An Elementary Affine λ-calculus with Multithreading and Side Effects*

Antoine Madet   Roberto M. Amadio

Laboratoire PPS, Université Paris Diderot

{madet,amadio}@pps.jussieu.fr

Abstract

Linear logic provides a framework to control the complexity of higher-order functional programs. We present an extension of this framework to programs with multithreading and side effects focusing on the case of elementary time. Our main contributions are as follows. First, we provide a new combinatorial proof of termination in elementary time for the functional case. Second, we develop an extension of the approach to a call-by-value λ-calculus with multithreading and side effects. Third, we introduce an elementary affine type system that guarantees the standard subject reduction and progress properties. Finally, we illustrate the programming of iterative functions with side effects in the presented formalism.

*Work partially supported by project ANR-08-BLANC-0211-01 "COMPLICE" and the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open grant number: 243881 (project CerCo).

Contents

1 Introduction

There is a well explored framework based on Linear Logic to control the complexity of higher-order functional programs. In particular, *light logics* [11, 10, 3] have led to a polynomial light affine λ-calculus [13] and to various type systems for the standard λ-calculus guaranteeing that a well-typed term has a bounded complexity [9, 8, 5]. Recently, this framework has been extended to a higher-order process calculus [12] and a functional language with recursive definitions [4]. In another direction, the notion of *stratified region* [7, 1] has been used to prove the termination of higher-order multithreaded programs with side effects.

Our general goal is to extend the framework of light logics to a higher-order functional language with multithreading and side effects by focusing on the case of elementary time [10]. The key point is that termination does not rely anymore on stratification but on the notion of depth which is standard in light logics. Indeed, light logics suggest that complexity can be tamed through a fine analysis of the way the depth of the occurrences of a λ-term can vary during reduction.

Our core functional calculus is a λ-calculus extended with a constructor '!' (the modal operator of linear logic) marking duplicable terms and a related $\mathsf{let}\,!$ destructor. The depth of an occurrence in a λ-term is the number of $!'s$ that must be crossed to reach the occurrence. Our contribution can be described as follows.

1. In Section 2 we propose a formal system called *depth system* that controls the depth of the occurrences and which is a variant of a system proposed in [13]. We show that terms well-formed in the depth system are guaranteed to terminate in elementary time under an arbitrary reduction strategy. The proof is based on an original combinatorial analysis of the depth system ([10] assumes a specific reduction strategy while [13] relies on a standardization theorem).

2. In Section 3, following previous work on an affine-intuitionistic system [2], we extend the functional core with parallel composition and operations producing side effects on an 'abstract' notion of state. We analyse the impact of side-effects operations on the depth of the occurrences and deduce an extended depth system. We show that it still guarantees termination of programs in elementary time under a natural call-by-value evaluation strategy.

3. In Section 4, we refine the depth system with a second order (polymorphic) elementary affine type system and show that the resulting system enjoys subject reduction and progress (besides termination in elementary time).

4. Finally, in Section 5, we discuss the expressivity of the resulting type system. On the one hand we check that the usual encoding of elementary functions goes through. On the other hand, and more interestingly, we provide examples of iterative (multithreaded) programs with side effects.

The λ-calculi introduced are summarized in Table 1.1. For each concurrent language there is a corresponding functional fragment and each language (functional or concurrent) refines the one on its left hand side. The elementary complexity bounds are obtained for the $\lambda^!_\delta$ and $\lambda^{!R}_\delta$ calculi while the progress property and the expressivity results refer to their typed refinements $\lambda^!_{EA}$ and $\lambda^{!R}_{EA}$, respectively. Proofs are available in Appendix A.

Functional	$\lambda^!$	$\supset$	$\lambda^!_\delta$	$\supset$	$\lambda^!_{EA}$
$\cap$					
Concurrent	$\lambda^{!R}$	$\supset$	$\lambda^{!R}_\delta$	$\supset$	$\lambda^{!R}_{EA}$

Table 1.1: Overview of the λ-calculi considered

2 Elementary Time in a Modal λ-calculus

In this section, we present our core functional calculus, a related depth system, and show that every term which is well-formed in the depth system terminates in elementary time under an arbitrary reduction strategy.

2.1 A Modal λ-calculus

We introduce a modal λ-calculus called $\lambda^!$. It is very close to the *light affine λ-calculus* of Terui [13] where the paragraph modality '§' used for polynomial time is dropped and where the '!' modality is relaxed as in elementary linear logic [10].

2.1.1 Syntax

Terms are described by the grammar in Table 2.1: We find the usual set of

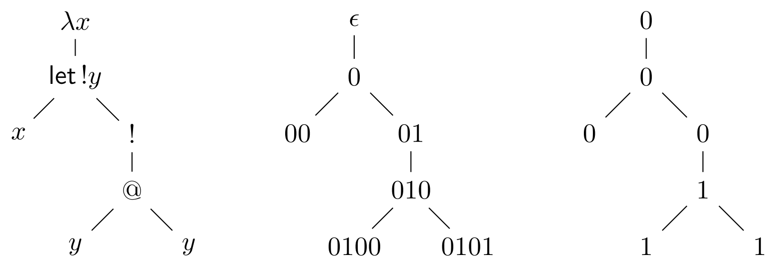
$$M, N ::= x, y, z \ldots \mid \lambda x.M \mid MN \mid !M \mid \mathsf{let}\ !x = N\ \mathsf{in}\ M$$

Table 2.1: Syntax of $\lambda^!$

variables, λ-abstraction and application, plus a modal operator '!' (read *bang*) and a $\mathsf{let}\,!$ operator. We define $!^0 M = M$ and $!^{n+1}M = !(!^n M)$. In the terms $\lambda x.M$ and $\mathsf{let}\ !x = N\ \mathsf{in}\ M$ the occurrences of x in M are bound. The set of free variables of M is denoted by $\mathsf{FV}(M)$. The number of free occurrences of x in M is denoted by $\mathsf{FO}(x, M)$. $M[N/x]$ denotes the term M in which each free occurrence of x has been substituted by the term N.

Each term has an *abstract syntax tree* as exemplified in Figure 2.1(a). A path starting from the root to a node of the tree denotes an *occurrence* of the program that is denoted by a word $w \in \{0, 1\}^*$ (see Figure 2.1(b)).

We define the notion of *depth*:



(a) (b) (c)

Figure 2.1: Syntax tree of the term $\lambda x.\mathsf{let}\ !y = x\ \mathsf{in}\ !(yy)$, addresses and depths

Definition 2.1 (depth)**.** The *depth* $d(w)$ of an occurrence w is the number of !'s that the path leading to w crosses. The depth $d(M)$ of a term M is the maximum depth of its occurrences.

In Figure 2.1(c), each occurrence is labelled with its depth. Thus $d(\lambda x.\mathsf{let}\ !y = x\ \mathsf{in}\ !(yy)) = 1$. In particular, the occurrence 01 is at depth 0; what matters in computing the depth of an occurrence is the number of ! that precedes strictly the occurrence.

2.1.2 Operational Semantics

We consider an arbitrary reduction strategy. Hence, an evaluation context E can be any term with exactly one occurrence of a special variable $[\,]$, the 'hole'. $E[M]$ denotes E where the hole has been substituted by M. The reduction rules are given in Table 2.2. The $\mathsf{let}\,!$ is 'filtering' modal terms and 'destructs' the

$$
\begin{array}{rcl}
E[(\lambda x.M)N] & \to & E[M[N/x]] \\
E[\mathsf{let}\ !x = !N\ \mathsf{in}\ M] & \to & E[M[N/x]]
\end{array}
$$

Table 2.2: Operational semantics of $\lambda^!$

bang of the term $!N$ after substitution. In the sequel, $\xrightarrow{*}$ denotes the reflexive and transitive closure of $\to$.

2.2 Depth System

By considering that deeper occurrences have less weight than shallow ones, the proof of termination in elementary time [10] relies on the observation that when reducing a redex at depth i the following holds:

(1) the depth of the term does not increase,

(2) the number of occurrences at depth $j < i$ does not increase,

(3) the number of occurrences at depth i strictly decreases,

(4) the number of occurrences at depth $j > i$ may be increased by a multiplicative factor k bounded by the number of occurrences at depth $i+1$.

Theses properties can be guaranteed by the following requirements:

(i) in $\lambda x.M$, x may occur at most once in M and at depth 0,

(ii) in $\mathsf{let}\ !x = M\ \mathsf{in}\ N$, x may occur arbitrarily many times in N and at depth 1.

Hence, the rest of this section is devoted to the introduction of a set of inferences rules called depth system. Every term which is valid in the depth system will terminate in elementary time. First, we introduce the judgement:

$$\Gamma \vdash^{\delta} M$$

where δ is a natural number and the context Γ is of the form $x_1 : \delta_1, \ldots, x_n : \delta_n$. We write $dom(\Gamma)$ for the set $\{x_1, \ldots, x_n\}$. It should be interpreted as follows:

> The free variables of $!^{\delta} M$ may only occur at the depth specified by the context Γ.

The inference rules of the depth system are presented in Table 2.3.

$$\frac{}{\Gamma, x : \delta \vdash^{\delta} x}$$

$$\frac{\Gamma, x : \delta \vdash^{\delta} M \quad \mathsf{FO}(x, M) \leq 1}{\Gamma \vdash^{\delta} \lambda x.M} \qquad \frac{\Gamma \vdash^{\delta} M \quad \Gamma \vdash^{\delta} N}{\Gamma \vdash^{\delta} MN}$$

$$\frac{\Gamma \vdash^{\delta} N \quad \Gamma, x : (\delta+1) \vdash^{\delta} M}{\Gamma \vdash^{\delta} \mathsf{let}\ !x = N\ \mathsf{in}\ M} \qquad \frac{\Gamma \vdash^{\delta+1} M}{\Gamma \vdash^{\delta} !M}$$

Table 2.3: Depth system: $\lambda^{!}_{\delta}$

We comment on the rules. The variable rule says that the current depth of a free variable is specified by the context. The λ-abstraction rule requires that the occurrence of x in M is at the same depth as the formal parameter; moreover it occurs at most once so that no duplication is possible at the current depth (Property (3)). The application rule says that we may only apply two terms if they are at the same depth. The $\mathsf{let}\,!$ rule requires that the bound occurrences of x are one level deeper than the current depth; note that there is no restriction on the number of occurrences of x since duplication would happen one level deeper than the current depth. Finally, the bang rule is better explained in a bottom-up way: crossing a modal occurrence increases the current depth by one.

Definition 2.2 (well-formedness)**.** A term M is *well-formed* if for some Γ and δ a judgement $\Gamma \vdash^{\delta} M$ can be derived.

Example 2.3. The term of Figure 1(a) is well-formed according to our depth system:

$$\dfrac{\dfrac{\dfrac{x : \delta \vdash^{\delta} x \qquad \dfrac{\dfrac{x : \delta, y : \delta + 1 \vdash^{\delta+1} y \qquad x : \delta, y : \delta + 1 \vdash^{\delta+1} y}{x : \delta, y : \delta + 1 \vdash^{\delta+1} yy}}{x : \delta, y : \delta + 1 \vdash^{\delta} !(yy)}}{x : \delta \vdash^{\delta} \mathsf{let}\ !y = x \ \mathsf{in}\ !(yy)}}{\vdash^{\delta} \lambda x.\mathsf{let}\ !y = x \ \mathsf{in}\ !(yy)}}{}$$

On the other hand, the followings term is not valid:

$$P = \lambda x.\mathsf{let}\ !y = x \ \mathsf{in}\ !(y!(yz))$$

Indeed, the second occurrence of y in $!(y!(yz))$ is too deep of one level, hence reduction may increase the depth by one. For example, $P!!N$ of depth 2 reduces to $!(!N!(!N)z)$ of depth 3.

Proposition 2.4 (properties on the depth system)**.** *The depth system satisfies the following properties:*

1. *If $\Gamma \vdash^{\delta} M$ and x occurs free in M then $x : \delta'$ belongs to Γ and all occurrences of x in $!^{\delta} M$ are at depth δ'.*
2. *If $\Gamma \vdash^{\delta} M$ then $\Gamma, \Gamma' \vdash^{\delta} M$.*
3. *If $\Gamma, x : \delta' \vdash^{\delta} M$ and $\Gamma \vdash^{\delta'} N$ then $d(!^{\delta} M[N/x]) \leq max(d(!^{\delta} M), d(!^{\delta'} N))$ and $\Gamma \vdash^{\delta} M[N/x]$.*
4. *If $\Gamma \vdash^{0} M$ and $M \to N$ then $\Gamma \vdash^{0} N$ and $d(M) \geq d(N)$.*

2.3 Elementary Bound

In this section, we prove that well-formed terms terminate in elementary time under an arbitrary reduction strategy. To this end, we define a measure on terms based on the number of occurrences at each depth.

Definition 2.5 (measure)**.** Given a term M and $0 \leq i \leq d(M)$, let $\omega_i(M)$ be the number of occurrences in M of depth i increased by 2 (so $\omega_i(M) \geq 2$). We define $\mu_n^i(M)$ for $n \geq i \geq 0$ as follows:

$$\mu_n^i(M) = (\omega_n(M), \ldots, \omega_{i+1}(M), \omega_i(M))$$

We write $\mu_n(M)$ for $\mu_n^0(M)$. We order the vectors of $n+1$ natural number with the (well-founded) lexicographic order $>$ from right to left.

We derive a termination property by observing that the measure strictly decreases during reduction.

Proposition 2.6 (termination)**.** *If M is well-formed, $M \to M'$ and $n \geq d(M)$ then $\mu_n(M) > \mu_n(M')$.*

Proof. We do this by case analysis on the reduction rules:

- $M = E[(\lambda x.M_1)M_2] \to M' = E[M_1[M_2/x]]$
  Let the occurrence of the redex $(\lambda x.M_1)M_2$ be at depth i. The restrictions on the formation of terms require that x occurs at most once in M_1 at depth 0. Then $\omega_i(M) - 3 \geq \omega_i(M')$ because we remove the nodes for application and λ-abstraction and either M_2 disappears or the occurrence of the variable x in M_1 disappears (both being at the same depth as the redex). Clearly $\omega_j(M) = \omega_j(M')$ if $j \neq i$, hence

  $$\mu_n(M') \leq (\omega_n(M), \ldots, \omega_{i+1}(M), \omega_i(M) - 3, \mu_{i-1}(M)) \qquad (2.1)$$

  and $\mu_n(M) > \mu_n(M')$.

- $M = E[\mathsf{let}\ !x = !M_2\ \mathsf{in}\ M_1] \to M' = E[M_1[M_2/x]]$
  Let the occurrence of the redex $\mathsf{let}\ !x = !M_2\ \mathsf{in}\ M_1$ be at depth i. The restrictions on the formation of terms require that x may only occur in M_1 at depth 1 and hence in M at depth $i+1$. We have that $\omega_i(M) = \omega_i(P) - 2$ because the $\mathsf{let!}$ node disappear. Clearly, $\omega_j(M) = \omega_j(M')$ if $j < i$. The number of occurrences of x in M_1 is bounded by $k = \omega_{i+1}(M) \geq 2$. Thus if $j > i$ then $\omega_j(M') \leq k \cdot \omega_j(M)$. Let's write, for $0 \leq i \leq n$:

  $$\mu_n^i(M) \cdot k = (\omega_n(M) \cdot k, \omega_{n-1}(M) \cdot k, \ldots, \omega_i(M) \cdot k)$$

  Then we have

  $$\mu_n(M') \leq (\mu_n^{i+1}(M) \cdot k, \omega_i(M) - 2, \mu_{i-1}(M)) \qquad (2.2)$$

  and finally $\mu_n(M) > \mu_n(M')$.

□

We now want to show that termination is actually in elementary time. We recall that a function f on integers is elementary if there exists a k such that for any n, $f(n)$ can be computed in time $\mathcal{O}(t(n,k))$ where:

$$t(n,0) = 2^n, \qquad t(n,k+1) = 2^{t(n,k)} .$$

Definition 2.7 (tower functions)**.** We define a family of tower functions $t_\alpha(x_1, \ldots, x_n)$ by induction on n where we assume $\alpha \geq 1$ and $x_i \geq 2$:

$$\begin{array}{rcll} t_\alpha() & = & 0 & \\ t_\alpha(x_1, x_2, \ldots, x_n) & = & (\alpha \cdot x_1)^{2^{t_\alpha(x_2, \ldots, x_n)}} & n \geq 1 \end{array}$$

Then we need to prove the following crucial lemma.

Lemma 2.8 (shift)**.** *Assuming $\alpha \geq 1$ and $\beta \geq 2$, the following property holds for the tower functions with $x, \mathbf{x}$ ranging over numbers greater or equal to 2:*

$$t_\alpha(\beta \cdot x, x', \mathbf{x}) \leq t_\alpha(x, \beta \cdot x', \mathbf{x})$$

Now, by a closer look at the shape of the lexicographic ordering during reduction, we are able to compose the decreasing measure with a tower function.

Theorem 2.9 (elementary bound)**.** *Let M be a well-formed term with $\alpha = d(M)$ and let t_α denote the tower function with $\alpha + 1$ arguments. If $M \to M'$ then $t_\alpha(\mu_\alpha(M)) > t_\alpha(\mu_\alpha(M'))$.*

Proof. We illustrate the proof for $\alpha = 2$ and the crucial case where

$$M = \mathsf{let}\ !x = !M_1\ \mathsf{in}\ M_2 \to M' = M_1[M_2/x]$$

Let $\mu_2(M) = (x, y, z)$ such that $x = \omega_2(M)$, $y = \omega_1(M)$ and $z = \omega_0(M)$. We want to show that:

$$t_2(\mu_2(M')) < t_2(\mu_2(M))$$

We have:

$$\begin{array}{rcll} t_2(\mu_2(M')) & \leq & t_2(x \cdot y, y \cdot y, z - 2) & \text{by inequality (2.2)} \\ & \leq & t_2(x, y^3, z - 2) & \text{by Lemma 2.8} \end{array}$$

Hence we are left to show that:

$$t_2(y^3, z-2) < t_2(y, z) \quad \textit{i.e.} \quad (2y^3)^{2^{2(z-2)}} < (2y)^{2^{2z}}$$

We have:

$$(2y^3)^{2^{2(z-2)}} \quad \leq \quad (2y)^{3 \cdot 2^{2(z-2)}}$$

Thus we need to show:

$$3 \cdot 2^{2(z-2)} < 2^{2z}$$

Dividing by 2^{2z} we get:

$$3 \cdot 2^{-4} < 1$$

which is obviously true. Hence $t_2(\mu_2(M')) < t_2(\mu_2(M))$. ☐

This shows that the number of reduction steps of a term M is bound by an elementary function where the height of the tower depends on $d(M)$. We also note that if $M \xrightarrow{*} M'$ then $t_\alpha(\mu_\alpha(M))$ bounds the size of M'. Thus we can conclude with the following corollary.

Corollary 2.10 (elementary time normalisation)**.** *The normalisation of terms of bounded depth can be performed in time elementary in the size of the terms.*

3 Elementary Time in a Modal λ-calculus with Side Effects

In this section, we extend our functional language with side effects operations. By analysing the way side effects act on the depth of occurrences, we extend our depth system to the obtained language. We can then lift the proof of termination in elementary time to programs with side effects that run with a call-by-value reduction strategy.

3.1 A Modal λ-calculus with Multithreading and Regions

We introduce a call-by-value modal λ-calculus endowed with parallel composition and operations to read and write *regions*. We call it $\lambda^{!\mathsf{R}}$. A region is an *abstraction* of a set of dynamically generated values such as imperative references or communication channels. We regard $\lambda^{!\mathsf{R}}$ as an abstract, highly non-deterministic language which entails complexity bounds for more concrete languages featuring references or channels (we will give an example of such a language in Section 5). To this end, it is enough to map the dynamically generated values to their respective regions and observe that the reductions in the concrete languages are simulated in $\lambda^{!\mathsf{R}}$ (see, *e.g.*, [2]).

3.1.1 Syntax

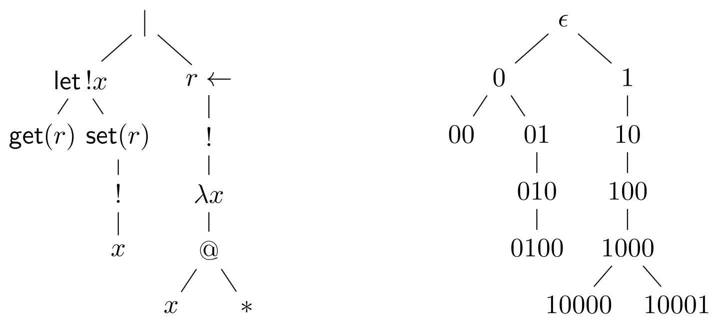
The syntax of the language is described in Table 3.1. We describe the new

$$
\begin{array}{lcll}
x, y, \ldots & & & \text{(Variables)} \\
r, r', \ldots & & & \text{(Regions)} \\
V & ::= & * \mid r \mid x \mid \lambda x.M \mid !V & \text{(Values)} \\
M & ::= & V \mid MM \mid !M \mid \mathsf{let}\ !x = M\ \mathsf{in}\ M & \\
 & & \mathsf{set}(r, V) \mid \mathsf{get}(r) \mid (M \mid M) & \text{(Terms)} \\
S & ::= & (r \leftarrow V) \mid (S \mid S) & \text{(Stores)} \\
P & ::= & M \mid S \mid (P \mid P) & \text{(Programs)} \\
E & ::= & [\,] \mid EM \mid VE \mid !E \mid \mathsf{let}\ !x = E\ \mathsf{in}\ M & \text{(Evaluation Contexts)} \\
C & ::= & [\,] \mid (C \mid P) \mid (P \mid C) & \text{(Static Contexts)}
\end{array}
$$

Table 3.1: Syntax of programs: $\lambda^{!\mathsf{R}}$

operators. We have the usual set of variable $x, y, \ldots$ and a set of regions $r, r', \ldots$. The set of values V contains the unit constant $*$, variables, regions, λ-abstraction and modal values $!V$ which are marked with the *bang* operator '!'. The set of terms M contains values, application, modal terms $!M$, a $\mathsf{let}\,!$ operator, $\mathsf{set}(r, V)$ to write the value V at region r, $\mathsf{get}(r)$ to fetch a value from region r and $(M \mid N)$ to evaluate M and N in parallel. A store S is the composition of several stores $(r \leftarrow V)$ in parallel. A program P is a combination of terms and stores. Evaluation contexts follow a call-by-value discipline. Static contexts C are composed of parallel compositions. Note that stores can only appear in a static context, thus $M(M' \mid (r \leftarrow V))$ is not a legal term. We define $!^n(P \mid P) = (!^n P \mid !^n P)$, and $!^n(r \leftarrow V) = (r \leftarrow V)$. As usual, we abbreviate $(\lambda z.N)M$ with $M;N$, where z is not free in N.

Each program has an *abstract syntax tree* as exemplified in Figure 3.1(a).



(a) (b)

Figure 3.1: Syntax tree and addresses of $P = \mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ \mathsf{set}(r, !x) \mid (r \leftarrow !(\lambda x.x*))$

3.1.2 Operational Semantics

The operational semantics of the language is described in Table 3.2. Programs

$$\begin{array}{rcll} P \mid P' & \equiv & P' \mid P & \text{(Commutativity)} \\ (P \mid P') \mid P'' & \equiv & P \mid (P' \mid P'') & \text{(Associativity)} \end{array}$$

$$\begin{array}{lllcll} E[(\lambda x.M)V] & & & \to & E[M[V/x]] & \\ E[\mathsf{let}\ !x = !V\ \mathsf{in}\ M] & & & \to & E[M[V/x]] & \\ E[\mathsf{set}(r, V)] & & & \to & E[*] & \mid\ (r \leftarrow V) \\ E[\mathsf{get}(r)] & \mid & (r \leftarrow V) & \to & E[V] & \\ E[\mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M] & \mid & (r \leftarrow !V) & \to & E[M[V/x]] & \mid\ (r \leftarrow !V) \end{array}$$

Table 3.2: Semantics of $\lambda^{!\mathsf{R}}$ programs

are considered up to a structural equivalence $\equiv$ which is the least equivalence relation preserved by static contexts, and which contains the equations for α-renaming and for the commutativity and associativity of parallel composition. The reduction rules apply modulo structural equivalence and in a static context C.

When writing to a region, values are accumulated rather than overwritten (remember that $\lambda^{!\mathsf{R}}$ is an abstract language that can simulate more concrete ones where values relating to the same region are associated with distinct addresses).

On the other hand, reading a region amounts to select non-deterministically one of the values associated with the region. We distinguish two rules to read a region. The first *consumes* the value from the store, like when reading a communication channel. The second *copies* the value from the store, like when reading a reference. Note that in this case the value read must be duplicable (of the shape $!V$).

Example 3.1. Program P of Figure 3.1 reduces as follows:

$$\begin{array}{ll} & \mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ \mathsf{set}(r, !x) \mid (r \leftarrow !(\lambda x.x*)) \\ \to & \mathsf{set}(r, !(\lambda x.x*)) \mid (r \leftarrow !(\lambda x.x*)) \\ \to & * \mid (r \leftarrow !(\lambda x.x*)) \mid (r \leftarrow !(\lambda x.x*)) \end{array}$$

3.2 Extended Depth System

We start by analysing the interaction between the depth of the occurrences and side effects. We observe that side effects may increase the depth or generate occurrences at lower depth than the current redex, which violates Property (1) and (2) (see Section 2.2) respectively. Then to find a suitable notion of depth, it is instructive to consider the following program examples where $M_r = \mathsf{let}\ !z = \mathsf{get}(r)\ \mathsf{in}\ !(z*)$.

$$\begin{array}{ll} (A) & E[\mathsf{set}(r, !V)] \\ (B) & \lambda x.\mathsf{set}(r, x); !\mathsf{get}(r) \\ (C) & !(M_r) \mid (r \leftarrow !(\lambda y.M_{r'})) \mid (r' \leftarrow !(\lambda y.*)) \\ (D) & !(M_r) \mid (r \leftarrow !(\lambda y.M_r)) \end{array}$$

(A) Suppose the occurrence $\mathsf{set}(r, !V)$ is at depth $\delta > 0$ in E. Then when evaluating such a term we always end up in a program of the shape $E[*] \mid (r \leftarrow !V)$ where the occurrence $!V$, previously at depth δ, now appears at depth 0. This contradicts Property (2).

(B) If we apply this program to $!V$ we obtain $!!V$, hence Property (1) is violated because from a program of depth 1, we reduce to a program of depth 2. We remark that this is because the read and write operations do not execute at the same depth.

(C) According to our definition, this program has depth 2, however when we reduce it we obtain a term $!^3*$ which has depth 3, hence Property (1) is violated. This is because the occurrence $\lambda y.M_{r'}$ originally at depth 1 in the store, ends up at depth 2 in the place of z applied to $*$.

(D) If we accept circular stores, we can even write diverging programs whose depth is increased by 1 every two reduction steps.

Given these remarks, the rest of this section is devoted to a *revised notion of depth* and to depth system extended with side effects. First, we introduce the following contexts:

$$\Gamma = x_1 : \delta_1, \ldots, x_n : \delta_n \qquad\qquad R = r_1 : \delta_1, \ldots, r_n : \delta_n$$

where δ_i is a natural number. We write $dom(R)$ for the set $\{r_1, \ldots, r_n\}$. We write $R(r_i)$ for the depth δ_i associated with r_i in the context R.

In the sequel, we shall call the notion of depth introduced in Definition 2.1 *naive depth*. We revisit the notion of naive depth as follows.

Definition 3.2 (revised depth)**.** Let P be a program, R a region context where $dom(R)$ contains all the regions of P and $d_n(w)$ the naive depth of an occurrence w of P. If w does not appear under an occurrence $r \leftarrow$ (a store), then the revised depth $d_r(w)$ of w is $d_n(w)$. Otherwise, $d_r(w)$ is $R(r)+d_n(w)$. The revised depth $d_r(P)$ of the program is the maximum revised depth of its occurrences.

Note that the revised depth is relative to a fixed region context. In the sequel we write $d(\_)$ for $d_r(\_)$. On functional terms, this notion of depth is equivalent to the one given in Definition 2.1. However, if we consider the program of Figure 3.1, we now have $d(10) = R(r)$ and $d(100) = d(1000) = d(10000) = d(10001) = R(r) + 1$.

A judgement in the depth system has the shape

$$R; \Gamma \vdash^{\delta} P$$

and it should be interpreted as follows:

> The free variables of $!^{\delta} P$ may only occur at the depth specified by the context Γ, where depths are computed according to R.

The inference rules of the extended depth system are presented in Table 3.3. We comment on the new rules. A region and the constant $*$ may appear at

$$\frac{}{R; \Gamma, x : \delta \vdash^{\delta} x} \qquad \frac{}{R; \Gamma \vdash^{\delta} r} \qquad \frac{}{R; \Gamma \vdash^{\delta} *}$$

$$\frac{\mathsf{FO}(x, M) \leq 1 \quad R; \Gamma, x : \delta \vdash^{\delta} M}{R; \Gamma \vdash^{\delta} \lambda x.M} \qquad \frac{R; \Gamma \vdash^{\delta} M_i \quad i = 1,2}{R; \Gamma \vdash^{\delta} M_1 M_2}$$

$$\frac{R; \Gamma \vdash^{\delta+1} M}{R; \Gamma \vdash^{\delta} !M} \qquad \frac{R; \Gamma \vdash^{\delta} M_1 \quad R; \Gamma, x : (\delta+1) \vdash^{\delta} M_2}{R; \Gamma \vdash^{\delta} \mathsf{let}\ !x = M_1\ \mathsf{in}\ M_2}$$

$$\frac{}{R, r : \delta; \Gamma \vdash^{\delta} \mathsf{get}(r)} \qquad \frac{R, r : \delta; \Gamma \vdash^{\delta} V}{R, r : \delta; \Gamma \vdash^{\delta} \mathsf{set}(r, V)}$$

$$\frac{R, r : \delta; \Gamma \vdash^{\delta} V}{R, r : \delta; \Gamma \vdash^{0} (r \leftarrow V)} \qquad \frac{R; \Gamma \vdash^{\delta} P_i \quad i = 1,2}{R; \Gamma \vdash^{\delta} (P_1 \mid P_2)}$$

Table 3.3: Depth system for programs: $\lambda^{!\mathsf{R}}_{\delta}$

any depth. The key cases are those of read and write: the depth of these two operations is specified by the region context. The current depth of a store is always 0, however, the depth of the value in the store is specified by R (note that it corresponds to the revised definition of depth). We remark that R is constant in a judgement derivation.

Definition 3.3 (well-formedness)**.** A program P is *well-formed* if for some R, Γ, δ a judgement $R; \Gamma \vdash^{\delta} P$ can be derived.

Example 3.4. The program of Figure 3.1 is well-formed with the following derivation where $R(r) = 0$:

$$\dfrac{\dfrac{R;\Gamma \vdash^0 \mathsf{get}(r) \qquad \dfrac{\dfrac{R;\Gamma, x:1 \vdash^1 x}{R;\Gamma, x:1 \vdash^0 !x}}{R;\Gamma, x:1 \vdash^0 \mathsf{set}(r, !x)}}{R;\Gamma \vdash^0 \mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ \mathsf{set}(r, !x)} \qquad \dfrac{\vdots}{R;\Gamma \vdash^0 (r \leftarrow !(\lambda x.x*))}}{R;\Gamma \vdash^0 \mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ \mathsf{set}(r, !x) \mid (r \leftarrow !(\lambda x.x*))}$$

We reconsider the troublesome programs with side effects. Program (A) is well-formed with judgement (i):

$$\begin{array}{lll} R;\Gamma \vdash^0 E[\mathsf{set}(r, !V)] & \text{with } R = r:\delta & (i) \\ R;\Gamma \vdash^0 !M_r \mid (r \leftarrow !(\lambda y.M_{r'})) \mid (r' \leftarrow !(\lambda y.*)) & \text{with } R = r:1, r':2 & (ii) \end{array}$$

Indeed, the occurrence $!V$ is now preserved at depth δ in the store. Program (B) is not well-formed since the read operation requires $R(r) = 1$ and the write operations require $R(r) = 0$. Program (C) is well-formed with judgement (ii); indeed its depth does not increase anymore because $!M_r$ has depth 2 but since $R(r) = 1$ and $R(r') = 2$, $(r \leftarrow !(\lambda y.M_{r'}))$ has depth 3 and $(r' \leftarrow !(\lambda y.*))$ has depth 2. Hence program (C) has already depth 3. Finally, it is worth noticing that the diverging program (D) is not well-formed since $\mathsf{get}(r)$ appears at depth 1 in $!M_r$ and at depth 2 in the store.

Theorem 3.5 (properties on the extended depth system)**.** *The following properties hold:*

1. *If $R;\Gamma \vdash^{\delta} M$ and x occurs free in M then $x : \delta'$ belongs to Γ and all occurrences of x in $!^{\delta} M$ are at depth δ'.*
2. *If $R;\Gamma \vdash^{\delta} P$ then $R;\Gamma, \Gamma' \vdash^{\delta} P$.*
3. *If $R;\Gamma, x:\delta' \vdash^{\delta} M$ and $R;\Gamma \vdash^{\delta'} V$ then $R;\Gamma \vdash^{\delta} M[V/x]$ and $d(!^{\delta} M[V/x]) \leq max(d(!^{\delta} M), d(!^{\delta'} V))$.*
4. *If $R;\Gamma \vdash^0 P$ and $P \to P'$ then $R;\Gamma \vdash^0 P'$ and $d(P) \geq d(P')$.*

3.3 Elementary Bound

In this section, we prove that well-formed programs terminate in elementary time. The measure of Definition 2.5 extends trivially to programs except that to simplify the proofs of the following properties, we assume the occurrences labelled with $|$ and $r \leftarrow$ do not count in the measure and that $\mathsf{set}(r)$ counts for two occurrences such that the measure strictly decreases on the rule $E[\mathsf{set}(r, V)] \to E[*] \mid (r \leftarrow V)$.

We derive a similar termination property:

Proposition 3.6 (termination)**.** *If P is well-formed, $P \to P'$ and $n \geq d(P)$ then $\mu_n(P) > \mu_n(P')$.*

Proof. By a case analysis on the new reduction rules.

- $P \equiv E[\mathsf{set}(r, V)] \to P' \equiv E[*] \mid (r \leftarrow V)$
  If $R; \Gamma \vdash^{\delta} \mathsf{set}(r, V)$ then by 3.5(4) we have $R; \Gamma \vdash^{0} (r \leftarrow V)$ with $R(r) = \delta$. Hence, by definition of the depth, the occurrences in V stay at depth δ in $(r \leftarrow V)$. However, the node $\mathsf{set}(r, V)$ disappears, and both $*$ and $(r \leftarrow V)$ are null occurrences, thus $\omega_\delta(P') = \omega_\delta(P) - 1$. The number of occurrences at other depths stay unchanged, hence $\mu_n(P) > \mu_n(P')$.

- $P \equiv E[\mathsf{get}(r)] \mid (r \leftarrow V) \to P' \equiv E[V]$
  If $R; \Gamma \vdash^{0} (r \leftarrow V)$ with $R(r) = \delta$, then $\mathsf{get}(r)$ must be at depth δ in $E[\ ]$. Hence, by definition of the depth, the occurrences in V stay at depth δ, while the node $\mathsf{get}(r)$ and $|$ disappear. Thus $\omega_\delta(P') = \omega_\delta(P) - 1$ and the number of occurrences at other depths stay unchanged, hence $\mu_n(P) > \mu_n(P')$.

- $P \equiv E[\mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M] \mid (r \leftarrow !V) \to P' \equiv E[M[V/x]] \mid (r \leftarrow !V)$
  This case is the only source of duplication with the reduction rule on $\mathsf{let}\,!$. Suppose $R; \Gamma \vdash^{\delta} \mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M$. Then we must have $R; \Gamma \vdash^{\delta+1} V$. The restrictions on the formation of terms require that x may only occur in M at depth 1 and hence in P at depth $\delta + 1$. Hence the occurrences in V stay at the same depth in $M[V/x]$, while the let, $\mathsf{get}(r)$ and some x nodes disappear, hence $\omega_\delta(P) \leq \omega_\delta(P') - 2$. The number of occurrences of x in M is bound by $k = \omega_{\delta+1}(P) \geq 2$. Thus if $j > \delta$ then $\omega_j(P') \leq k \cdot \omega_j(P)$. Clearly, $\omega_j(M) = \omega_j(M')$ if $j < i$. Hence, we have

$$\mu_n(P') \leq (\mu_n^{i+1}(P) \cdot k, \omega_i(P) - 2, \mu_{i-1}(P)) \tag{3.1}$$

  and $\mu_n(P) > \mu_n(P')$.

□

Then we have the following theorem.

Theorem 3.7 (elementary bound)**.** *Let P be a well-formed program with $\alpha = d(P)$ and let t_α denote the tower function with $\alpha + 1$ arguments. Then if $P \to P'$ then $t_\alpha(\mu_\alpha(P)) > t_\alpha(\mu_\alpha(P'))$.*

Proof. From the proof of termination, we remark that the only new rule that duplicates occurrences is the one that copies from the store. Moreover, the derived inequality (3.1) is exactly the same as the inequality (2.2). Hence the arithmetic of the proof is exactly the same as in the proof of elementary bound for the functional case. □

Corollary 3.8. *The normalisation of programs of bounded depth can be performed in time elementary in the size of the terms.*

4 An Elementary Affine Type System

The depth system entails termination in elementary time but does *not* guarantee that programs 'do not go wrong'. In particular, the introduction and elimination of bangs during evaluation may generate programs that deadlock, *e.g.*,

$$\mathsf{let}\ !y = (\lambda x.x)\ \mathsf{in}\ !(yy) \tag{4.1}$$

is well-formed but the evaluation is stuck. In this section we introduce an elementary affine type system ($\lambda^{!\mathsf{R}}_{EA}$) that guarantees that programs cannot deadlock (except when trying to read an empty store).

The upper part of Table 4.1 introduces the syntax of types and contexts. Types are denoted with $\alpha, \alpha', \ldots$. Note that we distinguish a special behaviour

$$\begin{array}{lcll}
t, t', \ldots & & & \text{(Type variables)} \\
\alpha & ::= & \mathbf{B} \mid A & \text{(Types)} \\
A & ::= & t \mid \mathbf{1} \mid A \multimap \alpha \mid !A \mid \forall t.A \mid \mathsf{Reg}_r A & \text{(Value-types)} \\
\Gamma & ::= & x_1 : (\delta_1, A_1), \ldots, x_n : (\delta_n, A_n) & \text{(Variable contexts)} \\
R & ::= & r_1 : (\delta_1, A_1), \ldots, r_n : (\delta_n, A_n) & \text{(Region contexts)}
\end{array}$$

$$\frac{}{R \downarrow t} \qquad \frac{}{R \downarrow \mathbf{1}} \qquad \frac{}{R \downarrow \mathbf{B}} \qquad \frac{R \downarrow A \quad R \downarrow \alpha}{R \downarrow (A \multimap \alpha)}$$

$$\frac{R \downarrow A}{R \downarrow !A} \qquad \frac{r : (\delta, A) \in R}{R \downarrow \mathsf{Reg}_r A} \qquad \frac{R \downarrow A \quad t \notin R}{R \downarrow \forall t.A}$$

$$\frac{\forall r : (\delta, A) \in R \quad R \downarrow A}{R \vdash} \qquad \frac{R \vdash \quad R \downarrow \alpha}{R \vdash \alpha}$$

$$\frac{\forall x : (\delta, A) \in \Gamma \quad R \vdash A}{R \vdash \Gamma}$$

Table 4.1: Types and contexts

type $\mathbf{B}$ which is given to the entities of the language which are not supposed to return a value (such as a store or several terms in parallel) while types of entities that may return a value are denoted with A. Among the types A, we distinguish type variables $t, t', \ldots$, a terminal type $\mathbf{1}$, an affine functional type $A \multimap \alpha$, the type $!A$ of terms of type A that can be duplicated, the type $\forall t.A$ of polymorphic terms and the type $\mathsf{Reg}_r A$ of the region r containing values of type A. Hereby types may depend on regions.

In contexts, natural numbers δ_i play the same role as in the depth system. Writing $x : (\delta, A)$ means that the variable x ranges on values of type A and may occur at depth δ. Writing $r : (\delta, A)$ means that addresses related to region r contain values of type A and that read and writes on r may only happen at depth δ. The typing system will additionally guarantee that whenever we use a type $\mathsf{Reg}_r A$ the region context contains an hypothesis $r : (\delta, A)$.

Because types depend on regions, we have to be careful in stating in Table 4.1 when a region-context and a type are compatible ($R \downarrow \alpha$), when a region context is well-formed ($R \vdash$), when a type is well-formed in a region context ($R \vdash \alpha$) and when a context is well-formed in a region context ($R \vdash \Gamma$). A more informal way to express the condition is to say that a judgement $r_1 : (\delta_1, A_1), \ldots, r_n : (\delta_n, A_n) \vdash \alpha$ is well formed provided that: (1) all the region names occurring in the types $A_1, \ldots, A_n, \alpha$ belong to the set $\{r_1, \ldots, r_n\}$, (2) all types of the shape $\mathsf{Reg}_{r_i} B$ with $i \in \{1, \ldots, n\}$ and occurring in the types $A_1, \ldots, A_n, \alpha$ are such that $B = A_i$. We notice the following substitution property on types.

Proposition 4.1. *If $R \vdash \forall t.A$ and $R \vdash B$ then $R \vdash A[B/t]$.*

Example 4.2. One may verify that $r : (\delta, \mathbf{1} \multimap \mathbf{1}) \vdash \mathsf{Reg}_r(\mathbf{1} \multimap \mathbf{1})$ can be derived while the following judgements cannot: $r : (\delta, \mathbf{1}) \vdash \mathsf{Reg}_r(\mathbf{1} \multimap \mathbf{1})$, $r : (\delta, \mathsf{Reg}_r \mathbf{1}) \vdash \mathbf{1}$.

A typing judgement takes the form:

$$R; \Gamma \vdash^{\delta} P : \alpha$$

It attributes a type α to the program P at depth δ, in the region context R and the context Γ. Table 4.2 introduces an elementary affine type system *with regions*. One can see that the δ's are treated as in the depth system. Note that a region r may occur at any depth. In the $\mathsf{let}\,!$ rule, M should be of type $!A$ since x of type A appears one level deeper. A program in parallel with a store should have the type of the program since we might be interested in the value the program reduces to; however, two programs in parallel cannot reduce to a single value, hence we give them a behaviour type. The polymorphic rules are straightforward where $t \notin (R; \Gamma)$ means t does not occur free in a type of R or Γ.

Example 4.3. The well-formed program (C) can be given the following typing judgement: $R; \_ \vdash^0 !(M_r) \mid (r \leftarrow !(\lambda y.M_{r'})) \mid (r' \leftarrow !(\lambda y.*)) : !!\mathbf{1}$ where: $R = r : (1, !(\mathbf{1} \multimap \mathbf{1})), r' : (2, !(\mathbf{1} \multimap \mathbf{1}))$. Also, we remark that the deadlocking program (4.1) admits no typing derivation.

$$\frac{R \vdash \Gamma \quad x : (\delta, A) \in \Gamma}{R; \Gamma \vdash^{\delta} x : A} \qquad \frac{R \vdash \Gamma}{R; \Gamma \vdash^{\delta} * : \mathbf{1}} \qquad \frac{R \vdash \Gamma \quad r : (\delta', A) \in R}{R; \Gamma \vdash^{\delta} r : \mathsf{Reg}_r A}$$

$$\frac{\mathsf{FO}(x, M) \leq 1 \quad R; \Gamma, x : (\delta, A) \vdash^{\delta} M : \alpha}{R; \Gamma \vdash^{\delta} \lambda x.M : A \multimap \alpha} \qquad \frac{R; \Gamma \vdash^{\delta} M : A \multimap \alpha \quad R; \Gamma \vdash^{\delta} N : A}{R; \Gamma \vdash^{\delta} MN : \alpha}$$

$$\frac{R; \Gamma \vdash^{\delta+1} M : A}{R; \Gamma \vdash^{\delta} !M : !A} \qquad \frac{R; \Gamma \vdash^{\delta} M : !A \quad R; \Gamma, x : (\delta + 1, A) \vdash^{\delta} N : B}{R; \Gamma \vdash^{\delta} \mathsf{let}\ !x = M\ \mathsf{in}\ N : B}$$

$$\frac{R; \Gamma \vdash^{\delta} M : A \quad t \notin (R; \Gamma)}{R; \Gamma \vdash^{\delta} M : \forall t.A} \qquad \frac{R; \Gamma \vdash^{\delta} M : \forall t.A \quad R \vdash B}{R; \Gamma \vdash^{\delta} M : A[B/t]}$$

$$\frac{r : (\delta, A) \in R \quad R \vdash \Gamma}{R; \Gamma \vdash^{\delta} \mathsf{get}(r) : A} \qquad \frac{r : (\delta, A) \in R \quad R; \Gamma \vdash^{\delta} V : A}{R; \Gamma \vdash^{\delta} \mathsf{set}(r, V) : \mathbf{1}} \qquad \frac{r : (\delta, A) \in R \quad R; \Gamma \vdash^{\delta} V : A}{R; \Gamma \vdash^{0} (r \leftarrow V) : \mathbf{B}}$$

$$\frac{R; \Gamma \vdash^{\delta} P : \alpha \quad R; \Gamma \vdash^{\delta} S : \mathbf{B}}{R; \Gamma \vdash^{\delta} (P \mid S) : \alpha} \qquad \frac{P_i \text{ not a store } i = 1, 2 \quad R; \Gamma \vdash^{\delta} P_i : \alpha_i}{R; \Gamma \vdash^{\delta} (P_1 \mid P_2) : \mathbf{B}}$$

Table 4.2: An elementary affine type system: $\lambda^{!\mathsf{R}}_{EA}$

Theorem 4.4 (subject reduction and progress)**.** *The following properties hold.*

1. (Well-formedness) *Well-typed programs are well-formed.*
2. (Weakening) *If $R; \Gamma \vdash^{\delta} P : \alpha$ and $R \vdash \Gamma, \Gamma'$ then $R; \Gamma, \Gamma' \vdash^{\delta} P : \alpha$.*
3. (Substitution) *If $R; \Gamma, x : (\delta', A) \vdash^{\delta} M : \alpha$ and $R; \Gamma' \vdash^{\delta'} V : A$ and $R \vdash \Gamma, \Gamma'$ then $R; \Gamma, \Gamma' \vdash^{\delta} M[V/x] : \alpha$.*
4. (Subject Reduction) *If $R; \Gamma \vdash^{\delta} P : \alpha$ and $P \to P'$ then $R; \Gamma \vdash^{\delta} P' : \alpha$.*
5. (Progress) *Suppose P is a closed typable program which cannot reduce. Then P is structurally equivalent to a program*

$$M_1 \mid \cdots \mid M_m \mid S_1 \mid \cdots \mid S_n \quad m, n \geq 0$$

where M_i is either a value or can be decomposed as a term $E[\mathsf{get}(r)]$ such that no value is associated with the region r in the stores $S_1, \ldots, S_n$.

5 Expressivity

In this section, we consider two results that illustrate the expressivity of the elementary affine type system. First we show that all elementary functions can be represented and second we develop an example of iterative program with side effects.

5.1 Completeness

The representation result just relies on the functional core of the language $\lambda^!_{EA}$. Building on the standard concept of Church numeral, Table 5.1 provides a representation for natural numbers and the multiplication function. We denote with

$$
\begin{array}{rcll}
\mathsf{N} & = & \forall t.!(t \multimap t) \multimap !(t \multimap t) & \text{(type of numerals)} \\
\\
\overline{n} & : & \mathsf{N} & \text{(numerals)} \\
\overline{n} & = & \lambda f.\mathsf{let}\ !f = f\ \mathsf{in}\ !(\lambda x.f(\cdots(fx)\cdots)) & \\
\\
\mathsf{mult} & : & \mathsf{N} \multimap (\mathsf{N} \multimap \mathsf{N}) & \text{(multiplication)} \\
\mathsf{mult} & = & \lambda n.\lambda m.\lambda f.\mathsf{let}\ !f = f\ \mathsf{in}\ n(m!f) &
\end{array}
$$

Table 5.1: Representation of natural numbers and the multiplication function

$\mathbb{N}$ the set of natural numbers. The precise notion of representation is spelled out in the following definitions where by strong β-reduction we mean that reduction under λ's is allowed.

Definition 5.1 (number representation)**.** Let $\emptyset \vdash^\delta M : \mathsf{N}$. We say M *represents* $n \in \mathbb{N}$, written $M \Vdash n$, if, by using a strong β-reduction relation, $M \xrightarrow{*} \overline{n}$.

Definition 5.2 (function representation)**.** Let $\emptyset \vdash^\delta F : (\mathsf{N}_1 \multimap \ldots \multimap \mathsf{N}_k) \multimap !^p\mathsf{N}$ where $p \geq 0$ and $f : \mathbb{N}^k \to \mathbb{N}$. We say F *represents* f, written $F \Vdash f$, if for all M_i and $n_i \in \mathbb{N}$ where $1 \leq i \leq k$ such that $\emptyset \vdash^\delta M_i : N$ and $M_i \Vdash n_i$, $FM_1 \ldots M_k \Vdash f(n_1, \ldots, n_k)$.

Elementary functions are also characterized as the smallest class of functions containing zero, successor, projection, subtraction and which is closed by composition and bounded summation/product. These functions can be represented in the sense of Definition 5.2 by adapting the proofs from Danos and Joinet [10].

Theorem 5.3 (completeness)**.** *Every elementary function is representable in* $\lambda^!_{EA}$.

5.2 Iteration with Side Effects

We rely on a slightly modified language where reads, writes and stores relate to concrete addresses rather than to abstract regions. In particular, we introduce terms of the form $\nu x\ M$ to generate a fresh address name x whose scope is M. One can then write the following program:

$$\nu x\ ((\lambda y.\mathsf{set}(y, V))x) \xrightarrow{*} \nu x\ * \mid (x \leftarrow V)$$

where x and y relate to a region r, *i.e.* they are of type $\mathsf{Reg}_r A$. Our type system can be easily adapted by associating region types with the address names. Next we show that it is possible to program the iteration of operations producing a side effect on an inductive data structure. Specifically, in the following we show how to iterate, possibly in parallel, an update operation on a list of addresses of the store. The examples have been tested on a running implementation of the language.

Following Church encodings, we define the representation of lists and the associated iterator in Table 5.2. Here is the function multiplying the numeral

$\mathsf{List}\ A$	$=$	$\forall t.!(A \multimap t \multimap t) \multimap !(t \multimap t)$	(type of lists)
$[u_1, \ldots, u_n]$	$:$	$\mathsf{List}\ A$	(list represent.)
$[u_1, \ldots, u_n]$	$=$	$\lambda f.\mathsf{let}\ !f = f\ \mathsf{in}\ !(\lambda x.fu_1(fu_2 \ldots (fu_nx)))$	
$\mathsf{list\_it}$	$:$	$\forall u.\forall t.!(u \multimap t \multimap t) \multimap \mathsf{List}\ u \multimap !t \multimap !t$	(iterator)
$\mathsf{list\_it}$	$=$	$\lambda f.\lambda l.\lambda z.\mathsf{let}\ !z = z\ \mathsf{in}\ \mathsf{let}\ !y = lf\ \mathsf{in}\ !(yz)$	

Table 5.2: Representation of lists

pointed by an address at region r:

$$\begin{array}{lcl} \mathsf{update} & : & !\mathsf{Reg}_r\mathsf{N} \multimap !\mathbf{1} \multimap !\mathbf{1} \\ \mathsf{update} & = & \lambda x.\mathsf{let}\ !x = x\ \mathsf{in}\ \lambda z.!((\lambda y.\mathsf{set}(x, y))(\mathsf{mult}\ \overline{2}\ \mathsf{get}(x))) \end{array}$$

Consider the following list of addresses and stores:

$$[!x, !y, !z] \mid (x \leftarrow \overline{m}) \mid (y \leftarrow \overline{n}) \mid (z \leftarrow \overline{p})$$

Note that the bang constructors are needed to match the type $!\mathsf{Reg}_r\mathsf{N}$ of the argument of update. Then we define the iteration as:

$$\mathsf{run} : !!\mathbf{1} \qquad \mathsf{run} = \mathsf{list\_it}\ !\mathsf{update}\ [!x, !y, !z]\ !!*$$

Notice that it is well-typed with $R = r : (2, \mathsf{N})$ since both the read and the write appear at depth 2. Finally, the program reduces by updating the store as expected:

$$\begin{array}{rl} & \mathsf{run} \mid (x \leftarrow \overline{m}) \mid (y \leftarrow \overline{n}) \mid (z \leftarrow \overline{p}) \\ \xrightarrow{*} & !!\mathbf{1} \mid (x \leftarrow \overline{2m}) \mid (y \leftarrow \overline{2n}) \mid (z \leftarrow \overline{2p}) \end{array}$$

Building on this example, suppose we want to write a program with three concurrent threads where each thread multiplies by 2 the memory cells pointed by a list. Here is a function waiting to apply a functional f to a value x in three concurrent threads:

$$\begin{array}{lcl} \mathsf{gen\_threads} & : & \forall t.\forall t'.!(t \multimap t') \multimap !t \multimap \mathbf{B} \\ \mathsf{gen\_threads} & = & \lambda f.\mathsf{let}\ !f = f\ \mathsf{in}\ \lambda x.\mathsf{let}\ !x = x\ \mathsf{in}\ !(fx) \mid !(fx) \mid !(fx) \end{array}$$

We define the functional F as run but parametric in the list:

$$F : \mathsf{List}\ !\mathsf{Reg}_r \mathsf{N} \multimap !!\mathbf{1} \qquad F = \lambda l.\mathsf{list\_it}\ !\mathsf{update}\ l\ !!*$$

And the final term is simply:

$$\mathsf{run\_threads} : \mathbf{B} \qquad \mathsf{run\_threads} = \mathsf{gen\_threads}\ !F\ ![!x, !y, !z]$$

where $R = r : (3, !\mathsf{N})$. Our program then reduces as follows:

$$\begin{array}{lccccccc} & \mathsf{run\_threads} & \mid & (x \leftarrow \overline{m}) & \mid & (y \leftarrow \overline{n}) & \mid & (z \leftarrow \overline{p}) \\ \xrightarrow{*} & !!!\mathbf{1} \mid !!!\mathbf{1} \mid !!!\mathbf{1} & \mid & (x \leftarrow \overline{8m}) & \mid & (y \leftarrow \overline{8n}) & \mid & (z \leftarrow \overline{8p}) \end{array}$$

Note that different thread interleavings are possible but in this particular case the reduction is confluent.

6 Conclusion

We have introduced a type system for a higher-order functional language with multithreading and side effects that guarantees termination in elementary time thus providing a significant extension of previous work that had focused on purely functional programs. In the proposed approach, the depth system plays a key role and allows for a relatively simple presentation. In particular we notice that we can dispense both with the notion of *stratified region* that arises in recent work on the termination of higher-order programs with side effects [1, 7] and with the distinction between affine and intuitionistic hypotheses [6, 2].

As a future work, we would like to adapt our approach to polynomial time. In another direction, one could ask if it is possible to program in a simplified language without bangs and then try to infer types or depths.

Acknowledgements We would like to thank Patrick Baillot for numerous helpful discussions and a careful reading on a draft version of this report.

References

[1] R. M. Amadio. On stratified regions. In *APLAS'09*, volume 5904 of *LNCS*, pages 210–225. Springer, 2009.

[2] R. M. Amadio, P. Baillot, and A. Madet. An affine-intuitionistic system of types and effects: confluence and termination. Technical report, Laboratoire PPS, 2009. http://hal.archives-ouvertes.fr/hal-00438101/.

[3] A. Asperti and L. Roversi. Intuitionistic light affine logic. *ACM Trans. Comput. Log.*, 3(1):137–175, 2002.

[4] P. Baillot, M. Gaboardi, and V. Mogbil. A polytime functional language from light linear logic. In *ESOP'10*, volume 6012 of *LNCS*, pages 104–124. Springer, 2010.

[5] P. Baillot and K. Terui. A feasible algorithm for typing in elementary affine logic. In *TLCA'05*, volume 3461 of *LNCS*, pages 55–70. Springer, 2005.

[6] A. Barber. Dual intuitionistic linear logic. Technical Report ECS-LFCS-96-347, The Laboratory for Foundations of Computer Science, University of Edinburgh, 1996.

[7] G. Boudol. Typing termination in a higher-order concurrent imperative language. *Inf. Comput.*, 208(6):716–736, 2010.

[8] P. Coppola, U. Dal Lago, and S. Ronchi Della Rocca. Light logics and the call-by-value lambda calculus. *Logical Methods in Computer Science*, 4(4), 2008.

[9] P. Coppola and S. Martini. Optimizing optimal reduction: A type inference algorithm for elementary affine logic. *ACM Trans. Comput. Log.*, 7:219–260, 2006.

[10] V. Danos and J.-B. Joinet. Linear logic and elementary time. *Inf. Comput.*, 183(1):123 – 137, 2003.

[11] J.-Y. Girard. Light linear logic. *Inf. Comput.*, 143(2):175–204, 1998.

[12] U. D. Lago, S. Martini, and D. Sangiorgi. Light logics and higher-order processes. In *EXPRESS'10*, volume 41 of *EPTCS*, pages 46–60, 2010.

[13] K. Terui. Light affine lambda calculus and polynomial time strong normalization. *Archive for Mathematical Logic*, 46(3-4):253–280, 2007.

A Proofs

A.1 Proof of theorem 3.5

1. We consider the last rule applied in the typing of M.

   - $\Gamma, x : \delta \vdash^{\delta} x$. The only free variable is x and indeed it is at depth δ in $!^{\delta}x$.
   - $\Gamma \vdash^{\delta} \lambda y.M$ is derived from $\Gamma, y : \delta \vdash^{\delta} M$. If x is free in $\lambda y.M$ then $x \neq y$ and x is free in M. By inductive hypothesis, $x : \delta' \in \Gamma, y : \delta$ and all occurrences of x in $!^{\delta}M$ are at depth δ'. By definition of depth, the same is true for $!^{\delta}(\lambda y.M)$.
   - $\Gamma \vdash^{\delta} (M_1 M_2)$ is derived from $\Gamma \vdash^{\delta} M_i$ for $i = 1, 2$. By inductive hypothesis, $x : \delta' \in \Gamma$ and all occurrences of x in $!^{\delta}M_i$, $i = 1, 2$ are at depth δ'. By definition of depth, the same is true for $!^{\delta}(M_1 M_2)$.
   - $\Gamma \vdash^{\delta} !M$ is derived from $\Gamma \vdash^{\delta+1} M$. By inductive hypothesis, $x : \delta' \in \Gamma$ and all occurrences of x in $!^{\delta+1}M$ are at depth δ' and notice that $!^{\delta+1}M = !^{\delta}(!M)$.
   - $\Gamma \vdash^{\delta} \mathsf{let}\ !y = M_1\ \mathsf{in}\ M_2$ is derived from $\Gamma \vdash^{\delta} M_1$ and $\Gamma, y : (\delta + 1) \vdash^{\delta} M_2$. Without loss of generality, assume $x \neq y$. By inductive hypothesis, $x : \delta' \in \Gamma$ and all occurrences of x in $!^{\delta}M_i$, $i = 1, 2$ are at depth δ'. By definition of depth, the same is true for $!^{\delta}(\mathsf{let}\ !y = M_1\ \mathsf{in}\ M_2)$.
   - $M \equiv *$ or $M \equiv r$ or $M \equiv \mathsf{get}(r)$. There is no free variable in these terms.
   - $M \equiv \mathsf{let}\ !y = \mathsf{get}(r)\ \mathsf{in}\ N$. We have

     $$\frac{R, r : \delta; \Gamma, y : (\delta + 1) \vdash^{\delta} N}{R, r : \delta; \Gamma \vdash^{\delta} \mathsf{let}\ !y = \mathsf{get}(r)\ \mathsf{in}\ N}$$

     If x occurs free in M then x occurs free in N. By induction hypothesis, $x : \delta' \in \Gamma$ and all occurrences of x in $!^{\delta}N$ are at depth δ'. By definition of the depth, this is also true for $!^{\delta}(\mathsf{let}\ !y = \mathsf{get}(r)\ \mathsf{in}\ N)$.
   - $M \equiv \mathsf{set}(r, V)$. We have

     $$\frac{R, r : \delta; \Gamma \vdash^{\delta} V}{R, r : \delta; \Gamma \vdash^{\delta} \mathsf{set}(r, V)}$$

     If x occurs free in $\mathsf{set}(r, V)$ then x occurs free in V. By induction hypothesis, $x : \delta' \in \Gamma$ and all occurrences of x in $!^{\delta}V$ are at depth δ'. By definition of the depth, this is also true for $!^{\delta}(\mathsf{set}(r, V))$.
   - $M \equiv (M_1 \mid M_2)$. We have

     $$\frac{R; \Gamma \vdash^{\delta} M_i \qquad i = 1, 2}{R; \Gamma \vdash^{\delta} (M_1 \mid M_2)}$$

If x occurs free in M then x occurs free in M_i, $i = 1, 2$. By induction hypothesis, $x : \delta' \in \Gamma$ and all occurrences of x in $!^\delta M_i$, $i = 1, 2$, are at depth δ'. By definition of depth, the same is true of $!^\delta(M_1 \mid M_2)$.

2. All the rules can be weakened by adding a context Γ'.

3. If x is not free in M, we just have to check that any proof of $\Gamma, x : \delta' \vdash^\delta M$ can be transformed into a proof of $\Gamma \vdash^\delta M$.

   So let us assume x is free in M.

   We consider first the bound on the depth. By (1), we know that all occurrences of x in $!^\delta M$ are at depth δ'. By definition of depth, it follows that $\delta' \geq \delta$ and the occurrences of x in M are at depth $(\delta' - \delta)$. An occurrence in $!^{\delta'} V$ at depth $\delta' + \delta''$ will generate an occurrence in $!^\delta M[V/x]$ at the same depth $\delta + (\delta' - \delta) + \delta''$.

   Next, we proceed by induction on the derivation of $\Gamma, x : \delta' \vdash^\delta M$.

   - $\Gamma, x : \delta \vdash^\delta x$. Then $\delta = \delta'$, $x[V/x] = V$, and by hypothesis $\Gamma \vdash^{\delta'} V$.
   - $\Gamma, x : \delta' \vdash^\delta \lambda y.M$ is derived from $\Gamma, x : \delta', y : \delta \vdash^\delta M$, with $x \neq y$ and y not occurring in N. By (2), $\Gamma, y : \delta \vdash^{\delta'} V$. By inductive hypothesis, $\Gamma, y : \delta \vdash^\delta M[V/x]$, and then we conclude $\Gamma \vdash^\delta (\lambda y.M)[V/x]$.
   - $\Gamma, x : \delta' \vdash^\delta (M_1 M_2)$ is derived from $\Gamma, x : \delta' \vdash^\delta M_i$, for $i = 1, 2$. By inductive hypothesis, $\Gamma \vdash^\delta M_i[V/x]$, for $i = 1, 2$ and then we conclude $\Gamma \vdash^\delta (M_1 M_2)[V/x]$.
   - $\Gamma, x : \delta' \vdash^\delta !M$ is derived from $\Gamma, x : \delta' \vdash^{\delta+1} M$. By inductive hypothesis, $\Gamma \vdash^{\delta+1} M[V/x]$, and then we conclude $\Gamma \vdash^\delta !M[V/x]$.
   - $\Gamma, x : \delta' \vdash^\delta \mathsf{let}\ !y = M_1\ \mathsf{in}\ M_2$, with $x \neq y$ and y not free in V is derived from $\Gamma, x : \delta' \vdash^\delta M_1$ and $\Gamma, x : \delta', y : (\delta + 1) \vdash^\delta M_2$. By inductive hypothesis, $\Gamma \vdash^\delta M_1[V/x]$ $\Gamma, y : (\delta + 1) \vdash^\delta M_2[V/x]$, and then we conclude $\Gamma \vdash^\delta (\mathsf{let}\ !y = M_1\ \mathsf{in}\ M_2)[V/x]$.
   - $M \equiv \mathsf{let}\ !y = \mathsf{get}(r)\ \mathsf{in}\ M_1$. We have

     $$\frac{R, r : \delta; \Gamma, x : \delta', y : (\delta + 1) \vdash^\delta M_1}{R, r : \delta; \Gamma, x : \delta' \vdash^\delta \mathsf{let}\ !y = \mathsf{get}(r)\ \mathsf{in}\ M_1}$$

     By induction hypothesis we get

     $$R, r : \delta; \Gamma, y : (\delta + 1) \vdash^\delta M_1[V/x]$$

     and hence we derive

     $$R, r : \delta; \Gamma \vdash^\delta (\mathsf{let}\ !y = \mathsf{get}(r)\ \mathsf{in}\ M_1)[V/x]$$

   - $M \equiv \mathsf{set}(r, V')$. We have

     $$\frac{R, r : \delta; \Gamma, x : \delta' \vdash^\delta V'}{R, r : \delta; \Gamma, x : \delta' \vdash^\delta \mathsf{set}(r, V')}$$

By induction hypothesis we get

$$R, r : \delta; \Gamma \vdash^{\delta} V'[V/x]$$

and hence we derive

$$R, r : \delta; \Gamma \vdash^{\delta} (\mathsf{set}(r, V'))[V/x]$$

- $M \equiv (M_1 \mid M_2)$. We have

$$\frac{R; \Gamma, x : \delta' \vdash^{\delta} M_i \qquad i = 1, 2}{R; \Gamma, x : \delta' \vdash^{\delta} (M_1 \mid M_2)}$$

  By induction hypothesis we derive

$$R; \Gamma \vdash^{\delta} M_i[V/x]$$

  and hence we derive

$$R; \Gamma \vdash^{\delta} (M_1 \mid M_2)[V/x]$$

4. We proceed by case analysis on the reduction rules.

- Suppose $\Gamma \vdash^0 E[(\lambda x.M)V]$. Then for some Γ' extending Γ and $\delta \geq 0$ we must have $\Gamma' \vdash^{\delta} (\lambda x.M)V$. This must be derived from $\Gamma', x : \delta \vdash^{\delta} M$ and $\Gamma' \vdash^{\delta} V$. By (3), with $\delta = \delta'$, it follows that $\Gamma' \vdash^{\delta} M[V/x]$ and that the depth of an occurrence in $E[M[V/x]]$ is bounded by the depth of an occurrence which is already in $E[(\lambda x.M)V]$. Moreover, we can derive $\Gamma \vdash^0 E[M[V/x]]$.
- Suppose $\Gamma \vdash^0 E[\mathsf{let}\ !x = !V \ \mathsf{in}\ M]$. Then for some Γ' extending Γ and $\delta \geq 0$ we must have $\Gamma' \vdash^{\delta} \mathsf{let}\ !x = !V \ \mathsf{in}\ M$. This must be derived from $\Gamma', x : (\delta + 1) \vdash^{\delta} M$ and $\Gamma' \vdash^{(\delta+1)} V$. By (3), with $(\delta + 1) = \delta'$, it follows that $\Gamma' \vdash^{\delta} M[V/x]$ and that the depth of an occurrence in $E[M[V/x]]$ is bounded by the depth of an occurrence which is already in $E[\mathsf{let}\ !x = !V \ \mathsf{in}\ M]$. Moreover, we can derive $\Gamma \vdash^0 E[M[V/x]]$.
- $E[\mathsf{set}(r, V)] \to E[*] \mid (r \leftarrow V)$
  We have $R; \Gamma \vdash^0 E[\mathsf{set}(r, V)]$ from which we derive

$$\frac{R; \Gamma \vdash^{\delta} V}{R; \Gamma \vdash^{\delta} \mathsf{set}(r, V)}$$

  for some $\delta \geq 0$, with $r : \delta \in R$. Hence we can derive

$$\frac{R; \Gamma \vdash^{\delta} V}{R; \Gamma \vdash^{0} (r \leftarrow V)}$$

  Moreover, we have as an axiom $R; \Gamma \vdash^{\delta} *$ thus we can derive $R; \Gamma \vdash^0 E[*]$. Applying the parallel rule we finally get

$$R; \Gamma \vdash^0 E[*] \mid (r \leftarrow V)$$

  Concerning the depth bound, clearly we have $d(E[*] \mid (r \leftarrow V)) = d(E[\mathsf{set}(r, V)])$.

- $E[\mathsf{get}(r)] \mid (r \leftarrow V) \to E[M[V/x]]$
  We have $R; \Gamma \vdash^0 E[\mathsf{get}(r)] \mid (r \leftarrow V)$ from which we derive

$$\frac{}{R; \Gamma \vdash^\delta \mathsf{get}(r)}$$

  and

$$\frac{}{R; \Gamma, x : \delta \vdash^\delta M}$$

  for some $\delta \geq 0$, with $r : \delta \in R$, and

$$\frac{R; \Gamma \vdash^\delta V}{R; \Gamma \vdash^0 (r \leftarrow V)}$$

  Hence we can derive

$$R; \Gamma \vdash^0 E[V]$$

  Concerning the depth bound, clearly we have $d(E[V]) = d(E[\mathsf{get}(r)] \mid (r \leftarrow V))$.

- $E[\mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M] \mid (r \leftarrow !V) \to E[M[V/x]] \mid (r \leftarrow !V)$
  We have $R; \Gamma \vdash^0 E[\mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M] \mid \mathsf{r}!V$ from which we derive

$$\frac{R; \Gamma', x : (\delta + 1) \vdash^\delta M}{R; \Gamma' \vdash^\delta \mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M}$$

  for some $\delta \geq 0$ with $r : \delta \in R$, and some Γ' extending Γ. We also derive

$$\frac{\dfrac{R; \Gamma \vdash^{\delta+1} V}{R; \Gamma \vdash^\delta !V}}{R; \Gamma \vdash^0 (r \leftarrow !V)}$$

  By (2) we get $R; \Gamma' \vdash^{\delta+1} V$. By (3) we derive

$$R; \Gamma' \vdash^\delta M[V/x]$$

  hence

$$R; \Gamma \vdash^0 E[M[V/x]]$$

  and finally

$$R; \Gamma \vdash^0 E[M[V/x]] \mid (r \leftarrow !V)$$

  Concerning the depth bound, by (3), the depth of an occurrence in $E[M[V/x]] \mid (r \leftarrow !V)$ is bounded by the depth of an occurrence which is already in $E[\mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M] \mid (r \leftarrow !V)$, hence $d(E[M[V/x]] \mid (r \leftarrow !V)) \leq d(E[\mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M] \mid (r \leftarrow !V))$.

A.2 Proof of proposition 3.6

We do this by case analysis on the reduction rules.

- $P = E[(\lambda x.M)V] \to P' = E[M[V/x]]$
  Let the occurrence of the redex $(\lambda x.M)V$ be at depth i. The restrictions on the formation of terms require that x occurs at most once in M at depth 0. Then $\omega_i(P) - 3 \geq \omega_i(P')$ because we remove the nodes for application and λ-abstraction and either V disappears or the occurrence of the variable x in M disappears (both being at the same depth as the redex). Clearly $\omega_j(P) = \omega_j(P')$ if $j \neq i$, hence

  $$\mu_n(P') \leq (\omega_n(P), \ldots, \omega_{i+1}(P), \omega_i(P) - 3, \mu_{i-1}(P)) \qquad \text{(A.1)}$$

  and $\mu_n(P) > \mu_n(P')$.

- $P = E[\mathsf{let}\ !x = !V\ \mathsf{in}\ M] \to P' = E[M[V/x]]$
  Let the occurrence of the redex $\mathsf{let}\ !x = !V\ \mathsf{in}\ M$ be at depth i. The restrictions on the formation of terms require that x may only occur in M at depth 1 and hence in P at depth $i+1$. We have that $\omega_i(P') = \omega_i(P) - 2$ because the $\mathsf{let}!$ node disappear. Clearly, $\omega_j(P) = \omega_j(P')$ if $j < i$. The number of occurrences of x in M is bounded by $k = \omega_{i+1}(P) \geq 2$. Thus if $j > i$ then $\omega_j(P') \leq k \cdot \omega_j(P)$. Let's write, for $0 \leq i \leq n$:

  $$\mu_n^i(P) \cdot k = (\omega_n(P) \cdot k, \omega_{n-1}(P) \cdot k, \ldots, \omega_i(P) \cdot k)$$

  Then we have

  $$\mu_n(P') \leq (\mu_n^{i+1}(P) \cdot k, \omega_i(P) - 2, \mu_{i-1}(P)) \qquad \text{(A.2)}$$

  and finally $\mu_n(P) > \mu_n(P')$.

- $P \equiv E[\mathsf{set}(r, V)] \to P' \equiv E[*] \mid (r \leftarrow V)$
  If $R; \Gamma \vdash^\delta \mathsf{set}(r, V)$ then by 3.5(4) we have $R; \Gamma \vdash^0 (r \leftarrow V)$ with $R(r) = \delta$. Hence, by definition of the depth, the occurrences in V stay at depth δ in $(r \leftarrow V)$. Moreover, the node $\mathsf{set}(r, V)$ disappears and the nodes $*$, $|$, and $r \leftarrow$ appear. Recall that we assume the occurrences $|$ and $r \leftarrow$ do not count in the measure and that $\mathsf{set}(r)$ counts for two occurrences. Thus $\omega_\delta(P') = \omega_\delta(P) - 2 + 1 + 0 + 0$. The number of occurrences at other depths stay unchanged, hence $\mu_n(P) > \mu_n(P')$.

- $P \equiv E[\mathsf{get}(r)] \mid (r \leftarrow V) \to P' \equiv E[V]$
  If $R; \Gamma \vdash^0 (r \leftarrow V)$ with $R(r) = \delta$, then $\mathsf{get}(r)$ must be at depth δ in $E[\ ]$. Hence, by definition of the depth, the occurrences in V stay at depth δ, while the node $\mathsf{get}(r)$ and $|$ disappear. Thus $\omega_\delta(P') = \omega_\delta(P) - 1$ and the number of occurrences at other depths stay unchanged, hence $\mu_n(P) > \mu_n(P')$.

- $P \equiv E[\mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M] \mid (r \leftarrow !V) \rightarrow P' \equiv E[M[V/x]] \mid (r \leftarrow !V)$
  This case is the only source of duplication with the reduction rule on $\mathsf{let}\,!$. Suppose $R; \Gamma \vdash^{\delta} \mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M$. Then we must have $R; \Gamma \vdash^{\delta+1} V$. The restrictions on the formation of terms require that x may only occur in M at depth 1 and hence in P at depth $\delta+1$. Hence the occurrences in V stay at the same depth in $M[V/x]$, while the let, $\mathsf{get}(r)$ and some x nodes disappear, hence $\omega_\delta(P) \leq \omega_\delta(P') - 2$. The number of occurrences of x in M is bounded by $k = \omega_{\delta+1}(P) \geq 2$. Thus if $j > \delta$ then $\omega_j(P') \leq k \cdot \omega_j(P)$. Clearly, $\omega_j(M) = \omega_j(M')$ if $j < i$. Hence, we have

  $$\mu_n(P') \leq (\mu_n^{i+1}(P) \cdot k, \omega_i(P) - 2, \mu_{i-1}(P)) \qquad \text{(A.3)}$$

  and $\mu_n(P) > \mu_n(P')$.

A.3 Proof of lemma 2.8

We start by remarking some basic inequalities.

Lemma A.1 (some inequalities)**.** *The following properties hold on natural numbers.*

1. $\forall\, x \geq 2, y \geq 0\ \ (y+1) \leq x^y$
2. $\forall\, x \geq 2, y \geq 0\ \ (x \cdot y) \leq x^y$
3. $\forall\, x \geq 2, y, z \geq 0\ \ (x \cdot y)^z \leq x^{(y \cdot z)}$
4. $\forall\, x \geq 2, y \geq 0, z \geq 1\ \ x^z \cdot y \leq x^{(y \cdot z)}$
5. *If* $x \geq y \geq 0$ *then* $(x-y)^k \leq (x^k - y^k)$

Proof. 1. By induction on y. The case for $y = 0$ is clear. For the inductive case, we notice:

$$(y+1)+1 \leq 2^y + 2^y = 2^{y+1} \leq x^{y+1}\ .$$

2. By induction on y. The case $y = 0$ is clear. For the inductive case, we notice:

$$\begin{array}{rcll} x \cdot (y+1) & \leq & x \cdot (x^y) & \text{(by (1))} \\ & = & x^{(y+1)} & \end{array}$$

3. By induction on z. The case $z = 0$ is clear. For the inductive case, we notice:

$$\begin{array}{rcll} (x \cdot y)^{z+1} & = & (x \cdot y)^z (x \cdot y) & \\ & \leq & x^{y \cdot z}(x \cdot y) & \text{(by inductive hypothesis)} \\ & \leq & x^{y \cdot z}(x^y) & \text{(by (2))} \\ & = & x^{y \cdot (z+1)} & \end{array}$$

4. From $z \geq 1$ we derive $y \leq y^z$. Then:

$$\begin{array}{rcll} x^z \cdot y & \leq & x^z \cdot y^z & \\ & = & (x \cdot y)^z & \\ & \leq & x^{y \cdot z} & \text{(by (3))} \end{array}$$

5. By the binomial law, we have $x^k = ((x-y)+y)^k = (x-y)^k + y^k + p$ with $p \geq 0$. Thus $(x-y)^k = x^k - y^k - p$ which implies $(x-y)^k \leq x^k - y^k$.

$\square$

We also need the following property.

Lemma A.2 (pre-shift)**.** *Assuming $\alpha \geq 1$ and $\beta \geq 2$, the following property holds for the tower functions with $x, \mathbf{x}$ ranging over numbers greater or equal than 2:*

$$\beta \cdot t_\alpha(x, \mathbf{x}) \leq t_\alpha(\beta \cdot x, \mathbf{x})$$

Proof. This follows from:

$$\beta \leq \beta^{2^{t_\alpha(\mathbf{x})}}$$

$\square$

Then we can derive the proof of the shift lemma as follows.

Let $k = t_\alpha(x', \mathbf{x}) \geq 2$. Then

$$\begin{array}{rll} t_\alpha(\beta \cdot x, x', \mathbf{x}) = \beta \cdot (\alpha \cdot x)^{2^k} & \leq (\alpha \cdot x)^{\beta \cdot 2^k} & \text{(by lemma A.1(3))} \\ & \leq (\alpha \cdot x)^{(\beta \cdot 2)^k} & \\ & \leq (\alpha \cdot x)^{2^{(\beta \cdot k)}} & \text{(by lemma A.1(3))} \end{array}$$

and by lemma A.2 $\beta \cdot t_\alpha(x', \mathbf{x}) \leq t_\alpha(\beta \cdot x', \mathbf{x})$.

Hence

$$(\alpha \cdot x)^{2^{(\beta \cdot k)}} \leq (\alpha \cdot x)^{2^{t_\alpha(\beta \cdot x', \mathbf{x})}} = t_\alpha(x, \beta \cdot x', \mathbf{x})$$

A.4 Proof of theorem 3.7

Suppose $\mu_\alpha(P) = (x_0, \ldots, x_\alpha)$ so that x_i corresponds to the occurrences at depth $(\alpha - i)$ for $0 \leq i \leq \alpha$. Also assume the reduction is at depth $(\alpha - i)$. By looking at equations (A.1) and (A.2) in the proof of termination (Proposition 3.6), we see that the components $i+1, \ldots, \alpha$ of $\mu_\alpha(P)$ and $\mu_\alpha(P')$ coincide. Hence, let $k = 2^{t_\alpha(x_{i+1}, \ldots, x_\alpha)}$. By definition of the tower function, $k \geq 1$.

We proceed by case analysis on the reduction rules.

- $P \equiv \mathsf{let}\ !x = !V\ \mathsf{in}\ M \to P' \equiv M[V/x]$
  By inequality (A.2) we know that:

$$\begin{array}{rcl} t_\alpha(\mu_\alpha(P')) & \leq & t_\alpha(x_0 \cdot x_{i-1}, \ldots, x_{i-1} \cdot x_{i-1}, x_i - 2, x_{i+1}, \ldots, x_\alpha) \\ & = & t_\alpha(x_0 \cdot x_{i-1}, \ldots, x_{i-1} \cdot x_{i-1}, x_i - 2)^k \end{array}$$

By iterating lemma 2.8, we derive:

$$\begin{array}{ll} & t_\alpha(x_0 \cdot x_{i-1}, x_1 \cdot x_{i-1}, \ldots, x_{i-1} \cdot x_{i-1}, x_i - 2) \\ \leq & t_\alpha(x_0, x_1 \cdot x_{i-1}^2, \ldots, x_{i-1} \cdot x_{i-1}, x_i - 2) \\ \leq & \ldots \\ \leq & t_\alpha(x_0, x_1, \ldots, x_{i-1}^i, x_i - 2) \end{array}$$

Renaming x_{i-1} with x and x_i with y, we are left to show that:

$$(\alpha x^i)^{2^{(\alpha \cdot (y-2))^k}} < (\alpha x)^{2^{(\alpha \cdot y)^k}}$$

Since $i \leq \alpha$ the first quantity is bounded by:

$$(\alpha x)^{\alpha \cdot 2^{(\alpha \cdot (y-2))^k}}$$

We notice:

$$\begin{array}{lll} & \alpha \cdot 2^{(\alpha \cdot (y-2))^k} & \\ = & \alpha \cdot 2^{(\alpha \cdot y - \alpha \cdot 2)^k} & \\ \leq & \alpha \cdot 2^{(\alpha \cdot y)^k - (\alpha \cdot 2)^k} & \quad \text{(by lemma A.1(5))} \end{array}$$

So we are left to show that:

$$\alpha 2^{(\alpha \cdot y)^k - (\alpha \cdot 2)^k)} \leq 2^{(\alpha \cdot y)^k}$$

Dividing by $2^{(\alpha \cdot y)^k}$ and recalling that $k \geq 1$, it remains to check:

$$\alpha \cdot 2^{-(\alpha \cdot 2)^k} \leq \alpha \cdot 2^{-(\alpha \cdot 2)} < 1$$

which is obviously true for $\alpha \geq 1$.

- $P \equiv (\lambda x.M)V \to P' \equiv M[V/x]$
  By equation (A.1), we have that:

  $$t_\alpha(\mu_\alpha(P')) \leq t_\alpha(x_0, \ldots, x_{i-1}, x_i - 2, x_{i+1}, \ldots, x_\alpha)$$

  and one can check that this quantity is strictly less than:

  $$t_\alpha(\mu_\alpha(P)) = t_\alpha(x_0, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_\alpha)$$

- $P \equiv \mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M \mid (r \leftarrow !V) \to P' \equiv M[V/x] \mid (r \leftarrow !V)$
  Let $k = 2^{t_\alpha(x_{i+1}, \ldots, x_\alpha)}$. By definition of the tower function, $k \geq 1$. By equation (A.3) we have

  $$\begin{array}{ll} t_\alpha(\mu_\alpha(P')) & \leq t_\alpha(x_0 \cdot x_{i-1}, \ldots, x_{i-1} \cdot x_{i-1}, x_i - 2, x_{i+1}, \ldots, x_\alpha) \\ & = t_\alpha(x_0 \cdot x_{i-1}, \ldots, x_{i-1} \cdot x_{i-1}, x_i - 2)^k \end{array}$$

  And we end up in the case of the rule for $\mathsf{let}\,!$.

- For the read that consume a value from the store, by looking at the proof of termination, we see that exactly one element of the vector $\mu_\alpha(P)$ is strictly decreasing during the reduction, hence one can check that $t_\alpha(\mu_\alpha(P)) > t_\alpha(\mu_\alpha(P'))$.
- The case for the write is similar to the read.

We conclude with the following remark that shows that the size of a program is proportional to its number of occurrences.

Remark A.3. The size of a program $|P|$ of depth d is at most twice the sum of its occurrences: $|P| \leq 2 \cdot \sum_{0 \leq i \leq d} \omega_i(P)$.

Hence the size of a program P is bounded by $t_d(\mu_d(P))$.

A.5 Proof of proposition 4.1

By induction on A.

- $A \equiv t'$
  We have
  $$\frac{R \vdash t' \quad t \notin R}{R \vdash \forall t.t'}$$
  If $t \neq t'$ we have $t'[B/t] \equiv t'$ hence $R \vdash [B/t]t'$. If $t \equiv t'$ then we have $t'[B/t] \equiv B$ hence $R \vdash t'[B/t]$.

- $A \equiv \mathbf{1}$
  We have
  $$\frac{R \vdash \mathbf{1} \quad t \notin R}{R \vdash \forall t.\mathbf{1}}$$
  from which we deduce $R \vdash \mathbf{1}[B/t]$.

- $A \equiv (C \multimap D)$
  By induction hypothesis we have $R \vdash C[B/t]$ and $R \vdash D[B/t]$. We then derive
  $$\frac{R \vdash C[B/t] \quad R \vdash D[B/t]}{R \vdash (C \multimap D)[B/t]}$$

- $A \equiv !C$
  By induction hypothesis we have $R \vdash C[B/t]$, from which we deduce
  $$\frac{R \vdash C[B/t]}{R \vdash !C[B/t]}$$

- $A \equiv \mathsf{Reg}_r C$
  We have
  $$\frac{\dfrac{R \vdash \quad r : C \in R}{R \vdash \mathsf{Reg}_r C} \qquad t \notin R}{R \vdash \forall t.\mathsf{Reg}_r C}$$

As $t \notin R$ and $r : (\delta, C) \in R$, we have $r : (\delta, C[B/t]) \in R$, from which we deduce

$$\frac{R \vdash \quad r : (\delta, C[B/t] \in R}{R \vdash \mathsf{Reg}_r\, C[B/t]}$$

- $A \equiv \forall t'.C$

  If $t \neq t'$: From $R \vdash \forall t.(\forall t'.C)$ we have $t' \notin R$ and by induction hypothesis we have $R \vdash C[B/t]$, from which we deduce

  $$\frac{R \vdash C[B/t] \qquad t' \notin R}{R \vdash (\forall t'.C)[B/t]}$$

  If $t \equiv t'$ we have $(\forall t'.C)[B/t] \equiv \forall t'.C$. Since we have

  $$\frac{R \vdash \forall t'.C \qquad t \notin R}{R \vdash \forall t.(\forall t'.C)}$$

  we conclude $R \vdash (\forall t'.C)[B/t]$.

A.6 Proof of theorem 4.4

Properties 1 and 2 are easily checked.

A.6.1 Substitution

If x is not free in M, we just have to check that any proof of $\Gamma, x : (\delta', A) \vdash^\delta M$ can be transformed into a proof of $\Gamma \vdash^\delta M$.

So let us assume x is free in M. Next, we proceed by induction on the derivation of $\Gamma, x : \delta' \vdash^\delta M$.

- $\Gamma, x : (\delta, A) \vdash^\delta x : A$. Then $\delta = \delta'$, $x[V/x] = V$, and by hypothesis $\Gamma \vdash^\delta V : A$.

- $\Gamma, x : (\delta', A) \vdash^\delta \lambda y.M : B \multimap C$ is derived from $\Gamma, x : (\delta', A), y : (\delta, B) \vdash^\delta M : C$, with $x \neq y$ and y not occurring in V. By (2), $\Gamma, y : (\delta, B) \vdash^{\delta'} V : A$. By inductive hypothesis, $\Gamma, (y : \delta, B) \vdash^\delta M[V/x] : C$, and then we conclude $\Gamma \vdash^\delta (\lambda y.M)[V/x] : B \multimap C$.

- $\Gamma, x : (\delta', A) \vdash^\delta (M_1 M_2) : C$ is derived from $\Gamma, x : (\delta', A) \vdash^\delta M_1 : B \multimap C$ and $\Gamma, x : (\delta', A) \vdash^\delta M_1 : B \multimap C$. By inductive hypothesis, $\Gamma \vdash^\delta M_1[V/x] : B \multimap C$ and $\Gamma \vdash^\delta M_2[V/x] : C$, and then we conclude $\Gamma \vdash^\delta (M_1 M_2)[V/x] : C$.

- $\Gamma, x : (\delta', A) \vdash^\delta !M : !B$ is derived from $\Gamma, x : (\delta', A) \vdash^{\delta+1} M : B$. By inductive hypothesis, $\Gamma \vdash^{\delta+1} M[V/x] : B$, and then we conclude $\Gamma \vdash^\delta !M[V/x] : !B$.

- $\Gamma, x : (\delta', A) \vdash^\delta \mathsf{let}\ !y = M_1\ \mathsf{in}\ M_2 : B$, with $x \neq y$ and y not free in V is derived from $\Gamma, x : (\delta', A) \vdash^\delta M_1 : C$ and $\Gamma, x : (\delta', A), y : (\delta+1, C) \vdash^\delta M_2 : B$. By inductive hypothesis, $\Gamma \vdash^\delta M_1[V/x] : C$ $\Gamma, y : (\delta+1, C) \vdash^\delta M_2[V/x] : B$, and then we conclude $\Gamma \vdash^\delta (\mathsf{let}\ !y = M_1\ \mathsf{in}\ M_2)[V/x] : B$.

- $M \equiv \mathsf{get}(r)$. We have $R, r : (\delta, B); \Gamma, x : (\delta', A) \vdash^\delta \mathsf{get}(r) : B$. Since $\mathsf{get}(r)[V/x] = \mathsf{get}(r)$ and $x \notin \mathsf{FV}(\mathsf{get}(r))$ then $R, r : (\delta, B); \Gamma \vdash^\delta \mathsf{get}(r)[V/x] : B$.

- $M \equiv \mathsf{set}(r, V')$. We have

$$\frac{R, r : (\delta, C); \Gamma, x : (\delta', A) \vdash^\delta V' : C}{R, r : (\delta, C); \Gamma, x : (\delta', A) \vdash^\delta \mathsf{set}(r, V') : \mathbf{1}}$$

By induction hypothesis we get

$$R, r : (\delta, C); \Gamma \vdash^\delta V'[V/x] : C$$

and hence we derive

$$R, r : (\delta, C); \Gamma \vdash^\delta (\mathsf{set}(r, V'))[V/x] : \mathbf{1}$$

- $M \equiv (M_1 \mid M_2)$. We have

$$\frac{R; \Gamma, x : (\delta', A) \vdash^\delta M_i : C_i \qquad i = 1, 2}{R; \Gamma, x : (\delta', A) \vdash^\delta (M_1 \mid M_2) : \mathbf{B}}$$

By induction hypothesis we derive

$$R; \Gamma \vdash^\delta M_i[V/x] : C_i$$

and hence we derive

$$R; \Gamma \vdash^\delta (M_1 \mid M_2)[V/x] : \mathbf{B}$$

A.6.2 Subject Reduction

We first state and sketch the proof of 4 lemmas.

Lemma A.4 (structural equivalence preserves typing)**.** *If $R; \Gamma \vdash^\delta P : \alpha$ and $P \equiv P'$ then $R; \Gamma \vdash^\delta P' : \alpha$.*

Proof. Recall that structural equivalence is the least equivalence relation induced by the equations stated in Table 3.2 and closed under static contexts. Then we proceed by induction on the proof of structural equivalence. This is is mainly a matter of reordering the pieces of the typing proof of P so as to obtain a typing proof of P'. □

Lemma A.5 (evaluation contexts and typing)**.** *Suppose that in the proof of $R; \Gamma \vdash^\delta E[M] : \alpha$ we prove $R; \Gamma' \vdash^{\delta'} M : \alpha'$. Then replacing M with a M' such that $R; \Gamma' \vdash^{\delta'} M' : \alpha'$, we can still derive $R; \Gamma \vdash^\delta E[M'] : \alpha$.*

Proof. By induction on the structure of E. □

Lemma A.6 (functional redexes). *If* $R;\Gamma \vdash^{\delta} E[\Delta] : \alpha$ *where* Δ *has the shape* $(\lambda x.M)V$ *or* $\mathsf{let}\ !x = !V\ \mathsf{in}\ M$ *then* $R;\Gamma \vdash^{\delta} E[M[V/x]] : \alpha$*.*

Proof. We appeal to the substitution lemma 3. This settles the case where the evaluation context E is trivial. If it is complex then we also need lemma A.5. □

Lemma A.7 (side effects redexes). *If* $R;\Gamma \vdash^{\delta} \Delta : \alpha$ *where* Δ *is one of the programs on the left-hand side then* $R;\Gamma \vdash^{\delta} \Delta' : \alpha$ *where* Δ' *is the corresponding program on the right-hand side:*

(1)	$E[\mathsf{set}(r,V)]$	$E[*] \mid (r \leftarrow V)$
(2)	$E[\mathsf{get}(r)] \mid (r \leftarrow V)$	$E[V]$
(3)	$E[\mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M] \mid (r \leftarrow !V)$	$E[M[V/x]] \mid (r \leftarrow !V)$

Proof. We proceed by case analysis.

1. Suppose we derive $R;\Gamma \vdash^{\delta} E[\mathsf{set}(r,V)] : \alpha$ from $R;\Gamma' \vdash^{\delta'} \mathsf{set}(r,V) : \mathbf{1}$. We can derive $R;\Gamma' \vdash^{\delta'} * : \mathbf{1}$ and by Lemma A.5 we derive $R;\Gamma \vdash^{\delta} E[*] : \alpha$ and finally $R;\Gamma \vdash^{\delta} E[\mathsf{set}(r,V)] \mid (r \leftarrow V) : \alpha$.

2. Suppose $R;\Gamma \vdash^{\delta} E[\mathsf{get}(r)] : \alpha$ is derived from $R;\Gamma \vdash^{\delta'} \mathsf{get}(r) : A$, where $r : (\delta', A) \in R$. Hence $R;\Gamma \vdash^{0} (r \leftarrow V) : \mathbf{B}$ is derived from $R;\Gamma \vdash^{\delta'} V : A$. Finally, by Lemma A.5 we derive $R;\Gamma \vdash^{\delta} E[V] : \alpha$.

3. Suppose $R;\Gamma \vdash^{\delta} E[\mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M] : \alpha$ is derived from

$$\frac{R;\Gamma' \vdash^{\delta'} \mathsf{get}(r) : !A \qquad R;\Gamma', x : (\delta'+1, A) \vdash^{\delta'} M : \alpha'}{R;\Gamma' \vdash^{\delta'} \mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M : \alpha'}$$

   where $r : (\delta', !A) \in R$. Hence $R;\Gamma \vdash^{0} (r \leftarrow !V) : \mathbf{B}$ is derived from $R;\Gamma \vdash^{\delta'+1} V : A$. By Lemma 3 we can derive $R;\Gamma' \vdash^{\delta'} M[V/x] : \alpha'$. Then by Lemma A.5 we derive $R;\Gamma \vdash^{\delta} E[M[V/x]] : \alpha$.

□

We are then ready to prove subject reduction. We recall that $P \to P'$ means that P is structurally equivalent to a program $C[\Delta]$ where C is a static context, Δ is one of the programs on the left-hand side of the rewriting rules specified in Table 3.2, Δ' is the respective program on the right-hand side, and P' is syntactically equal to $C[\Delta']$.

By lemma A.4, we know that $R;\Gamma \vdash^{\delta} C[\Delta] : \alpha$. This entails that $R';\Gamma' \vdash^{\delta'} \Delta : \alpha'$ for suitable $R', \Gamma', \alpha', \delta'$. By lemmas A.6 and A.7, we derive that $R';\Gamma' \vdash^{\delta'} \Delta' : \alpha'$. Then by induction on the structure of C we argue that $R;\Gamma \vdash^{\delta} C[\Delta'] : \alpha$.

A.6.3 Progress

To derive the progress property we first determine for each closed type A where $A = A_1 \multimap A_2$ or $A = !A_1$ the shape of a closed value of type A with the following classification lemma.

Lemma A.8 (classification)**.** *Assume* $R; - \vdash^{\delta} V : A$. *Then:*

- *if* $A = A_1 \multimap A_2$ *then* $V = \lambda x.M$*,*
- *if* $A = !A_1$ *then* $V = !V_1$

Proof. By case analysis on the typing rules.

- if $A = A_1 \multimap A_2$, the only typing rule that can be applied is

$$\frac{R; x : (\delta, A_1) \vdash^{\delta} M : A_2}{R; - \vdash^{\delta} \lambda x.M : A_1 \multimap A_2}$$

  hence $V = \lambda x.M$.

- if $A = !A_1$, the only typing rule that can be applied is

$$\frac{R; - \vdash^{\delta+1} V_1 : A_1}{R; - \vdash^{\delta} !V_1 : !A_1}$$

  hence $V = !V_1$.

□

Then we proceed by induction on the structure of the threads M_i to show that each one of them is either a value or a stuck get of the form $E[\Delta]$ where Δ can be $(\lambda x.M)\mathsf{get}(r)$ or $\mathsf{let}\ !x = \mathsf{get}(r)\ \mathsf{in}\ M$.

- $M_i = x$
  the case of variables is void since they are not closed terms.

- $M_i = *$ or $M_i = r$ or $M_i = \lambda x.M$
  these cases are trivial since $*$, r and $\lambda x.M$ are already values.

- $M_i = PQ$
  We know that PQ cannot reduce, which by looking at the evaluation contexts means that P cannot reduce. Then by induction hypothesis we have two cases: either P is a value or P is a stuck get.

  – assume P is a value. We have

$$\frac{R; - \vdash^{\delta} P : A \multimap B \qquad R; - \vdash^{\delta} Q : A}{R; - \vdash^{\delta} PQ : B}$$

    By Lemma A.8 we have $P = \lambda x.M$. Since PQ cannot reduce and $P = \lambda x.M$, by looking at the evaluation contexts we have that Q cannot reduce. Moreover Q cannot be a value, otherwise PQ is a redex. Hence by induction hypothesis Q is a stuck get of the form $E_1[\Delta]$. Hence PQ is of the form $E[\Delta]$ where $E = PE_1$.

– assume P is a stuck get of the form $E_1[\Delta]$. Then PQ is of the form $E[\Delta]$ where $E = E_1 Q$.

- $M_i = \mathsf{let}\ !x = P\ \mathsf{in}\ Q$
  We know that $\mathsf{let}\ !x = P\ \mathsf{in}\ Q$ cannot reduce, which by looking at the evaluation contexts means that P cannot reduce. Then by induction hypothesis we have two cases: either P is a value or P is a stuck get.

  – assume P is a value. We have

  $$\frac{R; - \vdash^{\delta} P : !A \qquad R; x : (\delta+1, A) \vdash^{\delta} Q : B}{R; - \vdash^{\delta} \mathsf{let}\ !x = P\ \mathsf{in}\ Q : B}$$

  By Lemma A.8 we have $P = !V$ hence $\mathsf{let}\ !x = !V\ \mathsf{in}\ Q$ is a redex and this contradicts the hypothesis that $\mathsf{let}\ !x = P\ \mathsf{in}\ Q$ cannot reduce. Thus P cannot be a value.

  – assume P is a stuck get of the form $E_1[\Delta]$. Then $\mathsf{let}\ !x = P\ \mathsf{in}\ Q$ is of the form $E[\Delta]$ where $E = \mathsf{let}\ !x = E_1\ \mathsf{in}\ Q$.

- $M_i = !P$
  We know that $!P$ cannot reduce, which by looking at the evaluation contexts means that P cannot reduce. Then by induction hypothesis we have two cases: either P is a value or P is a stuck get.

  – assume P is a value. Then $!P$ is also a value and we are done.

  – assume P is of the form $E_1[\Delta]$. Then $!P$ is of the shape $E[\Delta]$ where $E = !E_1$.

- $M_i = \mathsf{get}(r')$
  We know that $\mathsf{get}(r')$ cannot reduce which means that M_i is of the form $E[\Delta]$ where $r' = r$ and $E = []$ and that no value is associated with r in the store.

- $M_i = \mathsf{set}(r, V)$
  This case is void since $\mathsf{set}(r, V)$ can reduce is any case.

A.7 Proof of theorem 5.3

Elementary functions are characterized as the smallest class of functions containing zero, successor, projection, subtraction and which is closed by composition and bounded summation/product. We will need the arithmetic functions defined in Table A.1. We will abbreviate $\lambda^! x.M$ for $\lambda x.\mathsf{let}\ !x = x\ \mathsf{in}\ M$. Moreover, in order to represent some functions, we need to manipulate pairs in the language. We define the representation of pairs in Table A.2. In the following, we show that the required functions can be represented in the sense of Definition 5.2 by adapting the proofs from Danos and Joinet [10].

$$
\begin{array}{rcll}
\mathsf{N} & = & \forall t.!(t \multimap t) \multimap !(t \multimap t) & \text{(type of numerals)} \\
\\
\mathsf{zero} & : & \mathsf{N} & \text{(zero)} \\
\mathsf{zero} & = & \lambda f.!(\lambda x.x) & \\
\\
\mathsf{succ} & : & \mathsf{N} \multimap \mathsf{N} & \text{(successor)} \\
\mathsf{succ} & = & \lambda n.\lambda f.\mathsf{let}\ !f = f\ \mathsf{in} & \\
 & & \mathsf{let}\ !y = n!f\ \mathsf{in}!(\lambda x.f(yx)) & \\
\\
\overline{n} & : & \mathsf{N} & \text{(numerals)} \\
\overline{n} & = & \lambda f.\mathsf{let}\ !f = f\ \mathsf{in}\ !(\lambda x.f(\cdots(fx)\cdots)) & \\
\\
\mathsf{add} & : & \mathsf{N} \multimap (\mathsf{N} \multimap \mathsf{N}) & \text{(addition)} \\
\mathsf{add} & = & \lambda n.\lambda m.\lambda f.\mathsf{let}\ !f = f\ \mathsf{in} & \\
 & & \mathsf{let}\ !y = n!f\ \mathsf{in} & \\
 & & \mathsf{let}\ !y' = m!f\ \mathsf{in}\quad !(\lambda x.y(y'x)) & \\
\\
\mathsf{mult} & : & \mathsf{N} \multimap (\mathsf{N} \multimap \mathsf{N}) & \text{(multiplication)} \\
\mathsf{mult} & = & \lambda n.\lambda m.\lambda f.\mathsf{let}\ !f = f\ \mathsf{in} & \\
 & & n(m!f) & \\
\\
\mathsf{int\_it} & : & \mathsf{N} \multimap \forall t.!(t \multimap t) \multimap !t \multimap !t & \text{(iteration)} \\
\mathsf{int\_it} & = & \lambda n.\lambda g.\lambda x.\mathsf{let}\ !y = ng\ \mathsf{in} & \\
 & & \mathsf{let}\ !y' = x\ \mathsf{in}\ !(yy') & \\
\\
\mathsf{int\_git} & : & \forall t.\forall t'.!(t \multimap t) \multimap (!(t \multimap t) \multimap t') \multimap \mathsf{N} \multimap t' & \\
\mathsf{int\_git} & = & \lambda s.\lambda e.\lambda n.e(nts) &
\end{array}
$$

Table A.1: Representation of some arithmetic functions

A.7.1 Successor, addition and multiplication

We check that succ represents the *successor* function s:

$$
\begin{array}{l}
s : \mathbb{N} \mapsto \mathbb{N} \\
s(x) = x + 1
\end{array}
$$

Proposition A.9. $\mathsf{succ} \Vdash s$.

Proof. Take $\emptyset \vdash^{\delta} \underline{M} : \mathsf{N}$ and $M \Vdash n$. We have $\emptyset \vdash^{\delta} \mathsf{succ} : \mathsf{N} \multimap \mathsf{N}$. We can show that $\mathsf{succ}\ M \xrightarrow{*} \overline{s(n)}$, hence $\mathsf{succ}\ M \Vdash s(n)$. Thus $\mathsf{succ} \Vdash s$. $\square$

We check that add represents the *addition* function a:

$$
\begin{array}{l}
a : \mathbb{N}^2 \mapsto \mathbb{N} \\
a(x, y) = x + y
\end{array}
$$

$$
\begin{array}{rcll}
A \times B & = & \forall t.(A \multimap B \multimap t) \multimap t & \text{(type of pairs)} \\
\\
\langle M, N \rangle & : & A \times B & \text{(pair representation)} \\
\langle M, N \rangle & = & \lambda x.xMN & \\
\\
\mathsf{fst} & : & \forall t, t'.t \times t' \multimap t & \text{(left destructor)} \\
\mathsf{fst} & = & \lambda p.p(\lambda x.\lambda y.x) & \\
\\
\mathsf{snd} & : & \forall t, t'.t \times t' \multimap t' & \text{(right destructor)} \\
\mathsf{snd} & = & \lambda p.p(\lambda x.\lambda y.y) &
\end{array}
$$

Table A.2: Representation of pairs

Proposition A.10. $\mathsf{add} \Vdash a$.

Proof. For $i = 1, 2$ take $\emptyset \vdash^{\delta} M_i : \mathsf{N}$ and $M_i \Vdash n_i$. We have $\emptyset \vdash^{\delta} \mathsf{add} : \mathsf{N} \multimap \mathsf{N} \multimap \mathsf{N}$. We can show that $\mathsf{add} M_1 M_2 \xrightarrow{*} \overline{a(n_1, n_2)}$, hence $\mathsf{add} M_1 M_2 \Vdash a(n_1, n_2)$. Thus $A \Vdash a$. $\square$

We check that mult represents the multiplication function m:

$$
\begin{aligned}
&m : \mathbb{N}^2 \mapsto \mathbb{N} \\
&m(x, y) = x * y
\end{aligned}
$$

Proposition A.11. $\mathsf{mult} \Vdash m$.

Proof. For $i = 1, 2$ take $\emptyset \vdash^{\delta} M_i : \mathsf{N}$ and $M_i \Vdash n_i$. We have $\emptyset \vdash^{\delta} \mathsf{mult} : \mathsf{N} \multimap \mathsf{N} \multimap \mathsf{N}$. We can show that $\mathsf{mult}\ M_1 M_2 \xrightarrow{*} \overline{m(n_1, n_2)}$, hence $\mathsf{mult}\ M_1 M_2 \Vdash m(n_1, n_2)$. Thus $\mathsf{mult} \Vdash m$. $\square$

A.7.2 Iteration schemes

We check that $\mathsf{int\_it}$ represents the following iteration function it:

$$
\begin{aligned}
&it : (\mathbb{N} \mapsto \mathbb{N}) \mapsto \mathbb{N} \mapsto \mathbb{N} \mapsto \mathbb{N} \\
&it(f, n, x) = f^n(x)
\end{aligned}
$$

Proposition A.12. $\mathsf{int\_it} \Vdash it$.

Proof. We have $\emptyset \vdash^{\delta} \mathsf{int\_it} : \mathsf{N} \multimap \forall t.!(t \multimap t) \multimap !t \multimap !t$. Given $\emptyset \vdash^{\delta} M : \mathsf{N}$ with $M \Vdash n$, $\emptyset \vdash^{\delta} F : \mathsf{N} \multimap \mathsf{N}$ with $F \Vdash f$ and $\emptyset \vdash^{\delta} X : \mathsf{N}$ with $X \Vdash x$, we observe that $\mathsf{int\_it}\ M(!F)(!X) \xrightarrow{*} F^n X$. Since $F \Vdash f$ and $X \Vdash x$, we get $F^n X \xrightarrow{*} \overline{it(f, n, x)}$. Hence $\mathsf{int\_it} \Vdash it$. $\square$

The function it is an instance of the more general iteration scheme git:

$$
\begin{aligned}
&git : (\mathbb{N} \mapsto \mathbb{N}) \mapsto ((\mathbb{N} \mapsto \mathbb{N}) \mapsto \mathbb{N}) \mapsto \mathbb{N} \mapsto \mathbb{N} \\
&git(step, exit, n) = exit(\lambda x.step^n(x))
\end{aligned}
$$

Indeed, we have:

$$git(f, \lambda f.fx, n) = (\lambda f.fx)(\lambda x.f^n(x)) = it(f, n, x)$$

Proposition A.13. $\mathsf{int\_git} \Vdash git$.

Proof. Take $\emptyset \vdash^\delta M : \mathsf{N}$ with $M \Vdash n$, $\emptyset \vdash^\delta E : ((\mathsf{N} \multimap \mathsf{N}) \multimap \mathsf{N}) \multimap \mathsf{N}$ with $E \Vdash exit$, $\emptyset \vdash^\delta S : \mathsf{N} \multimap \mathsf{N}$ with $S \Vdash step$. Then we have $\mathsf{int\_git}\, S\, E\, M \xrightarrow{*} E(\lambda x.S^n x)$. Since $S \Vdash step$ and $E \Vdash exit$ we have $E(\lambda x.S^n x) \xrightarrow{*} \overline{exit(\lambda x.step^n(x))}$. Hence $\mathsf{int\_git} \Vdash git$. ☐

A.7.3 Coercion

Let $S = \lambda n^N.S'$. For $0 \geq i$, we define S'_i inductively:

$$\begin{aligned}&S'_0 = S'\\&S'_i = \mathsf{let}\ !n = n\ \mathsf{in}\ !S'_{i-1}\end{aligned}$$

Let $S_i = \lambda n.S'_i$. We can derive $\emptyset \vdash^\delta S_i : !^i\mathsf{N} \multimap !^i\mathsf{N}$. For $i \geq 0$, we define C_i inductively:

$$\begin{aligned}&C_0 = \lambda x.x\\&C_{i+1} = \lambda n.\mathsf{int\_it}(!S_i)(!^{i+1}\overline{0})n\\&\emptyset \vdash^\delta C_i : \mathsf{N} \multimap !^i\mathsf{N}\end{aligned}$$

Lemma A.14 (integer representation is preserved by coercion)**.** *Let $\emptyset \vdash^\delta M : \mathsf{N}$ and $M \Vdash n$. We can derive $\emptyset \vdash^\delta C_iM : !^i\mathsf{N}$. Moreover $C_iM \Vdash n$.*

Proof. By induction on i. ☐

Lemma A.15 (function representation is preserved by coercion)**.** *Let*

$$\emptyset \vdash^\delta F : !^{i_1}\mathsf{N}_1 \multimap \ldots \multimap !^{i_k}\mathsf{N}_k \multimap !^p\mathsf{N}$$

and $\emptyset \vdash^\delta M_j : \mathsf{N}$ with $M_j \Vdash n_j$ for $1 \leq j \leq k$ such that $F(!^{i_1}M_1 \ldots (!^{i_k}M_k)) \xrightarrow{*} \overline{f(n_1, \ldots, n_k)}$. Then we can find a term $\mathcal{C}(F) = \lambda\vec{x}^{\mathsf{N}}.F((C_{i_1}x_1)\ldots(C_{i_k}x_k))$ such that

$$\emptyset \vdash^\delta \mathcal{C}(F) : \mathsf{N} \multimap \mathsf{N} \multimap \ldots \multimap \mathsf{N} \multimap !^p\mathsf{N}$$

and $\mathcal{C}(F) \Vdash f$.

A.7.4 Predecessor and subtraction

We first want to represent *predecessor*:

$$\begin{aligned}&p : \mathbb{N} \mapsto \mathbb{N}\\&p(0) = 0\\&p(x) = x - 1\end{aligned}$$

We define the following terms:

$$ST = !(\lambda z.\langle \mathsf{snd}\ z, f(\mathsf{snd}\ z)\rangle)$$
$$f : (\delta+1, t \multimap t) \vdash^{\delta} ST : !(t \times t \multimap t \times t)$$

$$EX = \lambda g.\mathsf{let}\ !g = g\ \mathsf{in}\ !(\lambda x.\mathsf{fst}\ g\langle x, x\rangle)$$
$$\emptyset \vdash^{\delta} EX : !(t \times t \multimap t \times t) \multimap !(t \multimap t)$$

$$P = \lambda n.\lambda f.\mathsf{let}\ !f = f\ \mathsf{in}\ \mathsf{int\_git}\ ST\ EX\ n$$
$$\emptyset \vdash^{\delta} P : \mathsf{N} \multimap \mathsf{N}$$

Proposition A.16 (predecessor is representable)**.** $P \Vdash p$.

Proof. Take $\emptyset \vdash^{\delta} M : \mathsf{N}$ and $M \Vdash n$. We can show that $(PM)^{-} \xrightarrow{*} \overline{p(n)}$, hence $PM \Vdash p(n)$. Thus $P \Vdash p$. □

Now we want to represent (positive) subtraction s:

$$s : \mathbb{N}^2 \mapsto \mathbb{N}$$
$$s(x, y) = \begin{cases} x - y & \text{if } x \geq y \\ 0 & \text{if } y \geq x \end{cases}$$

Take

$$SUB = \lambda m.\mathsf{let}\ !m = m\ \mathsf{in}\ \lambda n.\mathsf{int\_it}\ !P\ !m\ n : !\mathsf{N} \multimap \mathsf{N} \multimap !\mathsf{N}$$
$$\emptyset \vdash^{\delta} SUB : !\mathsf{N} \multimap \mathsf{N} \multimap !\mathsf{N}$$

Proposition A.17 (subtraction is representable)**.** $\mathcal{C}(SUB) \Vdash s$.

Proof. For $i = 1, 2$ take $\emptyset \vdash^{\delta} M_i : \mathsf{N}$ and $M_i \Vdash n_i$. We can show that $(SUB(!M_1)M_2)^{-} \xrightarrow{*} \overline{s(n_1, n_2)}$. Hence by Lemma A.15, $\mathcal{C}(SUB) \Vdash s$. □

A.7.5 Composition

Let g be a m-ary function and G be a term such that $\emptyset \vdash^{\delta} G : \mathsf{N}_1 \multimap \ldots \multimap \mathsf{N}_m \multimap !^p\mathsf{N}$ (where $p \geq 0$) and $G \Vdash g$. For $1 \leq i \leq m$, let f_i be a k-ary function and F_i a term such that $\emptyset \vdash^{\delta} F_i : \mathsf{N}_1 \multimap \ldots \multimap \mathsf{N}_k !^{q_i}\mathsf{N}$ (where $q_i \geq 0$) and $F_i \Vdash f_i$. We want to represent the composition function h such that:

$$h : \mathbb{N}^k \mapsto \mathbb{N}$$
$$h(x_1, \ldots, x_k) = g(f_1(x_1, \ldots, x_k), \ldots, f_m(x_1, \ldots, x_k))$$

For $i \geq 0$ and a term T, we define T^i inductively as:

$$T^0 = T$$
$$T^i = \lambda \vec{x}^{!^i\mathsf{N}}.\mathsf{let}\ !\vec{x} = \vec{x}\ \mathsf{in}\ !(T^{i-1}\vec{x})$$

Let $q = max(q_i)$. We can derive

$$\emptyset \vdash^{\delta} G^{q+1} : !^{q+1}\mathsf{N}_1 \multimap \ldots \multimap !^{q+1}\mathsf{N}_m \multimap !^{p+q+1}\mathsf{N}$$

We can also derive

$$\emptyset \vdash^{\delta} F_i^{q-q_i} : !^{q-q_i}\mathsf{N}_1 \multimap \ldots \multimap !^{q-q_i}\mathsf{N}_k \multimap !^{q}\mathsf{N}$$

Then, applying coercion we get

$$\emptyset \vdash^{\delta} \mathcal{C}(F_i^{q-q_i}) : \mathsf{N}_1 \multimap \ldots \mathsf{N}_k \multimap !^{q}\mathsf{N}$$

and we derive

$$x_1 : (\delta+1, \mathsf{N}), \ldots, x_k : (\delta+1, \mathsf{N}) \vdash^{\delta} !(\mathcal{C}(F_i^{q-q_i})x_1 \ldots x_k) : !^{q+1}\mathsf{N}$$

Let $F_i' \equiv !(\mathcal{C}(F_i^{q-q_i})x_1 \ldots x_k)$. By application we get

$$x_1 : (\delta+1, \mathsf{N}), \ldots, x_k : (\delta+1, \mathsf{N}) \vdash^{\delta} G^{q+1}F_1' \ldots F_m' : !^{p+q+1}\mathsf{N}$$

We derive

$$\emptyset \vdash^{\delta} \lambda\vec{x}.\mathsf{let}\ !\vec{x} = \vec{x}\ \mathsf{in}\ G^{q+1}F_1' \ldots F_m' : !\mathsf{N}_1 \multimap \ldots \multimap !\mathsf{N}_m \multimap !^{p+q+1}\mathsf{N}$$

Applying coercion we get

$$\emptyset \vdash^{\delta} \mathcal{C}(\lambda\vec{x}^{!\mathsf{N}}.\mathsf{let}\ !\vec{x} = \vec{x}\ \mathsf{in}\ G^{q+1}F_1' \ldots F_m') : \mathsf{N}_1 \multimap \ldots \multimap \mathsf{N}_m \multimap !^{p+q+1}\mathsf{N}$$

Take

$$H = \mathcal{C}(\lambda\vec{x}^{!\mathsf{N}}.\mathsf{let}\ !\vec{x} = \vec{x}\ \mathsf{in}\ G^{q+1}F_1' \ldots F_m')$$

Proposition A.18 (composition is representable)**.** *$H \Vdash h$.*

Proof. We now have to show that for all M_i and n_i where $1 \leq i \leq k$ such that $M_i \Vdash n_i$ and $\emptyset \vdash^{\delta} M_i : \mathsf{N}$, we have $HM_1 \ldots M_k \Vdash h(n_1, \ldots, n_k)$. Since $F_i \Vdash f_i$, we have $F_iM_1 \ldots M_k \Vdash f_i(n_1, \ldots, n_k)$. Moreover $G \Vdash g$, hence

$$G(F_1M_1 \ldots M_k) \ldots (F_mM_1 \ldots M_k) \Vdash g(f_1(n_1, \ldots, n_k), \ldots, f_m(n_1, \ldots, n_k))$$

We can show that $HM_1 \ldots M_k \xrightarrow{*} G(F_1M_1 \ldots M_k) \ldots (F_mM_1 \ldots M_k)$, hence

$$HM_1 \ldots M_k \Vdash g(f_1(n_1, \ldots, n_k), \ldots, f_m(n_1, \ldots, n_k))$$

Thus $H \Vdash h$. ∎

A.7.6 Bounded sums and products

Let f be a $k+1$-ary function $f : \mathbb{N}^{k+1} \to \mathbb{N}$, where

$$\emptyset \vdash F : \mathsf{N}_i \multimap \mathsf{N}_1 \multimap \ldots \multimap \mathsf{N}_k \multimap !^{p}\mathsf{N}$$

with $p \geq 0$ and $F \Vdash f$. We want to represent

- bounded sum: $\sum_{1 \leq i \leq n} f(i, x_1, \ldots, x_k)$
- bounded product: $\prod_{1 \leq i \leq n} f(i, x_1, \ldots, x_k)$

For this we are going to represent $h : \mathbb{N}^{k+1} \to \mathbb{N}$:

$$\begin{aligned} h(0, x_1, \ldots, x_k) &= f(0, x_1, \ldots, x_k) \\ h(n+1, x_1, \ldots, x_k) &= g(f(n+1, x_1, \ldots, x_k), h(n, x_1, \ldots, x_k)) \end{aligned}$$

where g is a binary function standing for addition or multiplication, thus representable. More precisely we have $g : \mathbb{N}^2 \to \mathbb{N}$ such that $\emptyset \vdash^{\delta} G : \mathsf{N} \multimap \mathsf{N} \multimap \mathsf{N}$ and $G \Vdash g$.

For $i \geq 0$ and a term T we define T^i inductively:

$$\begin{aligned} T^0 &= T x_1 \ldots x_k \\ T^i &= \mathsf{let}\ !x_1 = x_1\ \mathsf{in}\ \ldots \mathsf{let}\ !x_k = x_k\ \mathsf{in}\ !T^{i-1} \end{aligned}$$

We define the following terms:

$$\begin{aligned} &ST = \lambda z.\langle S(\mathsf{fst}\ z), G^p(F x_1 \ldots x_k (S(\mathsf{fst}\ z)))(\mathsf{snd}\ z)\rangle \\ &\emptyset; x_1 : (\delta, \mathsf{N}), \ldots, x_k : (\delta, \mathsf{N}) \vdash^{\delta} ST : \mathsf{N} \times !^p\mathsf{N} \multimap \mathsf{N} \times !^p\mathsf{N} \end{aligned}$$

$$\begin{aligned} &EX = \lambda h.\mathsf{let}\ !h = h\ \mathsf{in}\ !\mathsf{snd}\ h\langle \overline{0}, F x_1 \ldots x_k \overline{0}\rangle \\ &\emptyset; x_1 : (\delta+1, \mathsf{N}), \ldots, x_k : (\delta+1, \mathsf{N}) \vdash^{\delta} EX : !(\mathsf{N} \times !^p\mathsf{N} \multimap \mathsf{N} \times !^p\mathsf{N}) \multimap !^{p+1}\mathsf{N} \end{aligned}$$

We derive

$$n : (\mathsf{N}), \vec{x} : (\delta, \mathsf{N}) \vdash^{\delta} \mathsf{let}\ !\vec{x} = \vec{x}\ \mathsf{in}\ \mathsf{let}\ !n = n\ \mathsf{in}\ \mathsf{int\_git}\ !ST\ EX\ n : !^{p+1}\mathsf{N}$$

Let $R = \mathsf{let}\ !\vec{x} = \vec{x}\ \mathsf{in}\ \mathsf{let}\ !n = n\ \mathsf{in}\ \mathsf{int\_git}\ !ST\ EX\ n$. By coercion and abstractions we get

$$\emptyset \vdash^{\delta} \mathcal{C}(\lambda n.\lambda \vec{x}.R) : \mathsf{N}_i \multimap \mathsf{N}_1 \multimap \ldots \multimap \mathsf{N}_k \multimap !^{p+1}\mathsf{N}$$

Take $H = \mathcal{C}(\lambda n.\lambda \vec{x}.R)$.

Proposition A.19 (bounded sum/product is representable)**.** *$H \Vdash h$.*

Proof. Given $M_i \Vdash i$ and $M_j \Vdash n_j$ with $1 \leq j \leq k$ and taking G for addition, we remark that

$$H M_i M_1 \ldots M_k \xrightarrow{*} \overline{f(i, n_1, \ldots, n_k) + \ldots + f(1, n_1, \ldots, n_k) + f(0, n_1, \ldots, n_k)}$$

Hence $H \Vdash h$. $\square$