-1} \omega_1(\sigma_i) + |\underline{v}| + (\omega(\pi) + I)[1/b'_1, \dots, b'_{l'}] + 3)^{d(\omega(\pi)+I)+1}.$$

$$\omega_1(\tau') + 2 \le (\omega_1(\tau) + 2) \cdot (\omega_1(\tau) + 2)^{d(\omega(\pi)+I)}.$$

And for $1 < j \le \alpha$,

$$\omega_j(\tau') + 2 \le \sum_{i=1}^{k-1} \omega_j(\sigma_i) + 2 = \omega_j(\tau) + 2 \le (\omega_j(\tau) + 2)(\omega_1(\tau) + 2)^{d(\omega(\pi)+I)}.$$

This proves $\mathcal{R}ed(\mu_\alpha(\tau) + \tilde{2}, \mu_\alpha(\tau') + \tilde{2})$.

- If $M_0 = [t_0]()$ and $M_1 = [t_1]()$ with $t_0 \to t_1$ in sℓT. We have a proof:

$$\tau \lhd \frac{\pi \lhd \cdot \vdash_{\mathsf{s}\ell\mathsf{T}} t_0 : U}{\Gamma \mid \Delta \vdash [t_0]() : \mathtt{type}(U)}$$

$$\forall n \in \mathbb{N}, \mu_n(\tau) = (1 + (\omega(\pi) + I)[1/b_1, \dots, b_l]) \cdot \mathbb{1}_1$$

where $\mathtt{ind}(U) = I$ and $\{b_1, \dots, b_l\} = \mathtt{Var}(I) \cup \mathtt{Var}(\omega(\pi))$. By Theorem 2, the main theorem of sℓT, we have a proof $\pi' \lhd \cdot \vdash_{\mathsf{s}\ell\mathsf{T}} t_1 : U$ with $\omega(\pi') < \omega(\pi)$. So we can construct the following proof:

$$\tau' \lhd \frac{\pi' \lhd \cdot \vdash_{\mathsf{s}\ell\mathsf{T}} t_1 : A^I}{\Gamma \mid \Delta \vdash [t_1]() : A}$$

Let us denote $\{b'_1, \dots, b'_{l'}\}$ all the index variables in I, $\omega(\pi)$ and $\omega(\pi')$.
$\forall n \in \mathbb{N}, \mu_n(\tau') = (1 + (\omega(\pi') + I)[1/b'_1, \dots, b'_{l'}]) \cdot \mathbb{1}_1$.
We directly see that the depth does not increase. Remark that the depth of τ is greater than 1 in this case.
We have by Lemma 2, $(\omega(\pi') + I)[1/b'_1, \dots, b'_{l'}] < (\omega(\pi) + I)[1/b'_1, \dots, b'_{l'}]$.
And so, for $\alpha \ge depth(\tau) \ge 1$, $\mu_\alpha(\tau') < \mu_\alpha(\tau)$, and so we have $\mathcal{R}ed(\mu_\alpha(\tau) + \tilde{2}, \mu_\alpha(\tau') + \tilde{2})$.
Remark that as opposed to all the precedent cases, $\mu_0(\tau)$ and $\mu_0(\tau')$ are equal, and so we need to look at position 1 to see that the measure strictly decreases. This remark is essential in the proof of Lemma 13.

- If $M_0 = [\underline{v}]()$ and $M_1 = \underline{v}$. The fact that M_0 can be typed by τ indicates that $\underline{v}$ is either an actual integer, a word or a boolean. With this remark, the typing τ' of M_1 is just the usual typing for those values. Moreover, we know the weight in sℓT and the measure in sEAL for the typing proof of a value, in sℓT the weight is 0 and in sEAL the measure is $|\underline{v}| \cdot \mathbb{1}_1$. Furthermore, if $\pi \lhd \cdot \vdash_{s\ell T} \underline{v} : U$, then we know that $|\underline{v}| \leq \mathtt{ind}(U)$. With this, we have $\mu_n(\tau) = (1 + I[1/\mathtt{Var}(I)]) \cdot \mathbb{1}_1$ and $\mu_n(\tau') = |\underline{v}| \cdot \mathbb{1}_1$. By Lemma 2, we have $|\underline{v}| \leq I[1/\mathtt{Var}(I)]$ and so for $n \geq 1$, $\mu_n(\tau') < \mu_n(\tau)$. This gives us $\mathcal{R}ed(\mu_\alpha(\tau) + \tilde{2}, \mu_\alpha(\tau') + \tilde{2})$. And the fact that the depth does not increase is direct.
  Remark that as the precedent case, we need to look at position 1 to see that the measure strictly decreases.

Now we need to work on the reductions under a context. For this we work by induction on contexts, and what we have done previously is the base case. For any inductive case of context except the ! case, the proof is straightforward, it is a direct application of the induction hypothesis.

When the context has the form $C = !C'$, the notion of depth is crucial. Indeed, suppose $M \to M'$, $M_0 = !M$ and $M_1 = !M'$. With the proof τ for M_0, we obtain a proof π for M, this gives us by induction hypothesis a proof π' for M', and this gives us a proof τ' for M_1. Moreover,

$$\forall n \in \mathbb{N}, \mu_n(\tau) = (1, \mu_{n-1}(\pi)) \text{ and } \mu_n(\tau') = (1, \mu_{n-1}(\pi')).$$

As $depth(\pi') \leq depth(\pi)$ we have:

$$depth(\tau') = depth(\pi') + 1 \leq depth(\pi) + 1 = depth(\tau).$$

And for $\alpha \geq depth(\tau)$, then $(\alpha - 1) \geq depth(\pi)$. By induction hypothesis, we have $\mathcal{R}ed(\mu_{\alpha-1}(\pi) + \tilde{2}, \mu_{\alpha-1}(\pi') + \tilde{2})$. From this, we can easily deduce $\mathcal{R}ed(\mu_\alpha(\tau) + \tilde{2}, \mu_\alpha(\tau') + \tilde{2})$.

Remark that this proof shows that if we had $\mathcal{R}ed(\mu_n(\pi) + \tilde{2}, \mu_n(\pi') + \tilde{2})$ we obtain $\mathcal{R}ed(\mu_{n+1}(\tau) + \tilde{2}, \mu_{n+1}(\tau') + \tilde{2})$. This remark is important for the proof of Lemma 13. □

A.4. Simulation of a Turing machine in sEAL

In order to prove Theorem 8, we show the converse of Lemma 15. Formally, we want to show the following lemma.

Lemma 22. *Let k be an integer. Let TM be a Turing machine on binary words such that, for an input word w, TM works in time $2_{2k}^{P(|w|)}$, where P is a polynomial function. Then, TM can be simulated in sEAL by a term of type* $!\mathsf{W} \multimap !^{k+1}\mathsf{B}$.

The first thing we prove is the existence of a term in sℓT to simulate n steps of a deterministic Turing-machine on a word w. Suppose given two variables $w : \mathsf{W}^{a_w}$ and $n : \mathsf{N}^{a_n}$, we note $\mathsf{Conf_b}$ the type $\mathsf{W}^{a_w+b} \otimes \mathsf{B} \otimes \mathsf{W}^{a_w+b} \otimes \mathsf{B}^q$, with q an integer and B^q being q tensors of booleans. This type represents a configuration on a Turing machine after b steps, with B^q coding the state, and then $\underline{w_0} \otimes x \otimes \underline{w_1}$ represents the tape, with x being the head, w_0 represents the reverse of the word before b, and w_1 represents the word after x. We then define some terms in sℓT that works with this encoding. First we have a term $\mathtt{init}$ such that $w : \mathsf{W}^{a_w}, n : \mathsf{N}^{a_n} \vdash \mathtt{init} : \mathsf{Conf_1}$ and $\mathtt{init}$ computes the initial configuration of the Turing machine. Then, we have a term $\mathtt{step}$ with $\cdot \vdash \mathtt{step} : \mathsf{Conf_b} \multimap \mathsf{Conf_{b+1}}$ that computes the result of the transition function from a configuration to the next one, and finally we have a term $\mathtt{final}$ with $\cdot \vdash \mathtt{final} : \mathsf{Conf_b} \multimap \mathsf{B}$ verifying if the final configuration is accepted or not. More precisely, this is given by:

- Given an initial state s of size q, we can code this state in a term $s : B^q$. Then we pose:

$$\mathtt{init} = \epsilon \otimes (\mathtt{ifw}(\lambda w'.\mathtt{ff} \otimes w', \lambda w'.\mathtt{tt} \otimes w', \mathtt{ff} \otimes \epsilon)\ w) \otimes s.$$

- Given for each state s of size q and boolean b a transition function $\delta(b, s) \subset \{left, right, stay\} \times \{0, 1\} \times \{0, 1\}^q$, we can construct the term $\mathtt{step}$:

$$\mathtt{step} = \lambda c.\mathtt{let}\ x \otimes b \otimes y \otimes s = c\ \mathtt{in}\ (\mathtt{case}_{q+1}(t_{0^{q+1}}, \ldots, t_{1^{q+1}}))\ (b \otimes s).$$

  The term $\mathtt{case}$ is a notation for a sequence of conditionals on the tensor of booleans of type $\mathsf{B}^{(q+1)}$, in this case $b \otimes s$. For a given boolean b and a state s, we define $t_{b \otimes s}$ according to $\delta(b, s)$. For example, if $\delta(b, s) = (left, b', s')$, we define:

$$t_{b \otimes s} = (\mathtt{ifw}(\lambda w.w \otimes \mathtt{ff}, \lambda w.w \otimes \mathtt{tt}, \epsilon \otimes \mathtt{ff})\ x) \otimes s_{b'}(y) \otimes s'.$$

- Given for states of size q a function $\mathtt{accept} : \mathsf{B}^q \to \mathsf{B}$ (constructed with $\mathtt{case}_q$), we can construct the term $\mathtt{final}$:

$$\mathtt{final} = \lambda c.\mathtt{let}\ x \otimes b \otimes y \otimes s = c\ \mathtt{in}\ \mathtt{accept}(s).$$

Now, suppose given a one-tape deterministic Turing machine TM on binary words such that for words w, TM works in time $2_{2k}^{P(|w|)}$. As usual, we suppose that TM has an infinite tape, this means that on an input w, the Turing-machine can read outside the bound of w and in this case, it reads a 0. We can compute a term in sEAL t_{TM} such that $\cdot \mid \cdot \vdash t_{TM} :!\mathsf{W} \multimap !^{k+1}\mathsf{B}$ and on an input $!w$, the term reduces to the term $!^{k+1}b$ with $b = \mathtt{tt}$ if w is accepted by TM, and $b = \mathtt{ff}$ otherwise. For this, we show how to decompose the work in order to construct this term.

1. We duplicate the word given in input.
2. With one of those words, we compute the length of the word, and we keep the other one as a copy.
3. Now that we have the length, we can compute $2_{2k}^{P(|w|)}$. So we obtain $!w \otimes !^{k+1}n$ with n representing $2_{2k}^{P(|w|)}$. By using the coercion, we obtain $!^{k+1}w \otimes !^{k+1}n$. We can then give this word and this integer as an input for a sℓT program using the sℓT-call of sEAL.
4. In sℓT, by using the previously defined term $\mathtt{init}$ with the word $w : \mathsf{W}^{a_w}$ and the integer $n : \mathsf{N}^{a_n}$, we obtain a configuration C_i of type Conf_1 representing the initial tape of the Turing machine.
5. By iterating n times (using the constructor $\mathtt{itern}$) the term $\cdot \vdash \mathtt{step} : \mathsf{Conf}_\mathsf{b} \multimap \mathsf{Conf}_{\mathsf{b}+1}$, from C_i, we obtain a term of type $\mathsf{Conf}_{\mathsf{a_n}}$. By definition, this term is a representation of the tape of the Turing machine after n steps, that is to say at the end of the computation.
6. Finally, with the term $\mathtt{final}$ we can extract the result of the computation as a boolean in sℓT.
7. As the word and the integer we used in the sℓT-call had the type $!^{k+1}\mathsf{W} \otimes !^{k+1}\mathsf{N}$, we obtain in sEAL the result of the computation as a boolean of type $!^{k+1}\mathsf{B}$.

In conclusion, we can simulate TM by a term of type $!\mathsf{W} \multimap !^{k+1}\mathsf{B}$.

References

[1] S. Bellantoni, S. Cook, A new recursion-theoretic characterization of the polytime functions, in: Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing, ACM, 1992, pp. 283–293.
[2] M. Hofmann, A mixed modal/linear lambda calculus with applications to Bellantoni-Cook safe recursion, in: International Workshop on Computer Science Logic, Springer, 1997, pp. 275–294.
[3] U. Dal Lago, P.P. Toldin, A higher-order characterization of probabilistic polynomial time, in: International Workshop on Foundational and Practical Aspects of Resource Analysis, Springer, 2011, pp. 1–18.
[4] J. Mitchell, M. Mitchell, A. Scedrov, A linguistic characterization of bounded oracle computation and probabilistic polynomial time, in: 39th Annual Symposium on Foundations of Computer Science, 1998. Proceedings, IEEE, 1998, pp. 725–733.
[5] J.-Y. Girard, Light linear logic, Inf. Comput. 143 (1998) 175–204.
[6] P. Baillot, On the expressivity of elementary linear logic: characterizing ptime and an exponential time hierarchy, Inf. Comput. 241 (2015) 3–31.
[7] V. Danos, J.-B. Joinet, Linear logic and elementary time, Inf. Comput. 183 (2003) 123–137.
[8] M. Hofmann, Linear types and non-size-increasing polynomial time computation, Inf. Comput. 183 (2003) 57–85.
[9] M. Hofmann, S. Jost, Static prediction of heap space usage for first-order functional programs, in: 30th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'11, Proceedings, ACM, 2003, pp. 185–197.
[10] J. Hoffmann, M. Hofmann, Amortized resource analysis with polynomial potential, in: 19th Euro. Symp. on Prog. (ESOP10), Springer, 2010, pp. 287–306.
[11] J. Hoffmann, K. Aehlig, M. Hofmann, Multivariate amortized resource analysis, in: 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'11, Proceedings, ACM, 2011.
[12] S. Jost, K. Hammond, H. Loidl, M. Hofmann, Static determination of quantitative resource usage for higher-order programs, in: Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL, 2010, 2010, pp. 223–236.
[13] G. Bonfante, J.-Y. Marion, J.-Y. Moyen, Quasi-interpretations a way to control resources, Theor. Comput. Sci. 412 (2011) 2776–2796.
[14] P. Baillot, U. Dal Lago, Higher-order interpretations and program complexity, Inf. Comput. 248 (2016) 56–81.
[15] P. Baillot, G. Barthe, U.D. Lago, Implicit computational complexity of subrecursive definitions and applications to cryptographic proofs, in: Proceedings of Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, in: LNCS, vol. 9450, Springer, 2015, pp. 203–218.
[16] P. Baillot, G. Barthe, U. Dal Lago, Implicit Computational Complexity of Subrecursive Definitions and Applications to Cryptographic Proofs (Long version), Research Report, ENS Lyon, 2015, https://hal.archives-ouvertes.fr/hal-01197456.
[17] P. Baillot, M. Gaboardi, V. Mogbil, A polytime functional language from light linear logic, in: European Symposium on Programming, Springer, 2010, pp. 104–124.
[18] A. Madet, R.M. Amadio, An elementary affine λ-calculus with multithreading and side effects, in: International Conference on Typed Lambda Calculi and Applications, Springer, 2011, pp. 138–152.
[19] U. Dal Lago, M. Gaboardi, Linear dependent types and relative completeness, Log. Methods Comput. Sci. 8 (2011).
[20] S. Alves, M. Fernández, M. Florido, I. Mackie, The power of linear functions, in: Computer Science Logic, Springer, Berlin, Heidelberg, 2006, pp. 119–134.
[21] J. Hughes, L. Pareto, A. Sabry, Proving the correctness of reactive systems using sized types, in: Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1996, pp. 410–423.
[22] M. Avanzini, U. Dal Lago, Automating sized-type inference for complexity analysis, Proc. ACM Program. Lang. 1 (2017) 43, 29 pp.
[23] P. Baillot, A. Ghyselen, Combining linear logic and size types for implicit complexity, in: 27th EACSL Annual Conference on Computer Science Logic (CSL 2018), in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 119, 2018, chapter 9, 21 pp.
[24] U. Dal Lago, B. Petit, Linear dependent types in a call-by-value scenario, Sci. Comput. Program. 84 (2014) 77–100.
[25] K. Terui, Light affine lambda calculus and polytime strong normalization, in: Logic in Computer Science, 2001. 16th Annual IEEE Symposium on Proceedings, IEEE, 2001, pp. 209–220.
[26] P. Baillot, E. De Benedetti, S.R. Della Rocca, Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, in: IFIP International Conference on Theoretical Computer Science, Springer, 2014, pp. 151–163.
[27] U. Dal Lago, B. Petit, The geometry of types, in: The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'13, Proceedings, ACM, 2013, pp. 167–178.
[28] P. Baillot, K. Terui, A feasible algorithm for typing in elementary affine logic, in: Proceedings of TLCA, vol. 5, Springer, 2005, pp. 55–70.